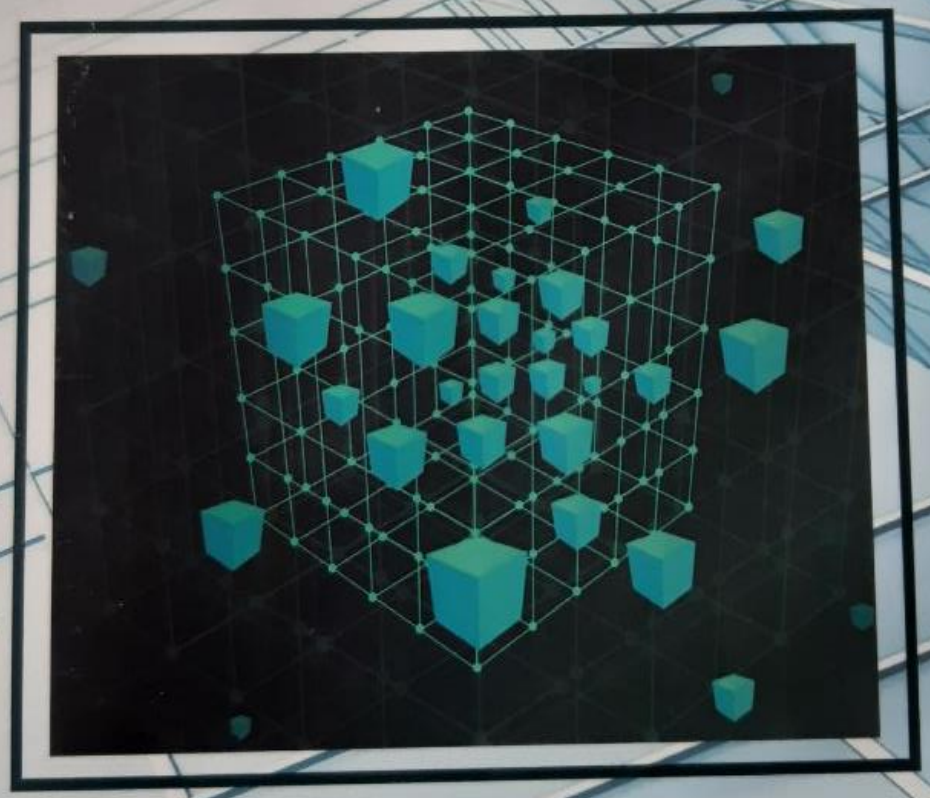
Ergashev A.Q., Akbaraliyev B.B., Yusupova Z.Dj.

MA'LUMOTLAR TUZILMASI VA ALGORITMLAR



O'ZBEKISTON RESPUBLIKASI RAQAMLI TEXNOLOGIYALAR VAZIRLIGI

MUHAMMAD AL-XORAZMIY NOMIDAGI TOSHKENT AXBOROT TEXNOLOGIYALARI UNIVERSITETI

Ergashev A.Q., Akbaraliyev B.B., Yusupova Z.Dj.

MA'LUMOTLAR TUZILMASI VA ALGORITMLAR

Muhammad Al-Xorazmiy nomidagi Toshkent axborot texnologiyalari universiteti tomonidan o'quv qo'llanma sifatida tavsiya etilgan

TATU barcha ta'lim yo'nalish talabalari va professor-o'qituvchilari uchun

Toshkent
"METODIST NASHRIYOTI"

UO'K: 004.043(075)
KBK: 32.973ya7
E – 74

Ergashev A.Q.
Ma'lumotlar tuzilmasi va algoritmlar. / o'quv qo'llanma/ Akbaraliyev B.B., Yusupova Z.Dj., , Toshkent: - "METODIST NASHRIYOTI", 2024. – 188 b.

O'quv qo'llanma dasturiy ta'minot ishlab chiqishni zamonaviy metodologiyasining asosi bo'lgan ma'lumotlar tuzilmasi va algoritmlarga bag'ishlangan. Mazkur o'quv qo'llanmada abstrakt ma'lumotlarni standart turlari va ular ustidagi amallar, abstrakt ma'lumotlar tuzilmasi va ulardagi amallar, qidiruv va saralash algoritmlari, xesh funksiya va xeshlash algoritmlari, interativ va rekursiv algoritmlar, sinf va funksiyalarning shablonlari, dasturlash tilidagi sinflar, standart ma'lumotlar tuzilmasi kutubxonasi bo'yicha ma'lumotlar, ro'yxat ko'rinishidagi ma'lumotlar tuzilmasi, ommaviy xizmat ko'rsatish va uning amalga oshirish yo'llari, daraxtsimon va ko'ptarmoqli ma'lumotlar tuzilmasi, dasturiy ta'minotni testlash va tekshirish bo'yicha ma'lumotlar keltirib o'tilgan.

O'quv qo'llanma 5 ta bo'limdan iborat:
– ma'lumotlar, tuzilmalar, rekursiya va obyektga yo'naltirilgan dasturlash;
– ma'lumotlarni qidirish va saralash usullari va algoritmlar;
– chiziqli ma'lumotlar tuzilmasi;
– chiziqsiz ma'lumotlar tuzilmasi;
– dasturiy ta'minotni testlash va tekshirish, ma'lumotlar tuzilmalarini modellashtirish.

O'quv qo'llanma o'quvchidan maxsus tayyorgarlikni talab etmaydi, faqatgina biror bir yuqori darajali dasturlash tillari, masalan, C++ bilan tanish bo'lishi yetarli.

O'quv qo'llanma Muhammad al-Xorazmiy nomidagi Toshkent axborot texnologiyalari universiteti va uning filiallarida tahsil olayotgan barcha ta'lim yo'nalishi talabalari uchun mo'ljallangan. Bundan tashqari o'quv qo'llanma dastur va algoritmlar ishlab chiqish bilan shug'ullanadigan mutaxasssislar va professorlar-o'qituvchilar uchun ham foydali bo'lishi mumkin.

Taqrizchilar:

M.S.Hodjayeva – O'zbekiston Xalqaro Islom akademiyasi «Zamonaviy axborot-kommunikatsiya texnologiyalari» kafedrasi dotsenti, t.f.n.

A.T.Rahmanov – Muhammad al-Xorazmiy nomidagi Toshkent axborot texnologiyalari universiteti «Tizimli va amaliy dasturlashtirish» kafedrasi dotsenti, f.-m.f.n.

Muhammad Al-Xorazmiy nomidagi Toshkent axborot texnologiyalari universitetining 2022–yil 22–dekabrdagi 5(727)–sonli qaroriga asosan nashr etishga ruxsat berilgan

ISBN 978-9910-03-142-7

KIRISH

Ma'lumotlar tuzilmasi (data structure) — bu dasturiy birlik bo'lib, u hisoblash texnikasida bir turdagi va/yoki mantiqan bog'langan ma'lumotlar to'plamini saqlash va qayta ishlash imkoniyatini beradi.

Axborot-kommunikatsiya texnologiyalarini rivojlantirish va uni amaliyotga joriy qilishda dasturlashning ahamiyati beqiyosdir.

Elektron hisoblash mashinalari (EHM) yaratilgan vaqtlarda, dastlab dasturlashning vazifasi EHM yordamida faqatgina matematik hisoblashlarni amalga oshirishga qaratilgan bo'lsa, hozirgi kunga kelib dasturlash obyektlari turli-tumanligi, murakkabligi va unga ta'sir etuvchi bir qator omillar dasturlarni ishlab chiqishga tizimli va ilmiy yondashuvni talab etmoqda. Ayniqsa, murakkab tuzilmaga ega jarayonlar uchun katta hajmdagi dasturlar ishlab chiqilayotganda bunday yondashuv naqadar muhimligini ko'rishimiz mumkin.

Ma'lumki, dasturlash usullari o'z ichiga ma'lumotlar tuzilmasining barcha variantlarini qamrab oladi. Dasturlar esa ma'lumotlarning ma'lum bir ifodasi va tuzilmasiga asoslangan holda abstrakt algoritmlar tahririni namoyon etadi.

Kompyuterda hisoblash jarayoni dasturlar va ma'lumotlar yordamida amalga oshiriladi. Ma'lumotlar esa dastur tomonidan foydalanilishi yoki shakllantirilishi mumkin. Boshqa tomondan qaraganda, dasturni o'zi ham ma'lumotlar majmuasini namoyon qilganligi sababli, kompyuter qayta ishlashi mumkin bo'lgan ixtiyoriy axborotni ma'lumotlar tasniflaydi, deb hisoblash mumkin. Bunda qayta ishlanayotgan axborot qaysidir ma'noda real olamning ma'lum bir qismini abstraksiyasini namoyon qiladi. Ma'lumotlar voqelikni abstraksiyasi hisoblanadi, chunki ular real obyektlarning hal qilinayotgan masala uchun ahamiyati muhim bo'lmagan ba'zi xossa va xususiyatlarini e'tiborga olmaydi. Boshqacha qilib aytganda abstraksiya bu voqelikni soddalashtirishdir.

Dastur ishlab chiqishda shuni e'tiborga olish muhimki, har doim ham abstrakt ma'lumotlarni bir xil ko'rinishda ifodalash murakkab bo'ladi, bunga sabab berilgan ma'lumotlar ustida turlicha amallar bajarilishi talab etilishi yoki boshqa cheklanishlar bo'lishi mumkin. Shu sababli, ma'lumotlarni ifodalanishini to'g'ri tanlash muhim ahamiyat kasb etadi.

Ma'lumki, matematikada o'zgaruvchilar biror-bir muhim xususiyatlarga mos ravishda sinflanadi. Ma'lumotlar qayta

ishlanayotganda ham shu kabi sinflashtirish muhim ahamiyat kasb etadi. Ma'lumotlar qayta ishlanayotganda har bir o'zgarmas (konstanta), o'zgaruvchi, ifoda yoki funksiya ma'lum bir toifaga tegishli bo'lishi lozim. Mazkur toifa konstanta, o'zgaruvchi, ifoda qabul qilishi yoki funksiya qayta ishlashi mumkin bo'lgan qiymatlar to'plamini tavsiflaydi. Toifa – elementar yoki notuzilmaviy berilganlarning muhim xususiyati hisoblanadi.

Ko'p hollarda, yangi toifalar oldindan mavjud bo'lgan ma'lumotlar toifalaridan foydalanib yaratiladi. Bunday toifalarga tegishli qiymatlar oldindan mavjud bo'lgan toifa komponentalari qiymatlari to'plami ko'rinishida bo'ladi. Bunday tarkibga ega qiymatlar tuzilmaviy qiymatlar deyiladi.

Ushbu o'quv qo'llanmada yuqorida keltirilgan mavzularga oid ma'lumotlar, misollarning C++ dasturlash tilidagi dastur kodlari keltirilgan bo'lib, Toshkent axborot texnologiyalari universitetining barcha ta'lim yo'nalishlari talabalari va professor-o'qituvchilar uchun mo'ljallangan.

I BO'LIM
MA'LUMOTLAR, TUZILMALAR VA OBYEKTGA YO'NALTIRILGAN DASTURLASH

Ma'lumotlar tuzilmasi tanlangan dasturlash tiliga bog'liq ravishda ma'lumotlar, ko'rsatkichlar va ular ustidagi amallar orqali shakllantiriladi.

Turli xil ma'lumotlar tuzilmalari turlicha dasturiy ilovalarni ishlab chiqishga, ba'zilari esa tor ixtisoslikka ega bo'lib, aniq bir toifadagi masalalarni hal qilishga mo'ljallangan bo'ladi. Masalan, B-daraxt ko'rinishidagi ma'lumotlar tuzilmasidan ma'lumotlar bazasini yaratishda, xesh-jadvallardan esa turli lug'atlarni, xususan, kompyuter internet-manzillari domen nomlarini ifodalashda foydalanish maqsadga muvofiq bo'ladi.

Dasturiy ta'minotni ishlab chiqishda dasturlarning amalga oshirish murakkabligi, samaradorligi va ishlash sifati ma'lumotlar tuzilmasiga bog'liq bo'lib, uni to'g'ri tanlash muhim ahamiyatga ega. Shuni esdan chiqarmaslik lozimki, zamonaviy dasturlashda dasturiy vosita arxitekturasi negizida algoritmlar emas, aynan ma'lumotlar tuzilmasi turadi. Shu sababli, nafaqat dasturiy ta'minot ishlab chiqishda, balki dasturlash tillarini yaratishda ham shunga e'tibor qaratish lozim bo'ladi.

Hozirda ko'plab zamonaviy dasturlash tillari turli ilovalarda ma'lumotlar tuzilmasidan xavfsiz qayta foydalanish imkonini beruvchi modullarga ega. Bularga misol qilib, Java, C# va C++ obyektga-yo'naltirilgan dasturlash tillarini keltirib o'tish mumkin.

Bir qator klassik ma'lumotlar tuzilmalari dasturlash tillarining standart kutubxonalariga yoki bevosita dasturlash tilining o'ziga kiritilgan. Masalan, xesh-jadval ma'lumotlar tuzilmasi Lua, Perl, Tcl, Python va boshqa bir qator dasturlash tillarining o'ziga kiritilgan. C++ tilida esa shablonlar standart kutubxonasidan keng foydalaniladi.

Ushbu bo'limda fanning asosiy tushunchalari, xususan, ma'lumot tushunchasi, ma'lumotlarni ifodalash bosqichlari, ma'lumotlar turlari va tarkibi, ma'lumotlar abstraksiyasi va ularning abstrakt tuzilmalari, shu bilan birga, obyektga yo'naltirilgan dasturlashdagi tushunchalar: dasturlash tilida sinflar (class), do'stona funksiyalar, istisno holatlarni qayta ishlash, xotirani ajratish va taqsimlash, inkapsulyatsiya, merosxo'rlik, polimorfizm, virtual funksiyalar, sinf va funksiyalar shablonlari, sinf va funksiyalar shablonlarini dasturlash, shablonlarning standart kutubxonasi (STL), STL-komponentalari keltirib o'tilgan.

1.1. Ma'lumot va ma'lumotlar tuzilmasi tushunchalari

1.1.1. Asosiy tushuncha va ta'riflar

Ma'lumot - bu biror bir obyekt, jarayon, hodisa yoki voqelikni ifodalab (tasniflab) beruvchi belgi yoki belgilar majmuasidir.

Ma'lumotlarning abstrakt turi (MAT)— bu matematik model va mazkur model doirasida aniqlangan amallar majmuasidir. Masalan, ma'lumotlarning abstrakt turiga oddiy misol sifatida to'plamlarning birlashmasi, kesishmasi va ayirmasi kabi amallarni o'z ichiga olgan butun sonlar to'plamini keltirish mumkin.

Abstraksiya (obyektga yo'naltirilgan dasturlashda) – bu qaralayotgan tizimda obyektni yetarlicha aniqlikda namoyon qiladigan tavsiflari. Abstraksiyaning asosiy g'oyasi qo'yilgan masalani yetarlicha aniqlikda hal qilish uchun obyektni ifodalovchi minimal sondagi maydonlar majmuasi va usullarini aniqlash.

Abstraksiya ham polimorfizm, merosxo'rlik va inkapsulyatsiyalar kabi obyektga yo'naltirilgan dasturlashning muhim vositalaridan biridir.

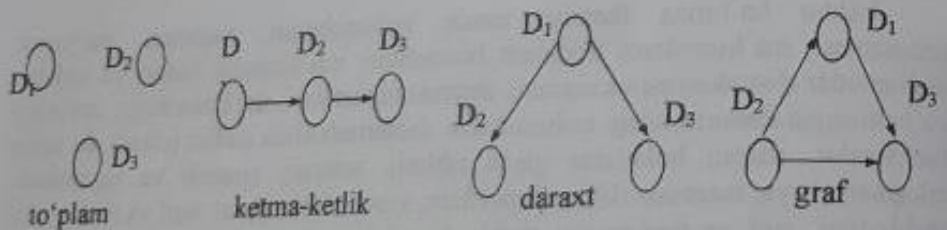
Ma'lumotlar abstraksiyasi – bu interfeys va amalga oshirishni (realizatsiya) ajratishga asoslangan dasturlash usuli, ya'ni tafsilotlarni ko'rsatmasdan dasturda kerakli axborotlarni taqdim etishdir. Masalan, C++ dasturlash tilida ***sort()*** funksiyasiga murojaat qiladigan bo'lsak, u yordamida berilgan massiv elementlarini saralashimiz mumkin, lekin bu funksiya saralashni qanday va qaysi algoritm asosida amalga oshirayotganligini bilmaymiz. Bu yerda ***sort*** interfeys bo'lib xizmat qiladi.

Ma'lumotlar tuzilmasi - bu tuzilmani tashkil qiluvchi elementlar (ma'lumotlar) va ular orasidagi bog'liqlikni ko'rsatib beruvchi munosabatlar majmuasidir.

Agar dasturlash nuqtai-nazaridan qaraydigan bo'lsak, u holda ma'lumotlar tuzilmasi hisoblash texnikasida bir turdagi va/yoki mantiqan bog'langan ma'lumotlar to'plamini saqlash va qayta ishlash imkoniyatini beruvchi dasturiy birlikdir.

Ma'lumotlar tuzilmasi o'zining quyidagi xossalari bilan tasniflanadi:

- ☐ qabul qilishi mumkin bo'lgan qiymatlari to'plami;
- ☐ mumkin bo'lgan amallar (operatsiyalar) majmuasi;
- ☐ tashkil etilganlik tasnifi.



1.1-rasm. Ma'lumotlar tuzilmasiga misollar (D_1, D_2, D_3 - elementlar)

1.1.2. Ma'lumotlarni ifodalash bosqichlari

Odatda, kompyuter xotirasida ma'lumotlarni ifodalash uch bosqichda amalga oshiriladi:

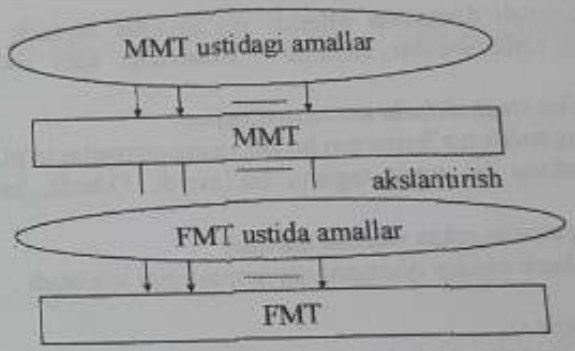
1) abstrakt (matematik) bosqich;
2) mantiqiy bosqich;
3) fizik bosqich.

Abstrakt (matematik) bosqichda ixtiyoriy tuzilmani <D,R> juftlik ko'rinishda ifodalash mumkin, bu yerda D – elementlarning chekli to'plami bo'lib, elementlar ma'lumotlar turlari yoki ma'lumotlar tuzilmasi bo'lishi mumkin, R – esa munosabatlar to'plami bo'lib, mazkur munosabatlar xususiyatlari abstrakt bosqichda ma'lumotlar tuzilmalari turlarini aniqlaydi.

Mantiqiy bosqich - ma'lumotlar tuzilmasini biror bir dasturlash tilida ifodalanishidir.

Fizik bosqich - informatsion obyektni mantiqiy tavsiflanishiga mos ravishda kompyuter xotirasiga akslantirilish. Kompyuter xotirasi chekli bo'lganligi sababli, xotirani taqsimlash va uni boshqari muammosi yuzaga keladi.

Mantiqiy bosqich bilan fizik bosqichlar bir biridan farq qilganligi uchun, hisoblash tizimlarida mantiqiy bosqichni fizik bosqichga va aksincha, fizik bosqichni mantiqiy bosqichga akslantirish amalga oshiriladi.



1.2-rasm. Ma'lumotlarni ifodalash bosqichlari

Bu yerda MMT – mantiqiy ma'lumotlar tuzilmasi, FMT – fizik ma'lumotlar tuzilmasi.

Ma'lumotlar tuzilmasini asosiy ko'rinishlari (turlari):

1. To'plam – munosabat to'plami bo'sh, ya'ni R=∅ bo'lgan elementlar majmuasi.
2. Ketma-ketlik – shunday abstrakt tuzilmaki, bunda R to'plam faqatgina bitta chiziqli munosabatdan iborat (ya'ni, birinchi va ohirgi elementdan tashqari har bir element uchun o'zidan oldin va keyin keladigan element mavjud.
3. Matritsa – shunday tuzilmaki, bunda R munosabatlar to'plami ikkita chiziqli munosabatdan tashkil topgan bo'ladi.
4. Daraxt – bunda R to'plam ierarxik tartibdagi bitta munosabatdan tashkil topgan bo'ladi.
5. Graf – bunda R munosabatlar to'plami faqatgina bitta binar tartibli munosabatdan tashkil topgan bo'ladi.
6. Gipergraf – bu shunday ma'lumotlar tuzilmasiki, bunda R to'plam ikki yoki undan ortiq turli tartibdagi munosabatlardan tashkil topgan bo'ladi.

1.1.3. Ma'lumotlar turlari

Turli adabiyot va standartlarda ma'lumotlar turi tushunchasiga turlicha ta'riflar keltirib o'tilgan. Masalan,

Ma'lumotlar turi (tur) — qiymatlar to'plami va mazkur qiymatlar ustidagi amallar *[IEEE Std 1320.2-1998 (R2004) IEEE Standard for Conceptual Modeling Language Syntax and Semantics for IDEFIX97:* **a set of values and operations on those values***]*.

Ma'lumotlar turi (tur) — sinf elementlari va ularga tatbiq qilish mumkin bo'lgan amallar orqali tavsiflanuvchi ma'lumotlar sinfi *[ISO/IEC/IEEE 24765-2010 Systems and software engineering — Vocabulary:* **a class of data, characterized by the members of the class and the operations that can be applied to them***]*.

Odatda, ko'plab dasturlash tillarida ma'lumotlar tayanch va keltirilgan turlarga ajratiladi. Ma'lumotlar turlarini 1.3-rasmdagi kabi sinflarga ajratish mumkin.

Quyida har bir turga alohida to'xtalib o'tamiz.

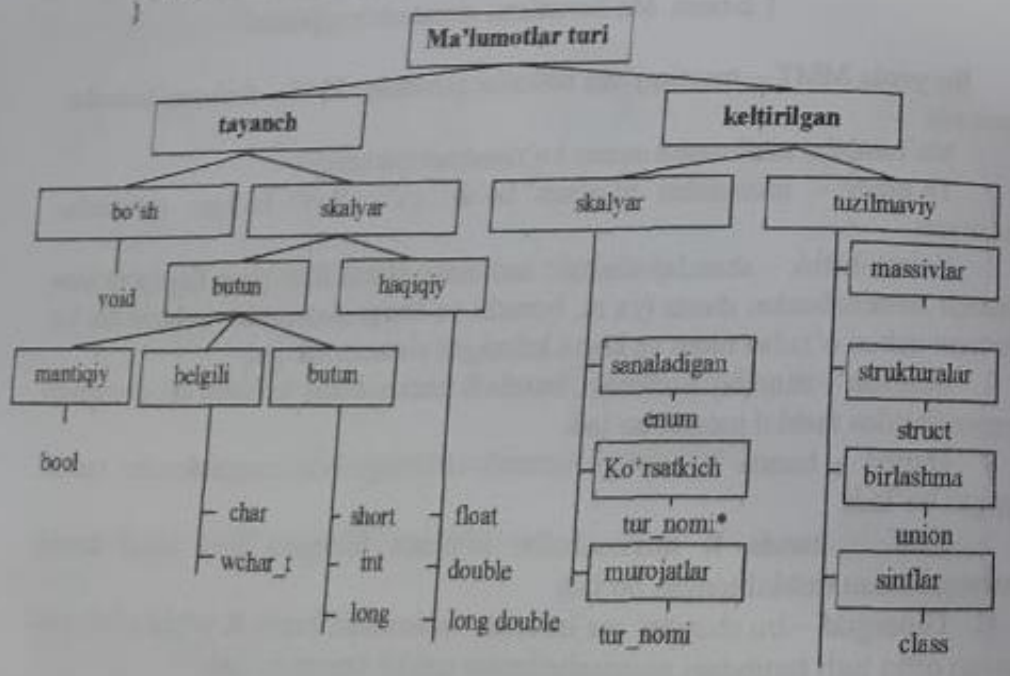
void. Bu eng sodda ma'lumot turi bo'lib, uning qiymatlar to'plami bo'sh, shu sababli, o'zgaruvchilar bu turga tegishli bo'lmaydi. Odatda, bu turdan kam foydalaniladi.

void turining asosan uchta vazifasi mavjud:

1. Funksiya hech qanday qiymatni qaytarmasligini anglatadi.

Masalan,

```
void writeValue(int x){
  std::cout << "The value of x is: " <<x << std::endl;
}
```



1.3-rasm. Ma'lumotlar turlarining sinflanishi

2. Funksiya parametrlarga ega emasligini anglatadi.

Masalan,

```
int getValue(void){
  int x;
  std::cin >> x;
  return x;
}
```

yoki

```
int getValue(){
  int x;
  std::cin >> x;
  return x;
}
```

3. Ixtiyoriy turdagi ma'lumotga ko'rsatkich.

Bu ko'rsatkichni maxsus turi bo'lib, e'lon qilinishi oddiy ko'rsatkich kabi bo'ladi, faqatgina ma'lumot turi o'rniga void kalit so'zidan foydalaniladi, ya'ni void *ptr;

Masalan,

```
int nResult;
float fResult;
struct Something{
  int n;
  float f;
};
Something sResult;
void *ptr;
ptr = &nResult;
ptr = &fResult;
ptr = &sResult;
```

Butun tur. Mazkur tur matematikadagi butun sonlar to'plamining biror bir qism to'plamini namoyon qilib, uning o'lchami mashina, ya'ni kompyuter konfiguratsiyasiga bog'liq ravishda o'zgarib turadi. Agar butun sonni mashinada tasvirlash uchun n ta razryaddan foydalanilsa, u holda x butun sonning qiymat qabul qilish oralig'i $-2^{n-1} <= x < 2^{n-1}$ dan iborat bo'ladi.

Ushbu turga kiruvchi sonlar ikkiga bo'linadi: ishorali (signed) va ishorasiz (unsigned). Ularning har biri uchun mos ravishda qiymat qabul qilish oralig'i mavjud:

a) ishorasiz sonlar uchun $(0..2^n-1)$;

b) ishoralilar uchun $(-2^{N-1}.. 2^{N-1}-1)$.

Butun sonlar ustida turli matematik amallarni, masalan, qo'shish, ayirish, ko'paytirish, darajaga oshirish, butun va qoldiqli bo'lish kabi amallar bilan bir qatorda taqqoslash, ya'ni binar amallarni ham bajarish mumkin. Binar amallarning natijalari butun turga emas, balki mantiqiy turga tegishli bo'ladi.

Butun qiymat qabul qiluvchi o'zgaruvchilarni e'lon qilish uchun *int*, *short int*, *long int* xizmatchi so'zlaridan foydalanish mumkin. Butun qiymatli turlar bo'yicha ba'zi ma'lumotlar quyidagi jadvalda keltirilgan:

tur	qiymatlar oralig'i	xotira hajmi
short int	signed: -32768 ... 32767 unsigned: 0 ... 65535	2 bayt
int	signed: -2147483648 ... 2147483647 unsigned: 0 ... 4294967295	4 bayt
long int	signed: -2147483648 ... 2147483647 unsigned: 0 ... 4294967295	4 bayt

Bu sanab o'tilgan turlar o'zlarining qiymatlar qabul qilish oralig'i va xotiradan egallagan joyining katta yoki kichikligi bilan farqlanadi. Shuning uchun, o'zgaruvchilarning qabul qiladigan qiymatlarini katta yoki kichikligiga qarab, yuqoridagi turlardan mosini tanlash maqsadga muvofiqdir. Yuqoridagi turlarni *signed* (ishorali), *unsigned* (belgisiz) kalit so'zlari bilan modifikatsiyalash mumkin. Bunda belgili tur uchun ajratilgan joyning eng chap biti ishora uchun, qolgan bitlar qiymatlarni saqlash uchun ishlatiladi, ya'ni 0 – plyus, 1- minus. Belgisiz turlarda esa barcha bitlar qiymatlarni saqlash uchun ishlatiladi.

Berilgan m va n butun sonlari ustida quyidagi arifmetik amallar bajarish dasturini ko'rib chiqaylik: m+n,m-n,m*n.

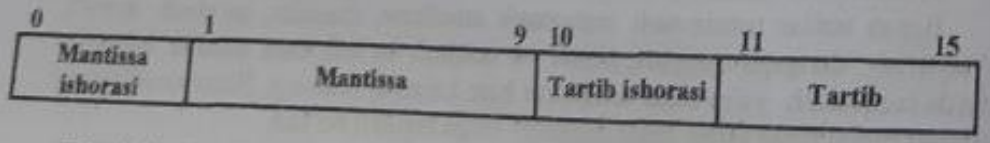
```
#include <iostream>
using namespace std;
int main(){
  int m,n;
  cin>>m>>n;
  int k1=m+n;
  int k2=m-n;
  int k3=m*n;
  cout<<k1<<k2<<k3;
  system("PAUSE");
}
```

Haqiqiy tur. Bu turga kasr qismlari bor chekli sonlar to'plami kiradi. Ular ustida turli matematik amallarni bajarish mumkin. Bu amallarning natijalari ham haqiqiy turga kiradi. Bu yerda ham binar amallarga nisbatan masalaning yechimlari mantiqiy turga tegishli bo'ladi.

Kompyuter xotirasida haqiqiy sonlar asosan qo'zg'aluvchan nuqta formatida saqlanadi. Bu formatda x haqiqiy son quyidagi ko'rinishda ifodalanadi:

$x = +/- M * q^{(+/-P)}$ – soning yarimlogarifmik shakldagi ifodalanishi quyidagi chizmada keltirilgan.

$937,56 = 93756 * 10^{-2} = 0,93756 * 10^{3}$



Haqiqiy (kasr) qiymatli turga tegishli o'zgaruvchilarni e'lon qilish uchun *float, double, long double* xizmatchi so'zlarni biridan foydalaniladi.

tur	qiymatlar oralig'i	xotira hajmi
float	3.4e-38 ... 3.4e+38	4 bayt
double	1.7e-308 ... 1.7e+308	8 bayt
long double	3.4e-4932 ... 3.4e+4932	<8 bayt

Berilgan *m* va *n* haqiqiy sonlari ustida quyidagi amallarni bajarish dasturini ko'rib chiqaylik.

```
#include <iostream>
using namespace std;
int main(){
  float m,n;
  cin>>m>>n;
  float k1=m+n;
  float k2=m-n;
  float k3=m*n;
  cout<<k1<<"  "<<k2<<"  "<<k3;
  system("PAUSE");
}
```

C++ da ushbu turlarni oldiga *signed* va *unsigned* kalit so'zlarini qo'yib turlarni modifikatsiyalash mumkin. Masalan,

signed float — unsigned float
signed double — unsigned double
signed long double — unsigned long double.

Mantiqiy tur. Bu turdagi o'zgaruvchi ***bool*** kalit so'z bilan e'lon qilinib, mantiqiy mulohazalarni rost yoki yolg'onligini aniqlashda foydalaniladi. Mazkur turdagi o'zgaruvchi, odatda, xotiradan 1 bayt joy egallab, 0 (***false-yolg'on***) va 0 qiymatdan farqli qiymat (***true-rost***) qabul qiladi. Ular qabul qiladigan qiymatlar matematik mantiq qonuniyatlariga bo'ysunadi.

tur	qiymatlari	xotira hajmi
bool	true , false	1 bayt

Mantiqiy mulohazalar ustida quyidagicha amallarni bajarish mumkin:

1) mantiqiy ko'paytirish - kon'yunksiya (va) (AND, &, *);
2) mantiqiy qo'shish - diz'yunksiya (yoki) (OR, |, +);
3) inkor – inversiya (yo'q) (NOT, ~, !);
4) "inkor-yoki" (xor, NEQV, ^);
5) ekvivalentlik (tenglik) (EQV, =, ==);
6) taqqoslash (>, <, <=, >=).

Quyida mantiqiy amallarning chinlik jadvali keltirilgan.

A	B	!A	A||B	A&&B
true	*true*	*false*	*true*	*true*
true	*false*	*false*	*true*	*false*
false	*true*	*true*	*true*	*false*
false	*false*	*true*	*false*	*false*

C++ da mantiqiy amallarni ifodalashning bir qancha ko'rinishlari mavjud, masalan, *and* yoki *&&*, *or* yoki *||*, *not* yoki *!* va "inkor-yoki" amali *xor* kabi yozilishi mumkin.

bool turiga misol keltiramiz.

```
#include <iostream>
using namespace std;
int main(){
  bool b=true;
  bool c=false;
  bool d1=not b || c;
  bool d2=b && c;
  bool d3=b xor c;
  cout<<d1<<" "<<d2<<" "<<d3;
  system("PAUSE");
}
```

Belgili tur. Bu turga belgilarning cheklı to'plami yoki liter, ularga lotin alifbosidagi xarflar va unda yo'q kirill xarflar, o'nlik raqamlar, matematik va maxsus belgilar kiradi. Belgili ma'lumotlar hisoblash texnikasi bilan inson o'rtasidagi aloqani o'rnatishda katta ahamiyatga ega. Belgili turdagi o'zgaruvchilar ustida turli matematik amallarni bajarish mumkin. Bunda amallar belgilarning ASCII kodidagi qiymatiga nisbatan bajariladi. Belgili turlarni taqqoslash ham mumkin va taqqoslashlarning natijalari bool turiga kiradi. C++ tilida belgili turdagi o'zgaruvchilarining qiymatlari apostrof (') ichida beriladi va u bitta belgidan iborat bo'lishi mumkin.

Tur	qiymatlar oralig'i	xotira hajmi
char(signed char)	-128...127	1 bayt
unsigned char	0...255	1 bayt
wchar_t (kengaytirilgan)	0...65535	2 bayt

Satr (qator) – bu qandaydir belgilar ketma-ketligi bo'lib, satr bo'sh bo'lishi, bir yoki bir nechta belgilar birlashmasidan iborat bo'lishi mumkin. C++ tilida satrlarni e'lon qilish belgilar massivi shaklida amalga oshiriladi. Bu haqida keyinroq batafsil to'xtalamiz.

Belgili turdagi o'zgaruvchilar ustida o'zlashtirish, taqqoslash va turli matematik amallarni bajarish mumkin. Bunda agar belgili turlar ustida matematik amallar bajariladigan bo'lsa, belgilarning ASCII kodlari olinadi.

Belgilar va qatorlarga doir quyidagi sodda dasturini keltiramiz:

```
#include<iostream.h>
using namespace std;
int main(){
  char x='a';
  char y='b';
  char min;
  cout<<x+y;//a va b belgilarni ASCII kodlarini yig'indisi - 195
  cout<<x<<" "<<y;//a b ni ekranga chiqarish
  if(x>y) min=y;
    else min=x;
```



```
  cout<<min;// ekranga a chiqadi
  system("pause");
}
```

Sanaladigan tur. Bir qancha qiymatlardan birini qabul qila oladigan o'zgaruvchiga sanaladigan turdagi o'zgaruvchilar deyiladi va bunday o'zgaruvchilarni e'lon qilishda **enum** kalit so'zi va undan keyin tur nomi hamda figurali qavs ichida vergullar bilan ajratilgan o'zgarmas qiymatlari ro'yxati ishlatiladi. Masalan,

```
enum Ranglar{oq,qora,qizil,yashil};
```

Bu yerda Ranglar nomli sanoqli tur yaratildi. Ushbu turning 4 ta o'zgarmas elementlari mavjud va ular dastlab 0 dan boshlab sanaladigan butun sonli qiymatga ega bo'ladilar. Ayrim hollarda foydalanuvchi tomonidan o'zgarmaslarga ixtiyoriy sonli qiymat ham o'zlashtirilishi mumkin. O'zgarmaslarga qiymatlar o'sish tartibida berilishi kerak. Masalan,

```
enum Ranglar{oq=100,qora=200,qizil,yashil=400};
```

Bu yerda qizil o'zgarmasni qiymati 201 ga teng bo'ladi. Endi shu turdagi birorta o'zgaruvchini e'lon qilish mumkin.

```
Ranglar r=qizil;
```

Endi r o'zgaruvchi Ranglar turida aniqlangan o'zgarmaslardan ixtiyoriy birini qiymat sifatida qabul qila oladi. Masalan.

```
#include<iostream.h>
using namespace std;
int main(){
  enum kunlar{du=1,se,chor};
  kunlar hafta;
  hafta=chor;
  cout<<hafta;
  int kun;
  cout<<"\nbugun qaysi kun";
  cin>>kun;
  if(kun==chor) cout<<"\ntalabalar bilan uchrashuvingiz bor";
  system("pause");
}
```

Ko'rsatkichli tur. Bu tur ma'lumotlarni emas, balki bu ma'lumotlar joylashgan xotiradagi manzilni o'z ichiga oladi. Ko'rsatkichlar xotirada bor yo'g'i 4 bayt joyni egallab, u ko'rsatayotgan ma'lumotlar ancha katta joyni egallagan bo'lishi mumkin. Ko'rsatkichlar qanday ishlashini bilish uchun mashina xotirasi tashkil etilishining tayanch tamoyillarini bilish lozim. Mashina xotirasi 16 lik sanoq sistemasida raqamlangan yacheykalar ketma-ketligidan iboratdir. Har bir o'zgaruvchining qiymati uning adresi deb ataluvchi alohida xotira yacheykasida saqlanadi. Ko'rsatkichli turdagi o'zgaruvchilar o'zida ana shu kabi o'zgaruvchilar yoki boshqa ma'lumotlarning xotiradagi adresini saqlaydilar. C++ da o'zgaruvchini ko'rsatkichli turda e'lon qilish uchun o'zgaruvchi nomidan oldin * belgisi qo'yiladi. Har bir o'zgaruvchini turi bilan e'lon qilingani kabi ko'rsatkichli o'zgaruvchilar ham ma'lum bir tur bilan e'lon qilinadi. Bunda ko'rsatkichli turdagi o'zgaruvchining turi – shu ko'rsatkich ko'rsatayotgan xotira yacheykasidagi ma'lumotning turi bilan bir xil bo'lishi kerak. Masalan, int a=1 bo'lsin. Ushbu o'zgaruvchini adresini o'zida

saqlovchi b ko'rsatkichli o'zgaruvchini e'lon qilishda ham int turi ishlatiladi, ya'ni *int *b*. Endi bunday turdagi o'zgaruvchiga a o'zgaruvchini adresini o'zlashtirish uchun a ning oldiga & - adres operatorini qo'yish zarur, ya'ni b=&a.

Misol.

```
#include<iostream.h>
using namespace std;
int main(){
  short int a=1234567;
  short int *b;
  b=&a;
  cout<<b; // a o'zgaruvchining adresi 0x22ff76 ni ekranga chiqaradi
  system("pause");
}
```

Ko'rsatkichli turlar yordamida fayllarga ham murojaat qilsa bo'ladi, masalan:

```
#include <stdio.h>
#include <iostream.h>
using namespace std;
int main(){
  FILE *p;
  char s[100];
  if((p=fopen("f.txt","r"))==NULL)
       cout<<"o'hshamadi";
  else  cout<<"ulandi";
  fclose(p);
  system("pause");
}
```

Massivlar (bir va ikki o'lchamli). Massiv – bu eng sodda statik va chiziqli tartiblangan tuzilmadir. Bu tuzilmadagi elementlar orasidagi munosabat ularning qat'iy ketma-ketlik ko'rinishida ifodalanishidir. Massiv elementlarining barchasi faqatgina bitta turga tegishli va ularning soni oldindan aniq bo'lishi lozim.

Massivning har bir elementi uning massivdagi o'rnini belgilovchi o'zining indeksiga ega bo'ladi. Ko'pgina dasturlash tillarida, xususan, C++ da massiv elementlarini raqamlash 0 dan boshlanadi. Massivning elementiga murojaat qilish uchun massiv nomi va elementning indeksini ko'rsatish kifoya bo'ladi. Dasturda massivni e'lon qilish uchun uning nomini, elementlar sonini va ularning turini ko'rsatish lozim.

Bir o'lchamli, n ta xadli (n=10) massivning xadlari yig'indisini topish dasturini keltiramiz.

```
#include <iostream.h>
using namespace std;
int main(){
 int a[10],s=0;
 for(int i=0;i<10;i++){
      cin>>a[i];
      s+=a[i];
      }
cout<<s;
system("pause");
}
```


Ikki o'lchamli massiv elementiga murojaatni amalga oshirish uchun uning indeksi qiymatlari zarur bo'ladi. Fizik bosqichda ikki o'lchamli massiv ham xuddi bir o'lchamli massiv kabi ko'rinishga ega bo'ladi hamda translyatorlar massivni qator yoki ustun ko'rinishida ifodalaydi.

```
#include <iostream.h>
using namespace std;
int main(){
 int a[2][3],s=0;
 for(int i=0;i<2;i++)
 for(int j=0;j<3;j++){
      cin>>a[i][j];
      s+=a[i][j];
      }
 cout<<s;
 system("pause");
}
```

Vektorlar. Garchi vektor turidagi tuzilma massivlarga o'xshash bo'lsada, ulardan foydalanish ancha xavfsiz va bir qator qo'shimcha imkoniyatlarga ega. Masalan, oddiy massivlarda berilgan chegaradan, ya'ni massiv o'lchamidan chiqib ketilganligi nazorat qilinmaydi, vektorlarda esa bunday holat yuzaga kelsa, dastur xatolik mavjudligi bo'yicha xabar beradi. Bundan tashqari, vektor dinamik tuzilma bo'lib, uni tashkil etuvchilari soni dastur mobaynida o'zgarib turishi mumkin.

Dasturda vector turidagi tuzilma bilan ishlash uchun vector sarlavha faylini yuklab olish zarur, ya'ni

```
#include <vector>
```

Quyida vektor bilan ishlashga doir misol keltiramiz.

```
#include <iostream>
#include <cstdlib>
#include <vector>
using namespace std;
int main(){
    vector<int> v(3); // 3 ta elementga ega vector
    cout << "Vector o'lchami v = " << v.size() << endl;
    for(int i = 0; i < v.size(); i++) // vectorni to'ldirish
       v[i] = 2 * i + 1;
    cout << "Vector:\n";
    for(int i = 0; i < v.size(); i++) // Vectorni chop etish
       cout << "v[" << i << "] = " << v[i] << endl;
    //cout << "v[-1] = " << v[-1] << endl; // -1 – mumkin bo'lmagan indeks
    //cout << "v[3] = " << v[3] << endl; // 3 - mumkin bo'lmagan indeks
    system("pause");
    return 0;
}
```

Dastur natijasi:

Vector o'lchami v = 3
Vector:
v[0] = 1
v[1] = 3
v[2] = 5

Agar quyidagi satrlar boshidagi izoh belgisi olib tashlansa, ya'ni

```
cout << "v[-1] - " << v[-1] << endl; // -1 - mumkin bo'lmagan indeks
cout << "v[3] - " << v[3] << endl; // 3 - mumkin bo'lmagan indeks
```

u holda dastur ularning birinchisida xatolikni ko'rsatuvchi quyidagicha oyna chiqadi:

```
Vector o'lchami v = 3
Vector:
v[0] = 1
v[1] = 3
v[2] = 5
v[-1] = 335687736
v[3] = 0
Для продолжения нажмите любую клавишу . . .
```

Strukturalar. Strukturalar turli turdagi maydonlardan tashkil topgan yozuv hisoblanadi. Strukturalarni e'lon qilish uchun **struct** kalit so'zi ishlatiladi. Undan keyin turga nom beriladi va {} qavs ichida maydonlar turlari va nomlari e'lon qilinadi. Yaratilgan tur bilan e'lon qilingan o'zgaruvchilar yozuv hisoblanadi, massivlar esa jadvalni tashkil etadi.

Masalan,

```
#include <iostream.h>
using namespace std;
int main(){
 struct Guruh{
      int n;
      char fio[30];
      };
 Guruh talaba[5];
 for(int i=0;i<5;i++){
     talaba[i].n=i+1;
     cin>>talaba[i].fio;
     }
 for(int i=0;i<5;i++)
     cout<<talaba[i].n<<"  "<<talaba[i].fio<<endl;
 system("pause");
}
```

Birlashmalar (union). Birlashmalar xuddi strukturalarga o'xshash tur hisoblanadi, farqi shuki, birlashmalarda bir vaqtning o'zida faqat uning bitta elementigagina murojaat qilish mumkin. Birlashma turi quyidagicha aniqlanadi:

```
union{1-elementni tavsiflash;
      ...
      n-elementni tavsiflash;
     };
```

Birlashmalarning asosiy xususiyati shuki, e'lon qilingan har bir element uchun xotiraning bitta hududi ajratiladi, ya'ni ular bir-birini qoplaydi. Bu yerda xotiraning shu qismiga istalgan element bilan murojaat qilsa bo'ladi, lekin buning uchun element shunday tanlanishi kerakki, olinadigan natija ma'noga ega bo'lishi kerak. Birlashmaning elementiga murojaat xuddi struktura elementiga murojaat kabi

amalga oshiriladi. Birlashmalar qo'llaniladigan xotira obyektini initsializatsiya qilish maqsadida ishlatiladi, agarda har bir murojaat vaqtida bir qancha obyektlardan faqat bittasi faollashtirilsa.

Birlashma turidagi o'zgaruvchi uchun ajratiladigan xotira hajmi ushbu turning eng uzun elementi uchun ketadigan xotira hajmi bilan aniqlanadi. Kichik uzunlikdagi element ishlatilganda, birlashma turidagi o'zgaruvchi uchun ajratilgan xotira soxasining ayrim qismi ishlatilmaydi. Birlashmaning barcha elementi uchun xotiraning bitta adresdan boshlanuvchi bitta soxasi ajratiladi. Masalan:

```
union {  char fio[30];
         char adres[80];
         int yoshi;
         int telefon;  } inform;
union {  int ax;
         char al[2];   } ua;
```

Birlashma turidagi inform obyektini ishlatganda qiymat qabul qilgan elementinigina qayta ishlash mumkin, ya'ni masalan inform.fio elementiga qiymat berilgandan keyin boshqa elementlarga murojaat ma'noga ega emas. ua birlashmasi al elementining kichik ua.al[0] va katta ua.al[1] baytlariga alohida murojaat qilish mumkin. Birlashma turiga oid misol ko'rib chiqamiz.

```
#include <iostream.h>
using namespace std;
int main(){
  union Guruh{
      int n;
      int m;
      };
  Guruh w;
  w.n=12;// w birlashmasining n elementiga qiymat berish
  w.m=23; // w birlashmasining m elementiga qiymat berish
  cout<<w.n<<"  "<<w.m; //bu yerda w  uchun ajratilgan joyga oxirgi marta m uchun 23
qiymati yozilgani sababli ekranga 23  23 javobi chiqariladi
  system("pause");
}
```

Sinflar (class). Sinf – bu dasturchi tomonidan ixtiyoriy kiritilgan, mavjud turlar asosida yaratilgan strukturalangan tur hisoblanadi. Sinflar lokal va global o'zgaruvchilar va ular ustida amal bajaradigan funksiyalar to'plamidan iborat bo'lishi mumkin. Sinflar quyidagicha ifodalanadi:

```
class sinf_nomi{
     <local va global o'zgaruvchilar ro'yxati>;
     <funksiyalar>
};
```

Sinflarga oid misol:

```
#include <iostream.h>
using namespace std;
class daraxt{
 public:
   unsigned int uzunligi ;
   unsigned int yoshi;
 int o_sish(int i){
```

```
    i++;
    return i;
    };
};
int main(){
  int k=2;
  daraxt olma_daraxt;
  olma_daraxt.uzunligi=5;
  olma_daraxt.yoshi=7;
  cout<<olma_daraxt.o_sish(k);
  system("pause");
}
```

1.1.4. Ma'lumotlar tuzilmalarini sinflashtirish

Ma'lumotlar tuzilmalarini tuzilishi, o'zgaruvchanligi, elementlar orasidagi munosabatlar, elementlarning tartiblanganligi, elementlarni xotirada joylashishi va boshqalar bo'yicha bir qator sinflarga ajratish mumkin. Quyida bularga qisqacha to'xtalib o'tamiz.

Masalan,

1) ***tuzilishiga ko'ra: sodda (tayanch) va murakkab (kompozit).***

Sodda – bu tarkibiy qismlarga bo'linmaydigan ma'lumotlar tuzilmasidir. Bularga sonli (butun, haqiqiy), belgili, mantiqiy, ko'rsatkichli turlar kiradi.

Murakkab ma'lumotlar tuzilmasi tarkibiy qismlardan iborat bo'ladi. Uning tarkibiy qismlari sodda yoki murakkab tuzilma bo'lishi mumkin. Masalan, to'plam, massiv, yozuv, ro'yxat va boshqalar.

2) ***O'zgaruvchanligiga ko'ra: statik, yarimstatik va dinamik.***

Agar dastur bajarilishi mobaynida tuzilmani tashkil etuvchi elementlar soni va/yoki ular orasidagi munosabatlar o'zgarsa, u holda bunday ma'lumotlar tuzilmasi dinamik, aks holda statik deyiladi. Masalan, statik tuzilmalar - massiv, to'plam, yozuv, jadval; yarim statik tuzilmalar – stek, navbat, dek, satr; dinamik tuzilmalar - ro'yxatlar, daraxtlar, graflar.

3) ***Elementlar orasidagi munosabatlarga ko'ra: bog'lanmagan va bog'langan.***

Tuzilmani tashkil etuvchi elementlar orasidagi munosabatlar ikki xil ko'rinishda berilishi mumkin: oshkor va oshkormas. Bog'lanmagan ma'lumotlar tuzilmasida elementlar orasidagi munosabatlar oshkormas ko'rinishda, bog'langan ma'lumotlar tuzilmasida esa oshkor ko'rinishda beriladi.

Vektor, massiv, satr, stek, dek, navbat - bog'lanmagan tuzilmalar, bog'langan ro'yxat, daraxt va graf esa bog'langan tuzilmalarga misol bo'ladi.

4) ***Elementlarning tartiblanganligiga ko'ra: chiziqli va chiziqsiz.***

Agar tuzilma elementlari orasidagi munosabat biror bir shartni bajarilishiga bog'liq bo'lmasa, u holda bunday tuzilma chiziqli, aks holda chiziqsiz deyiladi. Ma'lumotlar tuzilmasini tashkil etuvchi elementlarning xotiradagi o'zaro joylashuviga ko'ra chiziqli tuzilmalar ikkiga bo'linadi: ketma-ket va ixtiyoriy joylashgan. Masalan, vector, massiv, satr, stek, dek, navbat – elementlari ketma-ket joylashgan

chiziqli tuzilma; bir va ikki bog'lamli ro'yxatlar – elementlari ixtiyoriy joylashgan chiziqli tuzilma. Ko'p bog'lamli ro'yxatlar, daraxt va graflar esa chiziqsiz tuzilmalarga misol bo'ladi.

Nazorat savollari

1. Ma'lumotlar tuzilmasi deganda nimani tushunasiz?
2. Ma'lumotlarni tasvirlash bosqichlarini keltirib o'ting.
3. Ma'lumotlar tuzilmasi klassifikatsiyasi qanday amalga oshiriladi?
4. Ma'lumotlar tuzilmasini foydalanuvchi dasturidagi klassifikatsiyasi qanday?
5. Ma'lumotlar tuzilmasini operativ xotiradagi klassifikatsiyasi qanday?
6. Ma'lumotlar tuzilmasini tashqi xotiradagi klassifikatsiyasi qanday?
7. Qanday ma'lumotlar dinamik yoki statik turdagi ma'lumotlar tuzilmasi deyiladi?

1.2. Dasturlash tilida sinflar

1.2.1. Obyektga yo'naltirilgan dasturlash tushunchasi

Kompyuter tizimlarining rivojlanishi, hal qilinishi lozim bo'lgan masalalar ko'lamining ortishi va ularni tobora murakkablashuvi dasturlashning turli modellarini (paradigmalarini) yaratilishiga turtki bo'ldi. Dasturlashda dastlab funksiyalardan foydalanishga asoslangan protsedura modelidan, rivojlanishning keyingi bosqichida esa tuzilmaviy modeldan foydalanilgan. Tuzilmaviy modelda dasturlar o'zaro bog'langan protseduralar va ular qayta ishlaydigan ma'lumotlar majmuasidan iborat.

Ma'limki, dastur hajmi va uning murakkablik darajasi ortishi bilan shunga mos ravishda dastur kodlarida ham xatoliklar ortib borishi ko'p kuzatilgan. Dasturiy ta'minotdagi xatoliklar tufayli nafaqat moddiy, hattoki inson hayotiga (masalan, avia yoki kosmik kemalarni boshqarishda) ham jiddiy ziyon yetishi mumkin.

Dasturlash va dasturiy ta'minotni ishlab chiqish bilan bog'liq bo'lgan ko'plab muammolarni hal qilish maqsadida turli ilmiy izlanishlar olib borilib, buning natijasida bir qator konsepsiyalar ishlab chiqildi:

- obyektga yonaltirilgan dasturlash (OYD - OOP);
- umumlashgan (unified) modellashtirish tili (UMT-UML);
- dasturiy ta'minot ishlab chiqishning maxsus vositalari.

OYD abstraksiya, inkapsulyasiya, vorislik (merosxo'rlik) va polimorfizm kabi muhim tushunchalarga asoslangan.

Obyektga yo'naltirilgan yondashuvning asosiy g'oyasi ma'lumotlar va ular ustida bajariladigan amallarni yaxlit bir ko'rinishda ifodalashdir. Mazkur ko'rinish obyekt deb ataladi.

OYDda metod bu – biror bir sinf yoki obyektga tegishli bo'lgan funksiya yoki protsedura bo'lib, u ma'lum bir amallarni bajaruvchi va kiruvchi argumentlar majmuasiga ega bir necha operatorlardan tashkil topadi.

Bundan tashqari, ulardan biror-bir sinf obyektiga murojaatni amalga oshiruvchi interfeys sifatida foydalanish mumkin.

Metodlar murojaatni taqdim etish darajasiga qarab quyidagilarga ajratiladi:

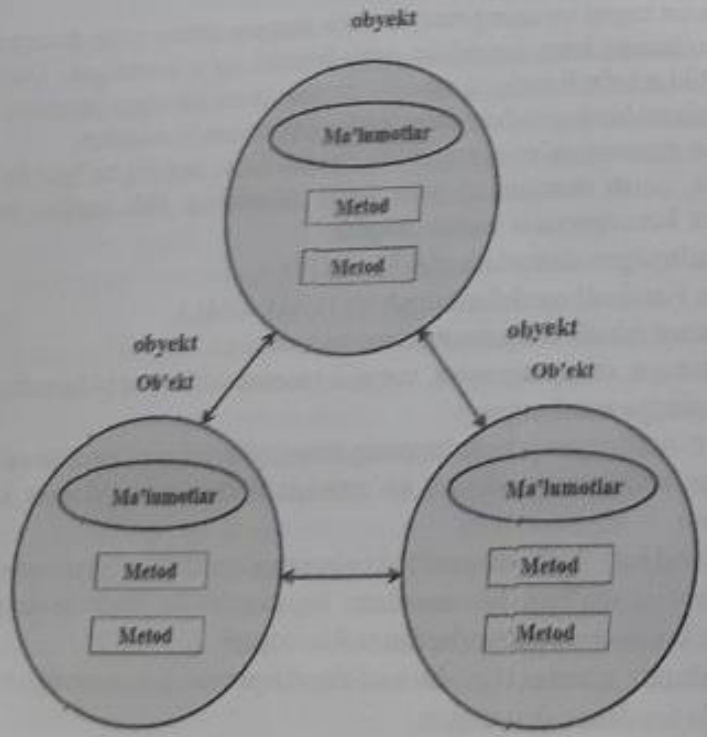
– ochiq interfeys (public) – berilgan sinfning barcha foydalanuvchilari uchun umumiy interfeys;
– himoyalangan interfeys (protected) – berilgan sinfning barcha vorislari uchun ichki interfeys;
– yopiq interfeys (private) – faqatgina berilgan sinfning ichidagi elementlar uchun interfeys.

Obyektning atrof muhit (foydalanuvchi, dasturni qolgan qismi, operatsion tizim va hokazolar) bilan aloqasi faqatgina o'zining metodlari orqali amalga oshiriladi, ya'ni obyekt holatiga tashqaridan murojaat yo'q. Masalan, agar obyekt atrof muhitga o'zining biror-bir o'zgaruvchisi holati qiymati bo'yicha axborot berishi lozim bo'lsa, u holda buning uchun maxsus metod yaratiladi.

Atrof muhitdan obyekt ichki holati yopiqligi inkapsulyasiya xossasi deyiladi. Inkapsulyasiya obyekt o'z ichiga ma'lumotlar va ular ustida amal bajaruvchi metodlarni olishini anglatadi. Boshqacha qilib aytganda, obyekt atrof muhit uchun "qora qut" bo'lib, o'zining ichki tuzilmasini namoyon qilmagan holda kiruvchi ta'sirlarga mos ravishda reaksiya bildiradi.

Obyektlar orasidagi o'zaro aloqa xabarlar almashish orqali amalga oshriladi, xabarni qabul qilgan obyekt uni e'tiborsiz qoldirishi yoki undagi buyruqni bajarishi mumkin (o'zining biror-bir metodi orqali).

Obyektga yonaltirilgan yondashuvda dastur obyektlar to'plami ko'rinishida ifodalanib, bunda obyektlar bir-birini metodlarini chaqirish orqali o'zaro aloqada bo'ladi. Dastur tuzilmasining umumiy ko'rinishi quyidagi rasmda keltirib o'tilgan.



1.4-rasm. Obyektga yonaltirilgan yondashuv

1.2.2. Sinf va obyekt

Dasturlashda ma'lumotlarni guruhlashtirish imkonini beruvchi tuzilmalar va ma'lum bir vazifalarni bajaruvchi dasturning qismlari, ya'ni funksiyalar juda katta ahamiyatga ega. Bu ikki tushunchani birlashtirish natijasida sinf deb ataluvchi dasturning yangi elementi hosil qilinadi.

Misol.

```
#include <iostream>
using namespace std;
class smallobj{  // sinfni aniqlash
  private:
      int somedata; // sinf maydoni
  public:
void setdata(int d) { // maydon qiymatini o'zgartiruvchi sinf metodi
      somedata = d; }
void showdata(){// maydon qiymatini aks ettiruvchi sinf metodi
  cout << "Maydon qiymati=" << somedata << endl;}
};
int main(){
   smallobj s1, s2; // smallobj sinfining ikki obyektini aniqlash
   s1.setdata(1066); // setdata() metodini chaqirish
   s2.setdata(1776);
   s1.showdata(); // showdata() metodini chaqirish
   s2. showdata(); return 0;
}
```

Ushbu misolda aniqlangan smallobj sinfi bitta ma'lumotlar maydoni va 2 ta metodni o'z ichiga olgan.

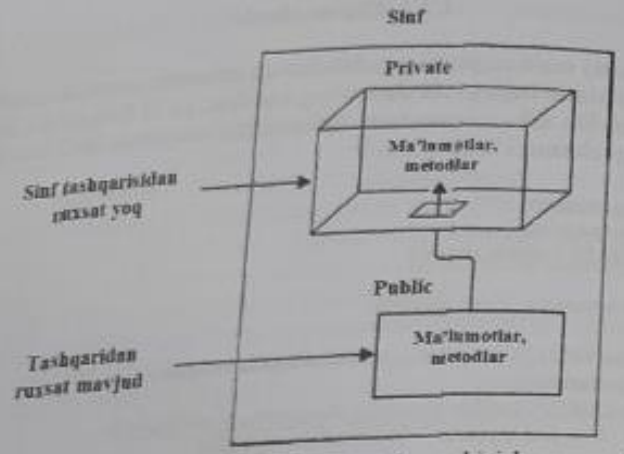
Ma'lumotlar bilan funksiyalarni birlashtirish OYDning tayanch g'oyalaridan biri bo'lib hisoblanadi.

Yuqorida keltirib o'tilgan misoldan ko'rinib turibdiki, sinfni e'lon qilishda *class* kalit so'zi va sinf nomi keltiriladi. Sinf tanasi esa figurali qavs ({}) ichida berilib, oxirida nuqtali vergul (;) qo'yiladi.

Sinf tanasi ichida e'lon qilingan o'zgaruvchi va funksiyalar sinf a'zolari deb nomlanadi. Sinfning funksiya-a'zolari shu sinf metodlari, o'zgaruvchi a'zolar esa sinf maydoni deb ataladi. Demak, keltirib o'tilgan misolda **somedata** sinf maydoni, **setdata()** va **showdata()** sinf metodlari, **s1** va **s2** esa sinf obyektlari bo'lib hisoblanadi.

E'tibor bersak, sinf tanasida **private** va **public** degan kalit so'zlar keltirib o'tilgan. Bu nimani anglatadi? Keling shularga to'xtalib o'taylik.

OYDning muhim xususiyatlaridan biri bu ma'lumotlarni yashirish imkoniyatidir, ya'ni sinf ichidagi ma'lumotlarni sinf tashqarisidagi funksiyalardan himoyalanganligini anglatadi. Agar qandaydir ma'lumotlarni himoyalash lozim bo'lsa, u holda ular private kalit so'zidan foydalanib sinf ichiga joylashtiriladi. Bunday ma'lumotlarga faqatgina sinf ichida murojaat qilishga ruxsat etilgan bo'ladi. Agar sinf ma'lumotlari public kalit so'zi orqali berilgan bo'lsa, ularga sinf tashqarisidan ham murojaat qilinishiga ruxsat etilgan bo'ladi.



1.5-rasm. Sinf a'zolariga murojaatni amalga oshirish sxemasi

Agar qo'shimcha aniqlashtirishlar berilmagan bo'lsa, u holda sinf ichida e'lon qilingan barcha metod va sinf maydonlari yopiq hisoblanadi, ya'ni sinfning yopiq qismiga tegishli bo'ladi. Bu esa o'z navbatida ulardan foydalanish faqatgina sinf ichida ruxsat etilganligini, tashqaridan ruxsat yo'qligini anglatadi.

Sinfda ochiq va yopiq qismlar bir nechta bo'lib, ular ixtiyoriy tartibda almashinib kelishi mumkin.

Sinfni e'lon qilishda xotira ajratilmaydi. Sinf e'lon qilinganda kompilyator faqat shunday sinf borligini, unda qanday qiymatlar saqlanishi va ular yordamida qanday amallarni bajarish mumkinligi haqida xabar beradi.

Misol. Faraz qilaylik, uch o'lchovli fazoda geometrik vektorlarni uzunligini aniqlash lozim bo'lsin. Bu masala uchun sinf quyidagicha bo'ladi.

```
class spatial_vector{
  public:
      double abs ( );
  private:
      double x, y, z;
};
```

Sinf ichidagi metodga murojaat qilinayotganda "." yoki "->" dan foydalaniladi, ya'ni

```
main ( ){
  spatial_vector a,b;
  double d;
  .....
  d = a.abs ( );
}
```

Ko'rinib turibdiki, spatial_vector sinfda e'lon qilingan abs() metod vektorni absolyut qiymatini qaytaradi. Lekin, dastur kompilyasiya bo'lishi uchun abs() metod e'lon qilinganidan keyin, bu metodni aniqlash lozim bo'ladi (ya'ni mazkur metod tanasini yozish lozim). Metod ham oddiy funksiya kabi aniqlanib, faqatgina metod

nomida qaysi sinfga tegishli ekanligini ko'rsatib o'tish lozim bo'ladi. Bunda ko'rish sohasini kengaytirish operatoridan foydalaniladi, ya'ni "::".

Quyidagi misolda berilgan sinfdagi ikkita metodni aniqlash keltirib o'tilgan.

```
#include <iostream>
#include <math.h>
using namespace std;
class spatial_vector{
  double x, y, z;
        public:
  void set ( double a,double b,double c );
  double abs ( );
};
void spatial_vector::set ( double a,double b,double c ){
  x=a;y=b;z=c;
}
double spatial_vector::abs ( ){
  return sqrt ( x*x + y*y + z*z );
}
main ( ){
  spatial_vector a;
  a.set ( 1, 2, 3 );
  cout << a.abs ( ) << endl;
}
```

1.2.3. Konstruktor va destruktorlar

Yuqorida ko'rib o'tilgan misolda **spatial_vector** sinfi uchun o'zgaruvchilarga qiymatlarni belgilash uchun **set** metodidan foydalandik. Umuman olganda, sinfning yopiq o'zgaruvchilariga murojaatni an'anaviy usulda ma'lumotni o'qish uchun o'zgaruvchi oldiga "**get**" va yozish uchun "**set**" qo'shimchali metodlar ishlatiladi:

```
class spatial_vector{
   double x, y, z;
public:
   double get_x ( );
   void set_x ( double x );
   ... .
double spatial_vector::get_x ( ) { return x; }
   ... .
}
```

O'zgaruvchilar soni ko'p bo'lganda bunday yondashuv noqulay hisoblanib, odatda, undan foydalanish tavsiya qilinmaydi.

C++da sinf obyektlari yaratilayotganda uning o'zgaruvchilarini avtomatik tarzda initsializatsiya qilishga mo'ljallangan metod yaratish imkoniyati mavjud. Mazkur metod konstruktor deb ataladi. Bunda dasturchi, o'z xohshiga ko'ra, masalan, elementlarga boshlang'ich qiymatlarni o'zlashtirish, xotirani dinamik ajratish va hokazolar bo'yicha konstruktorni aniqlashi mumkin. Agar dasturchi sinf konstruktorini aniqlamagan bo'lsa, u holda kompilyator avtomatik ravishda standart konstruktorni hosil qiladi (ya'ni, bo'sh va kiritish parametrlarisiz).

Konstruktor oshkor yoki oshkormas ko'rinishda chaqirilishi mumkin. Kompilyator o'zi sinf obyekti yaratilayotgan vaqtda konstruktorni chaqiradi.

C++da konstruktorlarni tavsiflashni quyidagicha o'ziga xosliklari mavjud:

- konstruktor nomi sinf nomi bilan bir xil bo'ladi;
- konstruktor hech qanday qiymat qaytarmaydi, bundan tashqari u tavsiflanayotganda **void** kalit so'zidan ham foydalanilmaydi.

Konstruktorga teskari funksiya bu destruktordir, ya'ni bu metod obyekt o'chirilayotganda chaqiriladi. Ta'kidlash lozimki, lokal obyektlar, ko'rish sohasidan tashqariga chiqilganda, global o'zgaruvchilar esa dastur yakunida o'chiriladi.

C++da destruktorlar sinf nomi oldiga "~" belgisini qo'yish orqali aniqlanadi, ya'ni "~sinf_nomi". Destruktor ham hech qanday qiymat qaytarmaydi, bundan tashqari u konstruktordan farqli ravishda oshkor ko'rinishda chaqirilmaydi.

Konstruktor sinfning yopiq qismida tavsiflanmaydi. Umumiy holda bunday cheklov destruktorga ham taalluqli. Quyida obyekt yaratish, uning metodini chaqirish va dastur so'ngida uni o'chirishni keltirib o'tamiz:

```
#include <iostream>
#include <math.h>
using namespace std;
class spatial_vector{
    double x, y, z;
public :
    spatial_vector ( );
    ~spatial_vector ( ) { cout << "Destruktor ishi\n "; }
    double abs ( ) { return sqrt ( x*x +y*y +z*z ); }
};
spatial_vector::spatial_vector ( ){
    //vector sinfi konstruktori
    x=y=z =0;
    cout << "Konstruktor ishi\n ";
}
main ( ){
    spatial_vector a;//nol qiymat bilan a obyekt yaratilmoqda
    cout << a.abs ( ) << endl;
}
```

Dastur natijasi:

```
Konstruktor ishi
0
Destruktor ishi
```

1.2.4. Do'stona funksiyalar

Faraz qilaylik, funksiya ikkita turli sinf obyektlari bilan ishlashi lozim. Masalan, funksiya ikkita sinf obyektlarini argument sifatida qabil qilib, ularni yashirin ma'lumotlarini qayta ishlashi zarur. Bunday vaziyatda do'stona, ya'ni friend funksiyadan foydalaniladi. Demak, do'stona funksiyalar ikki sinf orasidagi ko'prik vazifasini bajaradi.

Misol.

```
#include <iostream>
using namespace std;
class beta; // frifunc e'lon qilish uchun kerak
```



```
class alpha {
private:
    int data;
public:
    alpha() : data(3) { } //argumentsiz konstruktor
    friend int frifunc(alpha, beta); //do'stona funksiya
};
class beta{
private:
    int data;
public:
    beta() : data(7) { } // argumentsiz konstruktor
    friend int frifunc(alpha, beta); //do'stona funksiya
};
int frifunc(alpha a, beta b) {  //funksiyani aniqlash
    return( a.data + b.data );
}
int main(){
    alpha aa;
    beta bb;
    cout << frifunc(aa, bb) << endl; //funksiyani chaqirish
    return 0;
}
```

Keltirib o'tilgan dasturda ikkita sinf bor: **alpha** va **beta**. Mazkur sinflarning konstruktorlari yagona bo'lgan ma'lumotlariga fiksirlangan (mos ravishda 3 va 7) qiymatni bermoqda.

Talab etiladiki, **frifunc()** funksiya har ikkala sinfdagi yashirin ma'lumotga murojaat qila olsin. Buning uchun sinfni e'lon qilishda ularning har birining tana qismida **friend** kalit so'zidan foydalaniladi:

```
friend int frifunc(alpha, beta);
```

Mazkur e'lon qilinishi sinf tana qismining ixtiyoriy, ya'ni yopiq (private) yoki ochiq (public) qismida joylashishi mumkin.

Har bir sinf obyekti **frifunc()** funksiyaga parametr (argument) sifatida beriladi. Funksiya, o'z navbatida, mazkur argumentlar orqali har ikkala sinfning yashirin ma'lumotlariga murojaat qila oladi. Shundan keyin, funksiya o'ziga yuklatilgan vazifani (bizning holda ma'lumotlar yig'indisini topish) bemalol bajaraveradi. main() funksiyada mazkur funksiya chaqiriladi va natija ekranga chiqariladi.

Ta'kidlab o'tish lozimki, sinf dasturda e'lon qilinmaguncha, unga murojaat qilish mumkin emas. **alpha** sinfda e'lon qilinayotgan **frifunc()** funksiyada **beta** sinfga havola borligi uchun, **beta** sinf **alpha** sinfdan oldin e'lon qilinmoqda.

1.2.5. Istisno holatlarni qayta ishlash

Turli xil kutilmagan vaziyatlar (hatoliklar) natijasida dasturni normal ishlash jarayoni uzilib qolishi mumkin. Dastur bajarilishida vujudga keladigan shunday vaziyatlar (hatoliklar) istisno deyiladi.

Masalan, quyidagi dasturda sonni songa bo'lishda muammo yuzaga kelgan:

```
#include <iostream>
```

```
double divide(int, int);
int main(){
  int x = 500;
  int y = 0;
  double z = divide(x, y);
  std::cout << z << std::endl;
  std::cout << "The End..." << std::endl;
  return 0;
}
double divide(int a, int b){
  return a / b;
}
```

Agar dastur hajmi katta va unda turlicha istisno holatlar yuzaga kelishi havfi bo'lsa, dastur o'z ishini mantiqan oxirigacha yetkazishi uchun, istisno holatlarni qayta ishlash zarurati paydo bo'ladi. Buning uchun, istisno bo'ladigan holatlarni aniqlash va unga mos yechimlarni taqdim etish lozim.

C++ dasturlash tili standartida istisnolarni qayta ishlashning sozlangan mexanizmi mavjud. Shuni esdan chiqarmaslik lozimki, bu mexanizm apparat yoki asinxron istisnolarni boshqara olmaydi, u faqatgina dasturning funksiyalari tomonidan yuzaga keladigan istisnolar uchun mo'ljallangan.

*C++*da istisnolarni boshqarishda uchta kalit so'zdan foydalaniladi: **try**, **catch** va **throw**.

Xizmatchi **try** kalit so'zi istisno vujudga kelishi mumkin bo'lgan dastur qismini (kod blok) belgilashda ishlatiladi. Bunda dasturning o'sha qismi figurali qavs ichiga olinadi. Bu qism himoyalangan yoki try-blok deb nomlanadi:

```
try{          // kodning himoyalangan bloki
}
```

try-blokdan chaqirilgan ixtiyoriy funksiyaning tanasi mazkur blokga tegishli bo'ladi. Faraz qilaylik, himoyalangan qismdagi bir yoki bir necha funksiyalar istisno keltirib chiqarishi mumkin. Agar biror-bir finksiya tomonidan shunday vaziyat sodir bo'lsa, u holda bu funksiya ishi to'xtatiladi, himoyalangan blokdagi barcha instruksiyalarga e'tibor qilinmaydi va boshqaruv blok tashqarisiga uzatiladi.

Agar istisno ro'y bersa, boshqaruv **catch** kalit so'zi orqali aniqlangan dastur qismiga o'tiladi. Bunda **catch** kalit so'zidan keyin oddiy qavs ichida istisno tasnifi (ya'ni, istisno turi va o'zgaruvchi) keltirib o'tiladi, **catch**ning tanasi, ya'ni istisnoni qayta ishlash, figurali qavs ichida beriladi:

```
catch (<istisno_turi> <istisno_o'zgaruvchi>){
<istisnoni qayta ishlovchi>
}
```

Dasturning bu qismi ***catch-blok*** yoki ***istisnolarni qayta ishlovchi*** deyiladi, u try-blokdan keyin aniqlanadi. Bu bloklar dasturda ketma-ket joylashadi. Istisno o'zgaruvchisi ixtiyoriy, hattoki foydalanuvchi tomonidan aniqlangan tur ham bo'lishi mumkin.

Bir **try**-блок ortidan bir necha **catch**-блок kelishi mumkin. **catch (...)** operatori, ya'ni qavs ichida uchta nuqta bo'lsa, u holda bu orqali ixtiyoriy turdagi istisnolarni qayta ishlash mumkin.

Dasturda istisnoni joyi va turini aniqlash uchun **throw** kalit so'zidan foydalaniladi. Uning sintaksisi quyidagicha: throw <ifoda>;

Endi yuqorida keltirib o'tilgan misolda istisnolarni qayta ishlashni amalga oshirilgan variantini keltirib o'tamiz:

```
#include <iostream>
double divide(int, int);
int main(){
  int x = 500;
  int y = 0;
  try{
    double z = divide(x, y);
    std::cout << z << std::endl;
  }
  catch (const char* msg) {
    std::cout << msg << std::endl;
  }
  std::cout << "The End..." << std::endl;
  return 0;
}
double divide(int a, int b){
  if (b == 0)
    throw "Division by zero!";
  return a / b;
}
```

1.2.6. Vorislik (merosxo'rlik), virtual funksiyalar va polimorfizm

Vorislik tushunchasiga to'xtalishdan oldin misol ko'rib o'taylik.

Misol.

```
class Shaxs{
public:
    std::string name;     // ism
    int age;              // yosh
    void display(){
      std::cout << "Name: " << name << "\tAge: " << age << std::endl;
    }
};
class Xodim{
public:
    std::string name;     // ism
    int age;              // yosh
    std::string company; // tashkilot
    void display(){
      std::cout << "Name: " << name << "\tAge: " << age << std::endl;
    }
};
```

E'tibor berilsa, Xodim sinfi Shaxs sinfining xususiyatlarini o'z ichiga olmoqda, ya'ni o'zgaruvchilar: ism va yosh; metod - display.

Shunday vaziyatlarda Xodim sinfini e'lon qilishda qulayroq variant yo'qmi?

Faraz qilaylik, bir yoki bir necha sinflar mavjud va siz yaratmoqchi bo'lgan yangi sinfning qaysidir xususiyatlari ushbu sinf yoki sinflarda o'z aksini topgan

bo'lsin. U holda yangi sinfni to'liq qaytadan tavsiflab chiqmasdan, o'sha sinf yoki sinflar imkoniyatlaridan to'liq foydalana oladigan holda sinfni shakllantirish mumkin. Bunday imkoniyat vorislik deb ataladi, ya'ni yangi shakllantirilgan sinf foydalanilgan sinf (yoki sinflar)ning vorisi deyiladi. Albatta bu sinf mavjudlaridan farqli ravishda qo'shimcha imkoniyatlarga ega bo'lishi mumkin. Bunda yangi sinf hosilaviy, mavjudlari asosiy yoki tayanch sinf deyiladi.

Yuqorida keltirib o'tilgan misolda Xodim sinfini vorislik mexanizmidan foydalanib shakllantiramiz:

```
class Shaxs{
public:
  std::string name;     // ism
  int age;              // yosh
  void display(){
    std::cout << "Name: " << name << "\tAge: " << age << std::endl;
  }
};
class Xodim : public Shaxs{
public:
  std::string company;   // tashkilot
};
```

Bu misolda Shaxs sinfi – tayanch, Xodim sinfi – hosilaviy hisoblanadi.

Hosilaviy sinf e'lon qilinayotganda tayanch sinf oldiga ruxsat etish spesifikatori qo'yilishi ham, qo'yilmasligi ham mumkin. Bizning holda **public** spesifikatori qo'yilgan bo'lib, u hosilaviy sinfda tayanch sinfning barcha ochiq ma'lumotlaridan foydalanish imkoniyatini yaratadi. Agar ruxsat etish modifikatori bo'lmasa, u holda hosilaviy sinf tayanch sinf ma'lumotlarini bilmaydi.

Vorislik o'rnatilganidan keyin hosilaviy sinfdan tayanch sinfda mavjud o'zgaruvchilarni chiqarib tashlash mumkin.

```
#include <iostream>
#include <string>
class Shaxs{
public:
  std::string name;     // ism
  int age;              // yosh
  void display(){
    std::cout << "Name: " << name << "\tAge: " << age << std::endl;
  }
};
class Xodim : public Shaxs{
public:
  std::string company;   // tashkilot
};
int main(){
  Shaxs eshmat;
  eshmat.name = "Eshmat";
  eshmat.age = 23;
  eshmat.display();
  Xodim toshmat;
  toshmat.name = "Toshmat ";
```



```
  toshmat.age = 31;
  toshmat.company = "TATU";
  toshmat.display();
  return 0;
}
```

Bunday imkoniyat juda ko'plab qulayliklarga ega, masalan, dasturdagi mavjud kodlardan qayta foydalanish, vorislik mexanizmi yordamida turli vaziyatlarga moslashtirish va hokazolar.

Sinf xossalarni faqat bir tayanch sinfdan vorislik bilan olsa, *yakka (yoki oddiy) vorislik* deyiladi, agar obyekt bir nechta tayanch sinflardan olsa, u holda *to'plamli vorislik* deyiladi.

Quyidagi jadvalda turli vaziyatlarda sinf a'zolariga murojaat qilish imkoniyati keltirib o'tilgan.

1-jadval. Vorislik va murojaat imkoniyati

Murojaat spetsifikatori	Sinfni o'zidan murojaat	Hosilaviy sinfdan murojaat	Tashqi sinf va funksiyalardan murojaat
public	bor	bor	bor
protected	bor	bor	yo'q
private	bor	yo'q	yo'q

Ushbu jadvaldan ko'rinib turibdiki, hosilaviya sinfdan tayanch sinfning faqatgina **public** va **protected** kabi aniqlangan a'zolariga murojaat qilish huquqi berilgan. Agar biror-bir sinf o'z ma'lumotlaridan foydalanishni, ya'ni vorislikni taqiqlamoqchi bo'lsa, u holda sinf e'lon qilinayotganda final spesifikatoridan foydalanadi, masalan,

```
class User final{
};
```

Shuni e'tiborga olish lozimki, konstrukrorlar vorislikda meros bo'lib o'tmaydi. Bunday vaziyatda hosilaviy sinf o'zining konstruktorida tayanch sinfning kerakli bo'lgan konstruktorini chaqirishi lozim bo'ladi.

Masalan,

```
#include <iostream>
#include <string>
class Shaxs{
public:
  Shaxs(std::string n, int a) {
    name = n; age = a;
  }
  void display() {
    std::cout << "Name: " << name << "\tAge: " << age << std::endl;
  }
private:
  std::string name;
  int age;
};
class Xodim : public Shaxs{
public:
  Xodim (std::string n, int a, std::string c): Shaxs(n, a) {
```

```
    company = c;
  }
private:
  std::string company;
};
int main(){
  Shaxs tom("Eshmat", 23);
  eshmat.display();
  Xodim bob("Toshmat", 31, "TATU");
  toshmat.display();
  return 0;
}
```

Nazorat savollari

1. Merosxo'rlik, inkapsulyatsiya va polimorfizmni tushuntiring?
2. Sinf nima? Obyekt nima? Ularning farqini tushuntiring.
3. Konstruktorga toifa beriladimi va qanday?
4. Sinf konktruktori bittadan ko'p bo'lishi mumkinmi? U qanday e'lon qilinadi?
5. Destruktor nima? Uning vazifasi nimadan iborat?
6. Try...catch() operatorining kirish ma'lumoti (argumentidagi e) nima uchun ishlatiladi?

II BO'LIM.
MA'LUMOTLARNI QIDIRISH VA SARALASH USULLARI

2.1. Ma'lumotlarni qidirish va xeshlash algoritmlari

2.1.1. Qidiruv tushunchasi va vazifasi

Kompyuterda ma'lumotlarni qayta ishlashda qidiruv asosiy amallardan biri bo'lib hisoblanadi. Uning vazifasi berilgan argument bo'yicha massiv ma'lumotlari ichidan mazkur argumentga mos ma'lumotlarni topishdan iborat.

Ixtiyoriy ma'lumotlar majmuasi jadval yoki fayl deb ataladi. Ixtiyoriy ma'lumot (yoki tuzilma elementi) boshqa ma'lumotdan biror bir belgisi orqali farq qiladi. Mazkur belgi kalit deb ataladi. Kalit noyob bo'lishi, ya'ni mazkur kalitga ega ma'lumot jadvalda yagona bo'lishi mumkin. Bunday noyob kalitga birlamchi kalit deyiladi. Ikkilamch kalit bir jadvalda takrorlansada u orqali ham qidiruvni amalga oshirish mumkin. Ma'lumotlar kalitini bir joyga yig'ish (boshqa jadvalga) yoki yozuv sifatida ifodalab bitta maydonga kalitlarni yozish mumkin.

Ta'rif. Agar kalitlar ma'lumotlar jadvalidan ajratib olinib alohida fayl sifatida saqlansa, u holda bunday kalitlar tashqi kalitlar deyiladi. Aks holda, ya'ni yozuvning bir maydoni sifatida jadvalda saqlansa ichki kalit deyiladi.

Kalitni berilgan argument bilan mosligini aniqlovchi algoritmga berilgan argument bo'yicha qidiruv deb ataladi.

Qidiruvning maqsadi - quyidagi jarayonlarning birini bajarilishidan iborat:

- topilgan yozuvni o'qish;
- qidirilayotgan yozuv topilmasa, uni jadvalga qo'yish;
- topilgan yozuvni o'chirish.

Jadvaldagi ma'lumotlarning tuzilmasiga qarab qidiruvni bir necha turlari mavjud.

2.1.2. Qidiruv algoritmlari

Faraz qilaylik, k – kalitlar massivi. Har bir k(i) uchun r(i) – ma'lumot mavjud. key – qidiruv argumenti.

1. Chiziqli (ketma-ket) qidiruv

Chiziqli qidiruv eng sodda algoritm hisoblanib, bunda barcha ma'lumotlar butun jadval bo'yicha ma'lum bir tartibda ketma-ket qarab chiqiladi, masalan, operativ xotirada kichik adresdan to katta adresgacha yoki aksincha. Mazkur algoritmdan agar jadvaldagi ma'lumotlar tartibsiz yoki ularning tuzilishi noaniq bo'lganda foydalaniladi.

Massivda ketma-ket qidiruv (search o'zgaruvchi topilgan element raqamini saqlaydi).

i	k	r
1	k_1	...
2	k_2	...
3	k_{p_3}	...
...	...	...
$n-1$	$k_{p_{n-1}}$	...
...	...	...
n	k_n	...

Massivda chiziqli qidiruvni amalga oshirish *(C++)*:

```
int search(int a[], int N, int key){
    int i=0;
    while (i!=N)
      if (a[i]==key) return i;
      else i++;
    return -1;
}
```

Agar ma'lumotlar jadvali bir bog'lamli ro'yxat ko'rinishida berilgan bo'lsa, u holda ketma-ket qidiruv ro'yxatda amalga oshiriladi.

Ro'yxatda chiziqli qidiruvni amalga oshirish (C++):

```
struct TNode {
    int value;
    TNode* pnext;
    TNode(int val): pnext(0), value(val) {}
};
TNode* Find(TNode *phead, int key){
    TNode *p=phead;
    while(p)
      if (p->value==key) return p;
      else p = p->pnext;
    return 0;
}
```

Qidiruv algoritmlarining samaradorlik mezonlari sifatida quyidagilarni keltirib o'tish mumkin:

- kalitlarni taqqoslashlar soni;
- dasturning ishlab chiqishga sarflangan vaqt;
- dasturni ishlashi uchun sarflangan vaqt;
- talab qilinadigan xotira hajmi.

Izoh: *Qidiruv algoritmlari ishlab chiqilayotganida, odatda, asosan, kalitlarni taqqoslashlar soniga e'tibor qaratiladi.*

Faraz qilaylik, qidiruv jadvalidagi ma'lumotlar, ya'ni elementlar soni - n va taqqoslashlar soni – C bo'lsin. Agar ma'lumotlar jadvalda teng ehtimollik bilan taqsimlangan bo'lsa, u holda chiziqli (ketma-ket) qidiruv algoritmi samaradorligi:

$$C = 1 \div n, C_{o'rtacha} = \frac{n+1}{2} \Rightarrow O(n).$$

Eslatma: *Massiv va bog'langan ro'yxatda kerakli elementni bor yoki yo'qligini aniqlash samaradorligi bir xil, ammo topilgan elementni o'chirish yoki biror element jadvalda bo'lmasa, uni jadvalga qo'yish talab qilingan bo'lsa, u holda qidiruvni amalga oshirish ro'yxatda samaraliroq bo'ladi.*

Umuman olganda ketma-ket qidiruv samaradorligini oshirish mumkin.

Faraz qilaylik, jadvalda qidirilayotgan element mavjud. U holda qidiruv amalga oshirilayotgan barcha jadvalni diskret holatga ega tizim sifatida qarash mumkin hamda unda qidirilayotgan elementni topish ehtimolligi – bu tizim i-chi holati ehtimolligi $p(i)$ deb olish mumkin. Unda quyidagi tenglik o'rinli bo'ladi:

$$\sum_{i=1}^{n} p(i) = 1.$$

Jadvalni diskret tizim sifatida qaraganimizda, undagi taqqoslashlar soni diskret tasodifiy miqdorlar qiymatlarini matematik kutilmasini ifodalaydi:

$Z=S=1p(1)+2p(2)+3p(3)+...+np(n)$.

Bundan ko'rinadiki, $p(1) \geq p(2) \geq ... \geq p(n)$ shart bajarilishi maqsadga muvofiq bo'ladi.

Eslatma. Ushbu shart taqqoslashlar sonini kamaytirib, samaradorlikni oshiradi.

Chiziqli qidiruvni mukammallashtirish yo'llari

1) Topilgan elementni jadval boshiga qo'yish orqali jadvalni qayta tartiblash;
2) Transpozitsiya usuli.

Birinchi usulni mag'zi shundan iboratki, berilgan kalitga teng kalitli element jadvalda birinchi element deb o'zlashtiriladi, qolganlari esa suriladi.

Keltirilgan algoritm ro'yxat uchun ham massiv uchun ham o'rinli. Biroq bu algoritm massiv uchun tavsiya qilinmaydi, sababi elementlarni o'rinlashtirishlar soni ancha yuqori bo'ladi.

Topilgan elementni jadval boshiga qo'yishni massivda amalga oshirish

```
int search(int *k, int n, int key){
    int temp;       // almashtirish uchun yordamchi o'zgaruvchi
    for (int i = 0; i<n; i++){
        if (k[i] == key){
            temp = k[i];
            k[i] = k[0];
            k[0] = temp;
            return i;
        }
```

```
    }
    return -1;
}
```

Transpozitsiya usulida topilgan element jadvalda bitta oldingi element bilan o'rin almashtiriladi. Agarda mazkur elementga ko'p murojaat qilinsa, bir qadam oldinga surilib borib natijada jadval boshida bo'lib bo'ladi.

Ushbu usul nafaqat ro'yxatda, balki massivda ham qulay (sababi faqatgina ikkita yonma-yon turgan element o'rin almashtiriladi).

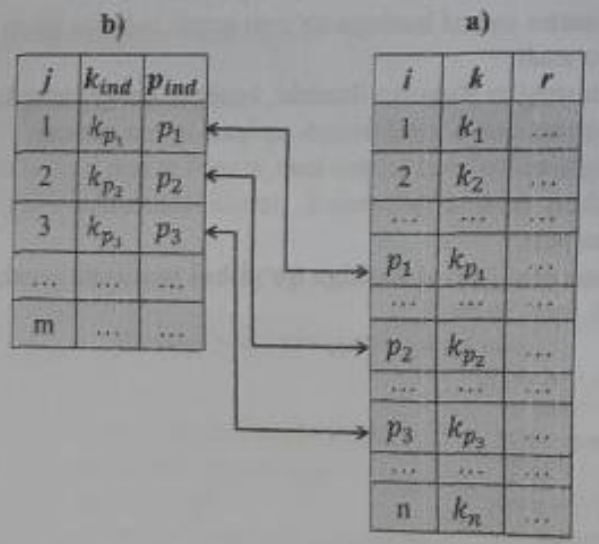
Transpozitsiya usulini massivda amalga oshirish

```
int search(int *k, int n, int key){
    int temp;        // almashtirish uchun yordamchi o'zgaruvchi
    for (int i = 0; i<n; i++){
      if (k[i] == key){
                if (i == 0)
                return i;
                temp = k[i];
                k[i] = k[i - 1];
                k[i - 1] = temp;
                return i;

      }
    }
    return -1;
}
```

2. Indeksli ketma-ket qidiruv

Faraz qilaylik, qidiruv jadvali elementlari ma'lum bir tartibda saralangan bo'lsin.

Indeksli ketma-ket qidiruv amalga oshirilayotganda ikkita jadval tashkil qilinadi: asosiy ma'lumotlar jadvali va indekslar jadvali. Indekslar jadvali asosiy jadvaldan ma'lumotlarni ma'lum bir tartibda olish orqali shakllantiriladi (2.1 -rasm).



2.1-rasm. a) asosiy ma'lumotlar jadvali; b) indekslar jadvali

Dastlab, qidirilayotgan kalit joylashishi mumkin bo'lgan oraliq indekslar jadvali orqali aniqlanadi. Natijada asosiy ma'lumotlar jadvalida qidiruv oralig'ini quyi chegarasi (low) va yuqori chegarasi (hi) o'rnatiladi.

Qidiruv to'liq jadval bo'yicha emas, balki low dan hi gacha olib boriladi.

Indeksli ketma-ket qidiruvni amalga oshirish

```
int InSeqsearch(int realArray[], int N, int kind[2][1000],int m,int key, int *t){
    int i=0,
    low = 0,
    hi = 0;
    while ((i<m) && (kind[0][i]<=key)){
        i++;
        (*t)++;
    }
    (*t)++,
    if (i==0)   low=0;
     else   low=kind[1][i-1];
    if (i==m)   hi=N;
      else   hi=kind[1][i]-1;
    for (int j=low; j<=hi; j++) {
      (*t)++;
      if( key==realArray[j] ){
        return j;
      }
    }
    return -1;
}
```

Indeksli ketma-ket qidiruv usulining samaradorligi

Agar bo'lishi mumkin bo'lgan barcha holatlar teng ehtimolli deb olinsa, u holda qidiruv samaradorligini quyidagicha aniqlash mumkin:

$C = (m+1)/2 + (p+1)/2 = (n/p+1)/2 + (p+1)/2 = n/2p+p/2+1,$

bu yerda n – asosiy ma'lumotlar jadvali o'lchami, m – indekslar jadvali o'lchami ($m = n / p$), p – qadam o'lchami.

Bundan ko'rinib turibdiki, indeksli ketma-ket qidiruv algoritmi samaradorligi qadam o'lchamiga bog'liq ekan.

$dC/dp=(d/dp)\ (n/2p+p/2+1) = -n / 2p^2 + 1/2 = 0,$

$$p^2 = n \rightarrow p_{opt} = \sqrt{n}.$$

Demak,

$C_{opt} = \sqrt{n} + 1.$

Indeksli ketma-ket qidiruv samaradorligi $O(\sqrt{n})$ ga teng.

3. Binar qidiruv usuli (oraliqni ikkiga bo'lish orqali)

Faraz qilaylik, o'sish tartibida tartiblangan sonlar massivi berilgan bo'lsin. Binar qidiruv usulini asosiy g'oyasi shundan iboratki, tasodifiy qandaydir a_m element olinadi va u x qidiruv argumenti bilan taqqoslanadi. Agar $a_m = x$ bo'lsa, u holda qidiruv yakunlanadi; agar $a_m < x$ bo'lsa, u holda indekslari m dan kichik yoki teng bo'lgan barcha elementlarni kelgusi qidiruvdan chiqarib yuboriladi. Xuddi shuningdek, agar $a_m > x$ bo'lsa.

M ixtiyoriy tanlanganda ham taklif qilinayotgan algoritm korrekt ishlaydi. Shu sababali m ni shunday tanlash lozimki, tadqiq qilinayotgan algoritm samaraliroq natija bersin, ya'ni uni shunday tanlaylikki, iloji boricha kelgusi jarayonlarda ishtirok etuvchi elementlar soni kam bo'lsin. Agar biz o'rtacha elementni, ya'ni massiv o'rtasini tanlasak yechim mukammal bo'ladi.

Binar qidiruvni amalga oshirish (C++ da)

```
int Binsearch(int a[], int N, int key, int *t){
    int l=0, r=N-1, mid=(l+r)/2;
    while (l<=r){
     *t+=1;
     if (a[mid]==key) return mid;
     if (a[mid]>key) r=mid-1;
     else l=mid+1;
     mid=(l+r)/2;
    }
    a[N]=key;
    return N;
}
```

Binar qidiruv samaradorligi: $O(log_2 n)$.

2.1.3. Xesh jadval va xesh funksiyalar

Yuqorida ko'rib o'tilgan barcha qidiruv usullarida algoritm samaradorligi qidiruv jadvali hajmi va ma'lumotlarning joylashuvuga bog'liq bo'lib, eng yaxshi holatda samaradorlik $O(log_2 n)$ ga proporsional edi. Quyida biz qidiruvni qidiruv jadvali hajmiga bog'liq bo'lmagan holda amalga oshirish yo'llarini ko'rib chiqamiz.

To'g'ridan-to'g'ri murojaat jadvali usuli

Ideal tez qidiruvni ta'minlab beruvchi jadval – bu to'g'ridan-to'g'ri murojaat jadvali usulidir. Bunda kalit jadvaldagi yozuv adresi bo'lib hisoblanib, o'zaro teng bo'lmagan kalitlar turli adreslarga akslantiriladi. Dastlab, jadval yaratishda, barcha yozuvlarni saqlash uchun xotira ajratiladi va u bo'sh yozuvlar bilan to'ldiriladi. Keyin har bir yozuv o'zining kaliti yordamida jadvaldagi o'z o'rniga joylashtiriladi. Qidiruv amalga oshirilayotganda kalitdan adres sifatida foydalaniladi va mazkur adresdagi yozuv tekshiriladi. Agar ko'rsatilgan adres bo'sh bo'lsa, u holda jadvalda mazkur kalitga ega bo'lgan yozuv yo'q hisoblanadi. Foydalanishda to'g'ridan-to'g'ri murojaat jadvali juda samarali, lekin uni tatbiq qilish ko'lami ancha chegaralangan.

Yozuv kalitlarining nazariy jihatdan qabul qilishi mumkin bo'lgan barcha qiymatlar to'plami kalitlar fazosi, jadvalni saqlash uchun ajratilgan xotira yacheykalari to'plami esa yozuvlar fazosi deb ataladi.

To'g'ridan-to'g'ri murojaat jadvali usulidan yozuvlar fazosi o'lchami kalitlar fazosi o'lchamiga teng bo'lgandagina foydalanish tavsiya etiladi. Afsuski, ko'pincha, yozuvlar fazosi o'lchami kalitlar fazosi o'lchamidan ancha kichik bo'ladi. Masalan, faraz qilaylik, xodimlarning ma'lumotlar bazasini kalit sifatida ismlardan foydalanib, to'g'ridan-to'g'ri murojaat jadvali usulida shakllantirish rejalashtirilgan. Agar har bir ism 9 tagacha harfdan iborat bo'lsa, u holda kalitlar fazosi

o'lchami 26^9 ga teng bo'ladi (agar alifbo 26 ta xarfdan iborat bo'lsa). Hisoblash tizimi bunday o'lchamdagi yozuvlarlarni kiritish resursiga ega bo'lgan taqdirda ham, jadvalni to'ldirish natijasida uning kattagina qismi bo'sh yozuvlardan iborat bo'ladi. Chunki, real bo'lishi mumkin bo'lgan kalitlar to'plami mazkur jadvalni qoplay olmaydi.

Shulami inobatga olgan holda hamda xotirani tejash maqsadida to'g'ridan-to'g'ri murojaat jadvali o'lchamini real yozuvlar to'plami o'lchamiga teng yoki undan biroz kattaroq qilib belgilash tavsiya etiladi.

Xeshlash (*hashing*) – bu yozuv(element)larni jadvalga joylashtirish usuludir. Bunda elementlarning jadvaldagi o'rni ularning kalit qiymatlarini massiv indekslari qiymatiga akslantirish orqali aniqlanadi. Shu sababli, xeshlash «*kalitlarni akslantirish*» deb ham ataladi. Akslantirishni amalga oshiruvchi funksiyaga xesh funksiya (*hash function*) deyiladi.

Umumiy holda xeshlash quyidagicha amalga oshiriladi:

$A = h(k)$, bu yerda h-xesh funksiya, k – element kaliti, A – jadvaldagi element adresi, ya'ni massiv indeksi, $0 \le A \le n-1$, n-massiv o'lchami.

Xesh funksiya orqali elementlar joylashtirilgan jadval xesh jadval (*hash table*) deyiladi

Boshqacha aytadigan bo'lsak, xesh jadval – juftliklarni (kalit yoki indeks + element) saqlovchi ma'lumotlar tuzilmasi bo'lib, unda uchta amal aniqlangan:

- ✓ qo'shish, ya'ni jadvalga yangi juftlikni kiritish;
- ✓ qidirish;
- ✓ o'chirish, ya'ni jadvaldan juftlikni o'chirish.

Xesh jadvallarda qidiruv ikki bosqichda amalga oshiriladi:

1) qidiruv kalitini jadval adresiga akslantiruvchi xesh funksiyani hisoblash;
2) ziddiyatlarni hal qilish jarayoni.

h - xesh funksiya ideal deyiladi, agar $\forall k_1, k_2 \in K$ va $k_1 \neq k_2$ uchun $h(k_1) \neq h(k_2)$ bo'lsa. Bu yerda K – kalitlar fazosi.

Agar $\exists k_1, k_2 \in K$ va $k_1 \neq k_2$ mavjudki, bular uchun $h(k_1) = h(k_2)$ o'rinli bo'lsa, u holda ziddiyat (collision) yuzaga keldi deyiladi. Bunday kalitlar sinonim deyiladi. Xesh jadvallardagi asosiy muammo bu ziddiyatlarni mavjudligidir.

Agar kalitlarni adreslarga akslantiruvchi xesh funksiya ziddiyat keltirib chiqarishi mumkin bo'lsa, u holda kalit ham yozuvning bir maydoni sifatida xesh jadvalda joylashishi lozim.

Umumiy holda xesh funksiyaga quyidagicha talablar qo'yiladi:

– mavjud kalitlarni oraliqlarga teng taqsimlasin;
– berilgan yozuvlar to'plami uchun imkon qadar kam ziddiyat vujudga keltirsin;
– sodda va tez hisoblansin.

2.1.4. Xesh funksiyalarga misollar

Quyida xesh funksiyasini aniqlashning eng keng tarqalgan usullarini keltirib o'tamiz.

1. Bo'lish usuli. Faraz qilaylik, k kalit qiymatlari butun sonlardan iborat va jadval o'lchami m bo'lsin. Mazkur usulda xesh funksiya quyidagicha aniqlanadi:

$h(k) = k \bmod m$.

Ushbu funksiyaning natijasi berilgan kalitni jadval o'lchamiga bo'lganda qolgan qoldiq bo'ladi.

Mazkur funksiya C++ da quyidagicha amalga oshiriladi:

```
int h(int key, int m) {
return key % m;
}
```

2. Additiv usul. Bunda kalit belgili satrlardan iborat bo'ladi. Bu usulda ham xesh funksiya *bo'lish usuli* kabi bo'lib, faqatgina, dastlab kalit qiymati butun songa akslantiriladi, ya'ni

```
int h(char *k, int m) {
  int s = 0;
  while(*k)
    s += *k++;
  return s % m;
}
```

Agar satrlar bir xil belgilardan tashkil topgan bo'lsa, u holda ziddiyatlar yuzaga keladi, masalan, $k_i = abc$ va $k_j = cab$.

Agar xesh funksiya tanlashdan oldin kalit qiymatlari bo'yicha qandaydir qo'shimcha ma'lumotga ega bo'lsak, u holda mazkur usulni nisbatan takomillashtirish mumkin, masalan, kalitni faqatgina birinchi va oxirgi belgisi qiymatlari yig'indisini olish orqali.

```
int h(char *k, int m) {
  int len = strlen(k), s = 0;
  if(len < 2) // agar kalit uzunligi 0 yoki 1 ga teng
    s = k[0]; // k[0] qaytar
  else
    s = k[0] + k[len-1];
  return s % m;
}
```

3. Kvadratlar o'rtasi usuli. Bunda kalit qiymati kvadratga oshiriladi va hosil bo'lgan qiymatning o'rtasidagi bir qancha raqamlaridan indeks sifatida foydalaniladi.

Masalan, kalit 32 bitli butun son, xesh funksiya uning kvadratini o'rtadagi 10 bitini qaytaradi:

```
int h(int k) {
  k *= k;
  k >>= 11; // 11 ta kichik bitlarni tashlab yuboramiz
  return k % 1024; // 10 ta kichik bitni qaytaramiz
}
```

Bundan tashqari kvadratlar o'rtasi usulidan foydalanib qurilayotgan xesh funksiyalarni turlicha olish mumkin. Masalan, $k = 234583$, demak $k^2 = 55029183889$. U holda xesh funksiyani jadval o'lchamiga nisbatan quyidagicha aniqlab olish mumkin.

Jadval o'lchami	h(k)
100	91
1000	918
10000	2918
736	918*0.736=676

4. O'ramlar usuli. Mazkur usulda kalitlar jadval o'lchamiga mos ravishda qismlarga ajratiladi. Adres, ya'ni xesh funksiya qiymati mazkur qismlarning yig'indisi ko'rinishida shakllantiriladi, agar bunda yig'indi katta razryadga o'tib ketsa, u e'tiborga olinmaydi. Masalan, k=3415768898.

Jadval o'lchami (necha xonali son)	h(k) (kalit chapdan o'ngga qismlarga ajratilgan)	h(k) (kalit o'ngdan chapga qismlarga ajratilgan)
2	34+15+76+88+98=11	34+15+76+88+98=11
3	341+576+889+9=014	3+415+768+898=084
4	3415+7688+98=1112	34+1576+8898=0508

Odatda, mazkur usuldan kalit qiymatlari katta bo'lgan hollarda foydalaniladi. Amaliyot shuni ko'rsatadiki, kalitlarni qismlarga o'ngdan chapga ajratish chapdan o'ngga ajratishga nisbatan afzalroq.

5. Ko'paytma usuli. Mazkur usulda xesh funksiyani qurishda tasodifiy $r \in (0,1)$ haqiqiy sondan foydalanadi:

$h(k) = [m * (k * r \bmod 1)]$.

Bu usuldan foydalanilganda, jadval o'lchami m va tasodifiy son r ni quyidagicha tanlash tavsiya etiladi:

$m = 2^p$, $r = (\sqrt{5} - 1)/2$.

Misol. Faraz qilaylik, $k = 123456$, $m = 1024$, $r = 0{,}61803$.

$h(k) = [1024 * (123456 * 0{,}61803 \bmod 1)] = [1024 * 0{,}51168] = 523$.

2.1.5. Ziddiyatlarni hal qilish

Yuqorida ko'rib o'tilgan xesh funksiyalar faqatgina kalitlarni xesh jadvalga akslantirishni ifodalab, ziddiyatlar yuzaga kelishini e'tiborga olmaydi. Shu sababli, xeshlash sxemasi ziddiyatlarni hal qilish algoritmini ham o'z ichiga olishi lozim bo'ladi.

Xeshlash sxemalari

Garchi, ko'plab masalalarda, ikki va undan ortiq kalitlar bir xil xeshlansada, lekin ular xesh jadvalda bitta adresga joylasha olmaydi. Bunday vaziyatlarda muammoni hal qilishning ikkita varianti mavjud: yoki yangi kalit uchun boshqa o'rin topish yoki xesh jadvalning har bir shunday indeksi uchun alohida ro'yxat yaratish.

Ushbu variantlar ikkita klassik sxemani namoyon qiladi:

– zanjirlar usuli bilan xeshlash;

– ochiq indeksatsiya usuli bilan xeshlash (yopiq xeshlash).

Zanjirlar usuli bilan xeshlash ko'pincha ochiq xeshlash deb ham nomlanadi. Mazkur usulda bir xil xeshga ega elementlar bog'lamli ro'yxat ko'rinishidagi bitta indeksga o'tadi, ya'ni, agar kalit akslantirilgan indeks band va bu kalit mazkur indeksni egallagan kalitdan farq qilsa, u holda yangi element kalit-qiymat juftligi ko'rinishida ro'yxatga kiritiladi.

Zanjirlar usulida jadvalga yangi element qo'shish samaradorligi $O(1)$, elementni topish (qidirish) samaradorligi esa ro'yxat uzunligiga bog'liq va eng yomon holatda $O(n)$ bo'ladi. Agar kalitlar soni – n va ular m – indeksga taqsimlanadigan bo'lsa, u holda to'ldirish koeffitsienti n/m ga teng bo'ladi.

Zanjirlar usuli C++ da quyidagicha amalga oshiriladi:

```
class LinkedHashEntry {
private:
    int k;
    int value;
    LinkedHashEntry *next;
public:
    LinkedHashEntry(int k, int value) {
        this->k = k;
        this->value = value;
        this->next = NULL;
    }
    int getKey() {
        return k;
    }
    int getValue() {
        return value;
    }
    void setValue(int value) {
        this->value = value;
    }
    LinkedHashEntry *getNext() {
        return next;
    }
    void setNext(LinkedHashEntry *next) {
        this->next = next;
    }
};
```

Ochiq indeksatsiya usulida kalit-qiymat juftliklari bevosita xesh jadvalda saqlanadi. Xesh jadvalga elementni qo'yish algoritmi yangi elementga joy topilgunga qadar, ma'lum bir tartibda, bo'sh kataklarni tekshirib boradi.

Mazkur usulni amalga oshirishning eng sodda yollaridan biri bu chiziqli zondlash (yoki chiziqli tekshirish), ya'ni agar ziddiyat yuzaga kelsa, bo'sh kataklar topilguniga qadar navbatdagi kataklar ketma-ket ravishda tekshirilib boriladi.

Qo'yish algoritmi qaysi tartibda ishlasa qidiruv algoritmi ham shu tartibga mos ravishda qidiruvni amalga oshiradi. Qidiruv toki kerakli element yoki bo'sh katak topilguncha davom etadi. Ikkinchi hol mazkur kalitga ega element jadvalda yo'qligini anglatadi.

Ochiq indeksatsiya uchun chiziqli zondlash usulini C++ da amalga oshirish:

```
class HashEntry {
private:
    int k;
    int value;
public:
    HashEntry(int k, int value) {
        this->k = k;
        this->value = value;
    }
    int getKey() {
        return k;
    }
    int getValue() {
        return value;
    }
    void setValue(int value) {
        this->value = value;
    }
};
```

Nazorat savollari

1. Qidiruv vazifasi nimadan iborat?
2. Noyob kalit deganda nimani tushunasiz?
3. Ro'yxatda berilgan kalitli element yo'q bo'lganda qaysi amal bajariladi?
4. Ketma-ket qidiruv va indeksli ketma-ket qidiruvlarning farqi nimadan iborat?
5. Ulardan qaysi biri samaraliroq va nima sababdan?
6. Jadvalni qayta tartiblashning qanday usullarini bilasiz?
7. Topilgan elementni boshiga qo'yish usulining transpozitsiya usulidan asosiy farqlari nimalardan iborat?
8. Mazkur usullar ma'lumotlar qanday ko'rinishda berilganda tezroq ishlaydi?
9. Ular qanday ro'yxatlarda ishlaydi, ya'ni tartiblangan yoki ixtiyoriy?
10. Binar qidiruvning mazmun va mohiyati nimadan iborat?
11. Binar qidiruvni massivda ishlatish mumkinmi?

2.2. Ma'lumotlarni saralash algoritmlari

2.2.1. Ma'lumotlarni saralash tushunchasi

Saralash – bu berilgan to'plam elementlarini biror bir tartibda joylashtirish jarayonidir. Saralashning maqsadi tartiblangan to'plamda kerakli elementni topishni osonlashtirishdan iborat. Saralash dasturlarni translyatsiya qilinayotganda, ma'lumotlar majmuasini tashqi xotirada tashkil qilinayotganda, kutubxonalar, kataloglar, ma'lumotlar bazasi yaratilayotganda tatbiq qilinadi. Ma'lumki, saralashning turli xil algoritmlari mavjud. Sababi, bironta tuzilmani saralash uchun juda ko'plab turli xil algoritmlardan foydalanish mumkin. Berilgan masalani hal qilishda ba'zilari mukammal bo'lishi mumkin. Shuning uchun saralash masalasida algoritmlarni qiyosiy tahlilini o'tkazish zarurati paydo bo'ladi.

Saralash masalasining qo'yilishini quyidagicha yozish mumkin.

Faraz qilaylik, $a_1, a_2, \ldots, a_n$, elementlar ketma-ketligi berilgan bo'lsin. U holda saralash algoritmi elementlarni massivga shunday joylashtiradiki, natijada ular

qandaydir munosabatga nisbatan $f(a_{k1}) \le f(a_{k2}) \le \ldots \le f(a_{kn})$ tartibga ega bo'ladi. Odatda f tartiblash funksiyasi qandaydir maxsus qoida bilan hisoblanmasdan, balki elementni kalit qiymati bo'yicha massiv elementlari tartiblanadi.

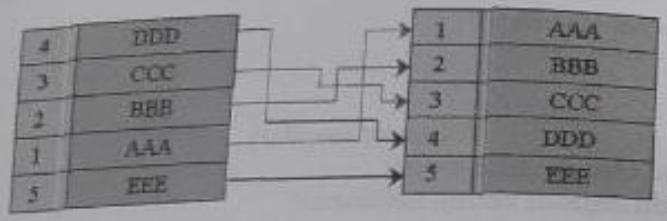
Ma'lumotlarga qayta ishlov berilayotganda ma'lumotni informatsion maydonini hamda uni mashinada joylashishini (adresini) bilish zarur.

Saralashni ikkita turi mavjud: ichki va tashqi:

– ichki saralash bu operativ xotiradagi saralash;

– tashqi saralash – tashqi xotirada saralash.

Saralash bu ma'lumotlarni kalitlari bo'yicha xotirada regulyar ko'rinishda joylashtirishdir. Regulyarlik deganda ma'lumotlar kalit qiymatlari bo'yicha massivda boshidan oxirigacha o'sishi yoki kamayishi tushiniladi.



Agar saralanayotgan yozuvlar xotirada katta hajmni egallasa, u holda ularni almashtirishlar katta sarf (vaqt va xotira ma'nosida) talab qiladi. Ushbu sarfni kamaytishi maqsadida, saralash kalitlar adresi jadvalida amalga oshiriladi. Bunda faqatgina ma'lumot ko'rsatkichlari almashtirilib, massiv o'z joyida qoladi. Mazkur usul adreslar jadvalini saralash usuli deyiladi.

Saralanayotganda bir xil kalitlar uchrashi mumkin, bu holda saralanagandan keyin bir xil kalitlilar boshlang'ich tartibda qanday joylashgan bo'lsa, ushbu tartibda qoldirilishi maqsadga muvofiq bo'ladi (Bir xil kalitlilar o'zlariga nisbatan). Bunday usulga turg'un saralash deyiladi.

2.2.2. Saralash algoritmlari va ularning samaradorligi

Saralash samaradorligini bir necha mezonlar bo'yicha baholash mumkin:

– saralashga ketgan vaqt;

– saralash uchun talab qilingan operativ xotira;

– dasturni ishlab chiqishga ketgan vaqt.

Birinchi mezonni qarab chiqaylik. Saralash bajarilganda taqqoslashlar yoki almashtirishlar sonini hisoblash mumkin.

Faraz qilaylik, $N = 0{,}01n^2 + 10n$ – taqqoslashlar soni. Agar $n < 1000$ bo'lsa, u holda ikkinchi qo'shiluvchi katta, aks holda ya'ni, $n > 1000$ bo'lsa, birinchi qo'shiluvchi katta bo'ladi.

Demak, kichkina n larda taqqoslashlar soni n ga teng bo'ladi, katta n larda esa n^2 ga teng bo'ladi.

Saralashda taqqoslashlar soni quyidagi oraliqlarda bo'ladi:

$O(n \log n)$ dan $O(n^2)$ gacha; $O(n)$ – ideal holatda.

Saralashni quyidagicha usullari bor:

– qat'iy (to'g'ridan-to'g'ri) usullar;

– yaxshilangan usullar.

Qat'iy usullar:

1. to'g'ridan-to'g'ri qo'shish usuli;
2. to'g'ridan-to'g'ri tanlash usuli;
3. to'g'ridan-to'g'ri almashtirish usuli.

Yuqorida keltirilgan uchala usulda ham almashtirishlar soni deyarli bir xil bo'ladi.

2.2.3. Saralashning oddiy algoritmlari

To'g'ridan-to'g'ri qo'shish usuli bilan saralash

Bunday usul karta o'yinida keng qo'llaniladi. Elementlar (kartlar) hayolan "tayyor" a(1),...,a(i-1) va boshlang'ich ketma-ketliklarga bo'linadi. Har bir qadamda (i=2 dan boshlanib, har bir qadamda bir birlikka oshirib boriladi) boshlang'ich ketma-ketlikdan i-chi element ajratib olinib tayyor ketma-ketlikning kerakli joyiga qo'shiladi.

Taklif qilinayotgan usulni quyidagi misolda ko'rib chiqamiz.

Faraz qilaylik, kalit qiymati 4, 5, 3, 8, 1, 7 bo'lgan elementlar berilgan bo'lsin.

i = 2	4	5	3	8	1	7
i = 3	4	3	5	8	1	7
	3	4	5	8	1	7
i = 4	3	4	5	8	1	7
i = 5	1	3	4	5	8	7
i = 6	1	3	4	5	7	8

Kerakli joyni qidirish jarayonini quyidagi tartibda olib borish qulay bo'ladi. Taqqoslashlar amalga oshirish mobaynida, navbatdagi a(j) element bilan solishtiriladi, keyin esa x bo'sh joyga qo'yiladi yoki a(j) o'nga suriladi va jarayon chapga "ketadi". Shuni e'tiborga olish lozimki, saralash jarayoni quyidagi shartlarni birortasi bajarilganda yakunlanadi:

1. x elementi kalitidan kichik kalitli a(j) element topildi.
2. tayyor ketma-ketlikning chap tomoni oxiriga yetib borildi.

Taklif etilayotgan usul algoritmi quyidagicha bo'ladi:

```
void StraightInsertion(int *a){
```

```
int x, j;
for (int i=1;j<n;i++){
  x=a[i]; j=i;
  while (x<a[j-1]){
        a[j]=a[j-1]; j=j-1; }

  a[j]=x;
 }
}
```

Algoritm samaradorligi

Faraz qilaylik, taqqoslashlar soni C, o'rinlashtirishlar soni M bo'lsin. Agar massiv elementlari kamayish tartibida bo'lsa, u holda taqqqoslashlar soni eng katta bo'lib, u $C_{max} = \frac{n(n-1)}{2}$ ga teng bo'ladi, ya'ni $O(n^2)$. O'rinlashtirishlar soni esa $M_{max} = C_{max} + 3(n-1)$ ga teng bo'ladi, ya'ni $O(n^2)$. Agar berilgan massiv o'sish tartibida saralangan bo'lsa, u holda taqqoslashlar va o'rinlashtirishlar soni eng kichik bo'ladi, ya'ni $C_{min} = n-1$, $M_{min} = 3(n-1)$.

To'g'ridan-to'g'ri tanlash usuli bilan saralash

Faraz qilaylik, $a_0, a_1, \ldots, a_{n-1}$ elementlar ketma-ketligi berilgan bo'lsin.

Mazkur usul quyidagi tamoyillarga asoslangan:

1. Berilgan elementlar ichidan eng kichik kalitga ega element tanlanadi.

2. Ushbu element boshlang'ich ketma-ketlikdagi birinchi element a_0 bilan o'rin almashadi.

3. Undan keyin ushbu jarayon qolgan n-1 ta element, n-2 ta element va hokazo, toki bitta eng "katta" element qolguncha davom ettiriladi.

Taklif qilinayotgan usul algoritmining C++ tilidagi dasturi quyidagicha bo'ladi:

```
int StraightSelection(int *a){
  int k;
  for(int i=0;j<n-1;j++)
  for(int j=i+1;j<n;j++)
    if(a[i]>a[j]) swap(a[i],a[j]);
}
```

Algoritm samaradorligi:

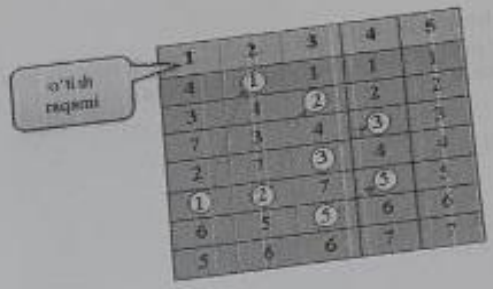
Taqqoslashlar soni $M = \frac{n}{2}(n-1) = \frac{n^2-n}{2}$.

Almashtirishlar soni $C_{min}=3(n-1)$, $C_{max}=3(n-1)\frac{n}{2}$ (n^2 tartib).

Ushbu usul bo'yicha saralash bajarilsa, eng yomon holda taqqoslashlar va almashtirishlar soni tartibi n^2 bo'ladi.

To'g'ridan-to'g'ri almashtirish usuli bilan saralash (pufaksimon)

Ushbu usulni g'oyasi quyidagicha: n - 1 marta massivda quyidan yuqoriga qarab yurib kalitlar jufti-jufti bilan taqqoslanadi. Agar pastki kalit qiymati yuqoridagi jufti kalitidan kichik bo'lsa, u holda ular o'rni almashtiriladi.



C++ tilidagi dasturi:

```
int bubble_sort(int *a){
    int x;
    for (int i=0;j<n-1;i++)
    for (int j=n-1;j>i;j--)
      if (a[j - 1] > a[j]) {
            x = a[j - 1];
            a[j - 1] = a[j];
            a[j] = x;
      }
}
```

Bizning holatda bitta o'tish "bekor" bo'ldi. Elementlarni ortiqcha o'rinlashtirmaslik uchun bayroqcha kiritish mumkin.

Pufaksimon usulni yaxshilangan usuli bu sheyker saralash usuli bo'lib, har bir o'tishdan keyin sikl ichida yo'nalish o'zgartiriladi.

Algoritm samaradorligi:

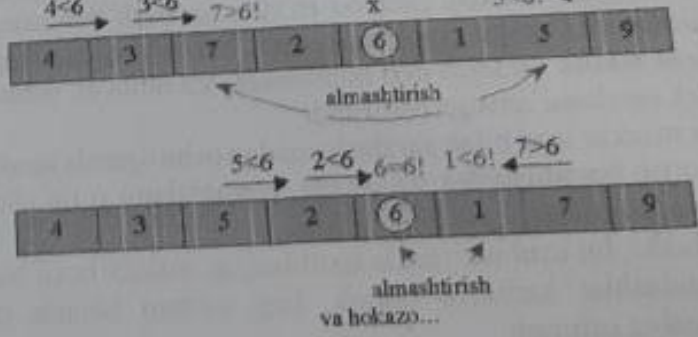
taqqoslashlar soni $M = \frac{n}{2}\cdot\frac{n}{2} = \frac{n^2}{4}$,

almashtirishlar soni $C_{max} = 3\frac{n^2}{4}$.

2.2.4. Takomillashtirilgan saralash algoritmlari

Quicksort – tez saralash usuli

G'oyasi: Bu usul almashtirish usulidagi saralashga tegishli bo'lib, uning asosini kalitlarni tanlangan kalitga nisbatan ajratish tashkil qiladi.



6 dan chap tomonda kalitlari kichik, o'ng tomonda esa kalitlari 6 dan katta bo'lgan elementlar joylashadi (yuqoridagi chizma).

```
int Sort(int L, int R){
 int i = L;
 int j = R;
 int x = a[(L + R) / 2];
 int y;
 while(i < j){
  while(a[i] < x) i = i + 1;
  while(a[j] > x) j = j - 1;
      if(i <= j){
      y = a[i];
      a[i] = a[j];
      a[j] = y;
      i = i + 1;
      j = j - 1;
      }
      }
      if(L < j) Sort (L, j);
      if(i < R) Sort (i, R);

}
int main(){
  Sort(0, n);
}
```

Algoritm samaradorligi:

$O(n \log n)$ – eng samarali usul.

Shella saralashi (qisqarib boruvchi qadamlar orqali saralash)

To'g'ridan-to'g'ri qo'yish orqali saralash usulini 1959 yilda D. Shell tomonidan mukammallashtirish taklif qilingan. Quyidagi chizmada ushbu usul tasvirlangan:

saralash	44	55	12	42	94	18	6	67
keyin to'rtlik	44	18	6	42	94	55	12	67
ikkilik	6	18	12	42	44	55	94	67
yakkalik	6	12	18	42	44	55	67	94

Boshida bir biridan 4 qadamda joylashgan elementlar o'zaro guruhlanib saralash amalga oshiriladi. Bunday jarayon to'rtlik saralash deb ataladi. Birinchi o'tishdan keyin elementlar qayta guruhlanib, endi har ikki qadamdagi elementlar taqqoslanadi. Bu esa ikkilik saralash deb nomlanadi. Va nihoyat, uchinchi o'tishda oddiy yoki yakkalik saralashi amalga oshiriladi.

Bir qarashda mazkur usul bilan saralash amalga oshirilganda saralash jarayoni kamayish o'rniga ortib boradigandek tuyulsada, elementlarni o'rin almashtirishlar nisbatan kam amalga oshiriladi.

Ko'rinib turibdiki, bu usul natijasida tartiblangan massiv hosil bo'lib, har bir o'tishdan keyin saralashlar kamayib boradi. Eng yomon holatda oxirgi ishni yakkalik saralash amalga oshiradi.

Baryer usulidan foydalanilganda har bir saralash o'zining baryeriga ega bo'lishi lozim hamda dastur uning joyini aniqlashi uchun uni iloji boricha osonlashtirish lozim.

Shella saralash algoritmining C++ tilidagi dastur kodini keltiramiz. Bunda a[n] - saralanayotgan massiv va k – saralash qadami.

```
int shellSort(int a[], int n){
        for (int k = n/2; k > 0; k /= 2) {
                for (int i = k; i < n; i += 1){
                        int t = a[i];
                        int j;
                        for (j = i; j >= k && a[j - k] > t; j -= k)
                                a[j] = a[j - k];
                        a[j] = t;
                }
        }
        return 0;
}
```

Umuman olganda, qanday qadamlar tanlanganda eng yaxshi natija olinishi isbotlanmagan bo'lsada, lekin bu qadamlar biri ikkinchisini ko'paytuvchilari bo'lmasligi lozimligi aniqlangan.

D. Knut qadamlarni quyidagicha ketma-ketligini taklif qilgan (teskari tartibda): 1,3,7,15,31,…,ya'ni: $h_{m-1}=2h_m+1$, $h_t=1$, $t = [\log_2 n] - 1$. Agar qadamlar ushbu ko'rinishda aniqlansa, algoritm samaradorligi tartibi $O(n^{1,2})$.

Nazorat savollari

1. Saralash deganda nimani tushunasiz?
2. Saralashning asosiy usullarini aytib bering.
3. Saralashning qaysi usulari qat'iy usulga tegishli?
4. Saralashning yaxshilangan usullarini aytib bering.
5. Qanday saralash turg'un deyiladi?
6. To'g'ridan-to'g'ri qo'shish usuli g'oyasi nimadan iborat?
7. To'g'ridan-to'g'ri tanlash usuli g'oyasi nimadan iborat?
8. To'g'ridan-to'g'ri almashtirish usuli g'oyasi nimadan iborat?
9. Yuqoridagi usullarning bir-biridan farqini aytib bering.
10. Qaysi saralash usuli eng samarali hisoblanadi?
11. Shella usuli qaysi asosiy saralash usuliga tegishli?

III BO'LIM.
CHIZIQLI MA'LUMOTLAR TUZILMASI

Chiziqli ma'lumotlar tuzilmasi - bu chiziqli munosabatga ega bo'lgan, bir xil toifadagi elementlardan iborat tuzilmadir. Bunday ma'lumotlar tuzilmasiga:

- ketma-ketlik (massiv, yozuv, jadval);
- yarimstatik tuzilmalar (stek, navbat, dek, ustuvor navbat);
- chiziqli ro'yxatlar;
- satrlar kiradi.

Chiziqli tuzilmalarda barcha elementlar o'zaro teng huquqlidir, ular orasida tabaqalanish va bo'ysunish kabi tushunchalar bo'lmaydi. Ular ustida quyidagi amallarni bajarish mumkin:

- yaratish;
- elementga murojaat qilish;
- yangi element kiritish va bironta elementni o'chirish;
- ikkita tuzilmani birlashtirish yoki tuzilmani ikkitaga ajratish;
- nusxalash;
- ro'yxatda elementar sonini aniqlash;
- saralash;
- qidirish;
- tuzilmani o'chirish.

Tanlangan tuzilma shakliga qarab uni xotirada tashkil etish va ustida amal bajarish amallarini tanlash mumkin. Misollar bilan har birini ko'rib chiqamiz.

3.1. Massivlar

3.1.1. Statik massivlar

Massiv – bu bir xil toifadagi elementlarning tartibli ketma-ketligidir. Massiv bironta nom va toifa orqali ifodalanadi (bu haqida 1.1.3-mavzuda qisqacha to'xtalib o'tilgan).

Massivlar:

- bir o'lchamli;
- ikki o'lchamli;
- ko'p o'lchamli

bo'lishi mumkin. Bir o'lchamli massivlar sodda bo'lib, undagi har bir element xotirada ketma-ket yacheykalarda joylashadi. Massiv uchun xotiradan joy ajratishda uning toifasidan kelib chiqqan holda har bir elementga sarflanadigan xotira hajmi elementlar soniga ko'paytmasi hisoblaniladi.

12	-3	6	...	94
a[0]	a[1]	a[2]	...	a[n]

Masalan, *int a[n]* massivning bitta elementiga 4 bayt joy sarflanadigan bo'lsa, massiv uchun ajratiladigan xotira hajmi 4*n bayt kabi hisoblanadi.

$H=\sum_{i=1}^{n} h$

H – massivga sarflanadigan xotira hajmi; h – bitta elementga ajratiladigan xotira hajmi.

Massivni e'lon qilish ikki xil usulda amalga oshirilishi mumkin.

1. Initsializatsiya qilmasdan e'lon qilish – bu holda massiv toifasi va nomi ko'rsatilib, kvadrat qavs ichida uning elementlari soni ko'rsatiladi: *int A[50]*.

2. Initsializatsiya qilish orqali e'lon qilish – bu holda massiv toifasi ko'rsatilib, elementlariga qiymat o'zlashtiriladi. Masalan, *int A[5]={1,2,3,5,4}* yoki *int A[]={1,2,3,4,5}*. Ikkinchi holatda massiv o'lchami kompilyator tomonidan avtomatik tarzda aniqlanadi.

Massiv elementlari bir toifaga tegishli bo'lgani uchun ular xotiradan bir xil hajmli joyni egallaydi va ular operativ xotirada joylashadi. Massiv dasturda foydalanilayotgan o'rniga qarab global yoki lokal bo'lishi mumkin.

Massivni global shaklda e'lon qilish uchun asosiy dastur tanasidan oldin, ya'ni *int main()* blokidan oldin uni e'lon qilish zarur, lokal massivni esa dasturni kerakli qismida e'lon qilinadi. Lokal massivdan foydalanilganda uni chegaralari dastur davomida aniqlanadi va qism dasturdan tashqarida bu massivdan foydalanib bo'lmaydi.

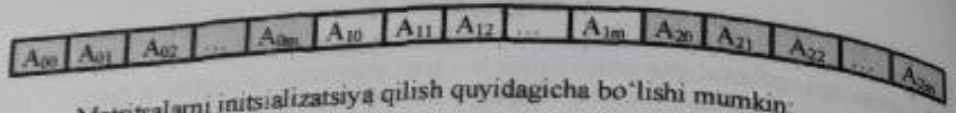
Ikki o'lchamli massivlarda bir nechta qator va ustunlar mavjud bo'lib, ustun va qatorlar kesishgan joyda massivning elementi joylashadi va u element massivning qator va ustun raqami bilan aniqlanadi. Masalan, beshinchi qator va uchinchi ustunda turgan B matritsaning elementi *B[4][2]* kabi belgilanadi (qator va ustunlarni nomerlash 0 dan boshlanadi). Massivlar ustida matematik amallarni bajarish mumkin.

Ikki o'lchovli massivlar matritsalar deb ham ataladi.

a[0][0]	a[0][1]	a[0][2]	...	a[0][m]
a[1][0]	a[1][1]	a[1][2]	...	a[1][m]
a[2][0]	a[2][1]	a[2][2]	...	a[2][m]
...	...	...	...	...
a[n][0]	a[n][1]	a[n][2]	...	a[n][m]

Bu yerda n – massiv qatorlari soni; m – massivdagi ustunlar soni. mxn – massiv elementlari soni.

Matritsalar ham statik tuzilma hisoblanadi. Chunki uning o'lchami dastur bajarilishidan oldin ko'rsatilishi kerak. Dastur ishga tushishidan oldin Matritsaning satr va ustunlar soni hamda toifasidan kelib chiqib, kompyuter uning uchun xotiradan joy ajratadi. Matritsa elementlari xotirada ketma-ket yacheykalarda joylashtiriladi, garchi uning alohida satr elementlari mantiqan quyidagicha keltirilsada, bitta satr elementlari xotirada ketma-ket joylashtirilgandan keyin, uning davomidan ikkinchi qator elementlari joylashtiriladi va uchunchi va x.k.



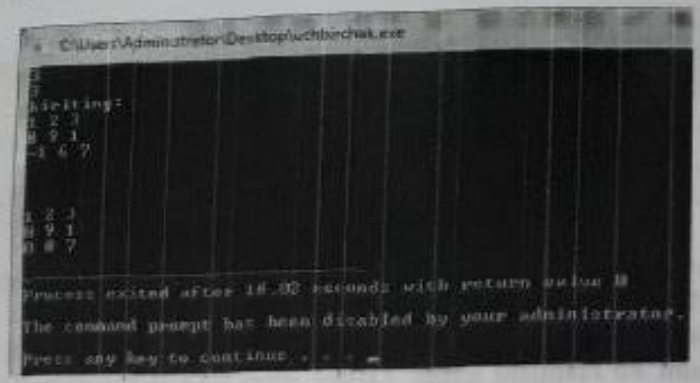
Matritsalarni initsializatsiya qilish quyidagicha bo'lishi mumkin:

- int a[5][3];
- int a[5][3] = { {4, 7, 8}, {9, 66, -1}, {5, -5, 0}, {3, -3, 30}, {1, 1, 1} }; - bunda ichki qavslarda satr elementlari keltiriladi.

Quyida matritsaning quyi uchburchak elementlarini aniqlab, ularni no'lga aylantiruvchi dastur kodi keltirilgan (C++ tilida):

```
int main(){
 int n,m,i,j;
 cin>>n>>m;
 int a[n][m];
 cout<<"kiriting:  "<<endl;
 for(i=0;i<n;i++){
        for(j=0;j<m;j++)
 cin>>a[i][j];}
        for(i=0;i<n;i++){
  for(j=0;j<m;j++)
          if(i>j) a[i][j]=0;}
        for(i=0;i<n;i++){
                for(j=0;j<m;j++){
  cout<<a[i][j]<<" ";}
       cout<<endl;}
 return 0;
 getch();
}
```

Dastur natijasi:



Statik massivlarni e'lon qilishda uning o'lchami faqat sonli konstanta bilan berilishi kerak. Ajratiladigan xotira hajmi kompilyatsiya jarayonida aniqlanadi va dastur bajarilish vaqtida o'zgarmas bo'ladi. Ammo ko'p hollarda massiv o'lchami oldindan aniq bo'lmaydi va bunday hollarda dasturchilar quyidagi kabi qiymati oldindan noma'lum bo'lgan o'zgaruvchi yordamida massiv e'lon qilishadi. Bu esa

statik tuzilmalarda nazariy jihatdan xato hisoblanadi, ammo zamonaviy kompilyatorlar bunday vaziyatlarda xatolik bermaydi va vaziyatni avtomatik tug'irlaydi, ya'ni dinamik massivlar mexanizmini qo'llaydi.

```
int n;
cin >> n;
string students[n]; /* noto'g'ri */
```

3.1.2. Dinamik massivlar

O'lchami dastur bajarilishi mobaynida ma'lum bo'ladigan va dastur bajarilishi vaqtida o'lchami o'zgaruvchan bo'lgan massivlarga dinamik massivlar deyiladi.

Dastur bajarilishi mobaynida o'zgaruvchan hajmda xotira ajratilishi dinamik massivlar bilan ishlashda yuz beradi. Yuqorida aytib o'tilganidek, statik massivlar o'lchami sonli konstanta bilan e'lon qilinishi kerak, o'zgaruvchi bilan emas. Ammo, ko'pincha massiv o'lchami oldindan ma'lum bo'lmaydi. Masalan, N ta elementdan iborat massiv yaratilishi kerak, ammo N ga qiymat dastur bajarilishi mobaynida foydalanuvchi tomonidan berilishi kerak bo'lsin. Bunday hollarda dinamik massivlarga xotira ajratish uchun ko'rsatkichlardan foydalanish mumkin. Dinamik massivga xotira ajratishda *new[]* va ajratilgan xotirani tozalash uchun *delete[]* operatorlari ishlatiladi:

```
toifa <ko'rsatkich>=new toifa[o'lcham];
delete[] <ko'rsatkich>;
```

Masalan:

```
int *a=new int[10];
delete[] a;
```

Dinamik massiv yaratishga doir dastur kodini ko'ramiz.

```
#include <iostream>
using namespace std;
int main()
{
  int n;
  cin >> n;
  int *p = new int[n];
  for (int i = 0; i < n; i++) {
    p[i]= i;
    cout << i << " -element: " << p[i] << endl;
  }
  delete [] p; // xotirani tozalash
  return 0;
}
```

Yuqorida aytib o'tilganidek, dinamik massivlarda ajratiladigan xotira hajmi dastur bajarilishi mobaynida ma'lum bo'ladi, ya'ni massiv uzunligi konstanta bilan emas, balki qiymati noma'lum bo'lgan va dastur bajarilishi mobaynida foydalanuvchi tomonidan kiritiladigan o'zgaruvchi bilan beriladi va dastur bajarilishi davomida uni o'zgartirish mumkin. Bunda dastur bajarilishi mobaynida dastlab ajratiljan xotira sohasi to'ladigan bo'lsa, kompilyator talabga qarab tuzilma joylashgan sohaning davomidan qo'shimcha xotira ajratadi. Agar buning iloji bo'lmasa,

boshqa joyda ikki baravar uzunroq bo'lgan yangi xotira sohasi ajratiladi va tuzilma elementlari uzunroq o'lchamdagi yangi sohaga ko'chirib o'tkaziladi. Bundan kelib chiqadiki, dinamik massivlar qulay bo'lishiga qaramasdan, uni tashkil etish jarayoni zamirida statik tuzilmalarda qo'llaniladigan yondashuvlar yotadi.

3.1.3. Massivlar bilan ishlash

Misol 1. Quyida dinamik massiv bilan ishlashga doir dastur kodi keltirilgan. Unda massivga tasodifiy qiymatlar berilib, ekranga chiqarish masalasi funksiyalar yordamida amalga oshirilgan.

```
#include <iostream>
#include <cstdlib>
#include <time.h>
using namespace std;
// min...max oraligida tasodifiy sonlarni hosil qilish funksiyasi
int rrand(int min, int max) {
  return rand() % (max - min + 1) + min;
}
int n = 12;
int limit = 5;
void set_random_values(int *a, int n) {
  // -limit...limit diapazonda tasodifiy sonlar bilan massiv elementlariga qiymat kiritish
  for (int i = 0; i < n; i++) {
    a[i] = rrand(-limit, limit);
  }
}
void print_values(int *a, int n) {
  for (int i = 0; i < n; i++) {
    cout << i << "-element: " << a[i] << endl;
  }
}
long summ_even(int *a, int n) {
  // juft indeksdagi elementlar yigindisi
  int sum = 0;
  for (int i = 0; i < n; i += 2) {
    sum += a[i];
  }
  return sum;
}
int main(void) {
  int *a = new int[n];      // massiv uchun joy ajratish
  set_random_values(a, n);
  print_values(a, n);
  cout << "juft indeksli elementlarning yigindisi " << summ_even(a, n) << endl;
  delete[] a;               // xotirani tozalash
  system("pause");
  return 0;
}
```

Misol 2. Butun sonlardan iborat massivning tub sonli elementlarini boshqa massivga ko'chirib o'tkazish dasturini tuzamiz.


```
#include <iostream>
#include <cstdlib>
#include <time.h>
using namespace std;
int n,m;
bool tub(int d){
        bool t=false;
        if(d==1) return true;
        for(int i=2;i<=d/2;i++)
                if(d%i==0){
                        t=true;
                        break;
                }
        return t;
}
void print(int *a,int n){
        for(int i=0;i<n;i++)
                cout<<a[i]<<"  ";
        cout<<endl;
}
void replace(int *a,int i,int *b){
        b[m++]=a[i];
        for(int j=i;j<n-1;j++)
                        a[j]=a[j+1];
        n--;
}
int main(void) {
   cin>>n;
   int *a = new int[n];      // massiv uchun joy ajratish
   int *b = new int[n];
        for (int i=0;i<n;i++){
        cout<<"el-tni kiriting=";
        cin>>a[i];
        }
        bool t=true;
        for(int i=0;i<n;i++){
                t=tub(a[i]);
                if(!t){
                        replace(a,i,b);
                        i--;
                }
        }
    cout<<"a[]= ";
                print(a,n);
    cout<<"tub sonlardan iborat massiv\n b[]=";
                print(b,m);
        delete[] a;             // xotirani tozalash
        delete[] b;
    system("pause");
    return 0;
}
```

Dinamik massivlar bilan ishlash uchun alohida sozlangan funksiyalar mavjud emas, ammo C++ tilining ***std::algorithm*** kutubxonasining ayrim funksiyalaridan foydalanish mumkin.

3.1.4. Chiziqli konteynerlar va ularni qo'llash

Konteyner – bu STL kutubxonasi sinfi bo'lib, turli ma'lumotlar tuzilmalari ustida amal bajarish imkonini beruvchi tarkibdan iborat. Konteynerlar o'zida elementlar to'plamini saqlaydi. Standart konteynerlarni shartli ravishda ikkita guruhga ajratish mumkin:

- chiziqli: *array, vector, stack, queue, deque, list, forward_list, string*;
- assosiativ: *set, map*.

Konteynerlar turiga qarab, elementlarga murojaat turlicha bo'lishi mumkin. Masalan, *array, vector* va *deque* konteynerlarida istalgan elementga uning indeksi bilan murojaat qilish mumkin, *stack* va *queue* konteynerlari esa faqat ma'lum bir elementlarga murojaat qilishga imkon beradi. Konteyner elementlariga murojaat qilish uchun *iteratorlar* tushunchasidan foydalaniladi.

Chiziqli konteynerlar bilan yaqindan tanishishdan oldin iteratorlarga alohida to'xtalib o'tishimiz zarur.

Iterator – bu konteynerga nisbatan ko'rsatkich vazifasini bajaruvchi obyekt hisoblanadi. Uning yordamida konteynerdagi elementlarga murojaat qilish va elementlarni skanerlash mumkin, xuddiki massiv elementlariga ko'rsatkich yordamida murojaat qilinganidek. Demak, iteratorlar yordamida konteyner elementlariga murojaat qilish mumkin. Iteratorlar *iterator* toifasi yordamida e'lon qilinadi. Masalan, turli konteynerlar uchun iteratorlarni e'lon qilishga misol keltiramiz:

```
list<int>::iterator it1;      //ro'yxat uchun iterator e'lon qilish
vector<int>::iterator it2;    //vektor uchun iterator e'lon qilish
```

Har bir konteyner turiga qarab konkret iteratorlar bir-biridan farq qilishi mumkin. Masalan, vektorlarda iteratorlar har ikkala tomonga harakat qilishi mumkin, ammo bir bog'lamli ro'yxatlarda buning iloji yo'q.

Iteratorlarda quyidagi amallarni qo'llash mumkin:

it1==it2 - ikkita iteratorni tenglikka solishtirish, agar ular aynan bitta elementni ko'rsatayotgan bo'lsa, true qiymat qaytariladi;

it1!=it2 - ikkita iteratorni tengsizlikka tekshirish, agar ular turli elementlarni ko'rsatayotgan bo'lsa, true qiymat qaytariladi;

++it - inkrement, ya'ni konteynerni navbatdagi elementiga o'tish;

--it - decrement, ya'ni konteynerda bitta oldingi elementga o'tish, ammo forward_list konteyneri iteratorlari bunday amalni bajara olmaydilar;

**it* - iterator ko'rsatayotgan element qiymatini olish.

it+k - k pozitsiya o'ngdagi elementga o'tish;

it+=k – iteratorni k pozitsiya o'ngga siljitish;

it-k - k pozitsiya chapdagi elementga o'tish;

it-=k – iteratorni k pozitsiya chapga siljitish;

Iteratorlar quyidagi metodlarga ega:

begin() – iteratorni konteynerning birinchi elementiga joylashtirish;

end() – konteynerning oxirgi elementidan keyingi o'rniga joylashtirish;

rbegin() – elementlarning teskari tartibida boshidagi elementga murojaat;

rend() – teskari tartibdagi elementlarning oxirgidan keyingi o'rinnga murojaat;

empty() – konteynerlarni bo'shlikka tekshirish;

size() - konteynerdagi elementlar sonini aniqlash;

clear() –konteynerni (array konteyneridan tashqari) tozalash.

Vektor konteynerining barcha elementlarini iterator yordamida ekranga chiqarish kodini keltiramiz.

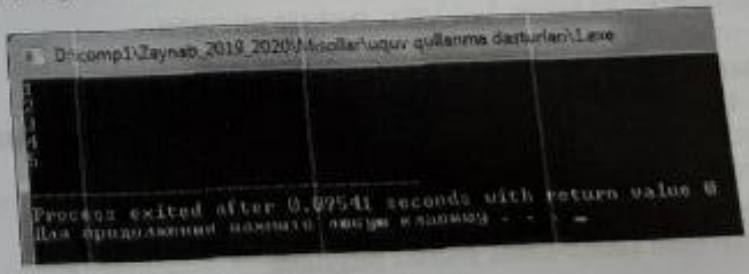
```
for (vector<int>::iterator it = a.begin(); it != a.end();it++)
  cout << *it << " ";
```

Iteratorlardan foydalanishga doir misollar keltiramiz.

Misol 1. Vektor konteyneri elementlarini iterator yordamida ekranga chiqaramiz.

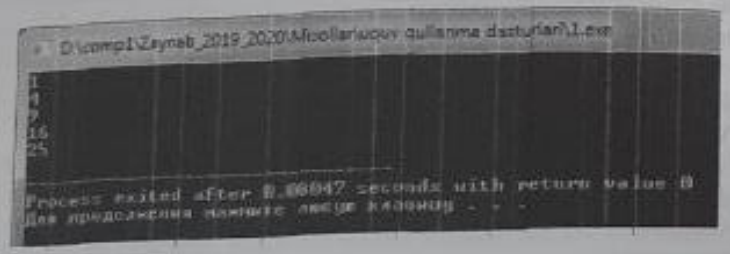
```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
  vector<int> v = {1, 2, 3, 4, 5};
  auto iter = v.begin();
  while(iter!=v.end())
  {
    cout << *iter << endl;
    ++iter;
  }
  return 0;
}
```

Dasturdagi ***auto*** kalit so'zi noaniq toifani bildirib, kompilyator initsializatsiya qilish vaqtida o'zgaruvchining toifasini avtomatik tarzda aniqlashi uchun ishlatiladi.



Misol 2. Iterator yordamida vektor konteyner elementlari kvadratlarini hisoblash dasturini keltiramiz.

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
  vector<int> v = {1, 2, 3, 4, 5};
  auto iter = v.begin();
  while(iter!=v.end())
  {
    *iter = (*iter) * (*iter);
    ++iter;
  }
  for(iter = v.begin(); iter!=v.end(); ++iter)
   cout << *iter << endl;
  return 0;
}
```



Dasturlash tilida mavjud bo'lgan sozlangan massivlardan tashqari ular bilan ishlashni osonlashtirish uchun C++ tilida *array* konteyneri mavjud. Dastur sarlavhasida konteyner kutubxonasi e'lon qilinadi.

#include <array>

Array konteyneri dasturlash tilining sozlangan massivi bilan bir xil ishlaydi va istalgan elementga uning indeksi yoki iterator yordamida murojaat qilish mumkin. Kutubxonada array konteyneri bilan ishlash uchun quyidagi funksiyalar mavjud:

array – array obyektini yaratadi;
assign – barcha elementlarni o'rin almashtiradi;
at – ma'lum bir pozitsiyadagi elemenetga murojaat qilish;
back – oxirgi elementga murojaat;
begin – qaralayotgan ketma-ketlikning boshiga murojaat;
cbegin – massivning birinchi elementiga konstanta iteratorni joylash;
cend – massivning oxirgi elementini ko'rsatuvchi konstanta iterator;
crbegin – teskari matritsa birinchi elementiga konstanta iteratorini yaratish;
crend – teskari matritsa oxirgi elementiga konstanta iterator yaratish;
data – birinchi element adresini olish;
empty – element mavjudligini tekshirish;
end – qaralayotgan ketma-ketlik oxirini belgilash;

fill – barcha elementlarga maxsus qiymatni o'zlashtirish;
front – birinchi elementga murojaat;
max_size – massiv uchun ajratilgan maksimal elementlar sonini aniqlash;
rbegin – qaralayotgan ketma-ketlikning teskari tartibi birinchi elementiga murojaat;
rend – qaralayotgan ketma-ketlikning teskari tartibi oxirgi elementiga murojaat;
size – elementlar sonini aniqlash;
swap – ikkita konteyner tarkibini o'rin almashtirish.

Array konteyneridan foydalanish oddiy massivlar bilan ishlashga qaraganda unchalik ham katta farq va qiyinchilik tug'dirmaydi, balki kutubxonaning tayyor operatorlari va funksiyalaridan foydalanish kabi qulayliklarni taqdim etadi.

Tilning sozlangan massividan foydalanish	Array konteyneridan foydalanish

#include <iostream>
 	
using namespace std;
int main()
{
int myarray[3] = {10,20,30};
for (int i=0; i<3; i++)
++myarray[i];
for (int i : myarray)
cout << i << '\n';
}
	#include <iostream>
	#include <array>
	using namespace std;
	int main()
	{
	array<int,3> myarray {10,20,30};
	for (int i=0; i<myarray.size();i++)
	++myarray[i];
	for (int i : myarray)
	cout << i << '\n';
	}

Kutubxona bilan ishlashga doir misol keltiramiz.

Misol 1.

```
#include <iostream>
#include <array>
using namespace std;
int main()
{
        array<int, 4> Myarray1= { 0, 1, 2, 3 };
        array<int, 4> Myarray2= { 4, 5, 6, 7 };
  for (int i : Myarray1)
      cout << " " <<i;
  cout << endl;
      swap(Myarray1,Myarray2);
      for (int i : Myarray1)
      cout << " " << i;
  return (0);
}
```

Misol 2.

```
#include <string>
#include <iterator>
#include <iostream>
#include <algorithm>
#include <array>
using namespace std;
int main(){
    array<int, 3> a1{1, 2, 3};
    array<int, 3> a2 = {1, 2, 3};
    array<string, 2> a3 = { "a", "b" };
    sort(a1.begin(), a1.end());
    reverse_copy(a2.begin(), a2.end(),ostream_iterator<int>(cout, " "));
  cout << '\n';
  for(string s: a3)
    cout << s << " ";
}
```

Qolgan konteynerlar bilan keyingi bo'limlarda batafsil to'xtalib o'tiladi.

3.2. "Ro'yxat" turidagi ma'lumotlar tuzilmalari

3.2.1. "Ro'yxat" turidagi ma'lumotlarning abstrakt turlari

Ro'yxat – bu biror bir toifadagi $a_1, a_2, \ldots a_n$, $n \geq 0$ elementlar ketma-ketligini o'zida saqlovchi konteyner bo'lib, n – ro'yxat uzunligi deyiladi. $n=0$ bo'lsa, ro'yxat bo'sh deyiladi va unda elementlar mavjud emas. Ro'yxat elementlari tuzilmada chiziqli tartiblangan bo'ladi, ya'ni har bir a_i element a_{i+1} elementdan oldin va a_{i-1} elementdan keyin keladi. Ro'yxatda shunday *NULL* pozitsiya borki, u ro'yxat tugaganligini bildiradi va u ro'yxat oxirgi elementidan keyin joylashadi. Ro'yxat elementlari soni dastur bajarilishi mobaynida o'zgarib turishi mumkin. Elementlar orasidagi munosabatga ko'ra, ro'yxatlar ikki turga bo'linadi:

1) Bog'lanmagan
2) Bog'langan

Roy'xatning bog'lanmagan turida uning elementlari orasidagi bog'liqlik oshkormas (noaniq) ko'rinishda bo'ladi. Bog'langan turida esa elementlarga ro'yxatda

o'zidan oldingi va keyingi elementlar bilan aloqasini bildiruvchi ko'rsatkich kiritiladi.

Stek, navbat va deklar bog'lanmagan ro'yxatlarga misol bo'ladi. Bundan tashqari ular ketma-ket ro'yxatga misol bo'lib, oshkormas bog'liqlik ularning ketma-ketligi orqali aks etadi. Ro'yxat ustida bajariladigan amallar:

- uzunligini o'zgartirish;
- elementlarni istalgan joyga kiritish va o'chirish;
- bo'shlikka tekshirish;
- kichik ro'yxatlarga arjatish yoki bir necha ro'yxatlarni birlashtirish va h.z.

Ro'yxatlar bilan ishlash uchun C++ tilida alohida kutubhona mavjud bo'lib, uni e'lon qilishdan oldin dasturning sarlavha qismida *<list>* kutubhonasini e'lon qilish talab etiladi. C++ tilida *<list>* konteyneri ikki bog'lamli ro'yxat ko'rinishida amalga oshirilgan (ikki bog'lamli ro'yxat haqida keyingi bo'limlarda batafsil to'xtalib o'tiladi). Ro'yxat e'lon qilishning bir nechta turi mavjud bo'lib, ayrim ko'rinishlariga misollar keltiramiz.

list<int> A – *int* toifali elementlarni saqlash uchun bo'sh konteyner e'lon qilish;

list<int> A(n, value) – *value* qiymatga ega bo'lgan n ta *int* toifali elementdan iborat ro'yxat konteyneri e'lon qilish.

<list> kutubhonasida mavjud bo'lgan ayrim funksiyalarni keltiramiz:

- *front()* – ro'yxatning birinchi elementiga murojaat;
- *back()* – oxirgi elementga murojaat;
- *push_back(value)* – *value* qiymatli yangi elementni ro'yxat oxiriga qo'shish;
- *push_front(value)* – yangi elementni ro'yxat boshiga qo'shish;
- *insert(it, value)* – *value* qiymatli elementni *it* iterator ko'rsatayotgan elementdan oldin joylash;
- *insert(i, n, value)* – *value* qiymatli *n* ta elementni *i* dan oldin joylshtirish;
- *pop_front()* – ro'yxatning birinchi elementini o'chirish;
- *pop_back()* – oxirgi elementni o'chirish; (qiymat qaytarmaydi);
- *erase(it)* – *it* iterator ko'rsatayotgan elementni o'chirish;
- *erase(start, fin)* – *start* va *fin* iteratorlari orasidagi barcha elementlar o'chiriladi, ammo fin iterator ko'rsatayotgan element o'chirilmaydi;
- *clear()* – ro'yxatni tozalash;
- *size()* – ro'yxatdagi elementlar sonini aniqlash;
- *empty()* – ro'yxatni bo'shlikka tekshirish;

Ro'yxatni yaratib ekranga chiqarish dasturini keltiramiz:

```
#include <iostream>
#include<list>;
using namespace std;
int main(){
int sum;
list<int> A = {1,2,3,4,5};
list<int>::iterator it;
for(it=A.begin(); it!=A.end(); it++)
{
   sum+=(*it);
```

```
}
cout<<sum;
return 0;
}
```

Iterator toifasi konteyner toifasi bilan aynan mos kelishi qat'iy talab qilinadi.

3.2.2. Roy'xatlarni statik va dinamik tarzda amalga oshirish

Ro'yxat konteynerini amalga oshirishning statik va dinamik usullari mavjud.

Statik ko'rinishda amalga oshirishda konteyner elementlarini e'lon qilish, saqlash va ustida amal bajarishda statik tuzilmalardan foydalaniladi, masalan massiv (dinamik massivdan ham foydalanish mumkin) yoki vektor.

Dinamik ko'rinishda amalga oshirishda esa bog'langan ro'yxatlardan foydalaniladi.

Har qanday ko'rinishda tashkil etilsa ham ro'yxatlar ustida aynan bir xil amallar bajariladi:

1. *insert(x, p)* – ushbu operator ro'yxatning *p* pozitsiyasiga *x* elementni kiritish amalini bajaradi. Bunda ro'yxatning *p* va undan keyin turgan elementlari bitta pozitsiya keyinga suriladi. Shunday qilib, *n* ta elementdan iborat ro'yxat $a_1, a_2, \ldots, a_{p-1}, x, a_p, \ldots, a_n$ ko'rinishga kelib qoladi. Agar *p* =*NULL* qiymat qabul qiladigan bo'lsa, u holda, $a_1, a_2, \ldots, a_n, x$ ko'rinishga keltiriladi.

Ushbu amal dasturini tuzishdan oldin sinflar yordamida oddiy usulda *List* konteynerini yaratamiz va unda 2 xil initsializatsiyani amalga oshiramiz.

```
class List{
public:
        List(){ this->n=10; }
        List(int n){    this->n=n; }
private:
        int n;
        int *a=new int[n];
        int R=0; //konteynerdagi oxirgi elementdan keyingi pozitsiyani ko'rsatuvchi
}
```

Bu yerda *int* toifadagi *n* ta elementdan iborat *a* dinamik massivini e'lon qildik. *List* sinfida 2 xil konstruktor e'lon qildik: agar foydalanuvchi ro'yxat e'lon qilishda *n* sonini kiritsa, shuncha elementdan iborat massiv e'lon qilinadi; agar *n* ga qiymat kiritilmasa, unga 10 soni beriladi. *R* o'zgaruvchisi konteynerdagi elementlar sonini aniqlaydi va o'zi esa oxirgi elementdan keyingi pozitsiyani ko'rsatib turadi. Agar konteyner bo'sh bo'lsa, *R=0* bo'ladi.

Dastlab oddiylik uchun, sinf ichida *push_back(x)* – *x* ni ro'yxat oxiriga kiritish funksiyasini ham yaratib olamiz.

```
int push_back(int x){
        if(!isFull()){
                a[R++]=x;
                return 1;
        }
        else cout<<"to'ldi"<<endl;
}
```


Ma'lumki, ro'yxat uzunligiga oldindan chegara qo'yiladigan bo'lsa, u to'lib qolishi mumkin. Shu sababli ajratilgan xotira sohasi to'lmaganligini tekshirish funksiyasini ham tuzib olamiz:

```
bool isFull(){
        if(R==n) return true; else return false;
}
```

Shu o'rinda konteynerni bo'shlikka ham tekshirish funksiyasini tuzib olamiz:

```
bool isEmpty(){
        if(R==0) return true; else return false;
}
```

Endi *insert(x, p)* funksiyasini amalga oshirish dastur kodini keltiramiz.

```
int insert(int x, int p){
    if(!isFull()){
    int s=getsize();
    if(p==0||p-s==1) {push_back(x); return 1;}
    if(p<=s) shiftAdd(x, p);
    if(p>s) cout<<"natija noma'lum";
    return 1;
    }
    else cout<<"to'ldi"<<endl;
}
```

Ko'rinib turibdiki, bu funksiyani ishlatish uchun *getsize()* – ro'yxat uzunligini aniqlash funksiyasi va *shiftAdd(x, p)* – *x* elementni ro'yxatning *p* pozitsiyasiga joylash funksiyalarini yaratib olishimiz talab etiladi. *List* sinfi ichida quyidagi funksiyalarni yozamiz:

```
int getsize(){
        return R;
}
void shiftAdd(int x, int p){
        for(int i=R-1;i>=p-1,i--)
                a[i+1]=a[i];
        R++;
        a[p-1]=x;
}
```

shiftAdd(x, p) funksiyasida *x* ni *p* pozitsiyaga joylash uchun *p* dan boshlab barcha elementlarni bitta pozitsiya o'ngga siljitiladi va *p*-pozitsiya o'rni *x* ni joylash uchun bo'shatiladi.

2. *print()* – ro'yxat elementlarini ekranga chiqarish. *List* sinfi tarkibida quyidagi funksiyani e'lon qilamiz:

```
void print(){
        if(R==0){cout<<"ro'yxat bo'sh"";
                return;
        }
        int i=0;
        while(i<R){
                cout<<a[i]<<"  ";
                i++;
        }
        cout<<endl;
}
```

Yuqorida keltirilgan funsiyalardan foydalanib, ro'yxat yaratish va ekranga chiqarish, x – yangi kiritilgan elementni 2-roy'xatning p pozitsiyasiga kiritish dasturini tuzishga harakat qilamiz.

```
#include <iostream>
using namespace std;
class List{
    public
        List(){ this->n=10; }
        List(int n){ this->n=n; }
        bool isEmpty(){
            if(R==0) return true;
        }
        bool isFull(){
            if(R==n) return true;
        }
        int getsize(){
            return R;
        }
        int push_back(int x){
            if(!isFull()){
                a[R++]=x;
                return 1;
            }
            else cout<<"to'ldi"<<endl;
        }
        void shiftAdd(int x, int p){
            for(int i=R-1;i>=p-1;i--)
                a[i+1]=a[i];
            R++;
            a[p-1]=x;
        }
        void print(){
            if(R==0){cout<<"ro'yxat bo'sh"";
                return;
            }
            int i=0;
            while(i<R){
                cout<<a[i]<<" ";
                i++;
            }
            cout<<endl;
        }
        int insert(int x, int p){
            if(!isFull()){
                int s=getsize();
                if(p==0||p-s==1) {push_back(x); return 1;}
                if(p<=s) shiftAdd(x, p);
                if(p>s) cout<<"natija noma'lum";
                return 1;
            }
            else cout<<"to'ldi"<<endl;
}
```



```
    private:
        int n;
        int *a=new int[n];
        int R=0;
};
int main(){
    int n; cin>>n; int k;
    List L1;
    for(int i=0;i<n;i++){
        cin>>k;
        L1.push_back(k);
    }
        cout<<"1-ro'yxat elementlari:";
    L1.print();
    int n2, int d;
    cin>>n2;
    List L2(20);
    for(int i=0;i<n2;i++){
        cin>>d;
        L2.push_back(d);
    }
    cout<<"2-ro'yxat elementlari:";
    L2.print();
    cout<<"\n yangi elementni va pozitsiyasini kiriting:";
    int p;
    cin>>d>>p;
    L2.insert(d,p);
    L1.print();
    cout<<endl;
    L2.print();
return 0;
}
```

```
D:\comp1\Zaynab_2019_2020\Misollar\uquv qullanma dasturlari\1.exe
4
1 2 3 4
1-ro'yxat elementlari:1  2  3  4
5
5 6 7 8 9
2-ro'yxat elementlari:5  6  7  8  9
 yangi elementni va pozitsiyasini kiriting:10 3
1  2  3  4

5  6  10  7  8  9

--------------------------------
Process exited after 12.58 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

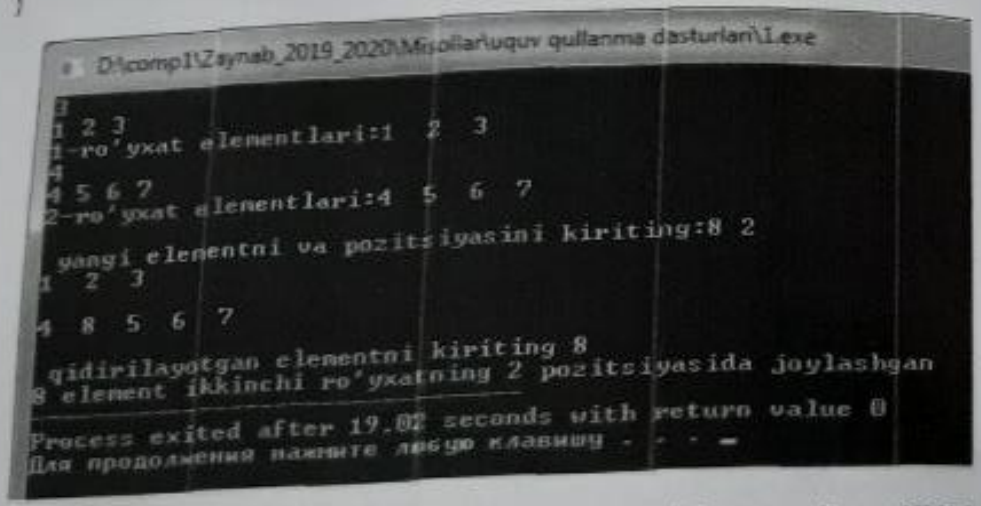
3. *locate(x)* – bu funksiya x elementni ro'yxatdagi pozitsiyasini qaytaradi. Agar x ro'yxatda bir necha marta uchrasa, 1-uchragan x ni turgan joyini qaytaradi.

Agar *x* ro'yxatda mavjud bo'lmasa, *NULL* qaytariladi. Sinf ichiga quyidagi funksiyani kiritamiz:

```
int locate(int x){
        for(int i=0;i<R;i++)
                if(x==a[i]) return i+1;

return 0;
}
```

Ushbu funksiyadan foydalanib, asosiy dastur tanasiga quyidagi kodni kiritamiz:

```
cout<<"\n qidirilayotgan elementni kiriting";
cin>>k;
p=L1.locate(k);
if(p)   cout<<k<<" element birinchi ro'yxatning "<<p<<" pozitsiyasida joylashgan";
if(!p){
        p=L2.locate(k);
        cout<<k<<" element ikkinchi ro'yxatning "<<p<<" pozitsiyasida joylashgan";
}
```



4. *retrieve(p)* – ro'yxatning *p* pozitsiyasidagi elementini qaytaradi. *p=NULL* bo'lsa yoki *p* pozitsiya mavjud bo'lmasa, natija chiqarilmaydi.

```
int retrieve(int p){
        if(p==NULL) return 0;
        else return a[p-1];

}
```

5. *delete(p)* – *p* pozitsiyadagi elementni o'chirish.

```
int del(int p){
        if(p==NULL) return 0;
        for(int i=p-1;i<R;i++){
                a[i]=a[i+1];
        }
        R--;
        return 1;
}
```


6. *next(p)* va *previous(p)* – funksiyalari *p* pozitsiyadan keyingi va oldingi elementlarni qaytaradi.
   - Agar *p* oxirgi pozitsiyani ko'rsatsa, *next(p)* funksiyasi *NULL* ni qaytaradi;
   - Agar *p=NULL* bo'lsa, natija qaytarmaydi;
   - Agar *p=1* bo'lsa, *previous(p)* natija qaytarmaydi;
   - Agar *p* pozitsiya mavjud bo'lmasa, har ikkala funksiya natija qaytarmaydi.

```
int next(int p){
        if(p==0) return 0;
        if(p>=R) return 0;
        if(p==R-1) return NULL;
        return a[p];

}
int previous(int p){
        if(p==0) return 0;
        if(p==1) return 0;
        if(p>R) return 0;
        return a[p-2];

}
```

7. *makenull()* – ro'yxatni tozalash.

```
int makenull(){
        delete []a;
        R=0;

}
```

8. *first()* – ro'yxat boshidagi elementni qaytarish funksiyasi. Agar ro'yxat bo'sh bo'lsa, *NULL* qaytariladi.

```
int first(){
        if(R==0) return -1;
        else return a[0];

}
```

Ro'yxatlarni dinamik tarzda amalga oshirish yuqorida keltirilgan yondashuvdan farq qiladi, ammo asosiy dastur tanasidan turib undan foydalanish bir xil. Bunda ko'rsatkichlardan foydalaniladi va bir yoki ikki bog'lamli ro'yxatlar ko'rinishida amalga oshirilishi mumkin. C++ tilining <list> kutubxonasi konteyner bog'lamli ro'yxat ko'rinishida amalga oshirilgan. Ularga alohida to'xtalib o'tamiz.

3.2.3. Ro'yxat ustida amal bajarishga doir misollar

C++ tilining *list* kutubxonasidan foydalangan holda, butun sonlardan iborat ro'yxatning toq va juft elementlaridan alohida ro'yxatlar hosil qilish dasturini tuzamiz. Ro'yxat elementlariga murojaat qilish uchun iteratordan foydalanamiz.

```
#include <iostream>
#include<list>
using namespace std;
int main(){
int n,s; cout<<"n=";cin>>n;
list<int> A, toq, juft;
int i=0;
  while(i<n){
        cin>>s;
```

```
            A.push_back(s);
            i++;
        }
    list<int>::iterator it;
            cout<<"dastlabki ruyhat el-tlari: ";
            for(it=A.begin(); it!=A.end(); it++)
                        cout<<*it<<" ";
            cout<<endl<<"saralangan ruyhat el-tlari: ";
            A.sort();
            for(it=A.begin(); it!=A.end(); it++){
                cout<<*it<<" ";
                if(*it%2==0) juft.push_back(*it);
                        else toq.push_back(*it);

            }
            cout<<"\ntoq el-tlar ruyhati: ";
            for(it=toq.begin();it!=toq.end();it++)
                cout<<*it<<" ";
            cout<<"\njuft el-tlar ruyhati: ";
            for(it=juft.begin();it!=juft.end();it++)
                cout<<*it<<" ";
            cout<<endl;
return 0;
}
```

Dastur natijasi:

```
n=10
7
4
6
9
3
10
2
8
5
1
dastlabki ruyhat el-tlari: 7  4  6  9  3  10  2  8  5  1
saralangan ruyhat el-tlari: 1  2  3  4  5  6  7  8  9  10
toq el-tlar ruyhati: 1  3  5  7  9
juft el-tlar ruyhati: 2  4  6  8  10
```

Yuqoridagi dastur kodida qo'yilgan masalani yechish uchun tayyor kutub- xon-adan foydalanildi. Tayyor kutubxonadan foydalanmasdan turib, ro'yxat tuzilmasini dinamik massiv ko'rinishida amalga oshiruvchi sinf yaratib, undan foydalangan holda yuqoridagi masalani yechish dasturini keltiramiz. Bunda dastur asosiy tanasi kodi deyarli o'zgarishsiz qolishi kerak.

```
#include <iostream>
using namespace std;
template <typename T>
class List{
```



```
private:
        int n;
        T *a=new T[n];
        int R=0;
public:
        List(){ this->n=10; }
        List(int n){ this->n=n; }
        bool isEmpty(){
                if(R==0) return true;
        }
        bool isFull(){
                if(R==n) return true;
        }
        int getsize(){
                return R;
        }
        int push_back(T x){
                if(!isFull()){
                        a[R++]=x;
                        return 1;
                }
                else cout<<"to'ldi"<<endl;
        }
        void sort(){
                for(int i=0;i<R-1;i++)
                for(int j=i+1;j<R;j++){
                        if(a[i]>a[j]) swap(a[i],a[j]);
                }
        }
    struct iterator {
        T *ptr;
        iterator (T* ptr_=0) : ptr(ptr_) {}
        T& operator*() { return *ptr; }
        T* operator->() { return ptr; }
        T* operator++() { return ptr++; }
        T* operator--() { return --ptr; }
        bool operator==(const iterator& other) const { return ptr == other.ptr; }
        bool operator!=(const iterator& other) const { return !(*this == other); }
    };
    size_t size() const { return n; }
    iterator begin () { return a; }
    iterator end () { return a+R; }
};
int main(){
int n,s; cout<<"n=";cin>>n;
List <int> A, toq, juft;
int i=0;
        while(i<n){
                cin>>s;
                A.push_back(s);
                i++;
        }
```

```
List<int>::iterator it;
    cout<<"dastlabki ruyhat el-tlari: ";
    for(it=A.begin(); it!=A.end(); ++it)
        cout<<*it<<" ";
    cout<<endl<<"saralangan ruyhat el-tlari: ";
    A.sort();
    for(it=A.begin(); it!=A.end(); ++it){
        cout<<*it<<" ";
        if(*it%2==0) juft.push_back(*it);
        else toq.push_back(*it);
        }
    cout<<"\ntoq el-tlar ruyhati: ";
    for(it=toq.begin();it!=toq.end();++it)
        cout<<*it<<" ";
    cout<<"\njuft el-tlar ruyhati: ";
    for(it=juft.begin();it!=juft.end();++it)
        cout<<*it<<" ";

return 0;
}
```

Natija esa xuddi yuqoridagi dastur natijasi kabi bo'ladi.

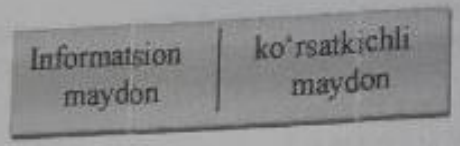
3.2.4. Bir bog'lamli ro'yxatlar va ular ustida bajariladigan oddiy amallar

Bizga ma'lumki, massivlar (statik tuzilmalar) dasturlash tillarida juda foydali va zaruriy tuzilmadir. Lekin uning ikkita kamchiligi bor:

- uning o'lchamini dastur bajarilishi mobaynida o'zgartirib bo'lmaydi;
- tuzilma orasiga element kiritish yoki o'chirish uchun qolganlarini surish kerak.

Bu kamchilik bog'langan ro'yxatlar bilan ishlashga olib keladi. Bo'glangan ro'yxatlar bir xil toifadagi elementlar (tugunlar) ketma-ketligi bo'lib, ular xotirada turli joylarga joylashtiriladi va o'zaro bir-biri bilan ko'rsatkichli maydonlar orqali bog'lanadi. Bo'glangan ro'yxatlarni dasturda turlicha amalga oshirish mumkn.

Bo'glangan ro'yxatlarda elementlarni quyidagicha hosil qilib olamiz:



3.1-rasm. Ro'yxat elementi tuzilishi

Informatsion maydonda foydalanuvchining foydali ma'lumoti yoziladi. Ko'rsatkichli maydonga keyingi elementning xotiradagi adresi yoziladi. Shunday elementlardan tashkil topadigan tuzilmaga *chiziqli bir bog'lamli ro'yxatlar* deyiladi.

Bog'langan ro'yxatlarda massivning kamchiliklari bartaraf qilinganligi sababli *tuzilma uzunligi* va *elementlar orasidagi munosabatlar* dastur bajarilishi mobaynida o'zgarib turadi. Bu dinamik tuzilma xususiyati hisoblanadi. *Dinamik tuzilma* deb:

- elementlari orasidagi munosabatlar;
- tuzilma uzunligi (elementlar soni)

dastur bajarilishi mobaynida o'zgarib turadigan tuzilmaga aytiladi. Dinamik tuzilmalarda elementlar xotirada istalgan joyda joylashishi mumkin. Shu sababli ular orasidagi munosabatlar ko'rsatkichlar orqali belgilanadi. Elementlar tuzilmaga kelib qo'shilgan paytda xotiradan bo'sh joy qidirib topiladi va elementlar joylashtiriladi. Shu sababli elementlar xotirada ketma-ket yacheykalarda joylashmagan bo'lishi mumkin. Agar fizik xotira tanqisligi sezilmasa, tuzilma uzunligi oshirilishi mumkin.

Bunday tuzilmalar bilan ishlashning o'ziga yarasha afzalliklari va kamchiliklari mavjud. Afzalligi shundaki, tuzilma uzunligiga oldindan chegara qo'yilmaydi. Unga element kiritish va o'chirish amallari massivga qaraganda oson kechadi. Chunki elementlar xotiraga istalgan joyga joylashtirilayotgan paytda oldin kelib tushgan elementlar joyidan qo'zg'atilmaydi. Faqat ularning ko'rsatkichlari to'g'irlab qo'yiladi, xolos.

Kamchiligi esa shundaki, oldindan mavjud bo'lgan tuzilmani massivlarda mavjud bo'lgan saralash algoritmlari bilan saralab bo'lmaydi, chunki bunday saralash elementlarning indekslari bilan bog'liq tushunchadir. Bog'langan ro'yxatlarda elementlarning indeksi degan tushuncha yo'qligi sababli elementlarga to'g'ridan to'g'ri murojaatning iloji yo'q, eng og'ir holatda oxirgi elementga N ta murojaat orqali yetib boriladi.

Qidiruv amali ham xuddi shunday. Ya'ni eng og'ir holatda oxirgi elementni N ta solishtirishda topish mumkin.

Bog'langan ro'yxatlar eng ko'p tarqalgan dinamik tuzilmalardan hisoblanadi. Ma'lumotlarni mantiqiy tasvirlash nuqtai nazaridan ro'yxatlar ikkitaga ajratiladi: chiziqli va chiziqsiz.

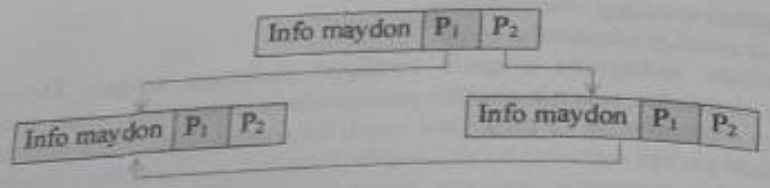
Chiziqli ro'yxatlarda elementlar orasidagi bog'liqlik qat'iy tartiblangan bo'ladi, ya'ni element ko'rsatkichi o'zidan oldingi yoki navbatdagi element manzilini saqlaydi.

Chiziqli ro'yxatlarga bir yoki ikki bog'lamli ro'yxatlar kiradi.

Chiziqsiz ro'yxatlarga esa ko'p bog'lamli ro'yxatlar kiradi. Umuman olganda, ro'yxat elementlari bir yoki bir nechta ko'rsatkichli maydonlarga ega bo'lishi mumkin. Va har bir ko'rsatkichi orqali istalgan elementga murojaat qilsa, bunday ro'yxatlar chiziqsiz ro'yxatlar deyiladi.

Tuzilmada elementlar o'zidan keyingi element bilan bog'langan bo'lsa, bunday ro'yxatga *bir bog'lamli ro'yxat* deyiladi. Agar har bir element o'zidan oldingi va o'zidan keyingi element bilan bog'langan bo'lsa, u holda bunday ro'yxatlarga *2 bog'lamli ro'yxatlar* deyiladi. Agar oxirgi element birinchi element ko'rsatkichi bilan bog'langan bo'lsa, bunday ro'yxatga *halqasimon ro'yxat* deyiladi. Ro'yxatning har bir elementi shu elementni identifikatsiyalash uchun *kalit*ga ega bo'ladi. Kalit, odatda, butun son yoki satr ko'rinishida ma'lumotlar maydonining bir qismi sifatida mavjud bo'ladi.

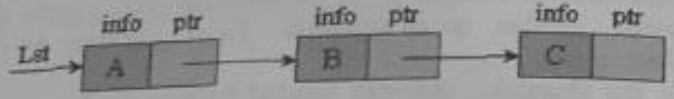
Dinamik tuzilma elementlarini o'zaro bog'lash uchun tuzilma elementlari tarkibiga informatsion maydondan tashqari ko'rsatkichlar maydoni ham kiradi (qarang, chizma) (tuzilmani boshqa elementlari bilan bog'liqligi).



P_1 va P_2 – o'zaro bog'langan elementlarni adreslarini o'z ichiga oluvchi ko'rsatkichlardir. Ko'rsatkichlar slot raqamini o'z ichiga oladi.

Umuman olganda, ro'yxat elementi bir yoki bir necha ko'rsatkichli maydonlarga ega bo'lishi mumkin.

Bir bog'lamli ro'yxat chizmasini keltiramiz.



Bir bog'lamli ro'yxatda ko'rsatkichni o'ziga xosligi shundan iboratki, joriy elementdan keyin keluvchi element adresini ko'rsatadi. Ro'yxat eng so'ngi elementning ko'rsatkich maydoni bo'sh bo'ladi *(NULL)*. *Lst* – ro'yxat boshiga ko'rsatkich. Umuman olganda, ro'yxat bo'sh ham bo'lishi mumkin, bu holda *Lst* *NULL* bilan ustma-ust tushadi, ya'ni teng bo'ladi.

Ro'yxatni dasturda dinamik tarzda, bir bo'glamli ro'yxat ko'rinishida amalga oshirish uchun element toifasini *Node* sinfi yordamida ifodalaymiz.

```
class Node{
public: //sinf ma'lumotlariga tashqaridan bo'ladigan murojaatga ruxsat berish
    int info;  // informatsion maydon
    Node *ptr; // ko'rsatkichli maydon
};
```

Endi elementlari *Node* toifali ro'yxatni e'lon qilish uchun *List* sinfini yaratamiz va unda ro'yxatni statik ko'rinishda tasvirlashda ishlatilgan amallarni bajaruvchi metodlarni ishlab chiqamiz.

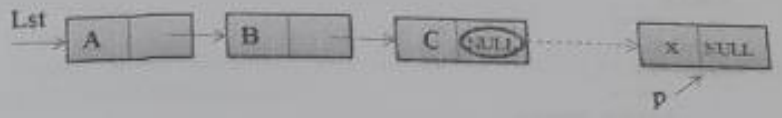
```
class List{
    Node *lst=new Node(), *last=new Node();
    public:
        List(){                    //konstruktor
            lst=NULL;
            last=NULL;
        }
};
```

Ushbu kodda *lst* – ro'yxat boshi ko'rsatkichi va *last* – ro'yxat oxirgi elementi ko'rsatkichidir. *last* ko'rsatkichdan maqsad shuki, yangi kelgan elementni oxirgi elementdan keyin ro'yxatga joylashtirishdir.

Ro'yxat elementiga murojat faqatgina ro'yxat boshidan amalga oshiriladi, ya'ni bu ro'yxatda teskari aloqa yo'q. Chunki har bir elementda faqat o'zidan keyingi elementning adresi saqlanadi.

Bir bog'lamli ro'yxatlar ustida bajariladigan oddiy amallarni keltiramiz.

- *push_back(x)* – *x* qiymatli yangi elementni ro'yxat oxiriga qo'shish;



Yuqorida keltirilgan sinfdan foydalanib, asosiy dastur tanasida yangi ro'yxat hosil qilish va unga *n* ta element kiritish amalini bajaramiz. Buning uchun *List* sinfi ichida *push_back(x)* – elementlarni ro'yxat oxiriga qo'shish funksiyasini yaratamiz. Buning uchun yangi *p* element uchun xotiradan joy ajratamiz.

```
Node *p=new Node()
```

va uning *info* maydiniga *x* ni kiritamiz. Yangi element ro'yxat oxiriga kiritilayotganligi sababli uning ko'rsatkichli maydoniga *NULL* yoziladi.

```
p->info=x;
p->ptr=NULL;
```

Yangi element tayyor bo'ldi. Endi uni ro'yxat oxiriga qo'shish lozim. Buning uchun avvalo, ro'yxat oxiriga qo'shish uchun roy'xatda bironta element bor-yo'qligini tekshirish kerak. Agar ro'yxat bo'sh bo'lsa, ya'ni *lst=NULL* bo'lsa, *p* yangi element ro'yxat boshiga joylashtiriladi va oxirgi element sifatida ham aynan shu yangi element belgilab qo'yiladi.

```
lst=p;
last=p;
```

Aks holda, ya'ni ro'yxat bo'sh bo'lmasa, *p* element oxirgi element *last* dan keyin joylanadi va *p* ni oxirgi kelib tushgan element sifatida belgilab qo'yiladi.

```
int puch_back(int x){
    Node *p=new Node();
    p->info=x;
    p->ptr=NULL;
    if(lst==NULL){
        lst=p;
        last=p;
    }
    last->ptr=p;
    last=p;
}
```

- *print()* – ro'yxatni ekranga chiqarish funksiyasini *List* sinfi ichida quyidagicha e'lon qilamiz:

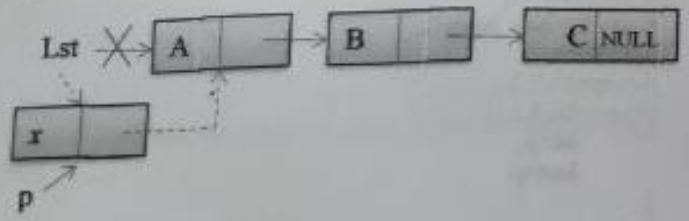
```
void print(){
    Node *p=lst;
    while(p){
        cout<<p->info<<" ";
        p=p->ptr;
    }
    cout<<endl;
}
```

Ushbu funksiyalardan foydalanib, yangi ro'yxat hosil qilish va ekranga chiqarish asosiy dastur tanasini keltiramiz.

```
int main(){
    List L1;
    int n; cin>>n;
    int i=0, k;
    while(i<n){
        cin>>k;
        L1.push_back(k);
        i++;
    }
    cout<<"1-roy'xat elementlari: ";
    L1.print();
return 0;
}
```

```
D:\comp1\Zaynab_2019_2020\Misollar\uquv qullanma dasturlari\3.exe
5
12
5
7
3
1-roy'xat elementlari: 12  5  9  7  3

Process exited after 10.45 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

- *push_front(x)* – yangi elementni ro'yxat boshiga qo'shish;



Faraz qilaylik, bir bog'lamli ro'yxat boshiga informatsion maydoni *x* bo'lgan yangi element qo'yish talab qilingan bo'lsin. Buning uchun *List* sinfi ichida *push_front(int x)* funksiyasini yaratamiz va unda bo'sh element e'lon qilamiz.

Node *p=new Node();

Ushbu elementning informatsion maydoniga *x* ning qiymati o'zlashtiriladi, ko'rsatkichli maydoniga esa ro'yxat boshi ko'rsatkichi o'zlashtiriladi, ro'yxat boshi ko'rsatkichi qiymatiga esa *p* ko'rsatkich qiymati o'zlashtiriladi.

p->info=x;
p->ptr = Lst;
Lst = p;

Ushbu amalning to'liq kodini keltiramiz.

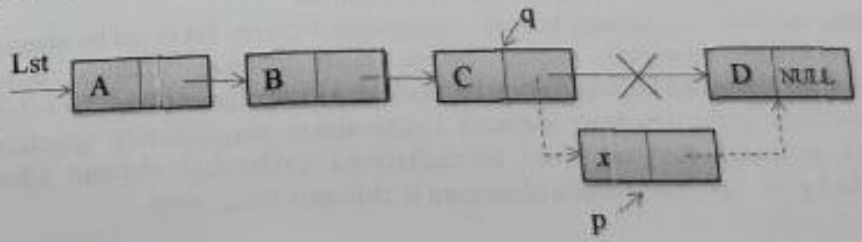

```
void push_front(int x){
    Node *p=new Node();
    p->info=x;
    p->ptr=lst;
    lst=p;
}
```

Asosiy dastur tanasida, ya'ni *int main()* ichida yuqoridagi funksiyadan quyidagicha foydalanish mumkin:

```
...
cout<<"ro'yxat boshida kiritiladigan element qiymatini kiriting: ";
cin>>k;
L1.push_front(k);
cout<<"ro'yxat elementlari: ";
L1.print();
...
```

```
D:\comp1\Zaynab_2019_2020\Misollar\uquv qullanma dasturlari\3.exe
4
12
5
7
3
1-roy'xat elementlari: 12  5  7  3
ro'yxat boshida kiritiladigan element qiymatini kiriting: 9
ro'yhat elementlari: 9  12  5  7  3

Process exited after 16.15 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

- *insert(x, c)* – *x* qiymatli elementni *info* maydoni *C* bo'lgan elementdan keyin joylash;



Yangi elementni *info* maydoniga joylash uchun *x* qiymat va bir bog'lamli ro'yxat o'rtasidagi bironta element *info* maydonidagi *C* qiymat berilgan bo'lsin. Xotiradan yangi element uchun joy ajratamiz.

Node *p=new Node;

Uning *info* maydoniga *x* ni kiritamiz.

p->info=x;

info maydoni *C* bo'lgan elementni topish uchun ro'yxat boshidan *q* ko'rsatkich orqali qidiramiz. Yangi elementning ko'rsatkichli maydoniga *q* dan keyin turgan ele-

ment adresi yozilishi kerak. Uni *q* ko'rsatayotgan elementning ko'rsatkichli maydonidan o'qib olamiz.

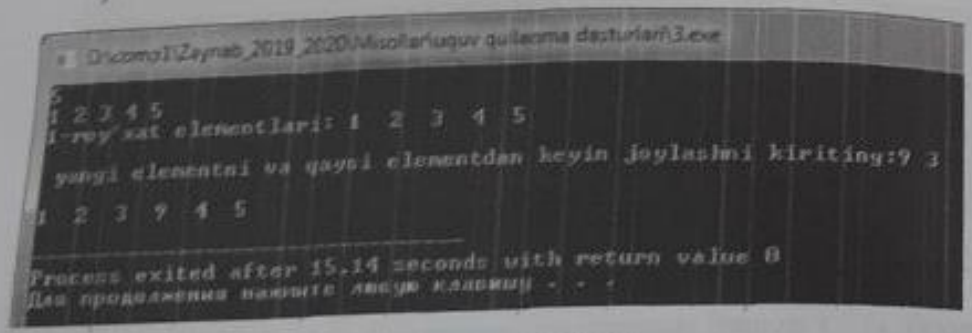
```
p->ptr=q->ptr;
```

Element tayyor bo'ldi, endi uni *q* elementdan keyin joylashtiramiz. Buning uchun *q* ko'rsatayotgan elementning ko'rsatkichli maydoniga yangi elementning adresi *p* ni yozib qo'yamiz.

```
q->ptr=p;
```

Ushbu funksiyaning to'liq kodini quyida keltiramiz.

```
int insert(int x, int c){
        Node *p=new Node();
        p->info=x;
        Node *q=lst;
        while(q->info!=c)
                q=q->ptr;
        p->ptr=q->ptr;
        q->ptr=p;
}
```



- *pop_front()* – ro'yxatning birinchi elementini o'chirish;

Faraz qilaylik, ro'yxatning birinchi elementini o'chirib, lekin ushbu element *info* maydonidagi qiymatni saqlab qolish talab qilinsin. Buning uchun o'chirilayotgan elementni ko'rsatuvchi *p* ko'rsatkich kiritib olamiz (*Node *p = lst*). *x* o'zgaruvchiga o'chirilayotgan element informatsion maydonining qiymatini beramiz (*x=p->info*). Ro'yxat boshi ko'rsatkichiga navbatdagi element adresi o'zlashtiriladi (*lst = p->ptr*) hamda elementni o'chiramiz (*delete(p)*).

```
int pop_front(){
        Node *p=lst;
        int x=p->info;
        lst=p->ptr;
        delete(p);
        return x;
}
```

- *pop_back()* – oxirgi elementni o'chirish;

Ushbu amalni bajarish uchun kerak bo'lgan oxirgi element adresi last ko'rsatkichda saqlanyapti va last elementni o'chirish uchun undan oldin turgan ele-ment adresi *q* ham ma'lum bo'lishi kerak. Chunki undan keyingi element o'chirilishi

uchun uning ko'rsatkichli maydoniga *NULL* qiymat berilishi kerak. Oxirgi elementdan oldin turgan *q* elementni topish uchun ro'yxat boshidan boshlab toki *last* element uchragunga qadar yangi *p* ko'rsatkich bilan ro'yxat oxiriga harakatlaniladi va shu bilan birga, *p* ko'rsatkichdan bir qadam orqada *q* ko'rsatkich ham harakatlanadi. Shu yo'l bilan *q* elementni topish mumkin.

```
int pop_back(){
        int x=last->info;
        Node *p=lst, *q=lst;
        while(p!=last){
                q=p;
                p=p->ptr;
        }
        q->ptr=NULL;
        delete(last);
        return x;
}
```

- *erase(x)* – *info* maydoni *x* bo'lgan elementni o'chirish;

Ro'yxat o'rtasidagi bironta *x* elementni o'chirish uchun ushbu element adresi *p* ni, undan oldin va undan keyin turgan elementlar adreslarini aniqlash zarur. Roy'xat boshidan boshlab *info* maydonlarni *x* bilan solishtirib ketamiz. Bunda bir qadam orqada *q* ko'rsatkich harakatlanadi.

```
void erase(int x){
        Node *p=lst, *q=lst;
        while(p){
                if(x==p->info){
                        if(p==lst) pop_front();
                        if(p==last)     pop_back();
                        q->ptr=p->ptr;
                        delete(p);
                }
                else {
                        q=p;
                        p=p->ptr;
                }
        }
}
```

- *clear()* – ro'yxatni tozalash; agar ro'yxat boshi ko'rsatkichi bo'sh bo'lmasa, ro'yxat boshidan elementni o'chirish funksiyasini ishlatamiz va keyingi elementga o'tiladi.

```
void clear(){
        while(lst){
                pop_front();
                lst=lst->ptr;
        }
}
```

- *size()* – ro'yxatdagi elementlar sonini aniqlash; ro'yxat boshi adresini yangi *p* ko'rsatkichga o'zlashtirib olamiz va toki ro'yxat oxirigacha navbatdagi elementga o'tishlar sonini aniqlaymiz.

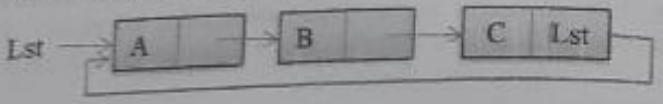
```
int size(){
```

```
if(empty()) return 0;
Node *p=lst;
int R=0;
while(p){
        p=p->ptr;
        R++;
}
return R;
}
```

- *empty()* – ro'yxatni bo'shlikka tekshirish; ushbu amalni bajarish uchun roy'xat boshi ko'rsatkichi *lst NULL* ga tengligi tekshirilishi yetarli.

```
bool empty(){
        if(lst==NULL) return true;
        else return false;
}
```

Halqasimon bir bog'lamli ro'yxat. Halqasimon bir bog'lamli ro'yxat oddiy bir bog'lamli ro'yxatda eng so'ngi element ko'rsatkichiga ro'yxat boshi elementi ko'rsatkichi qiymatini o'zlashtirish orqali hosil qilinadi (chizma).

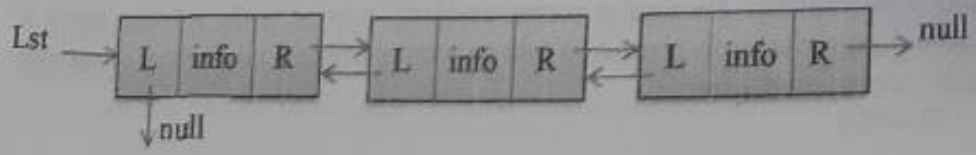


3.2.5. Ikki bog'lamli ro'yxatlar va ular ustida amal bajarish algoritmlari

Ko'pgina masalalarni hal qilishda bir tomonga yo'naltirilgan ro'yxatlardan foydalanish ma'lum bir qiyinchiliklarni keltirib chiqaradi. Sababi, bir tomonga yo'naltirilgan ro'yxatda har doim ro'yxatda bosh elementdan ro'yxatning so'ngi elementi tomoniga harakatlanish mumkin xolos. Lekin ko'pgina masalalar hal qilinayotganda ma'lum bir elementni qayta ishlash uchun undan oldin kelgan elementga murojaat qilish zarurati paydo bo'ladi. Ushbu holatda berilgan elementdan oldin kelgan elementga murojaat qilish bir bog'lamli ro'yxatda noqulay va ancha sekin amalga oshiriladi hamda uni amalga oshirish algoritmi murakkablashadi.

Ushbu noqulayliklarni yo'qotish maqsadida ro'yxatning har bir elementiga yana bitta maydon qo'shiladi. Ushbu maydon qiymati o'zidan oldin kelgan elementga murojaatdan iborat bo'ladi. Ushbu ko'rinishdagi elementlardan tashkil topgan dinamik tuzilmaga ikkitomonlama yo'naltirilgan yoki ikki bog'lamli ro'yxat deyiladi.

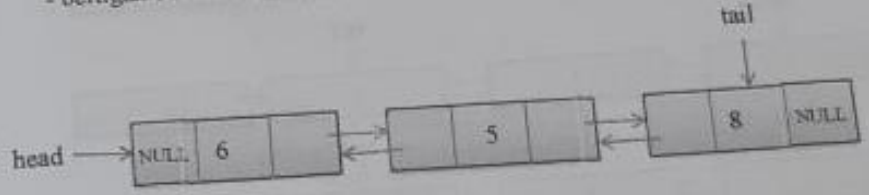
Ikki bog'lamli ro'yxatning har bir elementi ikkita ko'rsatkichga ega. Bittasi oldingi elementni ko'rsatadi (teskari), ikkinchisi navbatdagi elementni ko'rsatadi (to'g'ri) (chizma).



Umuman olganda, ikki bog'lamli ro'yxat bu elementlari soni bir hil, faqatgina teskari ketma-ketlikda yozilgan ikkita bir bog'lamli ro'yxatdir.

Ikki bog'lamli ro'yxat ustidagi amallar:

- ro'yxat elementini yaratish;
- ro'yxatda elementni qidirish;
- ro'yxatning ko'rsatilgan joyiga element qo'yish;
- berilgan elementni ro'yxatdan o'chirish.



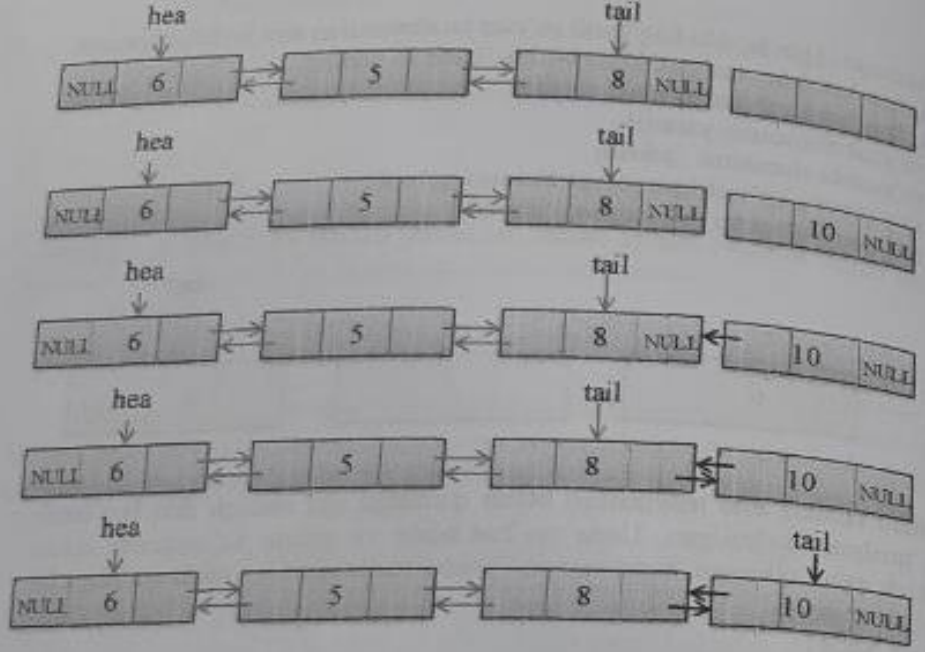
Ushbu rasmda *info* maydonlari butun qiymatga ega chiziqli ikki bog'lamli ro'yxat tuzilmasi keltirilgan. Unda roy'hat boshi va oxirini ko'rsatuvchi ikkita ko'rsatkich qabul qilingan, *head* va *tail*. Ushbu tuzilmani C++ tilida quyidagicha e'lon qilish mumkin:

```
Class Node{
 int info;
 Node *prev;
 Node *next;
}
Ushbu sinfga tegishli ro'yxat boshi ko'rsatkichini e'lon qilishimiz mumkin.
Node *head=new Node();
```

Ro'yxatga yangi elementni kiritish algoritmi. Bunning uchun yangi elementni yaratib, quyidagi qadamlar bilan ro'yxat oxiriga qo'shiladi:

- yangi element yaratiladi, uning 3 ta maydoni e'lon qilinadi;
- info maydoniga *el* sonini kiritamiz;
- *next* maydoniga NULL qiymat kiritamiz;
- *prev* maydoniga *tail* ni qiymatini yozib qo'yamiz, chunki bu element oxirgi turgan elementdan keyinga qo'shiladi va *prev* maydoni bilan o'zidan oldingi elementni ko'rsatib turishi kerak. Undan oldin keladigan element (xozircha oxirgi elementni *tail* ko'rsatyapti) adresi *tail* da saqlanyapti;
- yangi element kiritilgach, *tail* ko'rsatkichni ushbu yangi elementga o'matamiz. Chunki endi oxirgi element bo'lib, yangi element hisoblanadi;
- qachonki, yana yangi element kiritiladigan bo'lsa, xozirgina kiritilgan elementning *next* maydonidagi *NULL* ni o'rniga yangi kiritiladigan elementning adresi yoziladi.

Ushbu aytilgan harakatlarni 3.2-rasmda keltiramiz.



3.2-rasm. Chiziqli ikki bog'lamli ro'yxatga yangi element kiritish

Ushbu algoritmni C++ dagi dastur kodini keltiramiz.

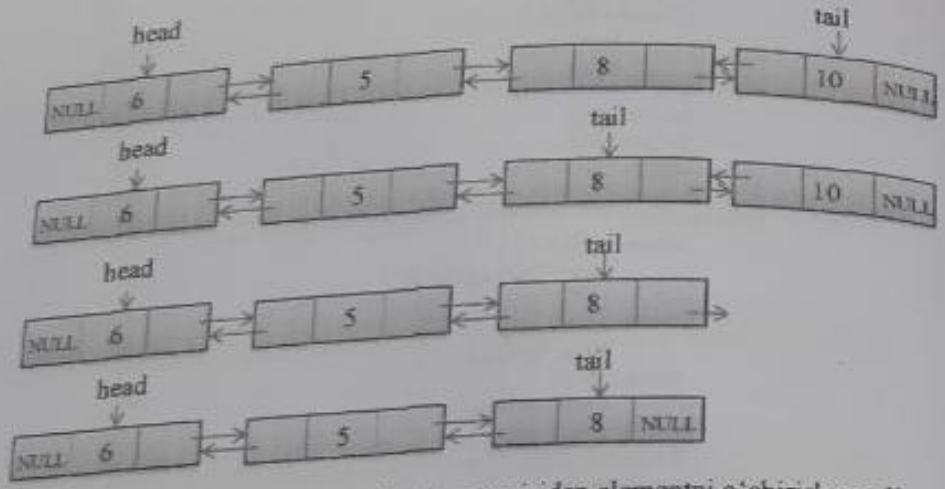
```
#include<iostream>
using namespace std;
class Node{
public:
    int info;
    Node *prev;
    Node *next;
};
class List{
    Node *head=new Node(), *tail=new Node();
    public:
        List(){
            head=NULL;
            tail=NULL;
        }
        bool isEmpty(){
            if(tail==NULL) return true;
            else return false;
        }
        int push_back(int x){
            Node *p=new Node();
            p->info=x;
```



```
            p->next=NULL;
            if(head==NULL){
                head=p;
                tail=p;
            }
            tail->next=p;
            p->prev=tail;
            tail=p;
        }
        void print(){
            if(head==NULL) {
                cout<<"ro'yxat bo'sh";
            }
            else{
                Node *p=head;
            while(p){
                cout<<p->info<<" ";
                p=p->next;
            }
            cout<<endl;
            }
        }
};
int main(){
    List L1;
    int n; cin>>n;
    int i=0,k;
    while(i<n){
        cin>>k;
        L1.push_back(k);
        i++;
    }
    cout<<"1-roy'xat elementlari: ";
    L1.print();
return 0;
}
```

Ikki bog'lamli ro'yxat oxiridan elementni o'chirish algoritmi. Oxiridan element o'chirish amalida tail ko'rsatkich ko'rsatayotgan element o'chiriladi. Bunda undan oldingi turgan elementning next maydoniga NULL yozib qo'yiladi. Keyin element o'chiriladi. Quyidagi amallar ketma-keltligini bajaramiz:

- o'chirilayotgan elementni *prev* maydonidagi adres bilan oldingi turgan element olinadi;
- uning *next* maydoniga *NULL* yoziladi;
- o'chirilayotgan elementni xotiradan tozalash mumkin.



3.3-rasm. Ikki bog'lamli ro'yxat oxiridan elementni o'chirish amali

Bu algoritmni bajarishda shu narsaga ahamiyat berish kerakki, tuzilma ustida amal bajarishda ro'yxat bo'sh yoki bo'sh emasligini tekshirish kerak. Ya'ni quyidagicha:

```
void pop_back(){
    if(head==NULL) {
        cout<<"ro'yxat bo'sh";
    }
    else if(head==tail){
        head=tail=NULL;
    }
    else{
        Node *p=tail;
        tail=tail->prev;
        tail->next=NULL;
        delete(p);
    }
}
```

Nazorat savollar

1. Dinamik tuzilmalar qanday xususiyatlarga ega?
2. Chiziqli va chiziqsiz tuzilmalarni mantiqiy tasvirlab bering.
3. Chiziqli bir bog'lamli ro'yxat nima?
4. Chiziqli bir bog'lamli ro'yxat ustida qanday amallar bajarish mumkin va ularning algoritmlarini tushuntirib bering.
5. Qanday dinamik turlarni bilasiz?
6. Dinamik obyektlarni o'ziga xosligi nimadan iborat?
7. Dinamik tuzilmada elementlar qanday bog'langan?

8. Bir bog'lamli ro'yxatlarning o'ziga xosligi nimalardan iborat?
9. Ko'rsatkich nima?
10. Bir bog'lamli ro'yxatga element kiritish uning elementlari soniga bog'liqmi?
11. Element kiritish va chiqarish jarayoni qaysi holda samaraliroq: ro'yxatdami yoki massivdami?

3.3. Steklar va navbatlar

Yarimstatik ma'lumotlar tuzilmalari deb nomlangan shunaqa tuzilmalar borki, ular ba'zi bir xususiyatlari bilan statik tuzilmalarga, ba'zi bir xususiyatlari bilan dinamik tuzilmalarga o'xshash bo'ladi. Ya'ni dastur bajarilishi mobaynida tuzilma uzunligining o'zgaruvchanligi dinamiklik xususiyati bo'lsa, elementlari xotirada ketma-ket joylashishi statik tuzilmalarga o'xshash bo'ladi. Yarimstatik ma'lumotlar tuzilmasi bir xil toifadagi elementlar ketma-ketligi hisoblanadi va unga:

- steklar;
- navbatlar;
- deklar

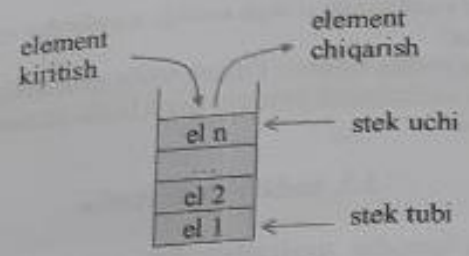
kiradi. Deyarli barcha zamonaviy dasturlash tillarida yuqorida keltirilgan tuzilmalar bilan ishlash uchun kutubxonalar mavjud va ular konteyner ko'rinishida amalga oshirilgan. Shuni aytib o'tish kerakki, bu yarimstatik tuzilmalarni dasturda statik yoki dinamik tuzilma ko'rinishida ifodalash mumkin, faqat yarimstatiklik shartlarini buzmagan holda. Bunday tuzilmalarning barchasida ixtiyoriy elementlarga tashqaridan murojaat qilib bo'lmaydi.

3.3.1. Steklarni mantiqiy tasvirlash va ustida amal bajarish algoritmlari

Stek - chiziqli ma'lumotlar tuzilmasi bo'lib, ma'lumotlarni kiritish va chiqarish uning bir tomonidan amalga oshiriladi. Stek bir tomoni yopiq, ikkinchi tomoni ochiq tuzilma hisoblanadi, shu sababli ham elementlar tuzilmaga aynan bir tomondan kiritib, chiqariladi. Bunday stek kafeteriyadagi tarelkalar to'plamini eslatadi. Yangi elementlar stekning uchiga qo'yiladi va yuqori qismidan olinadi. Oxirgi qo'yilgan element stek uchidan birinchi bo'lib olinadi. Shu sababli, stek LIFO (last input - first output) tuzilishidagi ma'lumotlar tuzilmasi deyiladi, ya'ni, "oxirgi kelgan birinchi ketadi" prinsipi bo'yicha ishlaydi.

Xizmat ko'rsatishni keltirilgan tartibiga ko'ra, stek tuzilmasida faqatgina bitta pozitsiyaga murojaat qilish orqali amalga oshirish mumkin. Bu pozitsiya stekning uchi deyilib, unda stekka vaqt bo'yicha eng oxirgi kelib tushgan element nazarda tutiladi. Biz stekga yangi element kiritsak, bu element oldingi stek uchida turgan element ustiga joylashtiriladi hamda stekni uchida joylashib qoladi. Elementni faqatgina stek uchidan tanlash mumkin; bunda tanlangan element stekdan chiqarib tashlanadi va stek uchini esa chiqarib tashlangan elementdan bitta oldin kelib tushgan element tashkil qilib qoladi (bunday tuzilmaga ma'lumotlarga cheklangan murojaat tuzilmasi deyiladi). Birinchi kiritilgan element stek tubiga tushib qoladi.

Stekni grafik ko'rinishida quyidagicha tasvirlash mumkin:



3.4-rasm. Stek tuzilishi

Stek o'lchami cheklangan bo'lsa, elementni stekka qo'yilishi stekda kamida bitta elementga joy bo'lgan holdagina amalga oshiriladi. Shuning uchun stek ustida amal bajarishdan oldin stek holatini tekshirish lozim bo'ladi, ya'ni:

- stekka element kiritilishidan oldin joy borligini tekshirish;
- stekdan elementni o'chirishdan oldin element borligini tekshirish.

Steklar bilan ishlash uchun C++ tilida tayyor kutubxona mavjud bo'lib, dastur boshida ***#include<stack>*** yozuvini kiritish kerak. Dasturda stek quyidagicha e'lon qilinadi:

Stack *<toifa>* stek_nomi;

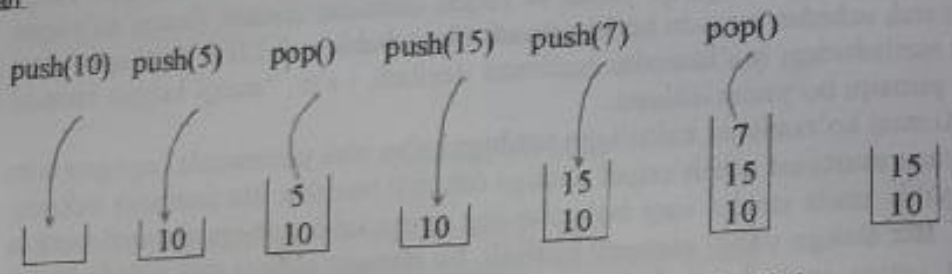
Misol uchun,

Stack <int> stek1;

Stek ustida quyidagi amallarni bajarish mumkin:

- **clear()** - stekni tozalash.
- **isEmpty()** - stekni bo'shlikka tekshirish;
- **push(el)** - stekka element kiritish;
- **pop()** - stekdan element o'chirish;
- **top()** - stekni uchidagi elementni o'chirmasdan o'qib olish.

Stekka element qo'shish va chiqarish amallari quyidagi 3.5 – rasmda keltirilgan.



3.5 – rasm. Stekda bajarilgan amallar ketma – ketligi

Unda bo'sh stekka 10 soni kiritilgan. Keyin stekka 5 soni kiritilgan, bu son 10 ni ustiga joylashadi, undan keyin chiqarish amali bajarilganda, stekdan 5 soni o'chiriladi. Chunki u 10 sonining yuqorisida joylashgan edi va 5 soni stekdan chiqariladi. Stekka 15 va 7 soni ketma - ket qo'yilgandan so'ng, eng yuqorida 7 soni

bo'ladi. Natijada chiqarish operatsiyasi amalga oshirilganda 7 soni stekni tark etadi. Chiqarish amalidan keyin stekda 10 soni 15 ning ostida qoladi.

Umuman olganda, stek ma'lumotlar saqlashda va ma'lumotlar to'plamidan elementlar teskari tartibda chiqarib olinadigan holatlarda ko'p foydalaniladi. Misol uchun kompyuterning mikroprotsessorida, arifmetik-mantiqiy qurilma tomonidan hisoblash jarayonlarida paydo bo'ladigan oraliq qiymatlarini saqlashda steklardan foydalanadi, kalkulyator dasturini tuzishda va yana ko'plab dasturiy ta'minotlarni tuzishda steklardan keng foydalaniladi. <stack> kutubxonasidan foydalanib, stek yaratish va ustida amal bajarishga doir misol keltiramiz.

Misol. Butun sonlardan iborat stekning juft o'rinda turgan elementlarini o'chirish dasturini keltiramiz.

```
#include<iostream>
#include<stack>
using namespace std;
int main(){
  int n,d;  cout<<"n="; cin>>n;
stack <int> st1,st2;
  for(int i=0;i<n;i++){
    cin>>d;
    st1.push(d);
    }
    cout<<"boshlangich holat: ";
  for(int i=0;i<n;i++){
    d=st1.top();
    cout<<d<<" ";
    st1.pop();
    st2.push(d);
    }
  for(int i=0;i<n;i++){
     d=st2.top();
     st2.pop();
     if(i%2!=0) st1.push(d);
     }
  cout<<"\n keyingi holat: ";
  n=st1.size();
  for(int i=0;i<n;i++){
     d=st1.top();
     cout<<d<<" ";
     st1.pop();
     st2.push(d);
     }
if(st1.empty()) cout<<"\nst1 bush\n";
system("pause");
}
```

Dastur natijasi:

```
n=6
1 2 3 4 5 6
boshlangich holat:  6  5  4  3  2  1
keyingi holat:  6  4  2
st1 bush
Для продолжения нажмите любую клавишу . . .
```

Yuqoridagi dastur kodida steklarni yaratish va ustida amal bajarishda tayyor kutubxonadan foydalanildi. Aslida shu kutubxonalarda steklarni qanday realizatsiya qilinganligiga qiziqadigan bo'lsak, uni yuqorida aytib o'tilganidek, dasturda ikki xil: statik va dinamik ko'rinishda amalga oshirish mumkin. Statik ko'rinishda amalga oshirishda massivlardan, dinamik ko'rinishda tasvirlashda esa bir bog'lamli ro'yxatlardan foydalanish mumkin. Quyida stekni yaratishda massiv- lardan foydalanishga doir misol keltiramiz. Unda stekni uzunligi *n* ga teng bo'lgan *a[n]* massiv ko'rinishida e'lon qilib, unda element kiritish – push(), element o'chirish – pop(), stek uchidagi elementni o'qib olish – top(), stekni bo'shlikka tekshirish – isEmpty(), stek elementlari sonini aniqlash – size() va stek massiv ko'rinishida amalga oshirilayotganligi sababli, uni to'lalikka tekshirish – isFull() funksiyalarini tuzib olamiz. Stekning sathi, ya'ni unda mavjud elementlar sonini R bilan belgilab olamiz va dastlab, stek bo'shligida R=0. Stekka yangi element kiritiladigan bo'lsa, uni R-pozitsiyaga joylab, keyingi keladigan element uchun bo'sh joyni ko'rsatib turishi uchun R ni qiymatini bittaga oshirib qo'yiladi.

```
#include<iostream>
using namespace std;
template<typename T>
class Stack{
    public:
        Stack(){ n=10; }
        Stack(int n){ this->n=n; }
        void push(T x){
            if(isFull())
                cout<<"stek to'ldi";
            else a[R++]=x;
        }
        void pop(){
          if(isEmpty())
                cout<<"stek bo'sh";
            else{  R--;
            }
        }
        T top(){
            if(isEmpty())
                cout<<"stek bo'sh";
            else{
                return a[R-1];
            }
        }
        void print(){
            if(isEmpty())
                cout<<"stek bo'sh";
            else{
                cout<<"\nstek elementlari: ";
                for(int i=R-1;i>=0;i--)
                    cout<<a[i]<<"  ";

                }
                cout<<endl;

            }
            bool isFull(){
                return (R>=n)?1:0;

            }
            bool isEmpty(){
                return (R==0)?1:0;

            }
    private:
            int n;
            T *a=new T[n];
            int R=0;

};
int main(){
        int n,d; cout<<"n=";cin>>n;
        Stack <int> st1(n);
        for(int i=0;i<n;i++){
            cin>>d;
            st1.push(d);

        }
        st1.print();
        int t=st1.top(); st1.pop();
        cout<<t<<" o'chdi \n";
        st1.print();
        int m; cout<<"m=", cin>>m;
        Stack<float> st2(m);
        float f;
        for(int i=0;i<m;i++){
            cin>>f;
            st2.push(f);

        }
        st2.print();
        f=st2.top(); st2.pop();
        cout<<f<<" o'chdi \n";
        st2.print();
        return 1;

}
```

Dastur natijasi:

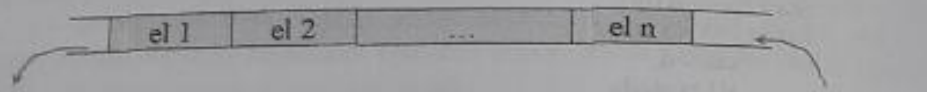
```
n=3
1 2 3
stek el-tlari: 3  2  1
3 o'chdi
stek el-tlari: 2  1
m=4
1.2  2.1  3.2  4.5
stek el-tlari: 4.5  3.2  2.1  1.2
4.5 o'chdi
stek el-tlari: 3.2  2.1  1.2
```

3.3.2. Navbatlarni mantiqiy tasvirlash va ustida amal bajarish algoritmlari

Kundalik xayotda deyarli har kuni har bir inson navbat tushunchasi bilan duch keladi. Umuman olganda, navbat elementi qandaydir xizmat ko'rsatishga buyurtma hisoblanadi: masalan, ma'lumotlar byurosidan kerakli ma'lumotni olish, kinoteatrlarda chipta olish, do'konda xarid qilib olingan mahsulotlarga kassada pul to'lash va boshqalar.

Navbat bu shunday tuzilmaki, u elementlar qo'shilishi bilan kengayib boradi va elementlarni faqatgina bir tomondan qabul qilib, ikkinchi tomondan chiqaruvchi tuzilma hisoblanadi. Stekdan farqli holda, navbat tuzilmasi har ikkala tomondan ochiq tuzilma hisoblanadi va element kiritish bir tomondan, chiqarish esa ikkinchi tomonidan amalga oshiriladi. Navbat FIFO(first input - first output – birinchi kelgan birinchi ketadi) ko'rinishidagi tuzilmadir.

Bunday ma'lumotlar tuzilmasi real navbatni modellashtirishda katta ahamiyatga ega. Bunda xizmat ko'rsatishga kelib tushgan talab, uning ijrosi, ya'ni xizmat ko'rsatish tartibini aniqlashda zarur bo'ladi.



3.6-rasm. Navbat tuzilmasi

Bu yerdan ko'rinib turibdiki, stekdan farqli ravishda navbatda birinchi kelgan elementga birinchi bo'lib xizmat ko'rsatiladi. Demak, navbatda elementni olish ro'yxat boshidan, yozish esa oxiridan amalga oshiriladi.

Kompyuter xotirasida real navbat elementlari soni chekli bo'lgan bir o'lchamli massiv ko'rinishida yaratiladi. Albatta, bunda navbat elementi turini ko'rsatish va navbat bilan ishlashni ko'rsatuvchi o'zgaruvchi zarur bo'ladi. Navbat fizik bos-qichda xotira sohasini ro'yxat ketma-ketligi bo'yicha to'laligicha egallaydi.

C++ tilida navbat tuzilmasini yaratish va ustida amal bajarish uchun alohida kutubxona mavjud.

#include<queue>

Navbatni dasturda e'lon qilish quyidagicha:

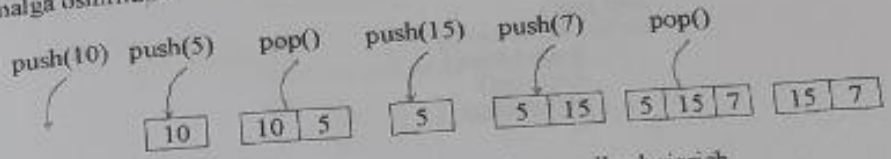
Queue <int> nav1;

Navbat ustida quyidagi amallarni bajarish mumkin:

- **clear()** - navbatni tozalash;
- **isEmpty()** – navbatni bo'shlikka tekshirish;
- **push(el)** — *el* elementni navbatga joylashtirish;
- **pop()** —navbatdan birinchi elementni olish;
- **front()** — navbatning birinchi elementini o'chirmasdan qaytarish.

Navbatda bajariladigan *push* va *pop* amallari 3.7-rasmda keltirilgan. Steklardan farqli ravishda, navbatlarda o'zgarishlar uning oxirida va boshida bo'lishi nazorat

qilinishi lozim. Elementlar navbatga oxiridan joylashtiriladi, olish esa boshidan amalga oshiriladi.



3.7-rasm. Navbatda ketma – ket amallar bajarish

Misol. <queue> kutubxonasidan foydalanib, butun sonlardan iborat navbat yaratish va uning o'rtasidagi elementni o'chirish dasturini keltiramiz. Bunda agar navbat uzunligi toq bo'lsa, o'rtadagi bitta element, juft bo'lsa, ikkita element o'chiriladi.

```
#include<iostream>
#include<queue>
using namespace std;
int main(){
  int n,s;cout<<"n=";cin>>n;
      queue <int> nav1;
  for(int i=0;i<n;i++){
    cin>>s;
    nav1.push(s);
    }
      int i=0;
      cout<<"\n boshlangich holat: ";
  while(i++<n){
    s=nav1.front();
    nav1.pop();
    nav1.push(s);
    cout<<s<<" ";
  }
      cout<<endl;
      if(n>=3){
            if(n%2==0){
                  for(int i=0;i<n/2-1;i++){
                        s=nav1.front();
                        nav1.pop();
                        nav1.push(s);
                  }
                  nav1.pop();nav1.pop();
                  for(int i=0;i<n/2-1;i++){
                        s=nav1.front();
                        nav1.pop();
                        nav1.push(s);
                  }
            }
            else{
                  for(int i=0;i<n/2;i++){
                        s=nav1.front();
```

```
                nav1.pop();
                nav1.push(s);
            }
            nav1.pop();
            for(int i=0;i<a/2;j++){
                s=nav1.front();
                nav1.pop();
                nav1.push(s);
            }
        }
    }
    i=0; n=nav1.size();
    cout<<"\n keyingi holat: ";
    while(i++<n){
      s=nav1.front();
      nav1.pop();
      nav1.push(s);
      cout<<s<<" ";
    }
    cout<<endl;
    system("pause");
}
```

Dastur natijasi:

```
n=6
1 2 3 4 5 6

boshlangich holat: 1  2  3  4  5  6

keyingi holat: 1  2  5  6
```

Yuqorida keltirilgan dasturda navbat tuzilmasini yaratish va ustida amal bajarish uchun <queue> kutubxonasidan foydalanildi. Mutaxassis sifatida bu tuzilma kutubxonalarda qanday realizatsiya qilinishi mumkinligiga qiziqadigan bo'lsak, stek singari navbatni ham dasturda statik yoki dinamik tarzda amalga oshirish mumkin. Navbat tuzilmasini massivlardan foydalanib realizatsiya qilishga to'xtalib o'tamiz. Uzunligi *n* ga teng bo'lgan *a[n]* massiv e'lon qilishdan boshlaymiz. Joriy vaqtda navbatda mavjud elementlar sonini R bilan belgilaymiz. Navbat bo'sh bo'lganda, R=0 va navbatga element kiritilganda, R-pozitsiyaga joylanadi va R oxirgi kelib tushgan elementdan bitta keyingi bo'sh joyga o'tkaziladi. Ana shu tartibda, navbatni statik tarzda amalga oshirishga doir misol ko'rib chiqsak. Yuqorida keltirilan misolda *queue* kutubxonasini o'chirib, o'rniga asosiy dastur tanasidan oldin quyidagicha navbatni massiv ko'rinishida amalga oshirish sinfini yozamiz.

```
template<typename T>
class queue{
    public:
        queue(){ this->n=10; }
        queue(int n){ this->n=n; }
        bool isEmpty(){ return (R==0)?1:0; }
        bool isFull(){ return (R>=n)?1:0; }
```



```
        void push(T x){
            if(isFull()) cout<<"navbat to'ldi\n";
            else a[R++]=x;
        }
        void pop(){
            if(isEmpty()) cout<<"navbat bo'sh\n";
            else{
                for(int i=0;i<R;i++) a[i]=a[i+1];
                R--;
            }
        }
        T front(){
            if(isEmpty()) cout<<"navbat bo'sh\n";
            else return a[0];
        }
        int size(){ return R; }
    private:
        int n, R=0;
        T *a=new T[n];
};
```

Bu *queue* sinfida 2 ta konstruktor e'lon qilingan bo'lib, birinchisi navbat uzunligi noma'lum bo'lsa, n=10 deb belgilash uchun, ikkinchi konstruktor esa navbat tuzilmasi initsializatsiya qilinayotganda uning uzunligi n ga qiymat berish uchun ishlatiladi. Dastur asosiy tanasi deyarli o'zgarishsiz qoladi va natija ham aynan bir xil chiqadi. Yuqorida keltirilgan dastur kodidan shunisi bilan farq qiladiki, bunda navbat tuzilmasi e'lon qilinayotganda quyidagicha initsializatsiya qilish talab qilinadi: *queue <int> nav1(n);*

3.3.3. Stek va navbatni bog'langan ro'yxat ko'rinishida tasvirlash

Yuqorida keltirilgan misollarda stek va navbat tuzilmalari statik ko'rinishda, ya'ni massivlardan foydalanib, amalga oshirildi. Bu tuzilmalarni dinamik tarzda, ya'ni bog'langan ro'yxat ko'rinishida amalga oshirish ham mumkin va bunda bir qator ustunliklarga ega bo'lish mumkin, ya'ni tuzilma uzunligiga oldindan chegara qo'yilmaydi va dastur bajarilishi mobaynida tuzilma uzunligi o'zgaruvchan bo'lishi mumkin.

Stekni bir bo'glamli ro'yxat ko'rinishida amalga oshirish protseduralarini *Stack* sinfi misolida ko'rib chiqamiz. Bunda stekni Node sinfiga tegishli elementlar ketmaketligi ko'rinishida ifodalaymiz. Elementlarni kiritish va chiqarish ro'yxat boshidan amalga oshiriladi. Ro'yxat boshi ko'rsatkichi **lst* o'zgaruvchisi va ro'yxat elementlari soni n o'zgaruvchisini qabul qilamiz.

```
Template<typename T>
class Stack{
    public:
        class Node{
          public:
            T info;
            Node *ptr;
        };
```

```
    Stack(){ this->lst=NULL; this->n=0; }
    bool isEmpty(){ return (lst==0)?1:0; }
    void push(T x){
        Node *p=new Node;
        p->info=x;
        if(isEmpty()) p->ptr=NULL;
            else  p->ptr=lst;
        lst=p;
        n++;
    }
    void pop(){
        if(isEmpty()) cout<<"stek bo'sh\n";
        else{
            Node *p=lst;
            lst=lst->ptr;
            delete p;
            n--;
        }
    }
    T top(){
        if(isEmpty()) cout<<"stek bo'sh\n";
        else return lst->info;
    }
    int size(){ return n; }
    void print(){
        if(isEmpty()) cout<<"stek bo'sh\n";
        else {
            cout<<"stek elementlari: ";
            Node *p=lst;
            while(p){
                cout<<p->info<<" ";
                p=p->ptr;
            }
            cout<<endl;
        }
    }
private:
    Node *lst=new Node;
    int n;
};
```

Xuddi shunday qilib, navbat tuzilmasini ham bir bog'lamli ro'yxat ko'rinishida tashkil qilish mumkin. Buning uchun *Queue* sinfini yaratamiz va unda *Node* sinfiga tegishli elementlar ketma-ketligini hosil qilish mumkin. Elementlarni navbatga kiritishda ro'yxat oxiriga qo'shib, chiqarishda ro'yxat boshidan olinadi. Navbatni bunday tarzda, bir bog'lamli ro'yxat ko'rinishida tashkil qilish va ustida amal bajarish protseduralarini quyidagicha tuzish mumkin.

```
template<typename T>
class Queue{
    public
        class Node{
            public:
                T info;
                Node *ptr;
};
Queue(){
        this->lst=NULL;
        this->last=NULL;
        this->n=0; }
bool isEmpty(){ return (lst==0)?1:0; }
void push(T x){
        Node *p=new Node;
        p->info=x;
        p->ptr=NULL;
        if(isEmpty()){
            lst=p;
            last=p;
        }
        else {
            last->ptr=p;
            last=p;
        }
        n++;
}
void pop(){
        if(isEmpty()) cout<<"navbat bo'sh\n";
        else{
            Node *p=lst;
            lst=lst->ptr;
            delete p;
            n--;
}
}
T front(){
        if(isEmpty()) cout<<"navbat bo'sh\n";
        else return lst->info;
}
int size(){ return n; }
void print(){
        if(isEmpty()) cout<<"navbat bo'sh\n";
        else {
            cout<<"navbat elementlari: ";
            Node *p=lst;
            while(p){
                cout<<p->info<<" ";
                p=p->ptr;
            }
            cout<<endl;
        }
    }
private
    Node *lst=new Node;
    Node *last=new Node;
    int n;
```

);

Shu o'rinda shuni ta'kidlash kerakki, stek va navbat tuzilmalarini bog'langan ro'yxat ko'rinishida tasvirlashda istalgan elementga uning adresi bilan tashqaridan murojaat qilishga yo'l qo'ymaslik kerak. Bu stek va navbat tuzilmalarining yarim-statiklik xususiyatlaridan biri hisoblanadi.

Nazorat savollari

1. Ro'yxat deb nimaga aytiladi?
2. Ro'yxat turlarini aytib o'ting.
3. Yarimstatik ma'lumotlar tuzilmasi nima va unga nimalar kiradi?
4. Stek va uning xususiyatlari?
5. Steklarni dasturda e'lon qilinishi?
6. Navbat nima va dasturda qanday ifodalanadi?
7. Bu tuzilmalar statik va dinamik tuzilmalardan nimasi bilan farq qiladi?
8. Navbat bilan stek qanday ma'lumotlar tuzilmasiga kiradi?
9. Stekdan elementni tanlash qanday amalga oshiriladi?
10. Stekning yuqori elementini o'chirmasdan o'qish qay yo'sinda amalga oshiriladi?
11. Qanday xizmat ko'rsatish turiga FIFO, qaysi biriga LIFO deb ataladi?
12. Navbatning bo'shligi belgisi qanday?

IV BO'LIM.
CHIZIQSIZ MA'LUMOTLAR TUZILMASI

Agar tuzilmani tashkil etuvchi elementlar bog'liqligi qat'iy tartiblanmagan bo'lsa, u holda bunday tuzilmaga chiziqsiz ma'lumotlar tuzilmasi deb ataladi. Chiziqsiz ma'lumotlar tuzilmasida elementlar orasidagi munosabatlar ixtiyoriy bo'lishi mumkin. Chiziqsiz tuzilmani 3 ta farqli belgisi mavjud:

- tuzilmani har bir elementi boshqa ixtiyoriy elementga murojaat qilishi mumkin;
- tuzilmani berilgan elementiga mazkur tuzilmaning ixtiyoriy sondagi elementi murojaat qilishi mumkin;
- murojaatlar og'irlikka, ya'ni murojaatlar ierarxik ko'rinishga ega bo'lishi mumkin.

Chiziqsiz malumotlar tuzilmasi sinflanishi:

- *ro'yxatlar*: chiziqsiz ikki bog'lamli, ko'p bog'lamli;
- *daraxtlar*: binar daraxtlar, ko'p o'lchamli daraxtlar;
- *graflar*: yo'naltirilgan graf(orgraf),yo'naltirilmagan graf(graf), gipergraf.

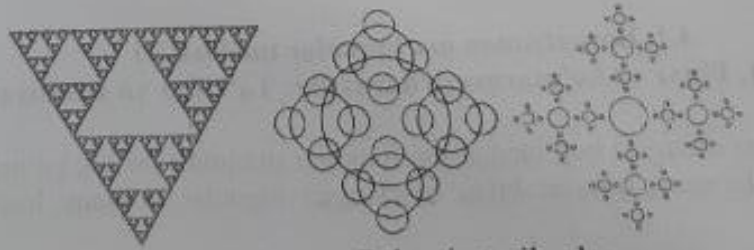
Chiziqsiz tuzimalar bilan ishlashda rekursiyadan foydalaniladi. Rekursiya tushunchasiga to'xtalib o'tsak.

Rekursiya – shunday jarayonki, bunda jarayonni borishi o'ziga o'zi murojaat qilish bilan bog'liq bo'ladi. Rekursiya tushunchasi jarayonlar, obyektlar, ma'lumotlar tuzilmalari, funksiyalar va algoritmlarga nisbatan ishlatilishi mumkin.

Rekursiv funksiya – bu o'zini o'zi chaqiruvchi funksiyadir. Funksiya rekursiv bo'lishi uchun bir nechta shartlarga javob berishi kerak:

- funksiya rekursiv bo'lishi uchun, albatta, uning kirish argumenti bo'lishi shart;
- har safar o'ziga o'zi murojaat qilish vaqtida argument qiymatida qandaydir o'zgarish berilishi kerak. Aks holda funksiya har safar o'zini o'zi chaqirganda aynan bir qiymtalar ustida bir xil amallar bajariladi va natijada rekursiv jarayon cheksiz bajarilishi mumkin;
- funksiya tarkibida o'ziga o'zi murojaat qilish holatini to'xtatishning sharti berilish kerak, aks holda funksiya cheksiz ravishda qayta chaqirilib yotaveradi.

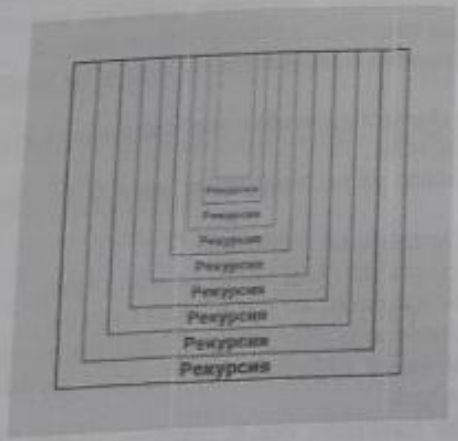
Rekursiv ma'lumotlar tuzilmasiga misol qilib shunday tuzilmalarni olish mumkinki, ushbu tuzilmalarning elementlarining o'zi ham xuddi shunday tuzilma kabi bo'ladi (4.1-rasm).



4.1-rasm. Rekursiv tuzilmalar

4.1-rasmdagi har bir shaklni tuzilma deb oladigan bo'lsak, uning har bir elementi ushbu tuzilma kabi ko'rinishga ega.

Rekursiyani bir nechta ichma-ich joylashgan aynan bir xil xonalarni biridan ikkinchisiga o'tib, oxiri xonalardan orqaga qaytib chiqish holatiga o'xshatish mumkin.



4.2-rasm. Rekursiya tushunchasi illyuziyasi

Chiziqsiz dinamik tuzilmalar, ayniqsa, daraxtsimon tuzilmalar bilan ishlashda rekursiyadan foydalanish samaralidir. Misol uchun, 13 ta dastlabki Fibonachchi sonlarini ekranga chiqarishda rekursiyadan foydalanish dasturini ko'rishimiz mumkin.

```
#include <iostream>
int fibonacci(int number){
  if (number == 0) return 0;
  if (number == 1) return 1;
  return fibonacci(number-1) + fibonacci(number-2);
}
int main(){
  for (int count=0; count < 13; ++count)
    std:: cout << fibonacci(count) << " ";
  return 0;
}
```

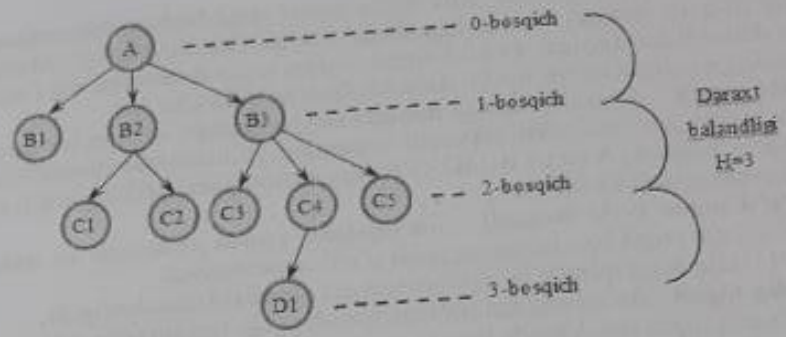
Natija:

0 1 1 2 3 5 8 13 21 34 55 89 144

4.1. Daraxtsimon ma'lumotlar tuzilmalari

4.1.1. Binar va ko'ptarmoqli daraxtlar. Ta'riflar va xususiyatlar

Daraxt – bu chiziqsiz bog'langan ma'lumotlar tuzilmasi bo'lib, yo'naltirilgan (yoki yo'naltirilmagan) qirralar bilan bog'langan tugunlar to'plami hisoblanadi (4.3-rasm).

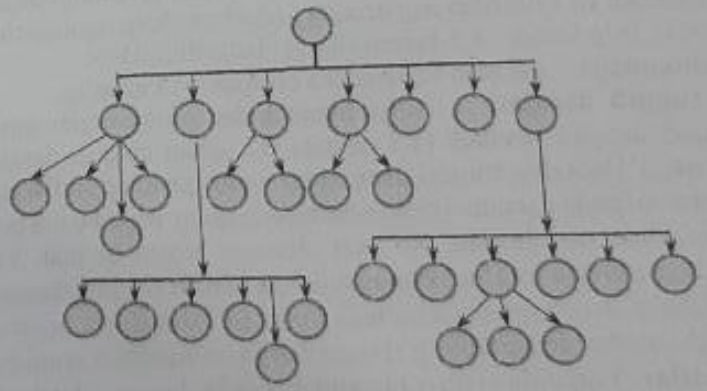


4.3-rasm. Daraxtsimon tuzilma

Daraxtlarning afzalliklari. Daraxtlar juda foydali va ko'p ishlatiladigan tuzilmadir, u quyidagi afzalliklarga ega:

- daraxtlar tuzilmaviy munosabatlarni aks ettiradi;
- daraxtlar ierarxiyani ifodalash uchun ishlatiladi;
- daraxtlar kiritish va qidirishda samarali hisoblanadi;
- daraxtlar juda moslashuvchan tuzilma bo'lib, undagi qismdaraxtlarni kam kuch-harakat bilan ko'chirishga imkon beradi.

Daraxtlar fayl tizimlari kabi dasturlarni modellashtirishda ham ishlatiladi (4.4-rasm).



4.4-rasm. Daraxt tuzilmasi

Daraxt tugunsiz bo'sh bo'lishi mumkin yoki bitta ildiz va bir nechta qismdaraxtlardan iborat bo'lishi mumkin. Daraxt o'zining quyidagi belgilari bilan tavsiflanadi:

- **ildiz** - daraxtda shunday bitta tugun borki, unga boshqa elementlardan murojaat yo'q va istalgan boshqa tugunlarga u orqali borish mumkin. Mazkur tugunga daraxt ildizi deyiladi, ya'ni A tugun. Ushbu tugun daraxtning eng yuqori cho'qqisida joylashgan bo'lib, har bir daraxtda faqat bitta ildiz bo'ladi.

- **ota tugun** - daraxtda ildizdan tashqari barcha tugunlar o'zidan bir daraja yuqorida joylashgan "ota" deb ataluvchi tugunning yo'naltirilgan qirrasi bilan bog'lanadi; 4.3-rasmda A tugun B1, B2 va B3 tugunlarning otasi, B2 tugun C1, C2 tugunlarga ota tugun hisoblanadi;

- **o'g'il tugun** (yoki farzand) – ota tugundan pastda joylashgan va undan chiquvchi qirralar orqali bog'langan tugunlar o'g'il tugun deyiladi;

- **yo'l** - daraxtdagi qirralar bo'ylab joylashgan tugunlar ketma-ketligidir;

- **oraliq tugun** - daraxtning har bir tuguni oraliq yoki terminal (barg) bo'lishi mumkin. Oraliq tugun deb kamida bitta farzand tugunga ega bo'lgan tugunga aytiladi. 4.3-rasmda B2, B3, C4 – oraliq tugunlar hisoblanadi;

- **barg tugun** – bironta farzand tugunga ega bo'lmagan tugunlarga barg yoki terminal tugunlar deyiladi, 4.3-rasmda B1, C1, C2, C3, D1, C5 – barglardir.

- **og'ayni tugunlar** - bitta otaning og'il tugunlari o'zaro og'ayni tugunlar hisoblanadi;

- **qismdaraxt** – biror tugunning avlodlarini anglatadi; 4.3-rasmda B2 va B3 ga tegishli qismdaraxtlar mavjud;

- **daraxtni ko'rikdan o'tkazish** – biror usul bilan daraxt tugunlarini ko'rib chiqish;

- **daraxt bosqishlari** – daraxt ildizi 0-bosqichda joylashadi, uning farzand tugunlari 1-bosqichda, nevara tugunlar esa 2-bosqichda joylashgan bo'ladi;

- **tugun balandligi** – tugundan eng quyida joylashgan barg tugungacha bo'lgan qirralar soni, ya'ni masofa, misol uchun B2 tugunning balandligi 1ga, B3 tugunning balandligi 2 ga, B1 tugunning balandligi 0 ga teng. Daraxt balandligi deganda ildiz balandligi tushuniladi va u ildizdan eng uzoqda joylashgan barg tugungacha bo'lgan yo'l uzunligi bilan belgilanadi. 4.3-rasmda daraxt balandligi H=3;

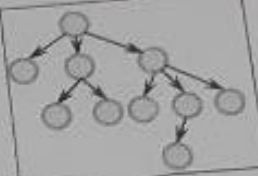
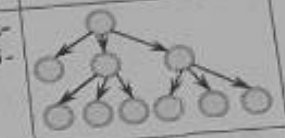
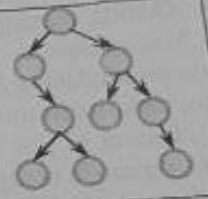
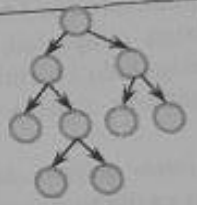
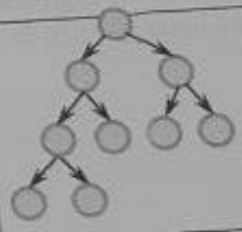
- **tugun chuqurligi** - ildizdan tugungacha bo'lgan yo'l uzunligi;

- **tugun chiqish darajasi** – daraxt tugunlaridan chiqayotgan qirralar soni tugunning chiqish darajasi deyiladi (4.3-rasmda B2 uchun chiqish darajasi 2, B3 uchun esa 3 ga teng). Daraxtlar chiqish darajasi bo'yicha quyidagi sinflarga ajratilad

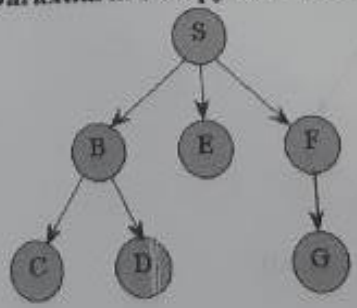
Kompyuter xotirasida daraxtni ifodalashning eng qulay usuli bu uni bog'langan ro'yxatlar ko'rinishida ifodalashdir. Ro'yxat elementi tugun qiymati va chiqish darajasini o'z ichiga oluvchi informatsion maydonga hamda chiqish darajasiga teng bo'lgan ko'rsatkichlar maydoniga ega bo'lishi lozim (4.5-rasm), ya'ni elementning har bir ko'rsatkichi ushbu elementni o'g'il tugunlariga yo'nalishini aniqlaydi.

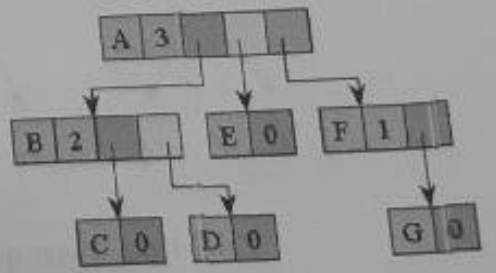
Binar daraxtlar. Yuqorida aytib o'tilganidek, har bir tuguni ikkitagacha o'g'il tugunga ega bo'lgan daraxtlarga binar daraxt deyiladi. Binar daraxtlar – ierarxik tuzilishga ega dinamik tuzilma bo'lib, elementlari xotirada turli sohalarda joylashishi mumkin va ular o'zaro ko'rsatkichlar bilan bog'lanadi. Binar daraxtning har bir elementi informatsion maydonga va ikkita ko'rsatkichli maydonga ega bo'ladi, ya'ni o'ng va chap. Binar daraxtlarning bir nechta turi mavjud bo'lib, ularga qisqacha to'xtalib o'tamiz:

- *piramida* (heap tree)– daraxtsimon tuzilma bo'lib, unda bitta bosqichda joylashgan barcha elementlar qiymatlari o'zidan yuqorida turuvchi element qiymatidan katta (yoki kichik) bo'ladi.

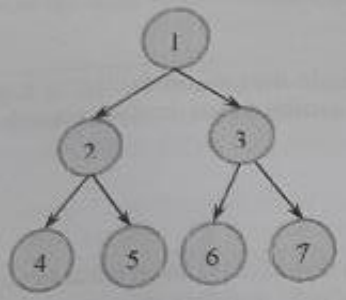
1	agar maksimal chiqish darajasi m bo'lsa, u holda bunday daraxt m-chi tartibli daraxt deyiladi; (rasmda 3-tartibli daraxt);	
2	agar chiqish darajasi 0 yoki m bo'lsa, u holda to'liq m-chi tartibli daraxt bo'ladi; quyidagi chizmada to'liq 3-tartibli daraxt keltirilgan;	
3	agar maksimal chiqish darajasi 2 bo'lsa, u holda bunday daraxt binar daraxt deyiladi;	
4	agar chiqish darajasi 0 yoki 2 bo'lsa, u holda to'liq binar daraxt deyiladi;	
5	agar binar daraxt to'liq bo'lsa va istisno tariqasida eng quyi darajasi chapdan o'ngga qarab to'ldiriladigan bo'lsa, tugallangan binar daraxt deyiladi.	

Daraxtlarni kompyuter xotirasida tasvirlash





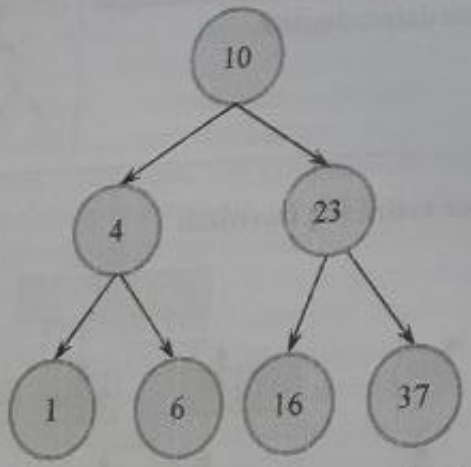
4.5-rasm. Daraxtning grafik shakldagi va chiziqsiz ro'yxat shaklidagi ifodalanishi



4.6-rasm. Piramida (heap tree) tuzilmasi

Odatda, bunday piramidalarni istalgan tartibdagi ketma-ketlikdan qurish mumkin. Misol uchun, $a_1, a_2, a_3, \ldots a_n$ butun sonlardan iborat ketma-ketlik berilgan bo'lsa, ularni piramidaning har bir bosqichini chapdan o'ngga qarab to'ldirish orqali ikkilik piramidani tuzish mumkin. Bunda a_i elementning chap o'g'liga a_{2i} va o'ng o'g'liga a_{2i+1} element to'g'ri keladi. Agar $a_1, a_2, a_3, \ldots a_n$ ketma-ketlik tartibsiz berilgan bo'lsa, piramida tuzilgach, eng quyi bosqichdan boshlab to ildizgacha, har bir kichik qismdaraxt (unda 3 ta element olinadi) elementlarining eng kichigi (yoki eng kattasi) yuqoriga chiqariladi, ya'ni ota tugun qilib joylashtiriladi. Shunday qilib, 4.6-rasmdagi kabi ko'rinishga ega bo'lgan piramida hosil qilinadi. Bunday piramidalar elementlarni o'sish yoki kamayish bo'yicha tartiblashda samarali tuzilma hisoblanadi;

- *binar qidiruv daraxti* – ota tugunga nisbatan chap o'g'il tugun qiymati kichik bo'lgan va ota tugunga nisbatan o'ng o'g'il tugun qiymati katta bo'lgan binar daraxtlarga binar qidiruv daraxti deyiladi.



4.7-rasm. Binar qidiruv daraxti

- AVL daraxti – muvozanatlangan binar daraxt bo'lib, tuzilma samaradorligini oshirish maqsadida element kiritish va o'chirish amallarini bajarishda daraxt har bir tugunining muvozanatlanganlik koeffisientlari tekshirilib turiladi. Agar qarayotgan tugunning chap va o'ng qismdaraxtlari balandliklari farqi 1 dan katta bo'lsa, daraxtni shu qismida muvozanatlash amali qo'llaniladi.

Binar daraxtlar eng ko'p foydalaniladigan daraxtlar turi hisoblanadi.

4.1.2. Daraxtlarni binar ko'rinishga keltirish algoritmi

Noformal algoritm:

1. daraxtning har bir tugunining chap shoxidan tashqari barcha shoxlari kesib tashlanadi;
2. kesib tashlangan shoxlardagi barcha tugunlar gorizontal chiziq bilan shu bosqichda joylashgan va otasi umumiy bo'lgan chapdagi tugunga ulanadi. Ushbu tugunlar o'zaro og'ayni tugunlar (siblings) hisoblanadi.

Algoritm amallar ketma-ketligi quyida keltirilgan.

4.8-rasm. m-o'lchovli daraxtni binar ko'rinishga keltirish

4.1.3. Binar daraxtlarni qurish algoritmi

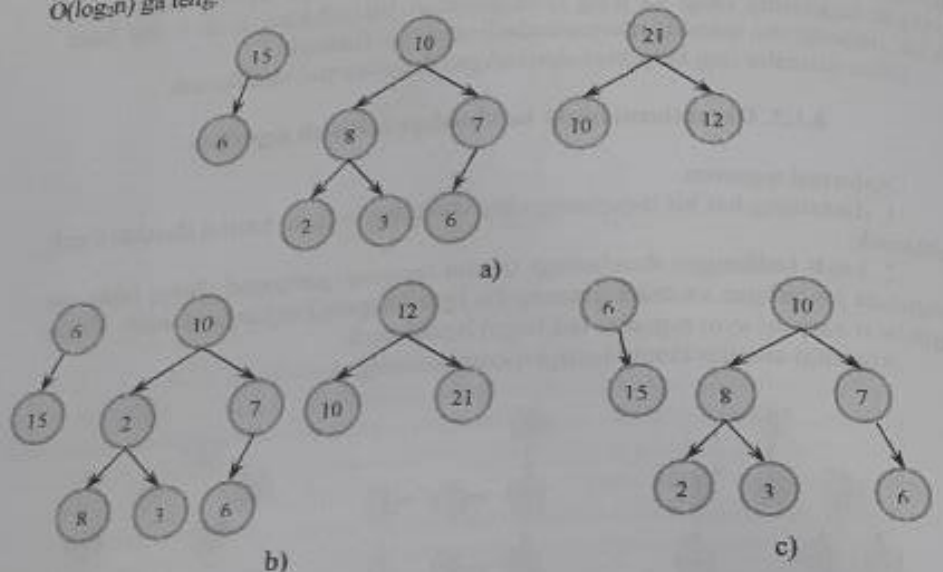
Piramida binar daraxtning bir turi bo'lib, ierarxik tuzilishga ega bo'lgan dinamik tuzilma hisoblanadi va uni inglizcha adabiyotlarda "heap tree" deb nomlanishini uchratish mumkin. "Heap tree" so'zi inglizchadan olingan bo'lib, ikkilik uyurma daraxti degan ma'noni anglatadi va quyidagi ikkita xususiyati bilan ajralib turadi:

- har bir tugun qiymati uning o'g'il tugunlari qiymatidan katta yoki teng (yoki kichik yoki teng bo'lishi ham mumkin);
- daraxt ideal muvozanatlangan, yoki daraxt barg tugunlari chapdan o'ngga qarab to'ldiriladi.

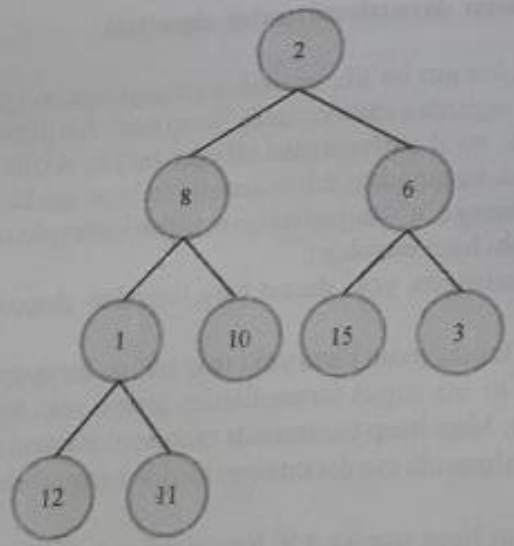
Agar har bir tugun o'g'il tugunlardan katta yoki teng bo'lsa, bu uyurma daraxtiga ***Max-heap***, aks holda ya'ni ota tugun farzandlardan kichik yoki teng bo'lsa, ***Min-heap*** deyiladi. Bu degani, Max-heap tuzilmasida maksimal element daraxt ildizida joylashadi, Min-heap tuzilmasida esa daraxtning ildizida minimal element joylashadi.

4.9, a-rasmdagi tuzilmalar heap tree va 4.9, b,c-rasmdagi tuzilmalar esa heap tree emas. Chunki b va c rasmda mos ravishda heap treening birinchi va ikkinchi xususiyatlari buzilgan. Qizig'i shuki, heap tree massiv yordamida yasalishi mumkin. Masalan, a[] = {2, 8, 6, 1, 10, 15, 3, 12, 11} butun sonlardan iborat massiv berilgan

bo'lsin. Undan yuqoridan pastga va chapdan o'ngga elementlarni joylab, daraxtni (heap tree bo'lmagan) hosil qilamiz (4.10-rasm). Bunday daraxtlarda bosqichlar soni O(log₂n) ga teng.



4.9-rasm. Heap tree (a) va heap tree bo'lmagan daraxtlar



4.10-rasm. Massivdan daraxt hosil qilish

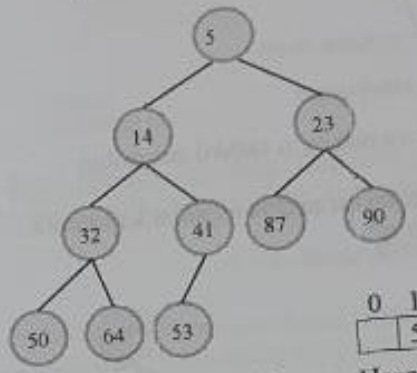
Bu daraxtni heap tree ko'rinishida qayta tashkil etish uchun uzunligi n ga teng *heap* massivini quyidagi shartlarga asoslanib tashkil etamiz:

$heap[i] \geq heap[2i]$, *for* $0 \leq i < \frac{n}{2}$ va

$heap[i] \geq heap[2i+1]$, *for* $0 \leq i < \frac{n-1}{2}$

Boshqacha qilib aytadigan bo'lsak, sonlar ketma-ketligidan heap tree tuzilmasini qurish uchun:

- 1-elementni daraxt ildizi qilib olamiz;
- qolgan har qanday i-element uchun quyidagi o'rinli:
  a) uning chap o'g'il tuguni 2*i indeksda;
  b) o'ng o'g'il tuguni esa 2*i+1 indeksda;
  c) uning ota tuguni i/2 indeksda bo'ladi.

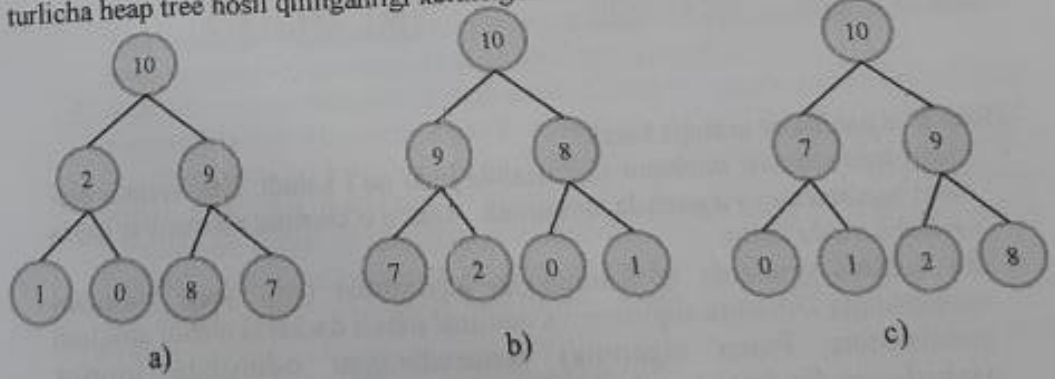


i-indeksdagi elementdan hosil qilingan tugunning chapida 2i-indeksdagi element, o'ngida 2i+1 – indeksdagi element va ota tugunida i/2-indeksdagi element joylashadi.

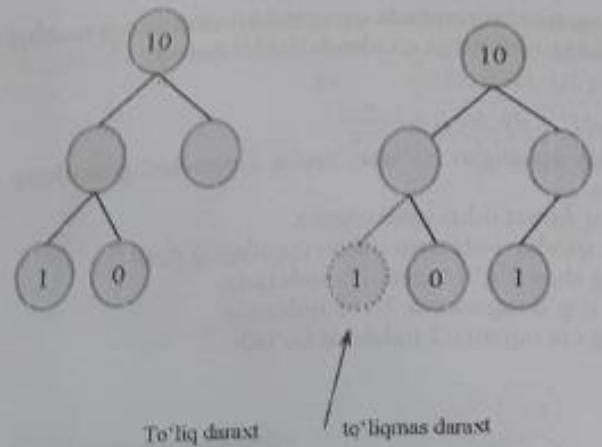
0	1	2	3	4	5	6	7	8	9	10
	5	14	23	32	41	87	90	50	64	53

4.11-rasm. Heap tree tuzilmasini tuzish

Ya'ni, 1-elementni ildiz qilib olingach, qolgan elementlarni chapdan o'ngga qarab, daraxt bosqichlarini to'ldirib joylashtirib boriladi. Har bir tugunda faqat ikkita o'g'il tugun chiqishi kerak. Agar shunday tartibda elementlar joylashtirilib chiqiladigan bo'lsa, har bir a[i]-o'rinda turgan ota tugunning chap tomoniga a[2*i]-element va o'ng tomoniga esa a[2*i+1]-element joylashadi. Quyida bir xil sonlardan turlicha heap tree hosil qilinganligi keltirilgan:

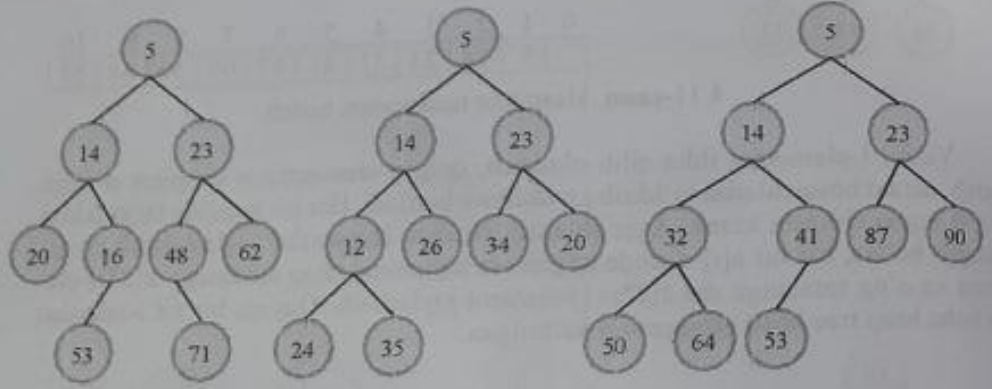


4.12 – rasm. Bir xil sonlardan tashkil topgan heap tree tuzilmalari



4.13-rasm. Heap treeni to'g'ri va noto'g'ri tashkil etilganligi

O'zlashtirishingizni tekshirib ko'rish uchun bir nechta misollar keltiramiz. Quyidagi binar daraxtlar min-heapmi yoki yo'q?



Heap tree tuzilmasi nimaga kerak?

- Heap tree ustuvor navbatni ifodalashda juda qo'l keladi. Eng kerakli element tuzilma eng yuqorisida joylashadi. Agar u o'chirilsa, elementlar qayta joylashtirilishi zarur.
- Bu tuzilma graflarda qo'llaniladigan algoritmlar (eng qisqa masofani aniqlashning Deykstra algoritmi va minimal narxli daraxt skeletini aniqlash masalasining Prima algoritmi) samaradorligini oshirishda ustuvor navbatlardan foydalanilganda qo'l keladi.

- Bundan tashqari heap tree - samaradorligi O(nlog2n) bo'lgan piramida saralash algoritmida ham qo'llaniladi.

Heap tree ko'rinishidagi tuzilmani kompyuterda oddiy massiv ko'rinishida tashkil etish mumkin va uning balandligi log2n (n-elementlar soni) ga teng. Min-heap tuzilmasini C++ da quyidagicha e'lon qilish mumkin.

```
class MinHeap{
  int *arr, // heap tuzilmasini massiv ko'rinishida tashkil etamiz va unga ko'rstakichni
        //e'lon qilamiz
  int max_size; // min heap uchun ajratilgan maksimal o'lcham
  int heap_size; // min heap elementlari soni
public:
  MinHeap(int n){              // konstructor
      heap_size = 0;
      max_size = n;
      arr = new int[n];
  }
};
int main(){
  MinHeap h(11);
}
```

Hosil qilingan Min-heap tuzilmasi ustida turli amal bajarish algoritmlari va ularni dasturda ifodalanishi bilan keying mavzularda tanishib chiqamiz.

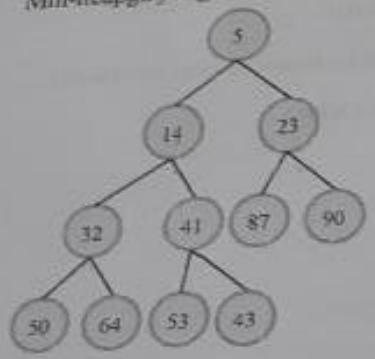
4.1.4. Binar daraxtlar ustida amallar

Min-heap tuzilmasi ustida quyidagicha amal bajarish algoritmlari mavjud:

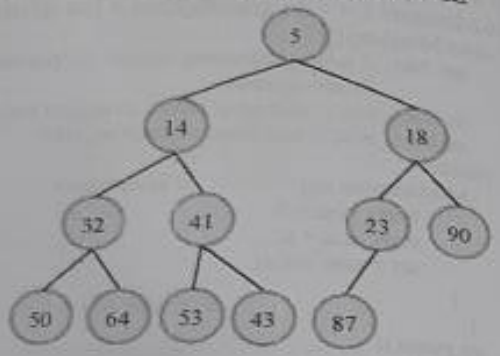
1. *getMin()* – Min-heap tuzilmasi ildizida joylashgan minimal elementni o'qib olish amali. Ushbu amalning samaradorligi O(1) ga teng.
2. *extractMin()* – Min-heap tuzilmasidan minimal elementni o'chirish amali. Ushbu algoritm samaradorligi O(logn) ga teng, chunki ildiz o'chirilgandan keyin Min-heap tuzilmasi qayta tuzib chiqilishi (*heapify()* amali deb ataladi) talab etiladi.
3. *decreaseKey()* – tuzilma elementi qiymatini kamaytirish. Bu amal samaradorligi O(log n) ga teng. Agar qiymati kamaytirilgan tugun ota tugun qiymatidan katta bo'lsa, qo'shimcha hech qanday amal bajarish shart emas. Aks holda, ushbu tugundan yuqoriga qarab barcha tugunlar joylashuvi Min-heap tuzilmasi shartlariga mosligi tekshirib chiqiladi.
4. *insert()* – yangi element kiritish amali O(log n) vaqtni talab etadi. Yangi element dastlab Min-heap daraxti oxiriga qo'shiladi. Agar yangi tugun ota tugun qiymatidan katta bo'lsa, hech qanday qo'shimcha amal bajarish shart emas. Aks holsa, tuzilmani ko'rikdan o'tkazib, Min-heap tuzilmasi shartlariga ko'ra, elementlar joylashuvini qayta tekshirib chiqish kerak bo'ladi. Min-heap tuzilmasiga yangi element kiritish algoritmi:
   - yangi elementni massivning navbatdagi indeksiga joylash;
   - yangi elementni ota tugun bilan solishtiriladi, agar yangi element otasidan kichik bo'lsa, ularni o'rin almashtiriladi;
   - bu jarayon takrorlanadi toki:

a) yoki yangi elementning otasi kichik yoki teng bo'lguncha;
b) yoki yangi element ildizga kelguncha (massivda 0 indeksga kelguncha).

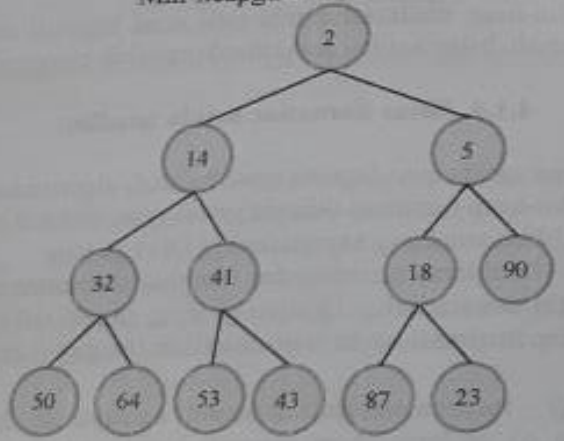
Min-heapga yangi 43 sonini kiritamiz.



Min-heapga 18 ni kiritamiz.



Min-heapga 2 ni kiritamiz.

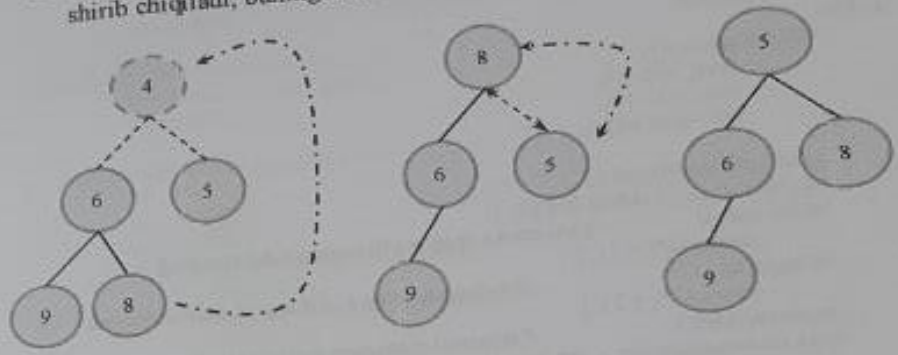


4.14-rasm. Min-heap tuzilmasiga yangi element kiritish

5. *delete()* – element o'chirish amali ham O(Log n) vaqtni talab qiladi. Bunda *decreaseKey()* amali yordamida o'chirilishi kerak bo'lgan element qiymati u e'lon qilingan toifaga tegishli eng kichik qiymat bilan o'rin almashtiriladi va shu yo'l bilan u Min-heap ildiziga chiqariladi. So'ngra *extracMin()* amali yordamida ushbu element tuzilmadan o'chiriladi. Ushbu amalni bajarishning yana boshqa bir algoritmini ham keltiramiz.

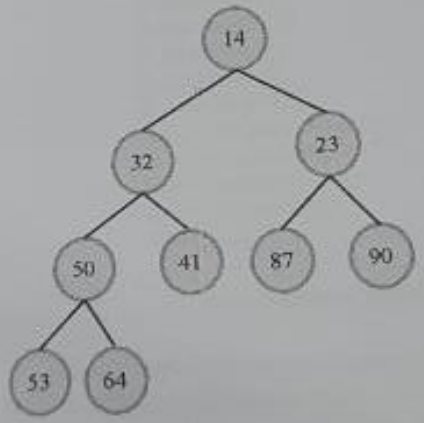
- O'chiriladigan element o'rniga daraxtdagi eng quyi darajada turgan eng o'ngdagi, ya'ni oxirgi element joylashtiriladi.
- O'rni o'zgargan shu element ikkita o'g'il tugunlari bilan solishtiriladi va agar ulardan katta bo'sa, kichik o'g'il tugunl bilan almashtiriladi.

- O'rinlashtirishda qatnashgan element ta'sir qiladigan qism daraxtlar tekshirib chiqiladi, buning uchun oxirgi ikkita amal takrorlanadi.

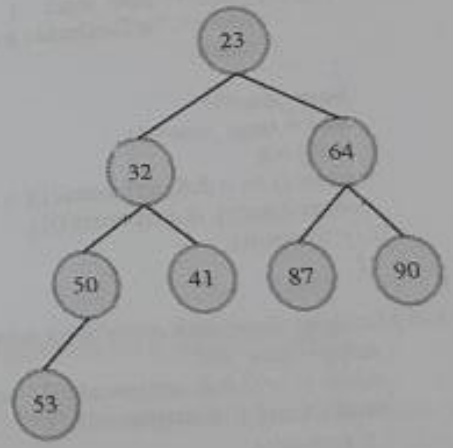


4.15-rasm. Min-heap tuzilmasidan elementni o'chirish amali

Masalan, 4.11-rasmdagi heap treedan 5 ni o'chiramiz. Quyidagi heap tree hosil bo'ladi.



Agar bundan 14 ni o'chirsak, quyidagi ko'rinishga keladi:



Min-heap tuzilmasini amalga oshirish dasturini keltiramiz.

```
#include<iostream>
#include<climits>
using namespace std;
void swap(int *x, int *y){
        int temp = *x;
        *x = *y;
        *y = temp;
};
```

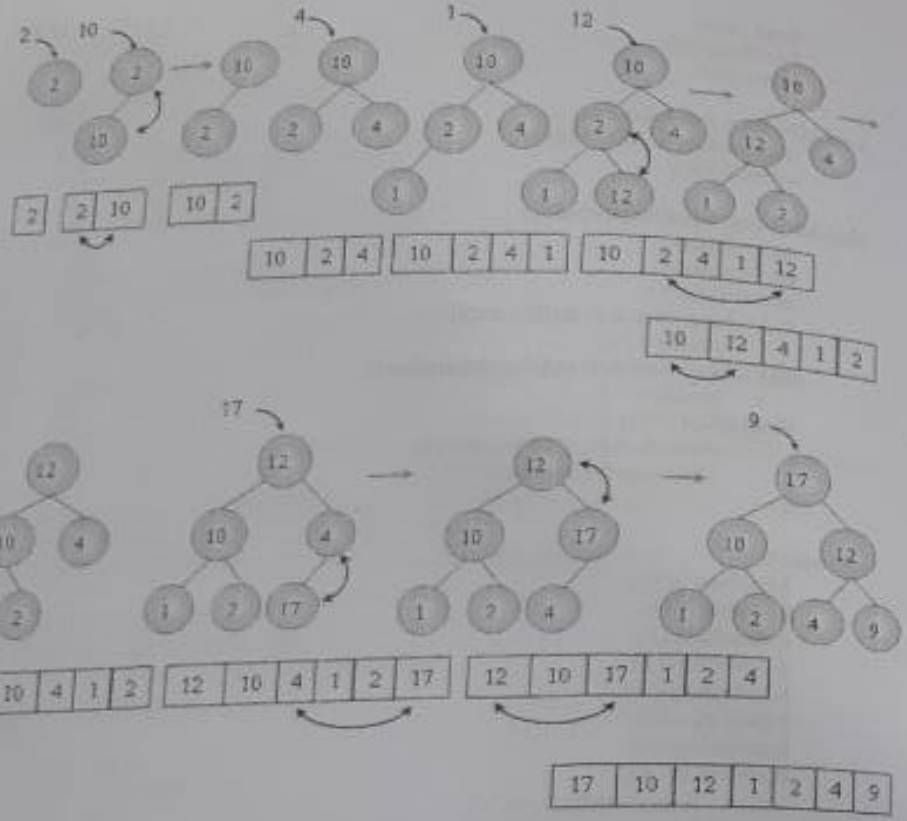
```
class MinHeap{
        int *arr;
        int max_size;
        int heap_size;
    public:
        MinHeap(int n){
            heap_size = 0;
            max_size = n;
            arr = new int[n]; ;
        }
        void MinHeapify(int );
        int parent(int i) { return (i-1)/2; }
        int left(int i){            // i-element chap o'g'il tuguni indeksini olish
            return (2*i + 1); }
        int right(int i) {          // i-element o'ng o'g'il tuguni indeksini olish
            return (2*i + 2); }
        int extractMin();           // minimal elementni tuzilmadan chiqarish
        void decreaseKey(int i, int new_val); // i-element qiymatini new_val qiymatga
                                              o'zgartirish
        int getMin() { return arr[0]; } // min heap ildizidagi minimal elementni olish
        void deleteKey(int i); // i-elementni o'chirish
        void insertKey(int k); // k – yangi elementni tuzilmaga kiritish
};
void MinHeap::insertKey(int k){
    if (heap_size == max_size)   {
        cout << "\nTuzilmada joy yo'q\n";
        return;
    }
    heap_size++;
    int i = heap_size - 1;
    arr[i] = k;
    while (i != 0 && arr[parent(i)] > arr[i]){
    swap(&arr[i], &arr[parent(i)]);
    i = parent(i);
    }
}
void MinHeap::decreaseKey(int i, int new_val){
    arr[i] = new_val;
    while (i != 0 && arr[parent(i)] > arr[i]){
    swap(&arr[i], &arr[parent(i)]);
    i = parent(i);
    }
}
int MinHeap::extractMin(){
    if (heap_size <= 0)
        return INT_MAX;
    if (heap_size == 1){
        heap_size--;
        return arr[0];
    }
    int root = arr[0];
    arr[0] = arr[heap_size-1];
```



```
    heap_size--;
    MinHeapify(0);
    return root;
}
void MinHeap::deleteKey(int i){
    decreaseKey(i, INT_MIN);
    extractMin();
}
void MinHeap::MinHeapify(int i){
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if (l < heap_size && arr[l] < arr[i])
        smallest = l;
    if (r < heap_size && arr[r] < arr[smallest])
        smallest = r;
    if (smallest != i){
        swap(&arr[i], &arr[smallest]);
        MinHeapify(smallest);
    }
}
int main(){
    MinHeap h(11);
    h.insertKey(3);
    h.insertKey(2);
    h.deleteKey(1);
    h.insertKey(15);
    h.insertKey(5);
    h.insertKey(4);
    h.insertKey(45);
    cout << h.extractMin() << " ";
    cout << h.getMin() << " ";
    h.decreaseKey(2, 1);
    cout << h.getMin();
    return 0; }
```

Heap treeni tashkil etish usullari va samaradorligi

Heap tree tuzilmasini dasturda massiv ko'rinishida ifodalash mumkin, ya'ni barcha heap tuzilmalarni massiv ko'rinishida ifodalash mumkin, lekin barcha massivlar heap tuzilmasi bo'lmaydi. Berilgan massiv elementlarini shunday joylash kerakki, natija heap tree tuzilmasini ifodalasin. Buning bir necha usullari mavjud. Eng soddasi bo'sh heap tuzilmasiga ketma-ket elementlarni joylash bilan amalga oshiriladi. Bu "yuqoridan-pastga" usuli (ya'ni elementlar heap tuzilmasiga yuqorida keltirilgan yangi element qo'shish algoritmi bilan kiritiladi) bo'lib, Jogn Williams tomonidan taklif etilgan. Quyida 4.16-rasmda "yuqoridan-pastga" algoritmi ifodalangan va dasturi keltirilgan.

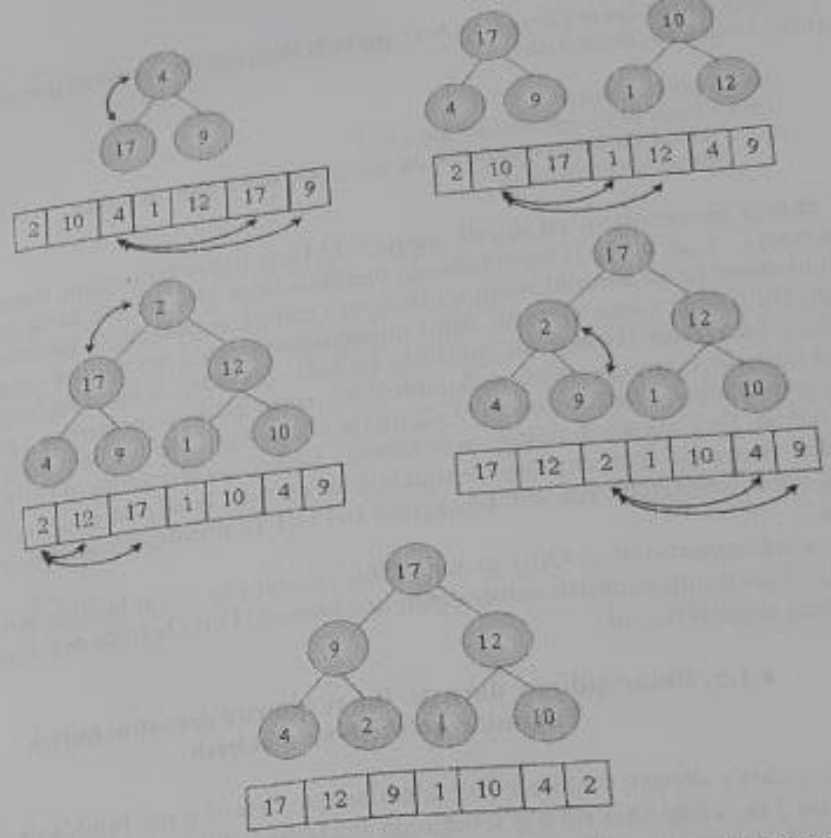


4.16-rasm. Max-heap tree tuzilmasini "yuqoridan-pastga" usuli bilan tashkil etish

Bu algoritm samaradorligini eng yomon holatda tekshiradigan bo'lsak, unga kiritilgan har bir element ildizgacha yuqoriga harakat qilishi kerak. Bunda k ta elementdan iborat bo'lgan MaxHeap tuzilmasida yangi kiritilgan element yuqoriga harakat qilishi uchun [lg k] ta o'rin almashtirishlar amalga oshirilishi kerak. Agar n ta yangi element kiritilsa, eng yomon holatda algoritm bajarilishi uchun quyidagicha o'rinlashtirishlar bajariladi, solishtirishlar ham xuddi shunday.

$$\sum_{k=1}^{n} \lfloor \lg k \rfloor \le \sum_{k=1}^{n} \lg k = \lg 1 + \cdots + \lg n = \lg(1 \cdot 2 \cdot \cdots \cdot n) = \lg(n!) = O(n \lg n)$$

Robert Floyd tomonidan taklif etilgan boshqa bir algoritmda MaxHeap tree "pastdan-yuqoriga" usuli yordamida amalga oshiriladi. Bunda kichik heap qismlar yaratiladi va davriy ravishda kattaroq heaplarga birlashtiriladi (4.17-rasm)



4.17-rasm. *Arr[2,10,4, 1, 12, 17, 9]* massivni "pastdan-yuqoriga" usuli bilan *MaxHeap tree* tuzilmasiga aylantirish

(quyidagi listingda keltirilgan moveDown() funksiyasi ildizdagi elementlarni quyiga harakat qildiradi.)

```
template<class T>
void movedown(T arr[], int first, int last){
    int largest=2*first+1;
    while(largest<=first){
        if(largest<last && arr[larges:]<arr[largest+1])
            largest++;
        if(arr[first]<arr[largest]){
            swap(arr[first],arr[largest]);
            first=largest;
            largest=2*largest+1;
        }
        else largest=last+1;
    }
};
```

Ushbu *movedown(T arr[], int first, int last)* funsiyasidan quyidagi psevdokodda keltirilganidek foydalaniladi:

```
FloydAlgorithm(arr[])
 for i = index of the last nonleaf down to 0
      restore the heap property for the tree whose root is arr[i] by calling
      moveDown(arr,i,n-1);
```

Bunda elementlarni tekshirish *arr[n/2-1]* barg tugun bo'lmagan elementdan boshlaymiz. Agar u o'g'il tugunlarining birontasidan kichik bo'lsa, katta qiymatli o'g'il element bilan almashtiriladi va jarayon yuqoridagi rasmdagi kabi davom ettiriladi. Bu usulda yangi element tahlil qilinayotgan paytda uning qism daraxti allaqachon *MaxHeap* daraxti ko'rinishida bo'ladi. Shunday qilib, *MaxHeap tree* daraxti pastdan yuqoriga qarab shakllantiriladi. Heap tuzilmasini bunday taskil qilishda, *moveDown()* funksiyasi (n+1)/2 marta chaqiriladi, har bir barg bo'lmagan tugun uchun. Eng yomon holatda, *moveDown()* funksiyasi elementni (n+1)/4 ta elementdan iborat bo'lgan eng quyi bosqichga ko'chiradi, bunda barg tugunlar bosqichiga yetib kelguncha har bir bosqichda (n+1)/4 ta o'rinlashtirishlarni amalga oshiradi.

Bu usul samaradorligi O(n) ga teng. Shu sababli eng o'g'ir holatda Williamning usuli Floydning usulidan samaraliroq hisoblanadi. O'rta holatda esa ikkala algoritm ham deyarli bir xil.

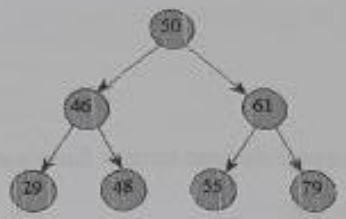
4.1.5. Binar qidiruv daraxti. Binar qidiruv daraxtini qurish. Tugunlar qo'shish va o'chirish

Binar qidiruv daraxti eng ko'p foydalaniladigan daraxtlar turi hisoblanib, unda har bir tugun 2 ta - chap va o'ng o'g'il tugunga ega bo'ladi. Shu sababli, binar qidiruv daraxti ikkilik daraxti hisoblanadi va unda har bir tugunning chapida o'zidan kichik va o'ngida o'zidan katta qiymatli elementlar joylashadi.

Binar daraxtni hosil qilish va unga yangi tugun qo'shish algoritmi. Butun sonlardan iborat kalitlar ketma-ketligi berilgan bo'lsin. Ularning birinchisi daraxt ildiziga joylashtiriladi. Qolgan barcha elementlar uchun quyidagi qoida o'rinli:

1. Qaralayotgan element dastlab, ildiz tugun bilan solishtiriladi. Agar ildizdan kichik bo'lsa, chap tomoniga o'tiladi, aks holda, ya'ni ildizdan katta bo'lsa, o'ng tomoniga o'tiladi.
2. O'tkazilgan tomon bo'sh bo'lsa, shu yerda o'g'il tugun sifatida joylashtiriladi. Aks holda, ya'ni o'tkazilgan tomonda oldindan tugun mavjud bo'lsa, endi bu tugun bilan solishtirilish uchun 1-qadamga o'tiladi. Bu jarayon toki bo'sh joy topilgunga qadar davom ettiriladi.

Shu tartibda kichik kalitga ega elementlar chap tomonga, katta qiymatli kalitga ega elementlar o'ng tomonga joylashtiriladi: *key(left)<key(Node)<key(right)*. Masalan, quyidagi elementlardan binar daraxt quramiz: 50, 46, 61, 48, 29, 55, 79. U quyidagi ko'rinishga ega bo'ladi:

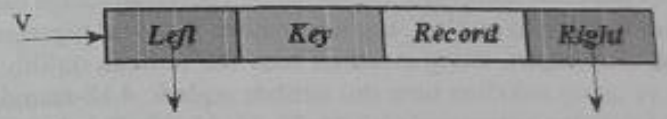


4.18-rasm. Binar qidiruv daraxti

Natijada, o'ng va chap qism daraxtlari bir xil bosqichga ega bo'lgan tartib- langan binar daraxt hosil qildik. Agar binar daraxtning ildizga nisbatan o'ng va chap qismdaraxtlari bosqichlari soni teng bo'lsa va barcha ota tugunlari aniq 2 ta o'gil tugunga ega bo'lsa, bunday binar daraxtga to'liq binar daraxt deyiladi. Yuqoridagi daraxt aynan shunday to'liq binar qidiruv daraxtiga kiradi.

To'liq binar qidiruv daraxtining har bir bosqichida 2^k ta element joylashadi. Bu yerda k – qaralayotgan bosqich raqami, odatda ildiz 0-bosqichda joylashgan hisoblanadi. Umuman olganda, tugallangan to'liq binar daraxt elementlari soni - 2^d-1 ga teng bo'lib, bu yerda d – daraxt bosqichlari soniga teng. Misol uchun, agar daraxtda 0-, 1- va 2-bosqichlarda elementlar joylashgan bo'lsa, demak elementlar bu daraxtda 3 ta bosqichga joylashgan. Shu sababli to'liq binar daraxtda elementlar soni 2^3-1=7 ga teng. Binar qidiruv daraxti ustida amal bajarish algoritmlari samaradorligi $O(\log_2 N)$ ga teng.

Binar qidiruv daraxtini kompyuter xotirasida fizik tasvirlanishiga ko'ra, har bir element kamida to'rtta maydonga ega yozuv hisoblanadi.



4.19-rasm. Binar qidiruv daraxti elementi tuzilishi

Har bir element ikkita ko'rsatkichli (o'ng va chap) maydon, kalit maydon va informatsion maydondan tashkil topadi (4.19-rasm). Kalit maydondan – tuzilmadagi elementlarni identifikatsiyalash uchun ishlatiladi, ya'ni har bir element informatsion maydonida nima yozilganidan qat'iy nazar, unikal bo'lgan, hech qaysi elementda uchramaydigan kalit ma'lumotga ega bo'lishi kerak. *Left* maydonga joriy tugunning chap o'gil tugunining xotiradagi adresi va *right* maydonga o'ng o'g'il tugunning xotiradagi adresi yoziladi. Informatsion maydonga esa ushbu elementning ixtiyoriy shakldagi ma'lumoti yoziladi.

Soddalik uchun, har bir elementi 3ta maydondan, ya'ni o'ng va chap o'g'il tugunlarga ko'rsatkichlar va kalit maydondan iborat tugunlardan tashkil topgan binar qidiruv daraxtini C++ tilida sinflardan foydalanib quyidagicha e'lon qilish mumkin:

```
class Node{
  public:
    int key;
    node *left;
    node *right;
};
```

Binar qidiruv daraxtiga yangi element kiritish funksiyasini quyidagicha tuzish mumkin:

```
int add(Node *&tree, Node *p){
    if(tree==NULL){
        tree=p; return 1;
    }
    else {
        if(tree->key>p->key) add(tree->left,p);
        else add(tree->right,p);
    }
}
```

Bu yerda *tree* – daraxt ildizi ko'rsatkichi, *p* – yangi element ko'rsatkichi. Ularni asosiy dastur kodida quyidagicha e'lon qilish mumkin:

```
Node *tree=NULL;
Node *p=new Node;
    p->key=k;
    p->left=p->right=NULL;
if(tree==NULL) tree=p;
    else add(tree,p);
```

Binar qidiruv daraxtini ko'rikdan o'tkazish algoritmi. Bu algoritm yordamida daraxt elementlarini ma'lum bir tartibda o'qib olish va zarur bo'lsa, ular ustida amal bajarish yoki ekranga chiqarish masalalarini ko'rish mumkin. Daraxt ko'ruvini 3 xil usuli mavjud:

1. ***To'g'ri (preorder traversal).*** Bunda daraxt yuqoridan pastga qarab o'qiladi. Dastlab ildiz tugun oq'iladi, keyin farzandlar. Farzand tugunlar o'qilganda dastlab chap tugun, uning avlodlari ham shu tartibda oq'ilib, keyin o'ng tugun va uning avlodlari ham shu tartibda oqiladi. 4.18-rasmdagi daraxtni to'g'ri ko'rikdan o'tkazganda 50, 46, 29, 48, 61, 55, 79 kabi ketma-ketlikda o'qiladi.
2. ***Teskari (postorder traversal).*** Bunda daraxt pastdan yuqoriga qarab o'qiladi. Dastlab eng pastki bosqichda joylashgan eng chapdagi element, keyin uning og'aynisi bo'lgan o'ng element va ularning otasi o'qiladi. Bu otaga og'ayni bo'lgan o'ng tomondagi qismdaraxt ham xuddi shu tartibda o'qiladi. Shu tartibda yuqoriga qarab davom ettirilib, oxirida ildiz o'qiladi. 4.18-rasmdagi daraxtni teskari ko'rikdan o'tkazilganda 29, 48, 46, 55, 79, 61, 50 kabi ketma-ketlikda o'qiladi.
3. ***Simmetrik (inorder traversal).*** Bu usulda daraxt chapdan o'ngga qarab ko'rikdan o'tkaziladi, ya'ni dastlab eng chapdagi element, keyin uning otasi va keyin o'ng o'g'il tugun o'qiladi. Simmetrik ko'ruvdan o'tkazganda elementlar o'sish tartibida kelib chiqadi. 4.18-rasmdagi daraxtni simmetrik ko'rikdan o'tkazilganda 29, 46, 48, 50, 55, 61, 79 kabi ketma-ketlikda o'qiladi.

Ushbu ko'rikdan o'tkazish algoritmlari dastur kodi quyida keltirilgan.

```
void printPostorder(Node* tree) {
  if (tree == NULL) return;
  printPostorder(tree->left);
  printPostorder(tree->right);
  cout << tree->key << " ";
}
void printInorder(Node* tree) {
  if (tree == NULL) return;
  printInorder(tree->left);
  cout << tree->key << " ";
  printInorder(tree->right);
}
void printPreorder(Node* tree) {
  if (tree == NULL) return;
  cout << tree->key << " ";
  printPreorder(tree->left);
  printPreorder(tree->right);
}
```

Binar qidiruv daraxtidan tugunni qidirish algoritmi. Daraxtdan tugunni qidirish algoritmi element qo'shish algoritmiga o'xshash bo'lib, qidiruv amali daraxt ildizidan boshlanadi.

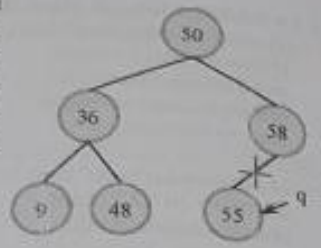
1. Qidirilayotgan element joriy tugun bilan solishtiriladi, agar teng bo'lsa, qidiruv to'xtatiladi va element topilgan hisoblanadi, aks holda 2-qadamga o'tiladi;
2. Qidirilayotgan element joriy tugundan kichik bo'lsa, qidiruv amali chap qismdaraxtda, aks holda o'ng qismdaraxtda davom ettirilishi uchun 1-qadamga o'tiladi.

Shu tarzda davom etib, terminal tugungacha qidiruv natija bermasa, daraxtda bunday element mavjud emas, degan xulosa chiqariladi. Quyidagi qidiruv funksiyasi kirish parametri *k* – qidirilayotgan element qiymati, *tree* - daraxt ildizi.

```
Node *search(Node *tree, int k){
    if(tree!=NULL){
        if(k==tree->info) {
            cout<<"\ndaraxtda ushbu el.t mavjud\n";
            return tree;
        }
        if(k<tree->info) search(tree->left, k);
        else search(tree->right, k);
    }
    else {
        cout<<"el.t mavjud emas\n";
        return 0;
    }
}
```

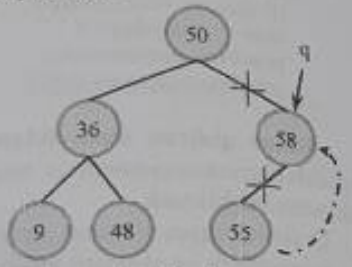
Binar qidiruv daraxtidan tugunni o'chirish algoritmi. Daraxtdan tugun o'chirilayotganda uning joylashgan o'rni inobatga olinishi kerak, sababi tugun o'chirilsa, uning o'g'il tugunlari qaerga joylashtirilishi hal qilinishi zarur. Bunda 3 xil holat bo'lishi mumkin:

1. O'chiriladigan element barg tugun bo'lsa, bu o'chirishning eng sodda usuli hisonlanadi. Bunda o'chiriladigan tugunni otasi bilan munosabati uzib tashlanadi va element joylashgan xotira sohasi tozalanadi. *q* – o'chirilayotgan tugun ko'rsatkichi, *otasi* – uning otasi ko'rsatkichi bo'lsin.




```
if(q==otasi->left) otasi->left=NULL;
else otasi->right=NULL;
delete(q);
```

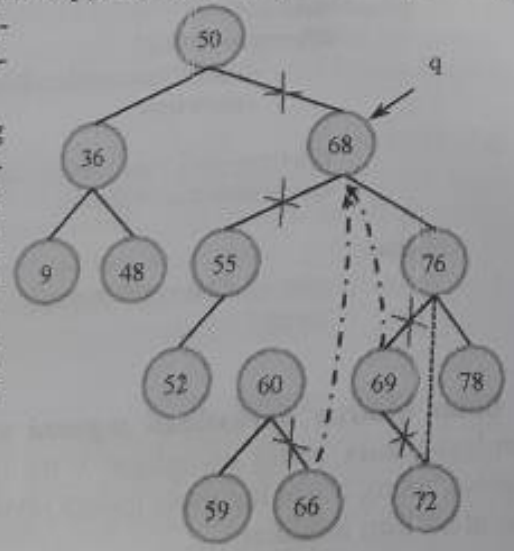
2. O'chiriladigan tugun 1 ta farzand tugunga ega bo'lsa, farzandi ota tugunni o'rnini egallaydi. Buning uchun o'chirilayotgan tugun otasi va farzandi bilan munosabati uziladi va ularni o'zari bir-biriga ulab qo'yiladi.




```
if(otasi->left==q){
    if(q->left!=NULL) otasi->left=q->left;
      else otasi->left=q->right;

}
else if(otasi->right==q){
      if(q->left!=NULL) otasi->right=q->left;
      else otasi->right=q->right;

}
    delete(q);
```

3. O'chiriladigan tugun 2 ta farzand tugunga ega bo'lsa, u holda daraxtda bu tugunni o'rniga qo'yiladigan konkret 2 ta voris tugun mavjud. Ularning ixtiyoriy birini o'chirilayotgan tugunni o'rniga qo'yish mumkin. Voris tugunlar quyidagicha topiladi:

- o'chiriladigan tugundan 1 qadam o'ngga, ya'ni o'ng qismdaraxtga o'tiladi va undagi eng kichik element olinadi, ya'ni eng chapdagi element tanlanadi;
- o'chiriladigan tugundan 1 qadam chapga, ya'ni chap qismdaraxtga o'tiladi va undagi eng katta element olinadi, ya'ni eng o'ngdagi element tanlanadi.



Agar tanlangan voris tugunning ham farzandi mavjud bo'lsa, 2-holatdagidek, farzand ota o'rnini egallaydi va bu tugun voris sifatida o'chirilayotgan tugun o'rniga joylashtiriladi. Quyidagi dastur kodida v – voris, s – vorisning farzandi, t – vorisning otasi deb belgilab olsak. Bu dastur kodida voris o'ng qismdaraxtdan tanlanadi.

```
Node *v=q, *s=v->right, *t=otasi;
    while(s!=NULL){
        t=v;
        v=s;
        s=s->left;
```



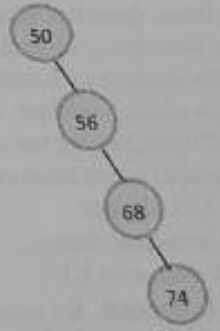
```
}
if(q!=t){
        t->left=v->right;
        v->right=q->right;

}
v->left=q->left;
if(q!=tree){
        if(q==otasi->left)
                otasi->left=v;
        else otasi->right=v;
        delete(q);

}
else{
        tree=v;
        delete(q);
        }
```

Tugunni o'chirish dasturini tuzishda, albatta, o'chirishga berilgan tugun ildiz bo'lishi ham mumkinligini inobatga olish zarur. Ushbu algoritmlarni bajarilishini internetda *https://www.cs.usfca.edu/~galles/visualization/BST.html* saytida vizual ko'rinishda kuzatishingiz mumkin.

Daraxt muvozanatlanganligi

Binar daraxtlarda elementlar kelib tushish ketma-ketligiga qarab daraxt ma'lum shaklga keladi. Binar daraxt balandligi uzaygan sari uning ustida amal bajarish qiyinlashib boradi. Binar daraxt ustida amal bajarish murak- kabligi uning balandigiga to'g'ri proporsonal. Daraxt shakllanganda, o'rtacha holatda uning samaradorligi O(log(n)) ga teng.

Agar daraxtning o'ng va chap qism daratlari bosqichlari va tugunlari soni (qismdaraxt vazni) bir xil bo'lsa, bunday daraxt *ideal muvozanatlangan daraxt* deyiladi. 4.18-rasmda hosil qilingan binar daraxt ideal muvozanatlangan daraxtga misol bo'ladi. Agar daraxtning har bir tuguni chap va o'ng qismdaraxti balandligi farqi 1 ga teng bo'lsa, bunday daraxtga muvozanatlangan daraxt deyish mumkin. Agar daraxt ixtiyoriy tuguni har ikkala qism daraxti balandligi farqi 1 dan katta bo'lsa, bunday daraxt *muvozanatlanmagan binar daraxt* deyiladi. Odatda, muvozanatlangan binar daraxtlar bilan ishlash samaradorligi $O(\log_2 N)$ ga teng. Ammo, tuzilmaga elementlar o'sish yoki kamayish tartibida kelib tushsa, daraxt bilan ishlash samaradorligi tushib ketadi va eng yomon holatda O(N) ga teng bo'lib qoladi, ya'ni eng yomon holatda bog'langan ro'yxat kabi samaradorlik bilan ishlaydi (4.20-rasm). Bunday hollarda binar daraxt bilan ishlash samaradorligini oshirish uchun uni muvozanatlash zarur. Buning bir nechta usullari mavjud.



4.20-rasm. O'sish tartibida kelib tushgan elementlardan hosil qilingan binar qidiruv daraxti

Binar qidiruv daraxtini muvozanatlanganlikka tekshirish protsedurasini ko'rib chiqamiz. Muvozanatlanganlikka tekshirish uchun har bir tugunning ikkala qismdaraxti balandliklarini hisoblash funksiyasini tuzib olamiz.

```
int height(Node *tree){
    if(tree==NULL) return 0;
    else{
        int l=height(tree->left);
        int r=height(tree->right);
        return (l>r)? l+1:r+1;
    }
}
bool isBalanced(Node *tree){
    if(tree==NULL) return 0;
    else{
        bool b=true;
        int l=height(tree->left);
        int r=height(tree->right);
        int k=l-r;
        if(k>1||k<-1){
            b=false;
            return b;
        }
        isBalanced(tree->left);
        isBalanced(tree->right);
        return b;
    }
}
```

Binar daraxtlar bilan ishlashda uni muvozanatlash zarur omil hisoblanadi, chunki daraxtning balandligi oshgan sari uning barg tugunlarini izlash va boshqa amallarni bajarishga ketadigan vaqt oshib boradi. Shu sababli daraxtlarni muvozanatlashga ehtiyoj seziladi. Binar daraxtlar bilan ishlaganda unga element qo'shish yoki o'chirishda uning muvozanatlanganligi buzilishi mumkin. Bunday hollarda uni muvozanatlab turish talab etiladi. Muvozanatlangan daraxt ko'rinishlari:

- AVL daraxt;
- Qora-qizil daraxt;
- B-daraxt va h.k.

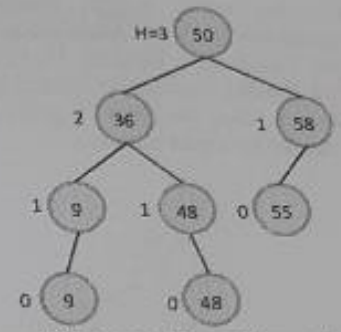
AVL daraxti. Bu muvozanatlangan binar qidiruv daraxti bo'lib, 1962 yilda rus olimlari Georgiy Maksimovich Adelson – Velskiy va Yevgeniy Mixaylovich Landis tomonidan taklif etilgan. Bunda daraxtni qurish yoki tugunlarni o'chirish jarayonida daraxt muvozanatlanganlikka tekshirilib turiladi va muvozanatlik buzilgan qismdaraxtda ma'lum algoritm bilan muvozanatlash amali bajariladi. Natijada binar daraxt bilan ishlash samaradorligi oshadi. AVL daraxtda har bir tugunning muvozanatlanganlik darajasi haqidagi ma'lumotni saqlash uchun har bir tugunga qo'shimcha maydon kiritiladi.

Asosiy g'oya: agar element qo'shish, o'chirish amallarida muvozanat buzilsa, u holda daraxt muvozanatlanadi.

Bu usulni ko'rib chiqish uchun quyidagi tushunchalarni kiritib o'tamiz.

1. Tugun balandligi (H) - bu undan eng uzoq joylashgan barg tugungacha bo'lgan yo'l uzunligi hisoblanadi. Bu haqda yuqorida to'xtalib o'tilgan edi. AVL daraxtida barg tugunning balandligi 0 ga va bo'sh qismdaraxtning balandligi -1 ga teng. Masalan, 4.21-rasmda daraxt tugunlari balandliklari keltirilgan.
2. Tugunning muvozanatlanganlik darajasi (balance factor) - bu tugunning chap va o'ng qismdaraxtlari balandliklari farqi.

Balance(node)=H(left)-H(right)

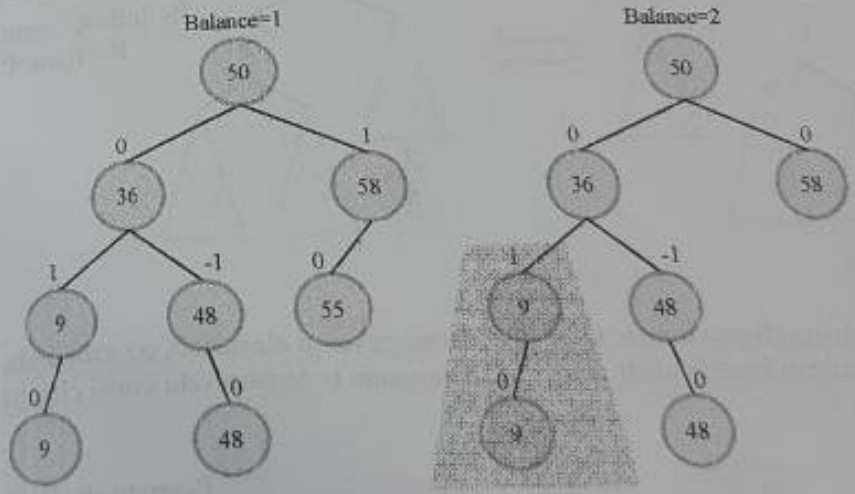


4.21-rasm. Binar qidiruv daraxti tugunlari balandliklari

AVL daraxtda har bir tugunning muvozanatlanganlik darajasi (-1, 0, 1) to'plamdan qiymat qabul qiladi (4.22-rasm).

Balance(9)=0-(-1)=1

Daraxtga yangi element qo'shilgach, tugunlarning muvozanatlanganlik darajalari qayta ko'rib chiqiladi. Agar bironta tugunning bu parametri 2 yoki undan katta bo'lsa (yoki -2 ga teng yoki undan kichik bo'lsa) u holda bu qismdaraxtni soat yo'nalishi yoki unga qarshi yo'nalishda burish usuli bilan muvozanatlash kerak.



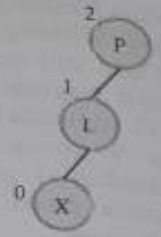
a b

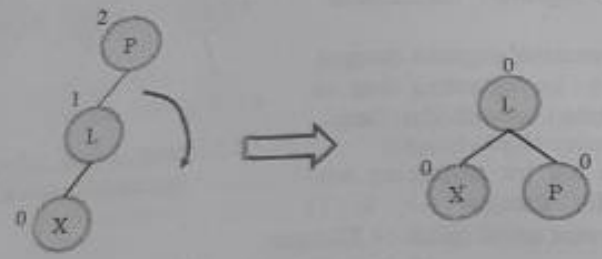
4.22-rasm. Tugunlar muvozanatlanganlik darajasi;
a-AVL daraxt, b-AVL daraxt emas

Muvozanatlanganlikning buzilishiga qarab, burib muvozanatlash algoritmining bir nechta turi qo'llaniladi.

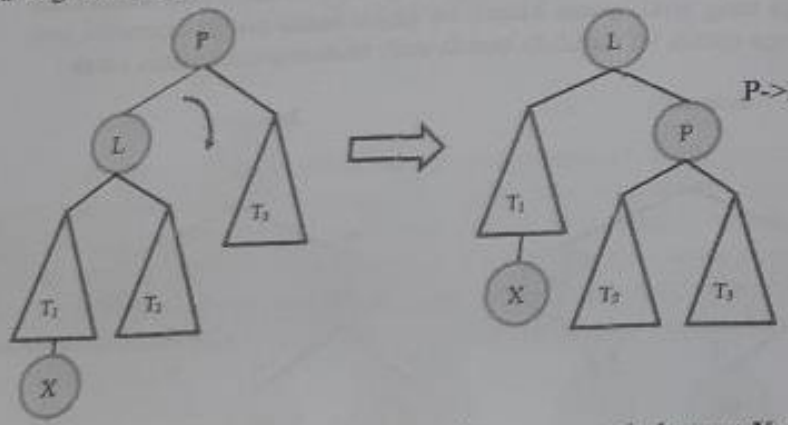
– R – bir marta o'ngga burish; tasavvur qiling, chap qismdaraxt- ga X qo'shildi. P tugunning muvozanatlanganligini tekshiramiz: H(Left)=1 > H(Right)=-1

O'ng (bo'sh) qismdaraxt balandligini oshirish talab etiladi. Ildiz va chap tugunni bog'lovchi yoyni o'ngga, soat yo'nalishi bo'yicha buramiz.



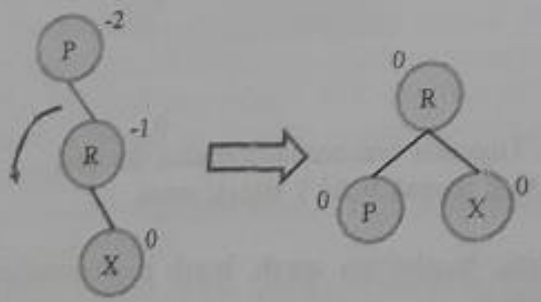


Agar L tugunni chap o'gil tuguni mavjud bo'lsa, quyidagicha muvozanatlash amalga oshiriladi:



P->left=R->right;
R->right=P;

– L – bir marta chapga burish; O'ng qismdaraxtga yangi element X qo'shilganda, ushbu usuldan foydalaniladi. Ildiz va o'ng tugunni birlashtiruvchi yoyni chapga buramiz.

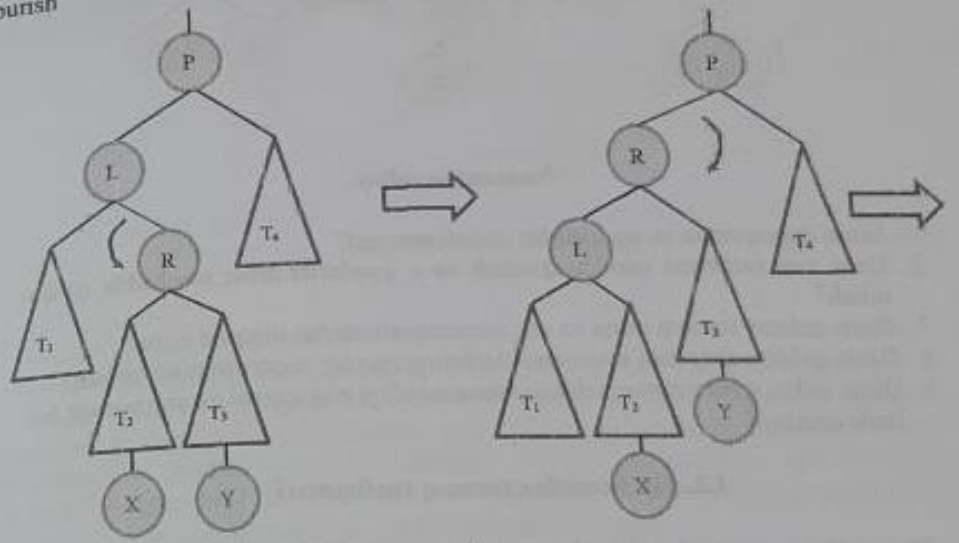


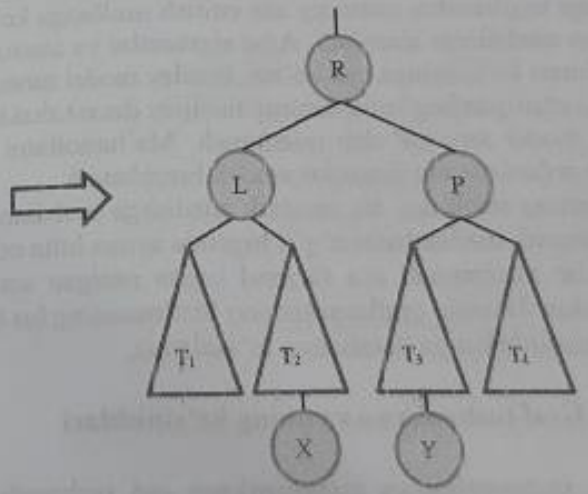
P->right=R->left;
R->left=P;

– LR – oldin chapga, keyin o'ngga burish; ushbu usul ildiz chap tugunining o'ng qismdaraxtiga yangi element kiritilganda qo'llaniladi.

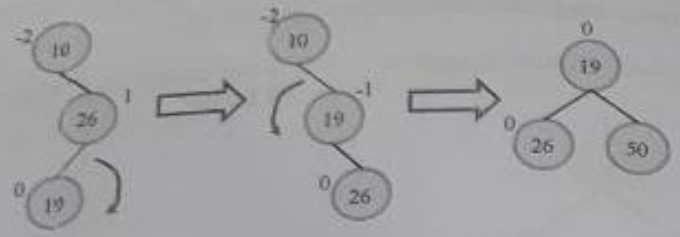
1. P ning chap qismini chapga L burish

2. Yangi daraxtning P tugunini o'ngga R burish





– RL – oldin o'ngga, keyin chapga burish. Bu usul o'ng qismdaraxtning chap tomoniga yangi element kiritilganda qo'llaniladi.



Nazorat savollar

1. Binar daraxt nima va uni qanday turlari mavjud?
2. Heap tree tuzilmasi qanday quriladi va u qaerlarda nima maqsadda qo'llaniladi?
3. Binar qidiruv daraxti nima va uni muvozanatlashdan maqsad nima?
4. Binar qidiruv daraxtini muvozanatlashning qanday algoritmlarini bilasiz?
5. Binar qidiruv daraxtining ishlash samaradorligi eng yaxshi va eng yomon holatda qanday?

4.2. Ma'lumotlar tarmoq tuzilmalari

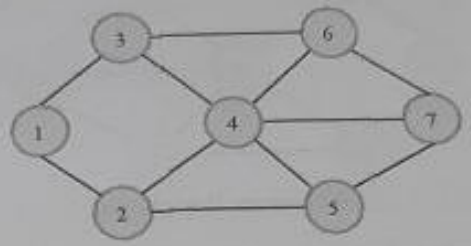
Elementlar orasidagi bog'lanishni mantiqiy aks ettirish usullariga ko'ra: *ierarxik, tarmoq* va *relyatsion* modellarga ajratiladi. Agar elementlar va ularning o'zaro munosabatlari graf tuzilmasi ko'rinishiga ega bo'lsa, bunday model *tarmoq modeli* deb nomlanadi. Agar aks ettirilgan bog'lanishlarning tuzilishi daraxt shaklida ifodalangan bo'lsa, u holda model *ierarxik* deb nomlanadi. Ma'lumotlarni jadvallar ko'rinishida taqdim etish *relyatsion ma'lumotlar modeli* hisoblanadi.

Ma'lumotlarning tarmoq tuzilmasi bu ierarxik tuzilishga nisbatan kengroq tushuncha hisoblanadi. Ierarxik tuzilmalarda o'g'il tugunda aynan bitta ota bo'lishi kerak; tarmoq ma'lumotlar tuzilmasida esa farzand tugun istalgan sondagi ota tugunga ega bo'lishi mumkin. Demak, graflarni tarmoq tuzilmasining bir ko'rinishi deyish mumkin. Graflar misolida bunga batafsilroq to'xtalamiz.

4.2.1. Graf tushunchasi va uning ko'rinishlari

Graf tushunchasi matematika va informatikaga oid tushuncha bo'lib, murakkab obyektlarning xususiyatlari va munosabatlarini o'zida aks ettiruvchi murakkab, chiziqsiz, ko'pbog'lamli, dinamik tuzilma hisoblanadi. Graf — bu tugunlar va ular orasidagi munosabatlarni ifodalovchi tuzilma hisoblanadi va *G(V,E)* kabi ifodalanadi. Bu yerda *V* – graf tugunlari (uchlari) to'plami, *E* esa – qirralar (yoylar) to'plami hisoblanadi, u va v tugunlar orasidagi qirraning matematik ifodasi (u,v) ko'rinishida beriladi. Graflar nazariyasi va uning sonli xarakteristkalarini

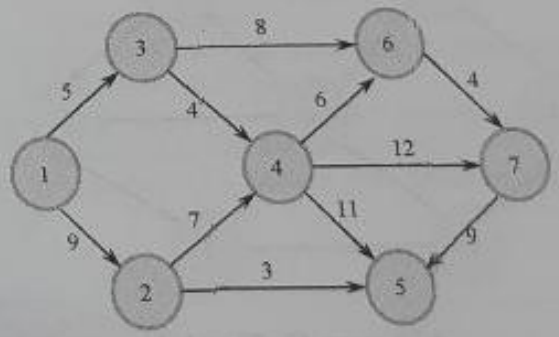
o'rganish orqali dinamik tizimlarni optimal boshqaruv masalalarini, geografik, iqtisodiy va bir qator yo'nalishdagi masalalarni yechish, o'yin dasturlarini tuzishda foydalanish mumkin.



4.23-rasm. Graf tuzilmasi

$G = \{V, E\}$
$V = \{1, 2, 3, 4, 5, 6, 7\}$
$E = \{(1,2), (1,3), (2,4), (2,5), (3,4), (3,6), (4,5), (4,6), (4,7), (5,7), (6,7)\}$

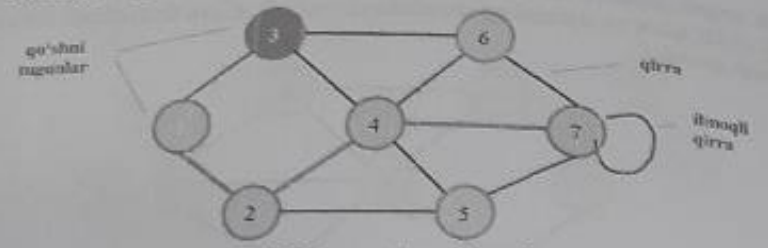
Graflar yo'naltirilgan va yo'naltirilmagan bo'lishi mumkin. Tugunlar orasidagi munosabatlar y'onalishga ega bo'lsa, ya'ni bir tomonlama bo'lsa, ***yo'naltirilgan graflar*** deyiladi. Grafning qirralari yo'nalishga ega bo'lmasa, ***yo'naltirilmagan graf*** deyiladi. 4.23-rasmdagi graf yo'naltirilmagan grafga misol bo'la oladi. Quyida yo'naltirilgan grafga misol keltiramiz:



4.24-rasm. Yo'naltirilgan va vazngа ega graf

Yo'naltirilgan ushbu grafda (1,2) va (2,1) munosabatlar teng emas. Ba'zi hollarda graflar qirralarning vazni (og'irligi) – w_{ij} ((i,j) qirraning og'irligi) bilan ham berilishi mumkin. Bunday grafga vazngа ega bo'lgan graf (weighted graph) deyiladi. Qirraning vazni aynan nima ma'noni anglatishi masalaning qo'yilishida beriladi, misol uchun yo'llarning tirbandlik darajasi, yuk tashish sarf-xarajati, punktlar orasidagi masofa yoki ketadigan vaqt bo'lishi ham mumkin.

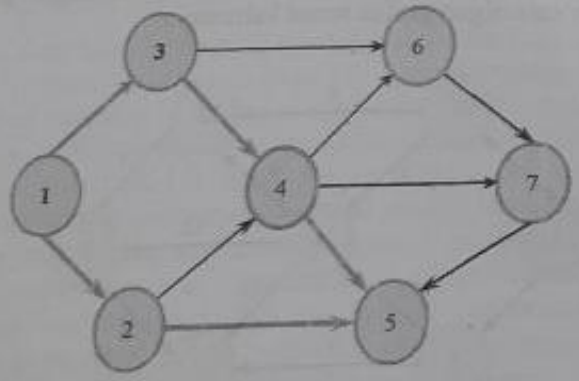
Graflarga tegishli tushunchalarni kiritib o'tamiz.



4.25-rasm. Ilmoqli graf

Qirraning har ikkala uchidagi tugunlar **qo'shni tugunlar** hisoblanadi, ya'ni o'zari bitta qirra bilan tutashgan tugunlar qo'shni tugunlar deyiladi. Qirraning har ikkala uchi aynan bir tugunga tutashgan bo'lsa, bunday qirraga **ilmoqli qirra** deyiladi (4.25-rasm). *V* to'plamning quvvati *n* ga teng bo'lsa, *n* soni **grafning tartibi** deyiladi. **Tugun darajasi** (vertex degree) – bu undan chiquvchi yoylar soni hisoblanadi. deg(7) = 3, deg(1) = 2. Tugunlar darajasiga nisbatan **juft yoki toq deyiladi**, agar ularning darajalari mos ravishda juft yoki toq qiymatga teng bo'lsa.

Yo'l – grafning 1 ta tugunidan boshqa tugunigacha bo'lgan tugunlar ketma-ketligiga aytiladi va (1,2,4,7) kabi ifodalanadi (4.25-rasm, qizil rang bilan ajratilgan).

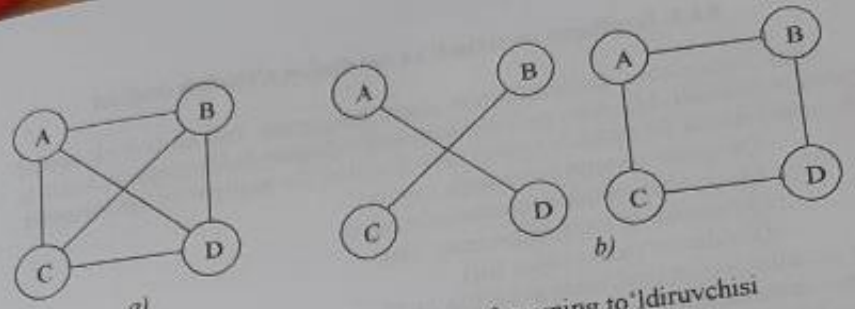


4.26-rasm. Grafdagi halqa

Halqa (cycle) – bu boshi va oxiri ayni bir tugunda tutashuvchi yo'l hisoblanadi: (1, 2, 5, 4, 3, 1) (4.26-rasm, qizil rangda)

To'liq graf (complete graph) – bu tugunlar orasida barcha bo'lishi mumkin bo'lgan munosabatlar o'rnatilgan graf hisoblanadi (4.27,a -rasm)

Grafni to'ldiruvchisi bu qaralayotgan grafni mavjud tugunlaridan iborat bo'lgan va to'liq graf bo'lishi uchun yetishmayotgan qirralardan tashkil topgan grafga aytiladi (4.27-rasm).



4.27-rasm.a) to'liq graf b) graf va uning to'ldiruvchisi

To'liq, yo'naltirilmagan grafda qirralar soni m quyidagi (1) formula orqali aniqlanadi:

$$m = \frac{n(n-1)}{2}. \quad (1)$$

Bu yerda n –tugunlar soni.

Grafning to'yinganligi D (density) deb quyidagi (2) formula bilan aniqlanuvchi parametrga aytiladi:

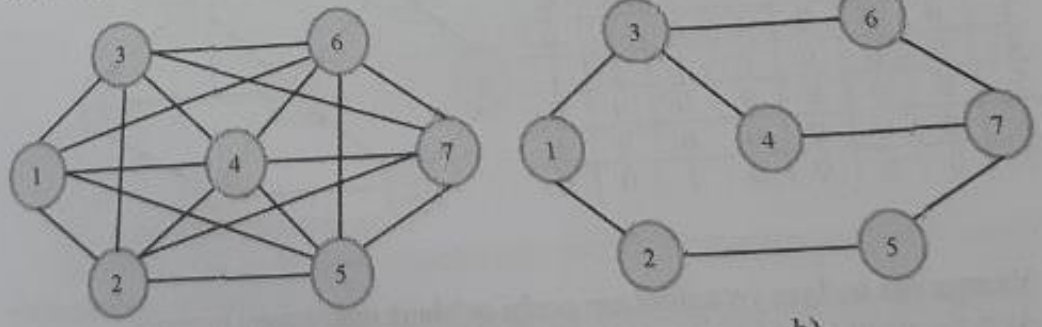
$$D = \frac{2m}{n(n-1)}. \quad (2)$$

Bu yerda n,m – (1) formuladagi kabi.

Grafning to'yinganlik koeffitsientiga qarab ikki xil graf ko'rinishini aniqlash mumkin: to'yingan graf va siyrak graf (4.28-rasm).

To'yingan graf (dense graph) – bu qirralar soni bo'lishi mumkin bo'lgan maksimalga yaqin bo'lgan graf hisoblanadi, ya'ni (D>0.5).

Siyrak graf (sparse graph) – bu qirralari soni tugunlar soniga yaqin bo'lgan grafdir, ya'ni (D<0.5).



4.28-rasm.a) to'yingan graf (D=0.76) b) siyrak graf (D=0.38)

4.2.2. Graflarni tasvirlash va ko'rikdan o'tkazish usullari

Graf masalalarini dasturlashtirish ularni kompyuter xotirasida fizik tashkil etishdan boshlanadi. Masalani qo'yilishidan kelib chiqqan holda graflarni xotirada statik yoki dinamik ko'rinishda tashkil etish mumkin. Bu usullarni ko'rib chiqamiz.

- Qo'shma matritsa (adjacency matrix);
- Munosabat matritsa (incidence matrix);
- Qo'shnilik ro'yxati (adjacency list);
- Qirralar ro'yxati (edges list).

Yo'naltirilmagan graf berilgan bo'lsin (4.23-rasm). Grafdagi tugunlar soni n=7. Qo'shma matritsa graf tugunlari soni n dan kelib chiqqan holda, A(n,n) kvadrat Matritsa ko'rinishida tasvirlanadi. Unda har bir satr va ustun tugunlarga tegishli bo'lib, tugunlar orasida munosabat mavjud bo'lsa, 1, aks holda 0 qo'yiladi. Ya'ni:

$A_{ij} = 1$, agar i va j tugunlar qirra bilan birlashtirilgan bo'lsa;

$A_{ij} = 0$, agar i va j tugunlar o'rtasida qirra mavjud bo'lmasa.

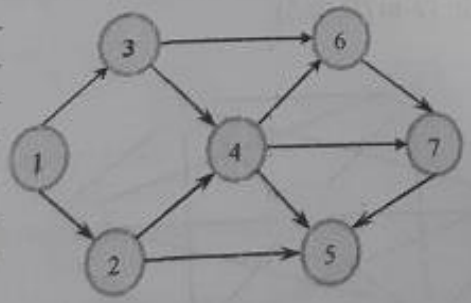
V	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0
2	1	0	0	1	1	0	0
3	1	0	0	1	0	1	0
4	0	1	1	0	1	1	1
5	0	1	0	1	0	0	1
6	0	0	1	1	0	0	1
7	0	0	0	1	1	1	0

Yo'naltirilgan grafni qo'shma matritsasi esa quyidagicha tashkil etiladi:

$A_{ij} = 1$, agar qirra i tugundan j tugunga yo'naltirilgan bo'lsa;

$A_{ij} = 0$, agar qirra i dan j tugunga yo'naltirilmagan bo'lsa

V	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0
2	0	0	0	1	1	0	0
3	0	0	0	1	0	1	0
4	0	0	0	0	1	1	1
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1
7	0	0	0	0	1	0	0

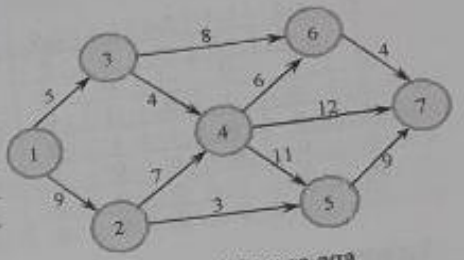


Vazнga ega bo'lgan yo'naltirilgan grafni qo'shma matritsasini tuzamiz.

$A_{ij} = w_{ij}$, ya'ni i dan j tugunga yo'naltirilgan qirra vazni yoziladi;

$A_{ij} = \infty$, agar i dan j tugunga qirra yo'naltirilmagan bo'lsa.

V	1	2	3	4	5	6	7
1	∞	9	5	∞	∞	∞	∞
2	∞	∞	∞	7	3	∞	∞
3	∞	∞	∞	4	∞	8	∞
4	∞	∞	∞	∞	11	6	12
5	∞	∞	∞	∞	∞	9	∞
6	∞	∞	∞	∞	∞	∞	4
7	∞	∞	∞	∞	∞	∞	∞



4.29-rasm. Vazнga ega bo'lgan yo'naltirilgan graf

Yo'naltirilmagan graflarning qo'shma matritsasi asosiy diagonaliga nisbatan semmitrik bo'ladi va diagonalda joylashgan elementlar 0 lardan iborat bo'ladi. Qo'shma matritsaning qulaylik tomonlari quyidagilar:

- qirra(yoy) qo'shish va o'chirish oson;
- tugunlar qo'shniligini tekshirish.

Qo'shma matritsaning noqulayliklari esa quyidagicha:

- tugunlarni kiritish yoki o'chirish;
- siyrak graflar bilan ishlash.

G grafning **munosabat matritsasi** bu n ta satr(tugunlar soni) va m ta ustunlar (qirralar soni) dan tashkil topgan B matritsa bo'lib, yo'naltirilmagan graf uchun quyidagicha hosil qilinadi:

$B_{ik} = 1$, agar i tugun k qirra bilan to'qnashgan bo'lsa;

$B_{ik} = 0$, agar i tugun k qirra bilan to'qnashmagan bo'lsa.

Misol uchun,4.1-rasmdagi grafning munosabat matritsasini keltiramiz.

V	1-2	1-3	2-4	3-4	2-5	3-6	4-5	4-6	4-7	6-7	5-7
1	1	1	0	0	0	0	0	0	0	0	0
2	1	0	1	0	1	0	0	0	0	0	0
3	0	1	0	1	0	1	0	0	0	0	0
4	0	0	1	1	0	0	1	1	1	0	0
5	0	0	0	0	1	0	1	0	0	0	1
6	0	0	0	0	0	1	0	1	0	1	0
7	0	0	0	0	0	0	0	0	1	1	1

Yo'naltirilgan graflarda esa:

$B_{ik} = -1$, $B_{jk} = 1$, agar qaralayotgan k-qirra i-tugundan j-tugunga tomon yo'naltirilgan bo'lsa;

$B_{ik} = 0$, agar i tugun k-qirra bilan to'qnashmagan bo'lsa.

Masalan, 4.4-rasmdagi yo'naltirilgan grafning munosabat matritsasi quyidagicha hosil qilinadi:

F	1-2	1-3	2-4	3-4	2-5	3-6	4-5	4-6	4-7	6-7	5-7
1	-1	-1	0	0	0	0	0	0	0	0	0
2	1	0	-1	0	-1	0	0	0	0	0	0
3	0	1	0	-1	0	-1	0	0	0	0	0
4	0	0	1	1	0	0	-1	-1	-1	0	0
5	0	0	0	0	1	0	1	0	0	0	0
6	0	0	0	0	0	1	0	1	0	-1	0
7	0	0	0	0	0	0	0	0	1	1	-1

Yo'naltirilgan graf vazn bilan birga berilgan bo'lsa, uni quyidagicha tashkil etiladi:

$B_{ik} = \pm w_{ik}$, agar i tugun k yoy boshi(oxiri) bo'lsa;

$B_{ik} = 0$, agar i tugun k yoy bilan to'qnashmagan bo'lsa.

Munosabat matritsasining qulaylik tomonlari:

- Qirra(yoy) o'lchamini yoki yo'nalishini o'zgartirish;
- Qirra(yoy)larni qo'shish yoki o'chirish;
- To'qnashuv(intsidentlik)ni tekshirish.

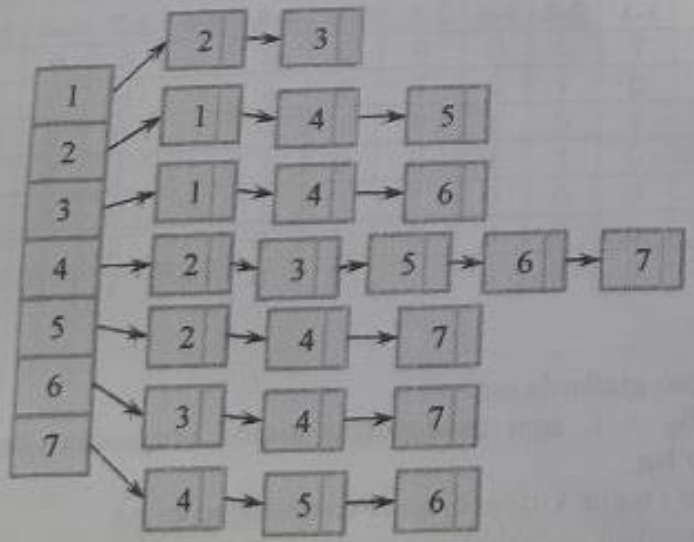
Munosabat matritsasining noqulayliklari esa quyidagilar:

- Tugunlarni qo'shish yoki o'chirish;
- Siyrak graflar bilan ishlash.

Qo'shnilik(qo'shni tugunlar) ro'yxati – bu A[n] massiv bo'lib, A[i] har bir elementi i tugun bilan qo'shni tugunlar ro'yxatini o'zida saqlaydi.

Qo'shnilik ro'yxati qulaylik tomonlari quyidagilar:

- joriy (berilgan) tugunga qo'shni tugunni izlash;
- tugun yoki qirra(yoy)larni qo'shish;
- siyrak graflar bilan ishlash.



4.30-rasm. Grafni qo'shnilik ro'yxati ko'rinishida ifodalanishi

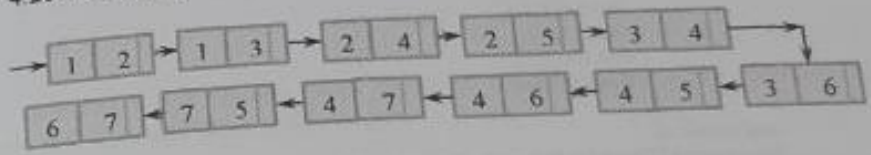
Qo'shnilik ro'yxati noqulayliklari esa quyidagicha:

- qirra(yoy)ning mavjudligini tekshirish;
- tugun yoki qirra(yoy)larni o'chirish.

Yo'naltirilgan grafning qo'shnilik ro'yxatida esa qirraning biror tugundan boshqasiga yo'naltirilganiga ko'ra ularni qo'shniligi belgilanishi mumkin, aks holda, ya'ni teskari yo'nalishga ega qirra bilan bog'langan qirralar qo'shni sifatida belgilanmaydi. Agar grafda qirralar vazni bilan berilsa, qo'shnilik ro'yxatida bog'langan ro'yxat elementlariga vazn haqida ma'lumot saqlash uchun qo'shimcha maydon kiritiladi.

Qirralar ro'yxati – qirralarning qo'shni tugunlar juftliklaridan iborat chiziqli ro'yxatdir.

4.26-rasmdagi grafning qirralar ro'yxati quyidagicha tashkil etiladi.



Yo'naltirilgan grafning qirralar ro'yxatida esa elementlardagi maydonlarga tugunlar qirraning yo'nalishi bo'yicha ketma-ketlikda yoziladi. Vazn berilsa, u haqidagi ma'lumotni yozish uchun ro'yxat elementlariga qo'shimcha maydon kiritiladi.

Qo'shnilik(qo'shni tugunlar) ro'yxati qulaylik tomonlari quyidagilar:

- qirra(yoy)larni qo'shish yoki o'chirish;
- yoylarning yuklanishi bo'yicha tartiblash;
- siyrak graflar bilan ishlash.

Qo'shnilik(qo'shni tugunlar) ro'yxati noqulayliklari esa quyidagicha:

- tugun va qirra(yoy)ning qo'shniligini aniqlash;
- berilgan tugunga insident qirra(yoy)larni tanlash.

Graflarni ko'rikdan o'tkazish (Graph traversal) orqali tuzilma elementlarini o'qib olish va ekranga chiqarish amallarini bajarish mumkin. Graf elementlari takror ko'rikdan o'tkazilmasligi uchun tashrif buyurilgan elementlarni bironta tuzilmada saqlab turish kerak bo'ladi. Buning uchun navbat yoki steklardan foydalanish mumkin. Demak, graflarni ko'rikdan o'tkazishning 2 xil usuli mavjud bo'lib,

- navbat tuzilmasi ishlatilsa, graflar eniga qarab ko'rikdan o'tkaziladi (Depth-First Search – DFS);
- stek ishlatilsa, graflar tubiga qarab ko'rikdan o'tkaziladi (Breadth-First Search – BFS).

Har ikkala holda ham, quyidagi algoritm bilan ko'rikdan o'tkaziladi:

1. grafni bironta tuguni o'qib olinadi, tashrif buyurilgan element sifatida belgilab qo'yiladi va navbat yoki stek tuzilmasiga joylashtiriladi.
2. navbat/stek tuzilmasi chiqishidagi element chiqarib olinadi va ekranga uzatiladi.

3. ekranga uzatilgan tugunning qo'shni tugunlari oldin tashrif buyurilmagan bo'lsa, o'qib olinadi va navbat/stek tuzilmasiga uzatiladi, ular tashrif buyurilgan tugun sifatida belgilab qo'yiladi.
3. agar navbat/stek tuzilmasi bo'sh bo'lmasa, 2-qadamga o'tiladi, ask holda algoritm tugatiladi.

4.1-rasmdagi grafni stek yordamida tubiga qarab ko'rikdan o'tkazish dastur kodini keltiramiz.

```
#include<iostream>
#include<stack>
using namespace std;
int main(){
    int a[7][7]={ 0,1,1,0,0,0,0,
                  1,0,0,1,1,0,0,
                  1,0,0,1,0,1,0,
                  0,1,1,0,1,1,1,
                  0,1,0,1,0,0,1,
                  0,0,1,1,0,0,1,
                  0,0,0,1,1,1,0,
                };
    stack<int> st;
    bool visited [7]={false,false,false,false,false,false,false};
    st.push(0);
    visited [0]=true;
    while(st.empty()!=true){
        int k=st.top();
        cout<<k+1<<" ";
        st.pop();
        for(int j=0;j<7;j++)
            if(a[k][j]==1&&visited[j]==false) {
                st.push(j);
                visited[j]=true;
            }
    }
}
```

Dastur natijasi:

1 8 3 6 7 5 4 2

4.2.3. Eng qisqa yo'lni aniqlash algoritmlari

Graflarda eng qisqa yo'lni aniqlash masalasi juda muhim va ko'p ishlatiladigan masala bo'lib, ushbu masalaning qo'yilishini ham bir necha turi mavjud:

- Ikkita tugun orasidagagi eng qisqa masofani aniqlash masalasi (single-pair shortest path problem). S tugundan d tugungacha bo'lgan eng qisqa yo'lni aniqlash talab etiladi.
- Berilgan tugundan barcha tugunlargacha bo'lgan eng qisqa yo'llarni aniqlash masalasi (single-source shortest path problem).

- Berilgan punktga yetib borishning qisqaroq yo'lini aniqlash masalasi (single-destination shortest path problem). Grafning barcha tugunlaridan V tugunga yetib borishning eng qisqa yo'llarini aniqlash talab etiladi.
- Barcha o'zaro tugunlar orasidagi qisqa masofani aniqlash masalasi (all-pairs shortest path problem). Har bir u tugundan har bir v tugungacha eng qisqa yo'llarni aniqlash masalasi.

Masalaning formal qo'yilishi:

$G = (V, E)$ og'irlikka ega bo'lgan graf berilgan. Har bir E(i, j) yoyning og'irligi berilgan - w_{ij}. Boshlang'ich tugun $s \in V$ va oxirgi tugun $d \in V$ berilgan. Ular orasidagi eng qisqa masofali yo'lni aniqlash talab etiladi. Yo'l uzunligi (path length, path cost, path weight) – unga kiruvchi yoylar og'irliklari yig'indisiga teng (3).

$$L = \sum w_{ij} \qquad (3)$$

Grafning turiga va eng qisqa masofani aniqlash masalasining qo'yilishiga qarab, turlicha algoritmlar taklif etilgan.

Algoritm	g'oyasi
Deykstra algoritmi	Grafning bitta tugunidan barcha tugunlargacha bo'lgan qisqa yo'llarni aniqlash algoritmi. Ushbu algoritm yo'naltirilmagan va vazni manfiy bo'lmagan graflarda qo'llaniladi. (w(i,j)>=0)
Bellman-Ford algoritmi	Vaznga ega bo'lgan grafning bitta tugunidan barcha tugunlargacha bo'lgan eng qisqa yo'llarni aniqlash algoritmi. Yoylar vazni manfiy bo'lishi mumkin.
A*ni izlash algo-ritmi	Bironta tugundan ikkinchi bir tugungacha eng kam sarf-xarajatga ega bo'lgan marshrutni aniqlash algoritmi.
Floyd-Warshell algoritmi	Vaznga ega bo'lgan yo'naltirilgan graflarda tugunlararo eng qisqa yo'llarni aniqlash algoritmi
Djonson algoritmi	Vaznga ega bo'lgan yo'naltirilgan graflarda barcha juftliklararo eng qisqa masofalarni aniqlash algoritmi
Li algoritmi (to'lqinli algoritm)	Grafning s va t tugunlari orasidagi eng kam to'siqqa (oradagi tugun) uchraydigan masofani aniqlash algoritmi

Algoritmlarning bajarilish vaqti graf tugunlari va qirralar soniga bog'liqligi quyidagi jadvalda keltirilgan:

Algorithm	Runtime
Bellman-Ford	$O(\|V\|\cdot\|E\|)$
Deykstra (to'yxat ko'rinishida tasvirlaganda)	$O(\|V\|^2)$
Topological Sort	$O(\|V\|+\|E\|)$
Floyd-Warshall	$O(\|V\|^3)$
Johnson	$O(\|E\|\cdot\|V\|+\|V\|^2\cdot\log_2(\|V\|))$

Eng qisqa masofani aniqlashning mashxur algoritmlardan biri Deykstra algoritmini ko'rib chiqamiz.

Deykstra algoritmi *manfiy vaznga ega bo'lmagan graflarda* bir tugundan boshqa tugunlargacha bo'lgan eng qisqa masofalarni aniqlashga yordam beradi.

Algoritm qadamlari:

1. Kerakli tugun tanlab olinadi va unga belgi sifatida 0, qolgan barcha tugunlarga '∞' belgisi beriladi. Barcha tugunlar hali tashrif buyurilmagan deb olinadi.

2. Tanlangan tugundan chiquvchi yoylarni hisobga olgan holda unga qo'shni tugunlargacha bo'lgan masofalar qaraladi. Bunda qo'shni tugun belgisi va ungacha bo'lgan masofa taqqoslanadi va kichik qiymat qo'shni tugun belgisiga o'zlashtiriladi. Qo'shni tugungacha bo'lgan masofa esa tanlangan tugun belgisi va qo'shni tugunga olib boruvchi qirraning vazni yig'indisiga teng.

$$D[v] = min(D[v], D[w] + C[w, v])$$

3. Qisqa yo'lning navbatdagi elementi sifatida qo'shni tugunlardan belgisi eng kichigini tanlaymiz.

4. Barcha tugunlar bir marotaba tashrif buyurilmaguncha, 2- va 3-qadamlar takrorlanadi. Barcha tugunlar belgisi doimiy bo'lgan payt, ya'ni '∞' dan farqli qiymatga ega bo'lganda, algoritm o'z nihoyasiga yetadi.

Deykstra algoritmi bo'yicha 1-tugundan qolgan barcha tugunlarga bo'lgan qisqa masofani aniqlash masalasini 4.8-rasmdagi graf misolida ko'rib chiqamiz.

Tugunlar	belgisi	1-qadam	belgisi	2-qadam	belgisi	3-qadam	belgisi
1	0		0		0		0
2	∞	***1->2: 0+9=9***	9		9		9
3	∞	***1->3: 0+5=5***	5		5		5
4	∞			***1->3->4: 5+4=9***	9		9

				1->2->4: 9+7=16			
5	∞			***1->2->5: 9+3=12***	12	1->3->4->5: 9+11=20	12
6	∞			***1->3->6: 5+8=13***	13	1->3->4->6: 9+6=15	13
7	∞					1->2->5->7: 12+9=21	17
						1->3->4->7: 9+12=21	
						1->3->6->7: 13+4=17	

Deykstra algoritmining ishlash tezligi:

- qo'shnilik matritsasi bilan ishlaganda $O(|V|^2)$;
- qo'shnilik ro'yxati bilan ishlaganda $O(|E|*\log|V|)$ ga teng.

Deykstra algoritmining asosiy kamchiligi manfiy vaznga ega graflarda ishlay olmaydi.

Qaralayotgan grafda eng qisqa masofani aniqlashning Deykstra algoritmi dastur kodini keltiramiz.

```
#include<iostream>
int dijkstra(int a[7][7], int start){
    int dis[7]={INT_MAX,INT_MAX,INT_MAX,INT_MAX,INT_MAX,INT_MAX,
                INT_MAX };
    bool visited[7]={false, false, false, false, false, false, false};
    dis[start]=0;
    for(int count=0;count<6;count++){
        int min=INT_MAX, u;
        for(int i=0;i<7;i++)
            if(!visited[i]&&min>dis[i]){
                min=dis[i];
                u=i;
            }
        visited[u]=true;
        for(int i=0;i<7;i++){
            if(a[u][i]&&!visited[i]&&dis[u]+a[u][i]<dis[i])
                dis[i]=dis[u]+a[u][i];
        }
    }
    for(int i=0;i<7;i++)
        if(i!=start){
            if(dis[i]==INT_MAX) cout<<start+1<<"->"<<i+1<<" : "<<"yul yuq"<<endl;
            else cout<<start+1<<"->"<<i+1<<" : "<<dis[i]<<endl;
        }
}
int main(){
    int a[7][7]={
```

```
        { 0, 9, 5, 0, 0, 0, 0 },
        { 0, 0, 0, 7, 3, 0, 0 },
        { 0, 0, 0, 4, 0, 8, 0 },
        { 0, 0, 0, 0, 11, 6, 12 },
        { 0, 0, 0, 0, 0, 9, 0 },
        { 0, 0, 0, 0, 0, 0, 4 },
        { 0, 0, 0, 0, 0, 0, 0 }
    };
    int start; cout<<"boshlangich tugunni kiriting="; cin>>start;
    dijkstra(a,start-1);
    return 0;
}
```

Dastur natijasi:

```
boshlangich tugunni kiriting=1
1->2 : 9
1->3 : 5
1->4 : 9
1->5 : 12
1->6 : 13
1->7 : 17

--------------------------------
```

Dastur natijasi ham yuqoridagi jadvalda olingan natijalar bilan aynan mos keladi.

Bellman-Ford algoritmi

Bellman-Ford Algoritmi ham bir tugundan boshqa barcha tugunlargacha bo'lgan eng qisqa masofalarni topish algoritmi bo'lib, Deykstra algoritmidan farqi u manfiy vaznga ega bo'lgan graflar bilan ham ishlash imkonini beradi. Bellman-Ford algoritmi qadamlari:

1. Eng avvalo, berilgan grafdagi qirrallar jadvalini tuzamiz va ularning vaznlarini yozib qo'yamiz.

2. Berilgan tugun belgisiga 0 ni, qolgan barcha tugunlar belgisiga '∞' ni o'zlashtiramiz.

3. Eng qisqa masofani saqlovchi D va eng qisqa yo'lni saqlovchi P massivlarni e'lon qilamiz.

4. |V|-1 marta qirrallar jadvalini tekshirib chiqib, D va P massivlarni o'zgartirib chiqamiz.

$$d[v] = \min(d[v], d[u] + w[u, v]);$$
$$p[v] = u.$$

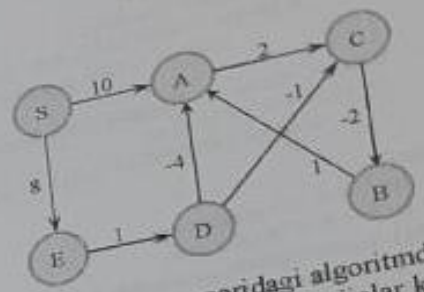
5. |V|-marta 4-qadamni takrorlab, grafda manfiy sikl mavjud emasligini tekshirib ko'ramiz.

Misol.

Bellman-Ford algoritmi bo'yicha eng qisqa masofani aniqlash masalasiga misol sifatida quyidagi grafni ko'rib chiqamiz. Unda S tugundan boshqa tugunlargacha bo'lgan eng qisqa masofalarni aniqlaymiz. Grafdagi qirralarni va ularning og'irliklari ro'yxatini tuzib olamiz. Grafda 6 ta tugun mavjudligi sababli bu algoritmda

qisqa masofalarni aniqlash ko'pi bilan |V|-1=5 ta iteratsiyada amalga oshiriladi. Dastlab boshlang'ich tugun belgisiga 0 ni, qolgan barcha tugunlar belgilariga '∞' belgisini o'zlashtiramiz.

SA=10
SE=8
AC=2
BA=1
CB=-2
DC=-1
DA=-4
ED=1



Quyidagi jadvalda har bir iteratsiyadan keyingi yuqoridagi algoritmda keltirilgan formula bo'yicha tugunlar belgilari qiymatlari hisoblanib, natijalar keltirilgan. Ko'rish mumkinki, 3- va 4-iteratsiyalardagi natijalar bir xilligi sababli oxirgi 5-iteratsiyani bajarishga ehtiyoj qolmagan.

Iteratsiya /V	S	A	B	C	D	E
1	0	∞	∞	∞	∞	∞
2	0	10	10	12	9	8
3	0	5	5	7	9	8
4	0	5	5	7	9	8
5						

Ushbu algoritmning C++ da dastur kodini keltiramiz. Qulaylik uchun S, A, B, C, D, E tugunlarni dasturda 0,1,2,3,4,5 deb ifodalab olamiz.

```
#include <bits/stdc++.h>
using namespace std;
void BellmanFord(int graph[][3], int V, int E,int src){
    // tugunlar belgisiga maksimal qiymat o'zlashtirish
    int dis[V];
    for (int i = 0; i < V; i++)
        dis[i] = INT_MAX;
    dis[src] = 0;    // boshlang'ich tugun belgisiga 0 ni o'zlashtirish
                     // qirralarni qarab chiqamiz va tugunlar belgilarini
                     // |V| - 1 marta formulaga ko'ra tekshiramiz
    for (int i = 0; i < V - 1; i++) {
    for (int j = 0; j < E; j++) {
    if(dis[graph[j][0]]!=INT_MAX&&dis[graph[j][0]]+graph[j][2]<dis[graph[j][1]])
        dis[graph[j][1]] = dis[graph[j][0]] + graph[j][2];
        }

    }
    // Yuqoridagi kod grafda manfiy sikl mavjud emasligiga kafolat beradi
```

```
    // manfiy sikl mavjudligini tekshirish
    // agar yanada qisqaroq yo'l topilsa, grafda manfiy sikl mavjudligi kelib chiqadi
    for (int i = 0; i < E; i++) {
        int x = graph[i][0];
        int y = graph[i][1];
        int weight = graph[i][2];
        if (dis[x] != INT_MAX && dis[x] + weight < dis[y])
            cout << "Graph contains negative weight cycle"<< endl;
    }
    cout << "Vertex   Distance from Source" << endl;
    for (int i = 0; i < V; i++)
        cout << i << "\t\t" << dis[i] << endl;
}
int main(){
    int V = 6; // tugunlar soni
    int E = 8; // qirralar soni
// graph[] massivi qirralarga tegishli 3ta qiymat (u, v, w), ya'ni u dan v gacha w qirra
//vazni qiymatlarini saqlash uchun ishlatiladi
    int graph[][3] = { { 0, 1, 10 }, { 0, 5, 8 },
                       { 1, 3, 2 },  { 2, 1, 1 },
                       { 3, 2, -2 }, { 4, 1, -4 },
                       { 4, 3, -1 }, { 5, 4, 1 }
                     };
    BellmanFord(graph, V, E, 0);
    return 0;
}
```

Dastur natijasi:

Vertex	Distance from Source
0	0
1	5
2	5
3	7
4	9
5	8

Dastur bajarilishida ham yuqoridagi jadvalda keltirilgan 4-iteratsiyadagi natija olindi. Bellman-Ford algoritmi ishlash tezligi: $O(|V| \cdot |E|)$.

4.2.4. Lug'atlar va ularni amalga oshirish

Odatdagi massivlar raqamlangan elementlar ketma-ketligi hisoblanadi, ya'ni bironta elementga murojaat qilish uchun uning raqamini, ya'ni indeksini ko'rsatish kerak bo'ladi. Ketma-ketlikda element raqami ushbu elementni identifikatsiyalash vazifasini bajaradi. Ammo to'plam elementlarini raqamlar orqali identifikatsiyalash har doim ham qulay bo'lavermaydi. Masalan, poyezdlar marshrutlari, avia-reyslar qatnovi sonli-xarfli yoki sonli-belgili kodlar bilan identifikatsiyalanadi, ya'ni bunda poyezdlar yoki samolyotlar reyslari haqidagi ma'lumotlarni saqlashda ularni

identifikatsiyalash uchun sonlar emas, satrlardan foydalanish ancha qulay hisoblanadi.

Elementlarini sonli indekslar bilan emas, balki erkin toifadagi ma'lumot yordamida identifikatsiyalash imkonini beruvchi tuzilmaga **lug'at** yoki **assotsiativ massiv** deyiladi va **map** - bu tuzilmaga mos keluvchi STL kutubxonasining sinfi hisoblanadi. Lug'atning vazifasi biror to'plam elementini (indeks hisonlangan kalit) boshqa to'plam elementi bilan solishtirishdir.

Lug'at - bu ma'lumotlar tuzilmasi bo'lib, unda elementlar ikkita asosiy qism: kalit va ma'lumotdan tashkil topadi. Lug'at elementlariga kalit orqali murojaat qilinadi. Kalit har doim bir lug'at ichida noyob bo'lishi kerak va zarur bo'lsa, ma'lumotlar takrorlanishi mumkin. Lug'at tuzilmasi ba'zi manbalarda *dictionary deb* ham ataladi. Lug'atning ishlash prinsipi saqlash xonasiga o'xshaydi: har qanday narsani saqlash mumkin bo'lgan seyflar mavjud, ammo bu seyflarga faqat noyob kalit yordamida murojaat qilish mumkin, shuning uchun uni topish har doim ham oson.

Xayotimizda lug'atlardan keng foydalaniladi. Masalan, qog'oz shaklidagi orfografik, lingvistik, texnik yoki izohli lug'atlar. Bularda elementlar, ya'ni maqolalar kaliti sifatida so'zlar – matn sarlavhasi olinadi, element qiymati sifatida – maqola matnining o'zi olinadi. Maqolani o'qish uchun uning kalit so'zini bilish kerak. Lug'atga misol qilib telefon katalogini ham keltirish mumkin. Unda kontakt ismlar – kalit, telefon raqamlar – element qiymati hisoblanadi.

Lug'atlarni quyidagi hollarda qo'llash mumkin:

- qandaydir obyektlarni sarhisobini yuritishda; bunda kalit sifatida - obyektlar va qiymat sifatida – ularning soni olinadi;
- obyektlar bilan bog'liq bironta berilganlarni saqlashda; bunda kalit – obyektlar, qiymat esa – ular bilan bog'liq berilganlar olinadi. Masalan, oy nomiga ko'ra uni tartib raqamini aniqlash kerak bo'lsa, u holda lug'atlardan foydalanish mumkin: Num["Yanvar"]=1; Num["Fevral"]=2; ...
- obyektlararo munosabatlarni o'rnatishda; masalan "ota-ona - avlod". Bunda kalit – obyekt, qiymat – unga mos keluvchi obyekt.

Shuni aytsh kerakki, lug'at elementlari tartiblanmagan hisoblanadi, zarur bo'lganda ularni kalit bo'yicha tartiblash va binar qidiruv usulini qo'llash mumkin. C++ tilida lug'atlar bilan ishlash uchun **#include<map>** sarlavha faylini faollashtirish talab etiladi.

Lug'at ustida quyidagi amallarni bajarish mumkin:

- yangi element kiritish;
- element o'chirish;
- element qiymatini yangilash;
- elementni o'qib olish.

Lug'atni amalga oshirishga misol keltirsak.

```
map <string, string> Capitals;
Capitals["Uzbekistan"] = "Tashkent";
Capitals["Ukraine"] = "Kiev";
Capitals["USA"] = "Washington";
```

```
    cout << "qaysi mamlakatda yashaysiz? ";
    cin >> country;
    if (Capitals.count(country))
      cout << "Bu mamlakat poytaxti " << Capitals[country] << endl;
    else{
      cout << "siz yashayotgan mamlakat poytaxti qanday ataladi? ";
      cin >> city;
      Capitals[country] = city;
    }
```

Bu misolda lug'at elementi kaliti sifatida mamlakat nomi va qiymati sifatida poytaxt nomi belgilangan. Map shablonidan foydalanib, lug'at e'lon qilganda, elementlar kalitlari va qiymatlari toifalari ko'rsatilishi zarur.

Lug'at elementlari ustida amal bajariluvchi *map* kutubxonasining ayrim asosiy funksiyalarini ko'rib chiqamiz.

- *begin()* – iteratorni lug'at birinchi elementiga joylash;
- *end()* – iteratorni lug'at oxirgi elementidan keyinga joylash;
- *size()* – lug'at elementlari sonini qaytaradi;
- *max_size()* – lug'atda saqlash mumkin bo'lgan elementlarning maksimal soni;
- *empty()* – lug'at bo'shligini tekshirish;
- *pair insert(keyvalue, mapvalue)* – lug'atga yangi element kiritish;
- *erase(iterator position)* – iterator ko'rsatayotgan pozitsiyadagi elementni o'chirish;
- *erase(const g)* – lug'atdan kaliti='g' bo'lgan elementni o'chirish;
- *clear()* – lug'at barcha elementlarini o'chirish.

Lug'at elementlari bilan ishlash. Lug'atda bajariladigan asosiy amallardan biri bu kalit bilan elementlar qiymatini o'qib olish uchun murojaat qilish amali bo'lib, dasturda ifodalash massiv bilan bir xil: **A[key]**. Agar bunday kalitli element mavjud bo'lmasa, sonli qiymatga ega lug'atlarda 0 qiymat qaytariladi, satrli qiymatga ega lug'atlarda bo'sh satr qaytariladi. Elementlarga murojaat qilish **at(key)** metodi bilan ham amalga oshirilishi mumkin. **Key** kalit lug'atda mavjudligini tekshirish **count(key)** metodi bilan bajariladi va u 0 yoki 1 qiymat qaytaradi. **Find(key)** metodi ham **key** kalitli elementni qidirishda ishlatiladi. Bunda iteratorni topilgan elementga joylashtiradi, aks holda, element topilmasa, **end()** qiymat qaytaradi. Lug'atga yangi element kiritish massivdagi kabi yoziladi: **A[key]=value**. Lug'atda **key** kalit bo'yicha elementga murojaat qilish va yangi element kiritish amali O(log n) (n-lug'at elementlari soni) vaqtda bajariladi. Element o'chirish **erase()** metodi orqali bajariladi va bu metodning parametri sifatida kalit berilishi yoki o'chirilishi talab qilinayotgan elementga joylashtirilgan iterator berilishi mumkin. Bunda ochirish amali, mos ravishda, O(log n) va O(1) vaqtni talab qiladi.

Lug'at elementlariga murojaat qilish amali. Lug'atlarda ham iteratorlar ishlatiladi. **find, upper_bound, lower_bound, begin, end, rbegin, rend** metodlari iteratorni lug'at elementlariga joylashtiradi. Bu metodlar **set** to'plamda ishlatiladigan iterator metodlari bilan bir xil. Iterator e'lon qilinganda pair tipli obyekt qaytariladi va uning **first** maydoni kalitlarga, **second** maydonida lug'at elementi qiymatlariga murojaat qiladi.


```
for (auto it = Capitals.begin(); it != Capitals.end(); ++it) {
    cout << "Mamlakat: " << (*it).first << endl;
    cout << "poytaxti: " << (*it).second << endl;
}
```

(*it).first va (*it).second kodi o'rniga "->" ixchamroq operatorini ham ishlatish mumkin:

```
    it->first, it->second.
```

Lug'at elementlarini quyidagi ixcham kod orqali ham ekranga chiqarish mumkin:

```
for (auto elem: Capitals) {
    cout << "Mamlakat: " << elem.first << endl;
    cout << "poytaxti: " << elem.second << endl;
}
```

Yuqorida keltirilgan lug'at bilan ishlash metodlari qo'llanilgan misolning to'liq dastur matnini keltiramiz.

```
#include <iostream>
#include <iterator>
#include <map>
using namespace std;
int main() {
  map<int, int> gquiz1;
  gquiz1.insert(pair<int, int>(1, 40));
  gquiz1.insert(pair<int, int>(2, 30));
  gquiz1.insert(pair<int, int>(3, 60));
  gquiz1.insert(pair<int, int>(4, 20));
  gquiz1.insert(pair<int, int>(5, 50));
  gquiz1.insert(pair<int, int>(6, 50));
  gquiz1.insert(pair<int, int>(7, 10));

  map<int, int>::iterator itr;
  cout << "\nThe map gquiz1 is : \n";
  cout << "\tKEY\tELEMENT\n";
  for (itr = gquiz1.begin(); itr != gquiz1.end(); ++itr) {
    cout << '\t' << itr->first<< '\t' << itr->second << '\n'; }
  cout << endl;
  // assigning the elements from gquiz1 to gquiz2
  map<int, int> gquiz2(gquiz1.begin(), gquiz1.end());
  // print all elements of the map gquiz2
  cout << "\nThe map gquiz2 after assign from gquiz1 is : \n";
  cout << "\tKEY\tELEMENT\n";
  for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr) {
    cout << '\t' << itr->first<< '\t' << itr->second << '\n'; }
  cout << endl;
  // remove all elements up to element with key=3 in gquiz2
  cout << "\ngquiz2 after removal of elements less than key=3 : \n";
  cout << "\tKEY\tELEMENT\n";
  gquiz2.erase(gquiz2.begin(), gquiz2.find(3));
  for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr) {
    cout << '\t' << itr->first << '\t' << itr->second << '\n'; }
  // remove all elements with key = 4
  int num;
```

```
    num = gquiz2.erase(4);
    cout << "\ngquiz2.erase(4) : ";
    cout << num << " removed \n";
    cout << "\tKEY\tELEMENT\n";
    for (itr = gquiz2.begin(); itr != gquiz2.end(); ++itr) {
      cout << '\t' << itr->first << '\t' << itr->second << '\n'; }
    cout << endl;
    // lower bound and upper bound for map gquiz1 key = 5
    cout << "gquiz1.lower_bound(5) : "<< "\tKEY = ";
    cout << gquiz1.lower_bound(5)->first << '\t';
    cout << "\tELEMENT = "<< gquiz1.lower_bound(5)->second << endl;
    cout << "gquiz1.upper_bound(5) : "<< "\tKEY = ";
    cout << gquiz1.upper_bound(5)->first << '\t';
    cout << "\tELEMENT = "<< gquiz1.upper_bound(5)->second << endl;
    return 0;
}
```

Dastur natijasi:

```
The map gquiz1 is :
        KEY     ELEMENT
        1       40
        2       30
        3       60
        4       20
        5       50
        6       50
        7       10
The map gquiz2 after assign from gquiz1 is :
        KEY     ELEMENT
        1       40
        2       30
        3       60
        4       20
        5       50
        6       50
        7       10
gquiz2 after removal of elements less than key=3 :
        KEY     ELEMENT
        3       60
        4       20
        5       50
        6       50
        7       10
gquiz2.erase(4) : 1 removed
        KEY     ELEMENT
        3       60
        5       50
```



```
        6       50
        7       10
gquiz1.lower_bound(5) :     KEY = 5     ELEMENT = 50
gquiz1.upper_bound(5) :     KEY = 6     ELEMENT = 50
```

Ba'zi hollarda lug'atlar bilan ishlaganda, elementlar qiymati toifasini e'lon qilishda, bog'langan ro'yxatlardan foydalanish ham mumkin:
std::map<std::color, std::list<std::car>>

Nazorat savollar

1. Graf tuzilmasi asosida ishlovchi qanday dasturiy ta'minotlarni shaxsiy hayotingizda ishlatib ko'rgansiz?
2. Graflarni kompyuter fizik xotirasida qanday ifodalash mumkin?
3. Graflarda minimal narxli daraxt skeletini aniqlash masalasi nima maqsadda va qaerlarda qo'llaniladi?
4. Eng qisqa masofani aniqlash algoritmlarining samaradorliklari qanday aniqlanadi?
5. Lug'at nima va u dasturda qanday ifodalanadi?
6. Lug'atlar bilan ishlashning qanday algoritmlarini bilasiz?

V BO‘LIM.
DASTURIY TA’MINOTNI TESTLASH VA TEKSHIRISH. MA’LUMOTLAR MODELI

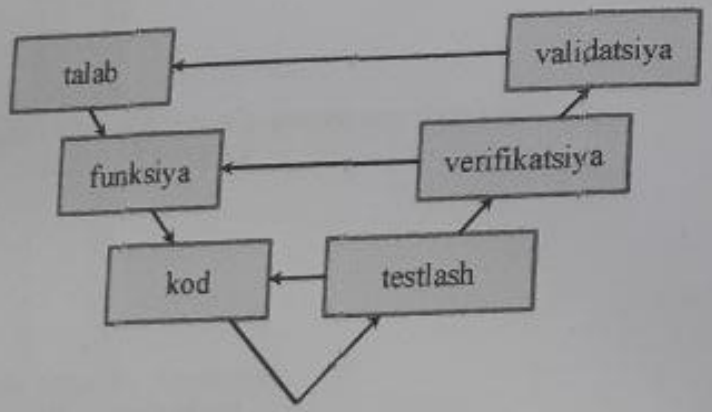
5.1. Testlash, validatsiya, verifikatsiya tushunchalari va ularning farqi

Testlash masalasiga to‘xtalganda, albatta, verifikatsiya va validatsiya tushunchalariga duch kelinadi. “Testlash”, “verifikatsiyalash” va “validatsiyalash” terminlari juda bir-biriga o‘xshash tuyulsada, obyektning, masalan bironta dasturiy ta’minotning (DT) korrekt ishlashini tekshirishning turli darajasini bildiradi.

DT ni **testlash** – bu DTni ishlab chiqish jarayonining bir qismi bo‘lib, unda xatoliklarni (nomuvofiqligi, to‘liqmasligi, noaniqligi va boshqalarni) aniqlashga (mavjudligini isbotlashga) yo‘naltirilgan protseduralarni bajarilishi bilan bog‘liq bo‘lgan faoliyat turi hisoblanadi. Testlash jarayoni, avvalambor, tizimni dasturiy realizatsiyasini korrekt amalga oshirilganligini, talablarga mos holda amalga oshirilganligini tekshirishdan iboratdir, ya’ni testlash - bu dasturiy ta’minotni nomuvofiq ishlashini va talabga javob bermasligini aniqlash maqsadida dastur bajarilishini tekshirishdir.

DT ni testlashdan maqsad uni har qanday sharoitda to‘g‘ri ishlash ehtimolligini oshirishdir. Testlash etaplari:

- mahsulotni tahlil qilish;
- talablar bilan ishlash;
- testlash strategiyasini ishlab chiqish va sifatni nazorat qilish protsedurasini rejalashtirish;
- testlash hujjatlarini yaratish;
- prototipni testlash;
- asosiy testlash;
- stabillash;
- ekspluatatsiya.



5.1-rasm. Testlash, verifikatsiya va validatsiya

Ushbu diagrammada testlashning turli darajalarini ko‘rish mumkin. Eng quyi qatlamda modulli testlash yotadi.

1. **Modulli testlash (Unit testing).** Modulli (komponentali) testlashda DT ning alohida qismlari (obyektlar, sinflar, funksiyalar,...) dastur kodi ishlashi tekshiriladi va nosozliklari qidiriladi.
2. **Integratsion testlash (Integration testing).** Modulli testlashdan o‘tgan komponentalarni o‘zaro aloqasi tekshiriladi.
3. **Tizimli testlash (System testing).** Bunda tizimda funksional va funksional bo‘lmagan talablarni bajarilishi tekshiriladi. Ya’ni tizim resurslaridan noto‘g‘ri foydalanish, foydalanuvchi ma’lumotlarini noto‘g‘ri kombinatsiyalash, reja qilinmagan yoki noto‘g‘ri funksionallik, qo‘llashning noqulayligi kabi nosozliklar tekshiriladi.
4. **Operatsion testlash (Release testing).** Garchi tizim barcha talablarga javob bersada, u foydalanuvchi ehtiyojlariga mos kelishi va ekspluatatsiya qilishda o‘ziga biriktirilgan vazifalarni to‘la bajara olishiga iqror bo‘lish mumkin hisoblanadi. Shuni inobatga olish kerakki, hatto biznes modelda ham xatolik bo‘lishi mumkin. Shu sababli, validatsiyaning yakuniy etapi sifatida operatsion test o‘tkazilishi zarur. Bunda boshqa tizimlar bilan nomutanosibliklar, ekspluatatsiya muhitida yetarlicha bo‘lmagan samaradorliklar kabi muammolar aniqlanishi mumkin.
5. **Qabul qilib olishda testlash (Acceptance Testing).** Testlashning formal jarayoni bo‘lib, tizimni talablarga javob berishi va mijoz tomonida tayyor mahsulot sifatida qabul qilinish yoki qilinmasligi aniqlanadi.

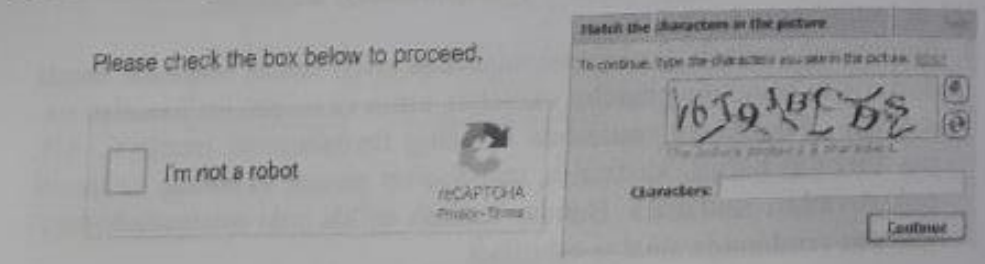
DT ni testlashning turlari:

1. Funksional testlash; funksional talablarga javob berishi testlanadi. Bunda dasturiy ta’minotning barcha vazifalari, kirish va chiqish ma’lumotlari tekshiriladi. Funksional testlashda DT ning foydalanuvchi interfeysi, API, ma’lumotlar bazasi, xavfsizlik, mijoz/server aloqasi va boshqa funksional imkoniyatlari testlanadi. Bunday testlash qo‘lda yoki avtomatlashtirilgan tizimlar yordamida amalga oshiriladi.

2. Nofunksional testlash: DT ni turli kattaliklar bilan o'lchanishi mumkin bo'lgan xarakteristikalarini aniqlash uchun zarur bo'lgan testlar orqali tasvirlanadi. Umumiy qilib aytganda, tizim umumiy ishlashini testlash hisoblanadi.
3. O'zgarish bilan bog'liq bo'lgan testlash: DT xatoliklarini bartaraf qilish, sozlash natijasida ro'y bergan o'zgarishlardan keyin muammo haqiqatda bartaraf etilganligini tasdiqlash uchun testlashdir.

DT ni **verifikatsiyalash** – bu testlashga nisbatan kengroq tushuncha bo'lib, verifikatsiyalashning maqsadi – ishlab chiqilgan DT ni talablarga javob berishiga, loyiha spetsifikatsiyasi va standartlarga mosligiga kafolat berishdan iboratdir. Verifikatsiya jarayoni o'z ichiga kodni testlash, natijalarni tahlil qilish, muammolar to'g'risida hisobotlarni shakllantirish va ularni tahlil qilishni oladi. Shunday qilib, testlash jarayoni verifikatsiyalashning bir qismi hisoblanadi.

Ko'p hollarda verifikatsiyalash deganda foydalanuvchi ma'lumotlari bilan ishlovchi DT larni xavfsizligi va ishlash qobiliyati kafolati tushuniladi. Tekshiruv protsedurasi ruxsatsiz murojaatlarni cheklash va internetda sayt ishlashi, servisi yoki dasturiy ta'minotini zararlash uchun mavjud bo'lmagan ma'lumotlar bilan yaratilgan akkauntlarning avtomatlashtirilgan tovlamachilik sxemalari riski pasaytiriladi. Shu sababli turli servislarda verifikatsiyalash o'tkaziladi. Misol uchun, bank kartasini olishda shartnomada ko'rsatilgan shaxsgina undan foydalanish xuquqiga ega ekanligini tasdiqlash talab etiladi. Buning uchun verifikatsiya tizimi qo'shimcha ma'lumot kiritishni so'raydi. Aytaylik, tovlamachiga sizning plastik kartangiz nomeri, amal qilish muddati, karta egasining ism-sharifi va kartaga tegishli kodlar ma'lum, ammo karta egasining telefoniga tovlamachi egalik qilolmaydi – tekshiruv SMS xabar orqali, internet-bank yoki operator qo'ng'irog'i bilan bajariladi. Ikki etapli autentifikatsiya tizimlari ham xuddi shu sxemada amalga oshirilgan, ya'ni foydalanuvchi login, parolidan tashqari ushbu akkauntga biriktirilgan telefon raqamiga SMS yuborish orqali tasdiqlash tamoyili amalga oshiriladi. Veb sayt ma'murlari murojaatlari ham verifikatsiyadan o'tkaziladi, misol uchun Yandex.Vebmaster tizimida. Ma'mur o'z shaxsini tanitishi uchun sayt asosiy sahifasining HTML kodida maxsus tegda maxsus kodni ishlatishi yoki FTP orqali o'zak katalogga HTML faylni yuklashi kerak bo'ladi. Internet tarmog'ida bunday holatlarga ko'p marta duch kelgan bo'lishingiz ham mumkin, misol uchun saytlarda "CAPTCHA" antispam verifikatsiyalash tizimida sayt foydalanuvchisi haqiqiy odam yoki avtomatlashtirilgan bot xizmati ekanligini tekshiruvdan o'tkazadi.



DT ni **validatsiyalash** – bu DT ni mijoz buyurtmasiga mosligini tasdiqlash. Boshqacha qilib aytganda, validatsiya – DT ni mijoz buyurtmasiga ko'ra, konkret holatga qo'llashga mo'ljallangan talablarga to'liq mosligini faktlar bilan asoslanishidir, ya'ni bunda talablarga mosligini faqatgina tasdiqlash emas, balki o'z sohasida qo'llash mumkinligini tasdiqlovchi fakt nazarda tutiladi.

Misol o'rnida keltirish mumkinki, tasavvur qiling, telefon sotib olish uchun do'konga bordingiz. Telefon apparati ishlab chiqarilishida unga qo'yilgan talablarga to'la javob berishi verifikatsiyadan o'tishi hisoblansa, uni aynan siz sotib oli- shingiz uchun sizning xohish-istagingiz va talablaringizga mos kelishi – siz tomondan validatsiyadan o'tkazilishi hisoblanadi.

Umumiy qilib aytganda,

- testlash – "bu qanday qilingan?" yoki "ishlab chiqilgan DT holati talab doirasidami?" degan savollarga;
- verifikatsiya – «nima qilindi?» yoki «ishlab chiqilgan tizim talablarga javob beradimi?» degan savollarga;
- validatsiya – «kerakli ish qilindimi?» yoki «ishlab chiqilgan tizim mijoz buyurtmasiga mosmi?» degan savollarga javob beradi.

Verifikatsiya va validatsiya tushunchalarini o'zaro solishtiradigan bo'lsak, verifikatsiya – talablar asosida umumiy tekshiruvdir. Nimanidir tekshiradigan bo'lsak, qo'limizda talablar ro'yxati bor, unga asoslanib verifikatsiyalash amalini bajarishimiz mumkin. DT shu talablarga javob bersa, u verifikatsiyalangan deyiladi. Qiyosiy misol kelturamiz, agar qandaydir bir dori preparati barcha tekshiruv ko'rikdan muvaffaqiyatli o'tgan bo'lsa, demak u verifikatsiyalangan. Ammo uni konkret bemorga qo'llash mumkinmi? Hech qanday nojo'ya ta'siri yo'qmi? Bu savollarga javobni dori preparatini aniq bir bemorga qo'llagandan keyingina berishimiz mumkin. Bu dori preparati hali konkret odamda validatsiya jarayonidan o'tkazilishi kerak. Verifikatsiya va validatsiya tushunchalarining farqlarini keltirib o'tamiz.

- validatsiya – testlovchi yoki foydalanuvchilar tomonidan, verifikatsiya esa – ishlab chiqaruvchi tomonidan amalga oshiriladi;
- verifikatsiya vazifasi – ishlab chiqarish talabiga rioya qilinishi bo'lsa, validatsiya vazifasi – pirovard mahsulot iste'molchi talabiga asosan ishlab chiqilganligini aniqlash jarayonidir;
- verifikatsiyalash majburiy tartibda o'tkaziladi, validatsiyalash esa standart yoki istemolchi talab qilgandagina amalga oshiriladi;
- verifikatsiyalash doimo obyektiv hisoblanadi, ya'ni ishlab chiqilgan mahsulot ko'rsatilgan texnik parametrlarga yoki mos keladi, yoki mos kelmaydi. Validatsiyalash esa sub'yektiv hisoblanadi, ya'ni insonlar tomonidan uni konkret sharoitlarda qanchalik muvaffaqiyatli qo'llash mumkinligi aniqlanadi.

5.2. Modulli testlash (oq, qora va kulrang quti)

Modulli testlash (unit testing) – DT ning alohida modullari dastur kodini testlashdan iborat jarayondir. Modulli testlashning maqsadi – dastur kodining har

qaysi birligini talab darajasida ishlashini tekshirishdan iborat. Testlashning ushbu turi dastur kodini tuzish etapida dasturchilar tomonidan bajariladi. Modulli testlar dastur kodining ma'lum bir qismini, masalan alohida funksiya, metod, protsedura, modul yoki obyektlarni izolyatsiyalab, uni ishlash qobiliyatini tekshiradi. Bunda har bir funksiya yoki metod uchun test ishlab chiqiladi. Bu DT ni navbatdagi o'zgartirishi ziddiyatga olib kelmasligini, ya'ni dasturni oldin testlangan qismlarida xatolik paydo bo'lmaganligini yetarlicha tezkor tekshirish imkonini beradi, shuningdek xatoliklarni aniqlashni va bartaraf qilishni yengillashtiradi.

Modulli testlash tekshirish jarayonining eng quyi darajasi bo'lib, unda bevosita dastur kodi matnini teskshirishga oid masalalar ko'riladi. Odatda, modulli testlash jarayoni tekshirilishi kerak bo'lgan dastur kodini frameworklar kabi dasturiy ta'minotni ishlab chiqish vositalari yoki otladka qilish instrumentlari yordamida amalga oshiriladi. Barcha aniqlangan nosozliklar turli rasmiyatchilik- larsiz to'g'irlanadi. Modulli testlashda funksiyalarning parametri sifatida aniq bir qiymatlar kiritiladi. Modulli testlashning yo'qligi keyingi testlash etaplarida no- sozliklar darajasini yetarlicha oshib ketishiga olib keladi.

- modulli testlash DT ni ishlab chiqishning dastlabki bosqichlarida xatolarni tuzatishga va xarajatlarni kamaytirishga yordam beradi;
- bu dasturchilarga loyihaning bazaviy kodini yaxshiroq tushunishga yordam beradi va mahsulotga o'zgartirishlarni tezroq va osonroq kiritishga imkon beradi;
- yaxshi modulli testlar loyiha hujjatlari bo'lib xizmat qiladi;
- modulli testlar koddan qayta foydalanishga yordam beradi. Oldin tuzilgan dastur kodini yangi loyihaga ko'chirib o'tkazish va kodni o'zgartirish oson bo'ladi.

Modulli testlash qo'lda yoki ko'pincha avtomatlashtirilgan holda amalga oshiriladi. Qo'lda modulli testlash qadamma-qadam bajariladi. Avtomatlashtirilgan modulli testlash esa quyidagi algoritm bilan bajariladi:

- dasturchi sinovdan o'tkazish uchun testlovchi dasturga kod birligini yozadi. Avval kodga izoh berishadi va keyin sinov kodini olib tashlashadi.
- Sifatli testlash uchun dasturchi kod birligini izolyatsiya qiladi. Ushbu amal kodni testlovchi muhitga nusxalash bilan ham bajarilishi mumkin. Kodni izolyatsiyalash - tekshirilayotgan kodlar orasidagi yoki boshqa modullar yoki berilganlar o'rtasidagi keraksiz bog'liqliklarni aniqlashga yordam beradi.
- Dasturchi, odatda, avtomatlashgan testlash tizimlarini ishlab chiqish uchun UniTest Framework ni ishlatadi. Avtomatlashtirish infratuzilmasini qo'llagan holda, dasturchi kodni korrekt bajarilishini tekshirish uchun testlash kriteriyasini o'rnatadi. Ko'pchilik frameworklar avtomatik tarzda testdan muvaffaqiyat bilan o'tolmagan holatlarni aniqlash va xabar qilish va navbatdagi testlash jarayonini to'xtatish imkonini beradi.
- Modulli testlash ketma-ketligi:
  a) testlash holatlarini yaratish;
  b) ko'zdan kechirish/qayta ishlash;
  c) testlash holatlarini amalga oshirish.

Modulli testlash afzalligi. Modul qanday funksional imkoniyatlarga ega ekanligini bilishni xohlovchi va uni qanday ishlatishni bilmoqchi bo'lgan dasturchilar modul haqida umumiy ma'lumot olish uchun modulli testlarni ko'rib chiqishlari mumkin. Modulli testlar dasturchiga regressiyon testlash bosqichida kodni qayta tuzatishga va uni to'g'ri ishlayotganiga iqror bo'lishga imkon beradi. Ushbu protsedura barcha funksiyalar va metodlar uchun nazorat misollarini yozishdan iborat, agar o'zgartirish natijasida xatoliklar kelib chiqadigan bo'lsa, bu holatlarda uni tezda aniqlash va tuzatish mumkin bo'ladi.

Modulli testning kamchiliklari. Barcha xatolarni aniqlay olmaydi. Hatto eng ahamiyatsiz dasturlarda ham barcha ijro yo'llarini baholash mumkin emas. Modulli testlash, o'z mohiyatiga ko'ra, kod birliklariga yo'naltirilgan. Shunday qilib, u integratsiyalash xatolarini yoki tizim darajasidagi xatolarni ushlay olmaydi. Shu sababli, modulli testni boshqa turdagi testlar bilan birgalikda ishlatgan ma'qul.

Test qilinadigan obyektning ichki tuzilishi ma'lum bo'lishiga ko'ra, ya'ni testdan o'tkazuvchi dasturchiga testlanadigan dastur kodining ma'lum bo'lishiga ko'ra, testlash 3 xil turli bo'ladi:

- oq qutini testlash;
- qora qutini testlash;
- kulrang qutini testlash.

Oq quti (ba'zan, shaffof quti deb ham ataladi) metodi bo'yicha testlashda testlovchi dasturchiga dastur kodi ochiq va ma'lum bo'ladi va testlanuvchi DT kutubxonasi bilan bog'liq bo'lgan testlash dasturi kodini yoza oladi. Bu modulli testlash turiga mansub bo'lib, tizimning alohida qismlari test qilinadigan dastur kodi ma'lum ekanligiga ko'ra unga kirish qiymatlari tanlab kiritiladi va ular ustida bajariladigan amal natijalari ham ma'lum. Oq quti yondashuvi bo'yicha testlash uchun testlanuvchi dasturning o'ziga xos xususiyatlari va realizatsiyasi ma'lum bo'lishi shart. Oq qutini testlashda tizim yoki komponentasining ichki tuzilishi tahlil qilinadi.

Misol. Testdan o'tkazuvchi shaxs dasturchi bo'lishi kerak. Veb saytning kiritish maydoni kodini realizatsiyasini o'rganib chiqadi. Foydalanuvchi tomonidan kiritiladigan, oldindan inobatga olingan (mumkin bo'lgan yoki mumkin bo'lmagan) qiymatlarni va hisobga olinmagan qiymatlarni ham qarab chiqadi. Dastur bajarilishining real natijasini kutilayotgan natija bilan solishtiradi. Bunda kutilayotgan natija dastur kodi qanday ishlashi kerakligiga qarab aniqlanadi.

Oq quti texnikasi testlashning turli darajalarida qo'llaniladi – modulli testlashdan to tizimli testlashgacha, ammo asosan dastur kodi muallifi tomonidan modulli testlash jarayonida qo'llaniladi.

Afzalligi. Oq quti metodi bo'yicha testlashning afzalligi shundaki, testlash malasi erta bosqichlarda qo'llanilishi mumkin, foydalanuvchi interfeysi yaratilishini kutish shart emas va dastur bajarilishini turli yo'llar bilan nisbatan chuqurroq tekshirish imkonini beradi.

Kamchiligi. Testlovchi dasturchidan katta bilim va tajriba talab qilinadi.

Qora quti metodi bo'yicha testlash - bu testlanadigan tizim tuzilishi ma'lum bo'lmaganda, tizimni tamomila tashqi interfeysi ishlashiga asoslanib testlash texnikasidir. Bunda testlanuvchi dastur testlovchiga qora, shaffof bo'lmagan, ichki tarkibi ko'rinmaydigan quti kabi bo'ladi. Qora quti bo'yicha testlash maqsadi - noto'g'ri tuzilgan yoki yetarlicha ishlab chiqilmagan funksyalardagi xatoliklar, interfeys xatoliklari, ma'lumotlar tuzilmasidagi xatoliklar yoki tashqi ma'lumotlar bazasiga murojaat qilishdagi xatoliklar, tizim xatti-harakatidagi xatoliklarni aniqlashdan iborat. Bunday testlashda dastur nima ish qilishiga ahamiyat qaratish zarur, bu ishni qanday bajarayotganligiga emas. Qora qutini testlash funksional, ham nofunksional bo'lishi mumkin. Funksional testlashda tizim funksiyalarining ishlashi tekshirilsa, nofunksional testlashda dasturning umumiy xarakteristikasi tekshiriladi. Qora quti texnikasi spetsifikatsiyasi mavjud bo'lgan testlashning barcha darajalarida qo'llanilishi mumkin (modulli testlashdan to qabul qilish testigacha). Misol uchun, tizimli yoki integratsion testlashda talablar yoki funksional spetsifikatsiyalar test-keyslarni yozish uchun asos hisoblanadi. Qora qutini ishla-tishga asoslangan test-dizayn texnikasi quyidagilarni o'z ichiga oladi:

- ekvivalent sinflar;
- chegaraviy qiymatlarni tahlil qilish;
- yechimlar jadvali;
- holatlarni o'zgarish diagrammasi;
- barcha juftliklarni testlash.

Afzalligi. Testlash yakuniy foydalanuvchi sifatida amalga oshiriladi va spetsifikatsiyadagi noaniqliklar va ziddiyatlarni aniqlashga yordam beradi. Testlovchi dasturlash tilini bilishi talab etilmaydi va dastur kodiga chuqur kirishi shart emas. Testlash dasturchilar jamoasi a'zosi bo'lmagan mutaxassis bilan dasturchi ishtirokisiz ham amalga oshirilishi mumkin. Spetsifikatsiya tayyor bo'lishi bilanoq test-keysni yozishni boshlash mumkin.

Kamchiligi. Dastur bajarilishining juda chegaralangan holatlarigina testlanishi mumkin. Aniq tasvirlangan spetsifikatsiya bo'lmasa, samarali test-keysni tuzish juda murakkab hisoblanadi. Agar modulli testlash bosqichida dasturchi tomonidan test qilingan bo'lsa, qora qutini testlash ortiqchalik qilishi mumkin.

Qora va oq qutini testlash texnikalarining farqini keltiramiz.

kriteriya	Qora quti	Oq quti
Ta'rifi	Tizim yoki uning komponentasini ichki tuzilishini bilmagan holda, funksional va nofunksional testlash	Tizim yoki uning komponentasini ichki tuzilishi tahliliga asoslangan testlash
Testlash texnikasini qo'llash darajasi	- qabul qilishda testlash; - tizimli testlash.	- modulli testlash; - integratsion testlash.
Test o'tkazuvchi?	Odatda, testlovchilar	Odatda, dasturchilar
Dasturlashtirishni bilish	Shart emas	shart
Amalga oshirishni bilish	Shart emas	shart
Test-keys uchun asos	Spetsifikatsiya, talablar	Loyihaviy hujjatlar

Kulrang quti. Kulrang quti - oq va qora quti yondashuvlarining kombinatsiyasi asosida dasturiy ta'minotni testlash metodi hisoblanadi. Bunda testlanadigan tizimni qanday amalga oshirilganligini ayrim xususiyatlarigina ma'lum bo'ladi, ya'ni dastur tuzilishining ayrim qismlarigina ma'lum bo'ladi. Maksimal samarali test-keyslarni yozish uchun DT algoritmi va ichki tuzilishiga murojaat qilinadi, ammo testlash jarayoni qora quti texnikasiga binoan, ya'ni foydalanuvchi nuqtai nazaridan amalga oshiriladi. Bu texnikani yarim shaffof quti metodi deb ham atashadi.

Bu usulda testlovchi dastur kodini ishlash tamoyillarini yaxshiroq anglash va uning bajarilishi mumkin bo'lgan yo'llarini o'rganish maqsadida dastur kodini o'rganib chiqadi. Buni bilish ma'lum funksionallikni tekshiruvchi test-keyslarni yozishda zarur bo'ladi. Kulrang qutini testlash texnikasi testlashning turli darajalarida qo'llanilishi mumkin, modulli testlashdan toki tizimli testlashgacha. Ammo, asosan integratsion testlash bosqichida dasturning turli modullarini o'zaro aloqasini tekshirishda qo'llaniladi.

5.3. Ma'lumotlar modeli va ularni ishlatish

Ma'lumotlar modeli – bu o'zaro bog'langan ma'lumotlar tuzilmalari, ular ustida bajariluvchi amallar va saqlanadigan ma'lumotlar uchun cheklovlar to'plamidir. Ma'lumotlar modeli yordamida bironta axborot obyektlari va ularning o'zaro aloqalarini ifodalash mumkin.

Ma'lumotlar modelidan foydalanishning asosiy maqsadi:

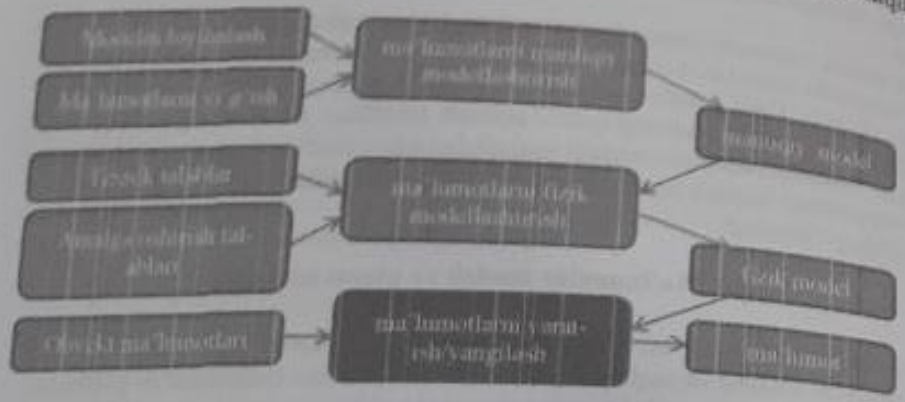
- ma'lumotlar bazasida ishlatiladigan barcha ma'lumotlar obyektlarining aniq ifodalanishini ta'minlaydi. Kerakli ma'lumotlarning tushirib qoldirilishi noto'g'ri hisobotlarni yaratishga va noto'g'ri natijalarga olib keladi;
- ma'lumotlar modeli ma'lumotlar bazasini konseptual, fizik va mantiqiy darajada loyihalashga yordam beradi;
- ma'lumotlar modeli tuzilmasi relyatsion jadvallarni, birlamchi va tashqi kalitlarni, o'rnatilgan protseduralarni aniqlashga yordam beradi;
- uning yordamida asosiy ma'lumotlar haqida aniq tasavvurga ega bo'lish mumkin va ma'lumotlar bazasini loyihalovchilar tomonidan fizik ma'lumotlar bazasini yaratish uchun foydalanilishi mumkin;
- yo'qolgan va keraksiz ma'lumotlarni aniqlashda juda ham qulaydir;
- ma'lumotlar modelini dastlabki yaratish ko'p vaqt va mehnat talab qilsada, kelajakda axborot texnologiyalari infratuzilmasini yangilash va texnik xizmat ko'rsatishni arzonroq va tezkor bajarilishini ta'minlaydi.

Ma'lumotlar modellari turlari ma'lumotlar, uning elementlari orasidagi munosabatlarni ifodalash va ularni ma'lumotlar bazasida qanday tashkil qilinishini ko'rsatish uchun ishlatiladi. Ma'lumotlar modellari 3ta asosiy toifaga ajratiladi (5.2-rasm):

1. konseptual (semantik) modellar; ushbu ma'lumotlar modeli tizim NIMAlardan iborat ekanligini aniqlaydi. Maqsad - asosiy tushunchalarni aniqlashtirish, tashkil qilish va tartibga solishdan iborat.

2. mantiqiy modellar: ma'lumotlar bazasini boshqarish tizimiga bog'liq bo'lmagan holda, tizim QANDAY amalga oshirilgandigini belgilaydi. Maqsad - ma'lumotlar tuzilmalari va qoidalarning texnik xarttasini ishlab chiqishdir.

3. fizik ma'lumotlar modeli; ushbu ma'lumotlar modeli ma'lum bir MBBT yordamida tizim QANDAY qilib amalga oshirilishini tavsiflaydi. Ushbu model odatda ma'lumotlar bazasi arxitektori va dasturchilar tomonidan ishlab chiqiladi. Maqsad - ma'lumotlar bazasini amalda qo'llashdir.



5.2-rasm. Ma'lumotlar modelini ishlab chiqish sxemasi

Ma'lumotlarning konseptual modeli bu ma'lumotlar bazasi tushunchalari va ularning o'zaro aloqalarini ifodalovchi tuzilmadir. Ma'lumotlarning konseptual modelini yaratishdan maqsad obyektlar, ularning atributlari va o'zaro aloqalarini o'rnatishdir. Ushbu modellashtirish darajasida ma'lumotlar bazasining haqiqiy tarkibida deyarli hech qanday tafsilot mavjud emas. Odatda, ishbilarmon manfaatdor tomonlar va ma'lumotlar arxitektorlari tomonidan ma'lumotlarning konseptual modeli yaratiladi. Konseptual modellarga quyidagi turdagi modellar kiradi:

- munosabat modellari (MM);
- semantik modellar (SM);
- obyektga yo'naltirilgan modellar (OYM).

Bugungi kunda eng keng tarqalgan mantiqiy modellarga esa quyidagi 3 ta turdagi modellar kiradi:

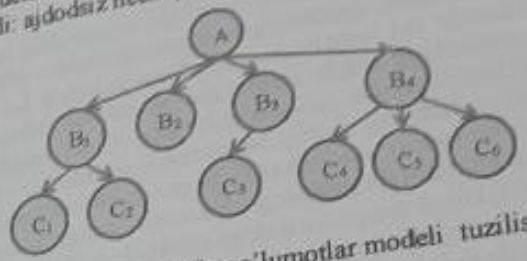
- ierarxik (IM);
- tarmoq (TM);
- relyatsion modellar (RM).

Har bir model o'ziga xos xususiyatlarga ega va ma'lumotlar bazasini boshqarish tizimlarida ma'lumotlar bazasini loyihalashda turli xil ma'lumotlar modellari ishlatiladi. Aynan qaysi turdagi ma'lumot modelini ishlatish qaralayotgan sohaning tuzilishi va xususiyatlariga bog'liq.

Ierarxik ma'lumotlar modeli. Ierarxik modellar obyektlarning ierarxik, ya'ni daraxtsimon bog'lanishidan iborat tuzilishga ega (5.3-rasm). Bunday modellar turli

darajalarda joylashgan tugunlar va ularni tutashtiruvchi yoylardan tashkil
Eng yuqori darajada ildiz va pastida uning avlodlari joylashadi. Har bir tugun faqat
bitta ajdod tugunga (ota tugun) va bir nechta avlod tugunga (o'g'il tugun) ega
bo'ladi. Istalgan tugunga ildizdan faqat yagona yo'l mavjud. Tugunlarda obyektlar
va yoylar ularning o'zaro munosabatlarini ifodalaydi.

Bu turdagi modelda ma'lumotlar o'zaro bo'ysunish tartibida bog'lanadi va bitta qoidaga asoslaniladi: ajdodsiz hech qanday avlod tugun bo'lishi mumkin emas.

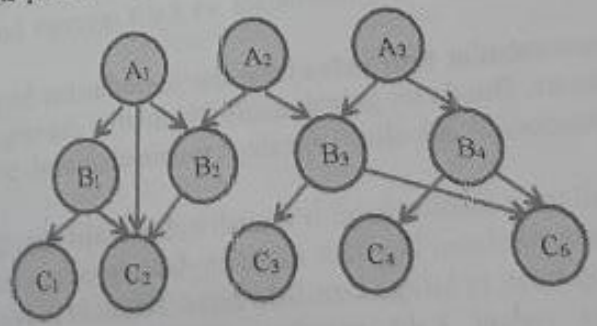


5.3-rasm. Ierarxik ma'lumotlar modeli tuzilishi

Afzalligi. Bunday modellar tushunishga va foydalanishga oson hisoblanadi. Ierarxik modellar amalda kompyuter xotirasidan samarali foydalanish va elementlar ustida tezkor amal bajarish qobiliyatini ko'rsatgan.

Kamchiligi. Tugunlarga murojaat bir nechta tugunlar ketma-ketligi orqali amalga oshiriladi. Faqat ildiz tugungagina kalit orqali bevotisa murojaat qilish mumkin. Elementlar soni oshib borgan sari va murakkab bog'lanishga ega tuzilmalar bilan ishlashda ierarxik modellar kattalashib, murakkablashib, oddiy foydalanuvchi uchun tushunish qiyinlashib boradi.

Tarmoq ma'lumotlar modeli. Ierarxik modellardagi kabi, bu modelda ham tugun, daraja va yoy tushunchalari mavjud bo'lib, tugunlari o'zaro istalgancha bog'lanishga ega bo'lgan tuzilishdagi model hisoblanadi. Modellashtirilayotgan predmet soha tarmoq tuzilishiga ega bo'lganda qo'llaniladi.



5.4-rasm. Tarmoq ma'lumotlar modeli

Bunda turli darajadagi tugunlar orasida bog'lanishlar o'rnatilishi mumkin va tugunlarning bir nechta ajdodlari bo'lishi mumkin.

Afzalligi. Bu model ierarxik modelga nisbatan umumiyroqdir. Tarmoq modelida kalit bilan nafaqat ierarxiyaning yuqori darajasidagi, balki ixtiyoriy darajadagi obyektga murojaat qilish mumkin. Bu model yordamida "ko'pga-ko'p" munosabatni o'rnatish mumkin va ma'lumotlar takrorlanishi uchramaydi.

Kamchiligi. Tarmoq modelining kamchiligi uning murakkabligidir, tushunchalarning, o'zaro bog'lanishlarning va realizatsiya xususiyatlarining xaddan ziyod ko'pligidir.

Relyatsion ma'lumotlar modeli. Elementlari o'zaro bog'langan bir nechta to'plamlar majmuasi sifatida qaralayotgan tizim tuzilishini ifodalash usulidir. Boshqacha qilib aytganda, ma'lumotlarni ikki o'lchamli jadval ko'rinishida tashkil etish usulidir. Bu modelning asosiy elementlari relyatsion jadvallar va ularning o'zaro bog'liqliklari bo'lib, har bir jadval o'zida biror obyekt haqida ma'lumotlarni saqlaydi. Relyatsion jadvalning asosiy tuzilmaviy elementi – bu maydon va yozuvdir. Maydon – bu jadval ustuni bo'lib, biror obyekt atributini ifodalaydi va ma'lumotlarni mantiqiy tashkil etishning elementar birligi hisoblanadi. Jadval ustunlari (maydonlari) unikal nomga va toifaga ega bo'ladi. Yozuv – bu jadval satri bo'lib, birorta obyektga tegishli bo'lgan mantiqiy bog'langan maydonlar to'plami hisoblanadi. Jadvallarda ayni bir xil satrlar mavjud bo'lmaydi.

№	ID	Familiya	ismi	Tug.yil.	manzil
1	193	Abdullayev	Axmad	1997	Toshkent sh.
2	205	Eshmatov	Komil	1995	Sirdaryo v.
3	87	Toshmatov	Alisher	1998	Jizzax v.

5.5-rasm. Relyatsion jadvalga misol

Jadvalning har bir yozuvini identifikatsiyalash uchun unikal bo'lgan birlamchi kalit tushunchasi ishlatiladi. Kalit sifatida bitta maydon ishlatilsa, unga oddiy kalit yoki kalit maydon deyiladi; bir nechta maydonlar ishlatilsa, tarkibli kalit deyiladi. 5.5-rasmdagi jadvalda kalit sifatida ID maydonni belgilash mumkin. Chunki bu maydon ma'lumotlari takrorlanmas hisoblanadi va kalit qiymati bo'yicha yagona yozuv qidiriladi.

Jadvallararo munosabatlar turli jadvallarni ma'lumotlardan birgalikda foydalanishlariga imkon beradi. Bunda bir jadval asosiy, ikkinchisi esa unga tobe', qaram hisoblanadi. Kalit maydonlar jadvallar orasidagi munosabatlarni o'rnatish uchun ishlatiladi.

Relyatsion jadvallararo munosabatni o'rnatish uchun kalit maydon ishlatiladi. Ikkita relyatsion jadvallarni o'zaro bog'lash mumkin, buning uchun bitta jadvalning kaliti boshqa jadvalning kaliti tarkibiga kiritilishi zarur. Ikkinchi jadval kaliti tarkibiga kiritilgan birinchi jadval kalit maydoni ikkinchi jadval uchun kalit hisoblanmasligi mumkin. Bunday holda, ushbu maydon tashqi kalit deb nomlanadi.

Relyatsion jadvallar ustida bajariladigan amallar - to'plamlar ustida odatdagi bajariluvchi amallardan iborat, ya'ni birlashtirish, kesishish, ajratish, dekart ko'paytirish va relyatsion algebra amallari – tanlash, proyeksiya va qo'shish.

Relyatsion modelning asosiy cheklovlari bu relyatsion jadvallar maydonlari orasidagi funksional bog'liqliklardir. Agar bironta jadval maydonining har bir qiymatiga ikkinchi jadval maydonining faqat bitta qiymati mos kelsa, ular funksional bog'liq hisoblanadi. Funksional bog'liqlikning ayrim turlari bazada ma'lumotlarni ortiqchaligiga olib keladi. Buni bartaraf etish uchun relyatsion ma'lumotlar modelini loyihalashda normallashtirish – bironta normal holatdan ikkinchi (yuqori) normal holatga o'tkazish amali bajariladi. Normal formalar (NF) ketma-ket va o'sish bo'yicha shakllantiriladi (1NF, 2NF, 3NF) va raqami qancha katta bo'lsa, jadvalda saqlanadigan qiymatlarga shunchalik katta cheklovlar qo'yiladi. Istalgan relyatsion jadval 1NF da hisoblanadi.

Afzalligi. Foydalanuvchilar uchun ma'lumotlarni jadval ko'rinishida tasvirlashning oddiyligidir. Ma'lumotlar tuzilmasini tashkil qilish va tushunish oson va ma'lumotlar ustida amal bajarish dasturini tuzish ham sodda hisoblanadi. Yana bir afzalligi shuki, ma'lumotlarni tasvirlashda ortiqcha ishlatilishini minimallashtirishning nazariy asoslangan usullari mavjud. Normallashtirishning maxsus usullari mavjud bo'lib, jumladan, ayrim MBBT lar yordamida berilgan jadvalni tahlil qilish natijasida normallashtirilgan jadvallarga ega bo'lish va ortiqcha ma'lumotlardan xolis bo'lish imkoni mavjud.

Kamchiligi. Jadvallarni birlashtirish amalining sekin bajarilishi. Normallashtirish natijasida ma'lumotlar sezilarli darajada bo'laklarga bo'linishi kuzatiladi, ko'pchilik masalalarda esa ma'lumot bo'laklarini birlashtirish muammosi ko'riladi. Model ko'p o'lchamli ko'rinishda tashkil qilinganligi sababli, ma'lumotlarni tezkor tahlil qilish uchun tahlilchilar tomonidan bazani ishlatish imkoni yo'qoladi. Oxirgi yillarda qaralayotgan yana bir kamchiligi shuki, bu model yordamida tizimning umumiy tuzilishi va ma'nosini anglash qiyin, ya'ni tizimning real holatini aks ettirish murakkab. Shuningdek, unda semantik imkoniyatlar cheklangan, ya'ni ma'lumotlar ma'nosini anglash murakkab hisoblanadi. Bu esa obyektga yo'naltirilgan modellarning paydo bo'lishiga olib keldi.

5.4. UML modellashtirish tili haqida

UML (Unified Modeling Language) - bu murakkab tizimlarni konseptual, mantiqiy va fizik modellashtirishni ta'minlaydigan vizual modellashtirish tili. U biror obyekt, murakkab tizimlar va xususan dasturiy vositalarni vizualizatsiya qilish, tahlil, spetsifikatsiya, loyihalash va hujjatlashtirish uchun mo'ljallangan. UML modellashtirish tili dasturiy ta'minot "shakl-shamoyili"ni yaratish uchun standart vosita hisoblanadi. Har qanday tizimni modellashtirishda UML tilidan foydalanish mumkin: koorporativ axborot tizimlaridan tortib toki veb-dasturlargacha va hattoki real vaqtda ishlovchi sozlangan tizimlarni ham.

Modellarni UML tilida yozishdan maqsad: tizim haqidagi ma'lumotni uzatish jarayonini osonlashtirishdir, yaqqol tasvirlangan model anglashni osonlashtiradi. Tizimning ba'zi xususiyatlarini matnda, ba'zilarini esa grafik ko'rinishda modellashtirgan ma'qul. Darhaqiqat, barcha muhim tizimlarda shunaqa tuzilmalar borki, ularni bitta dasturlash tili yordamida ifodalab bo'lmaydi. UML - bu tizim tuzilishlari

va xususiyatlarini grafik tarzda ifodalash tili hisoblanadi. UML shunchaki grafik belgilar to'plami emas. Ularning har biri aniq belgilangan semantikaga ega.

UML vizual dasturlash tili emas, lekin unda yaratilgan modellar to'g'ridan-to'g'ri turli dasturlash tillariga tarjima qilinishi mumkin. Boshqacha qilib aytganda, UML modelini Java, C ++, Visual Basic kabi dasturlash tillarida va hattoki relyatsion ma'lumotlar bazasining jadvallarida yoki obyektga yo'naltirilgan ma'lumotlar bazasi obyektlarida ifodalash mumkin. UML modellar asosida dasturlash tilida bevosita kodni yaratish mumkin.

UML tili quyidagi obyektga yo'naltirilgan tahlil va loyihalash tamoyillariga asoslanadi:

abstraksiyalash tamoyili – model o'z funksiyalarini bajarishiga bevosita aloqasi bor bo'lgan muhim xususiyatlarini inobatga olgan holda modelni loyihalshni nazarda tutadi;

inkapsulyatsiya tamoyili – abstraksiya elementlarini turli bo'limlarda ko'rinish va murojaat qilinish xususiyatlari bo'yicha qismlarga ajratish; alohida sinflarni interfeysini aniqlaydi va, odatda, obyektlar va ularning metodlari tuzilishi yashirin tarzda yaratiladi;

modullik tamoyili – loyihalanayotgan tizimni kuchli bog'langan va kuchsiz bog'langan modullar to'plamiga ajratish imkonini beradi;

ierarxiyalash tamoyili – abstraksiyaning ierarxik tuzilishini shakllantirishni anglatadi; bu tamoyil murakkab tizimlarni turli darajada detallashtirish, ya'ni batafsil yoritish orqali ierarxik modelini qurishni ta'minlaydi;

ko'pmodellik tamoyili – biror predmet sohani modellashtirishda murakkab tizimni loyihalashda uning turli holati va tuzilishini ifodalovchi turli modellarni ishlab chiqish zarurligini bildiradi;

Abstraktsiyaning asosiy omili bu obyekt. Har bir obyekt xususiyat, holat va metodlar bilan tavsiflanadi. *Xususiyat* – obyektni boshqalardan ajratib turuvchi xarakteristikasidir. *Holat* – obyektning joriy qiymatga ega bo'lgan xususiyatlari jamlanmasidir. *Metod* – obyektning boshqa obyektlarga ta'siri yoki uning boshqa obyektlarning ta'siriga bo'lgan munosabati. UML tilida ana shu obyektlararo bog'liqliklarni ifodalash mumkin. Tizimlarni loyihalash uchun UML da quyidagi diagrammalardan foydalanish mumkin:

1. foydalanish holatlari diagrammasi (Use Case Diagram); tizimni funksional vazifalarini ifodalaydi;
2. sinflar diagrammasi (Class Diagram); tuziladigan dastur kodini yozishga asos bo'lib xizmat qiladi. U dasturiy vositaning ichki tuzilishini, vorislik va sinflarning o'zaro munosabatlarini ifodalaydi;
3. xususiyatlar diagrammasi (Behavior Diagram); bu diagramma obyekt xatti-harakatini ifodalaydi va uning quyidagicha turlari mavjud:
   a) holatlar diagrammasi (Statechart Diagram) - bironta tas'irga javoban bir holatdan boshqa holatga mumkin bo'lgan o'tishlar ketma-ketligini ifodalaydi va model elementlari holatlarini tavsiflaydi;

   b) faoliyat diagrammasi (Activity Diagram) - tizimda bajariladigan amallarni algoritmik va mantiqiy realizatsiyasini modellashtiradi va obyektga yo'naltirilgan ilovalarda qo'llashga mo'ljallangan algo-ritmlar sxemasi analogi hisoblanadi;
   c) o'zaro aloqa diagrammasi (Interaction Diagram). Bu diagrammaning quyidagicha turlari mavjud:
      – ketma-ketlik diagrammasi (Sequence Diagram) – model obyektlarining vaqt bo'yicha o'zaro aloqasini ifodalovchi sinxron jarayonlarni tasvirlaydi. Bunday modelda vaqt oshkor ko'rinishda tasvirlanadi;
      – hamkorlik diagrammasi (Collaboration Diagram) – modelning o'zaro munosabatga ega obyektlari orasidagi tuzilmaviy aloqani ifodalaydi;
4. Amalga oshirish diagrammasi (Implementation Diagram). Bu diagrammaning quyidagi turlari mavjud:
   a. komponentalar diagrammasi (Component Diagram) – loyihalanayotgan tizimni fizik tasvirini ifodalaydi va modullar, bajariluvchi va berilgan dastur kodlari, fayllar atamalari yordamida uning ar-xitekturasini aniqlashtirishga imkon beradi;
   b. joriy qilish diagrammasi (Deployment Diagram) – tizim topologiyasi va umumiy konfiguratsiyasini, tarkibi qurilmaviy va dasturiy muhitlar, tugunlardan iborat bo'lgan bajariluvchi arxitekturasini ifodalaydi. Dasturiy komponentalarni tizimning alohida bo'limlarida ishlatilishini va ular orasida axborot uzatish marshrutlarini tasvirlaydi.

UML tilida foydalanish holatlari diagrammasi diagrammalar ichida boshlang'ich hisoblanadi va funksional talablar spetsifikasiyasini ishlab chiqishga asos bo'lib xizmat qiladi.

UML tilining modellari ikki turga bo'linadi:

1. *tuzilmaviy modellar* (statik modellar) tizim tuzilishi va tarkibiy qismlarini, shu jumladan ularning sinflari, atributlari, aloqalari, interfeyslarini tav-siflaydi; ushbu turdagi modellar foydalanish holatlari, sinflar, komponentalar, kuchaytirish diagrammalarini o'z ichiga oladi;
2. *xususiyatlar modellari* (dinamik modellar) tizim va tarkibiy qismlarining faoliyatini, shu jumladan ularning metodlarini, o'zaro aloqalarini, alohida tarkibiy qismlar va umuman tizim holatlarining o'zgarishini tavsiflaydi; ushbu turdagi model holat diagrammasi, faoliyat diagrammasi, ketma-ketlik diagrammasi, hamkorlik diagrammasini o'z ichiga oladi.

UML tili modellari 3 xil darajada loyihalanishi mumkin: konseptual, mantiqiy va fizik:

konseptual modellar modellashtirilayotgan tizimni tavsiflashning yuqori nisbatan umumiyroq va abstrakt darajasini aks ettiradi; ushbu daraja foydalanish holatlari diagrammasini o'z ichiga oladi; loyihalashtirilgan tizimni (dasturi

ta'minot vositasini) modellashtirish konseptual model darajasidan boshlanishi kerak.

mantiqiy modellar modellashtirilayotgan tizimni tasvirlashning ikkinchi darajasini aks ettiradi; ushbu darajada modelning elementlari jismoniy timsolga ega emas va real tizimining tuzilishi va xususiyatlarining mantiqan aks ettiradi; mantiqiy modellar konseptual modellardan keyin tuzilishi kerak; bu darajaga sinflar, holatlar, faoliyatlar, ketma-ketliklar, hamkorlik diagrammalari taalluqlidir;

fizik modellar modellashtirilayotgan tizimni eng quyi darajada tasvirlash imkonini beradi; bu darajada modellarning elementlari fizik tizimning konkret moddiy ashyolarini aks ettiradi; ushbu modellarni modellashtirish jarayonining eng oxirgi bosqichida qurish tavsiya etiladi; ushbu daraja komponenta va joriy qilish diagrammalarini o'z ichiga oladi.

Nazorat savollar.

1. Model va modellashtirish nima?
2. Ma'lumotlarning qanday modellarini dastur tuzishda ishlangansiz va ular dasturda qanday amalga oshiriladi?
3. Testlashning qanday usullarini bilasiz?
4. UML tilida obyektlar bilan ishlashning qanday funksiyalarini bilasiz?

FOYDALANILGAN ADABIYOTLAR

1. *Richard Wiener, Lewis J. Pinson.* Fundamentals of OOP and Data Structures in Java/Cambridge University Press, 2000. ISBN 9780521662208
2. *AdamDrozdek.* Data structure and algorithms in C++. Fourth edition. Cengage Learning, 2013.
3. Ma'ruza matnlari. Carnegie Mellon University – CORTINA. 2010. 15-121 Introduction to Data Structures. http://www.cs.cmu.edu/~tcortina/15-121sp10/lectures.html http://www.cs.cmu.edu/~tcortina/15-121sp10/Unit06B.pdf
4. Algorithms and data structures in C/C++ by Alex Allian. http://www.cprogramming.com/tutorial/computersciencetheory/heap.html
5. Алфред В. Ахо., Джон Э. Хопкрофт, Джефри Д. Ульман. Структура данных и алгоритмы//Учеб.пос., М.: Изд.дом: "Вильямс", 2006, — 384 с
6. Роберт Седжвик. Фундаментальные алгоритмы на C++. Анализ, Структуры данных, Сортировка, Поиск//К.: Изд. «ДиаСофт», 2001.- 688 с.

GLOSSARIY

Algoritm – biror masalani yechish uchun bajariladigan amallarning qat'iy ketma-ketligi.

Algoritm samaradorligi – bu ma'lum bir algoritmning bajarilishi uchun sarflanadigan hisoblash resurslari (vaqt sarfi, xotira hajmi,...) bilan bog'liq bo'lgan tushuncha bo'lib, algoritmni tashkil qiluvchi bosqichlar tahlili asosida aniqlanadigan xususiyatdir.

Axborot-kommunikatsiya texnologiyalari – axborotlarni yig'ish, saqlash, qayta ishlash va uzatishda qo'llaniladigan texnik vositalar, dasturiy ta'minotlar, usullar va texnologiyalar majmuasi.

Belgili toifa (tur) – kompyuter xotirasida ma'lumotlarni saqlashning bir ko'rinishi bo'lib, unda faqat bitta belgi (ixtiyoriy) saqlanadi va barcha dasturlash tllarida bunday toifa ishlatiladi.

Binar daraxt – bu elementlari o'zaro ierarxik ko'rinishda bog'langan tuzilma bo'lib, undagi tugunlarning chiqish darajasi maksimal 2 ga teng bo'lgan daraxt.

Butun toifa (tur) – butun sonlarni ifodalovchi ma'lumot turi.

Daraxt - bu elementlari o'zaro ierarxik ko'rinishda bog'langan, murakkab, nochiziqli dinamik tuzilma.

Dastur – axborotni qayta ishlash bo'yicha ko'rsatmalar, tartib – qoidalardan iborat kompyuterni boshqaruvchi vositalar yoki protseduralar.

Dastur kodi – algoritmni biror bir dasturlash tilida ifodalangan ko'rinishi.

Dek – bir xil toifali elementlardan iborat tuzilma bo'lib, bunday tuzilmaga elementlarni ham boshidan, ham oxiridan kiritish va chiqarish mumkin.

Dinamik tuzilma – uzunligi va elementlari orasidagi munosabat dastur bajarilishi mobaynida o'zgaruvchan bo'lgan tuzilma.

Gipergraf – grafning bir turi bo'lib, undagi yoylar faqat tugunlarni emas, balki tugunlar jamlanmasidan iborat qismto'plamlarni birlashtirishi mumkin.

Graf – elementlari o'zaro istalgancha bog'lanishga ega bo'lgan murakkab, nochiziqli, dinamik tuzilma.

Halqasimon ro'yxat – oxirgi elementi tuzilmaning birinchi elementini ko'rsatuvchi bog'langan ro'yxat tuzilmasi.

Haqiqiy toifa (tur) – haqiqiy qiymatlarni, ya'ni kasrli sonlarni ifodalovchi ma'lumot turi.

Inkapsulyatsiya – obyektga yo'naltirilgan dasturlash tamoyillaridan biri bo'lib, bu biror sinfda ma'lumotlar va metodlarni o'zaro birlashtirish va uning tarkibini amalga oshirish tafsilotlarini foydalanuvchidan yashirish imkonini beruvchi tizimning xususiyatidir.

Jadval – yozuvlar massividan iborat statik tuzilma.

Ko'rsatkich – xotirada joylashgan bironta ma'lumotni adresini o'zida saqlovchi o'zgaruvchi turi.

Ma'lumotlar abstraksiyasi – bu ma'lumotlarni faqat zaruriy jihatlarini tashqi muhit uchun ochiq qilib, orqa fondagi qismini yashirin tarzda ifodalashdir, ya'ni dasturda zaruriy ma'lumotlarni batafsil tavsilotlarsiz taqdim qilish holatidir.

Ma'lumotlar tuzilmasi (data structure) - ma'lum bir ko'rinishdagi munosabat bilan o'zaro bog'langan bir xil toifali elementlar ketma-ketligi.

Mantiqiy toifa – qiymati rost (true) yoki yolg'on (false) ko'rinishda bo'lgan ma'lumot turi.

Massiv – bir xil toifali elementlarning tartibli ketma-ketligidan iborat statik tuzilma.

Navbat – elementlarni bir tomondan kiritib, ikkinchi tomondan chiqarilishi mumkin bo'lgan yarimstatik tuzilma, ya'ni "1-kelgan 1-ketadi" tartibida ishlaydigan tuzilma (FIFO – first input first output)

Obyektga yo'naltirilgan dasturlash – bu dasturni obyektlar to'plami sifatida taqdim etishga asoslangan zamonaviy dasturlash metodologiyasi bo'lib, unda sinflar merosxo'rlik ierarxiyasini tashkil qiladi va obyektlarning har biri ma'lum bir sinfning namunasi hisoblanadi.

Operator – ma'lum vazifani bajaruvchi tilning konstruksiyasi.

Polimorfizm – OYD tamoyili bo'lib, funksiyalarning turli toifadagi ma'lumotlarni qayta ishlay olish qobiliyatini ifodalovchi tushunchadir.

Binar qidiruv daraxti – binar daraxtning bir turi bo'lib, undagi har bir tugunning chap o'g'il tugunida o'zidan kichik va o'ng o'g'il tugunida o'zidan katta qiymatli elementlar joylashgan bo'ladi.

Rekursiya - o'ziga o'zi murojaat qilish jarayoni bo'lib, bu tushuncha funksiyalar, ma'lumotlar tuzilmasi, algoritmlar va obyektlarga nisbatan qo'llanilishi mumkin.

Ro'yxat – uzunligi oldindan chegralanmagan va elementlari ko'rsatkichlar orqali oshkora bog'langan dinamik tuzilma bo'lib, dastur bajarilishi mobaynida elementlari soni va ularning o'zaro munosabati o'zgaruvchan bo'ladi.

Satr – uzunligi 256 belgidan oshmaydigan, belgilar ketma-ketligi ko'rinishidagi ma'lumot turi.

Sikl – ma'lum bir shartga ko'ra yoki chekli marta muayyan amallarni qayta bajaruvchi takrorlash operatori.

Statik tuzilma – elementlari soni oldindan aniqlanadigan va dastur bajarilishi mobaynida elementlari soni va ular orasidagi munosabat (ya'ni, tuzilmaga ajratiladigan fizik xotira hajmi va elementlarning fizik xotirada joylashuv tartibi) o'zgarmas bo'lgan tuzilma bo'lib, unga massiv, vektor (xotirada fizik tashkil etilishi nuqtai nazaridan), yozuv, jadval va to'plam kiradi.

Stek – elementlarni tuzilmaga kiritish va chiqarish faqat bir tomondan, ya'ni "1-kelgan oxirda ketadi, oxirda kelgan 1-ketadi" (LIFO – last input, first output) tartibida amalga oshiriladigan yarimstatik tuzilma.

To'plam – elementlari takrorlanmas bo'lgan (yagona qiymatli) tartibsiz, statik tuzilma.

Toifa – ma'lumotlarni fizik xotirada tasvirlanish turlari.

Vektor – bir xil toifali elementlarning tartibli ketma-ketligi. Massivdan farqi shuki, dastur bajarilish vaqtida vektor uzunligi o'zgarishi mumkin.

Vorislik (merosxo'rlik) – OYD tamoyillaridan biri bo'lib, oldindan mavjud bo'lgan (ona) sinf asosida yangi (voris) sinfni ifodalash mexanizmi, bunda ona

sinfning barcha funksiyalari va xususiyatlari voris sinfga meros sifatida o'zlashtirib olinadi.

Xotira – kompyuterning ma'lumotlarni saqlovchi fizik qurilmasi.

Yarimstatik tuzilma – elementlari qat'iy ketma-ketlikda joylashgan va dastur bajarilishi mobaynida ularning soni o'zgaruvchan bo'lgan tuzilma. Bunda yarim-statik tuzilmaning turiga qarab, tuzilmaning faqatgina ma'lum elementlarigagina tashqaridan murojaat qilish mumkin bo'ladi. Unga navbat, stek va dek kiradi.

Yo'naltirilgan graf - grafning bir turi bo'lib, elementlari o'zaro yo'nalishga ega bo'lgan yoylar bilan bog'lanadi.

Yo'naltirilmagan graf – grafning bir turi bo'lib, elementlari o'zaro yo'nalishga ega bo'lmagan yoylar bilan bog'lanadi.

Yozuv – turli toifadagi ma'lumotlarning tartibli ketma-ketligi.

ILOVA

TESTLAR

Ma'lumotlar tuzilmasi, asosiy tushuncha va ta'riflar. Ma'lumotlar tuzilmasi klassifikasiyasi

1. *INT* turi uchun qaysi amallar o'rinli?
A *qo'shish, ayirish, butun sonli bo'lish, qoldiqli bo'lish
B qo'shish, ayirish, bo'lish, mod, konkatenasiya
C ko'paytirish, ayirish, konkatenasiya
D ko'paytirish, ayirish, div, konkatenasiya

2. *FLOAT* turi uchun qaysi amallar o'rinli?
A *qo'shish, ayrish, ko'paytirish, bo'lish
B qo'shish, ayrish, bo'lish, mod
C ko'paytirish, ayrish, konkatenasiya
D qo'shish, ayrish, div, mod

3. STRUCT kalit so'zi yordamida qanday tuzilma yaratiladi?
A *Yozuv
B Birlashma
C Matritsa
D Standart toifa

4. Algoritm nima?
A *amallar ketma-ketligi
B Fayllarga murojaat
C Obyektlar majmuasini ifodalash
D To'plam elementlarini ifodalash

5. C++ tilida tuzilmani yaratish uchun ishlatiladigan kalit so'z qanday?
A *struct
B structure
C record
D object

6. Ma'lumotlar tuzilmasi nima?
A *bu ma'lumot elementlari va ular orasidagi munosabatlar majmuasi
B bu ma'lumot elementlari majmuasi
C bu elementlar orasidagi munosabatlar amali
D bu ma'lumot elementlari va ular orasidagi relyatsion munosabatlar majmuasi

7. Qaysi biri true kalit so'ziga mos qiymatini aniqlaydi?
A *1
B 0
C -1
D 66

8. Qaysi biri false kalit so'ziga mos qiymatini aniqlaydi?
A *0
B 1
C -1
D 66

9. Shartli operator if tanasi qachon bajariladi?
A *rost (true)
B yolg'on (false)
C Doimo bajariladi
D Hech qachon bajarilmaydi

10. Qaysi kalit so'z butun sonli o'zgaruvchi faqat musbat qiymatlarni qabul qilishini ko'rsatadi?
A *unsigned
B positive
C extern
D signed

11. C++ tilida kiritish oqimi qaysi?
A *cin >> X;
B cin << X;
C cout >> X;
D cout << X;

12. C++ tilida chiqarish oqimi qaysi?
A *cout << X;
B cin << X;
C cout >> X;
D cin >> X;

13. Massivning oxirgi elemenning tartib raqami nimaga teng bo'ladi, agar massiv o'lchami 19 teng bo'lsa?
A *18
B 19
C tartib raqami dasturchi aniqlaydi
D tartib raqami cheksiz bo'ladi

14. Ma'lumotlar tuzilmasi ustida qanday to'rtta asosiy amal bajariladi?
A *yaratish, o'chirish, tanlash (ruxsat olish), yangilash.
B yaratish, o'chirish, kengaytirish, yangilash.
C yaratish, tanlash (ruxsat olish), kengaytirish, yangilash.
D yaratish, o'chirish, kengaytirish, tanlash (ruxsat olish).

15. Ma'lumotlarni kompyuter xotirasda akslantirish nechta bosqichdan iborat?
A *3
B 4
C 5
D 6

16. Ma'lumotlar tuzilmasi mazmunli (matematik) bosqichda ...
A *konkret obyektning qayta ishlash, ularning xussusiyatlari va munosabatlarini tadqiq qilinadi.
B kompyuter xotirasida ma'lumotlarni aks ettirilishi tadqiq qilinadi.
C berilgan talabalar bo'yicha algoritmni ishlab chiqilishi tadqiq qilinadi.
D dasturni yaratish jarayoni tadqiq qilinadi.

17. Ma'lumotlar tuzilmasi mantiqiy bosqichda ...
A *berilgan talabalar bo'yicha algoritmni ishlab chiqilishi tadqiq qilinadi.
B kompyuter xotirasida ma'lumotlarni aks ettirilishi tadqiq qilinadi.

C konkret obyektning qayta ishlash, ularning xussusiyatlari va munosabatlarini tadqiq qilinadi.
D dasturni yaratish jarayoni tadqiq qilinadi.

18. Ma'lumotlar tuzilmasi fizik bosqichda ...
A *kompyuter xotirasida ma'lumotlarni aks ettirilishi tadqiq qilinadi.
B konkret obyektning qayta ishlash, ularning xussusiyatlari va munosabatlarini tadqiq qilinadi.
C berilgan talabalar bo'yicha algoritmni ishlab chiqilishi tadqiq qilinadi.
D dasturni yaratish jarayoni tadqiq qilinadi.

19. Bir xil tipdagi o'zaro takrorlanmaydigan elementlardan iborat majmua bu...
A *To'plam
B Massiv
C Yozuv
D Jadval

20. Bir xil tipdagi elementlar majmuasini ko'rsating.
A *Massiv
B Yozuv
C Jadval
D To'plam

21. Turli tipdagi ma'lumotlardan qanday tuzilma hosil qilinadi?
A *Yozuv
B Massiv
C To'plam
D Jadval

22. Turli tipdagi ma'lumot maydonlardan iborat tartibli tuzilma qaysi?
A * Jadval
B Massiv
C Yozuv
D To'plam

23. Ma'lumotlar tuzilmasini matematik qanday ifodalash mumkin?
A *$S = \{D, R\}$
B $G = \{V, E\}$
C $A = \{D(1..n)\}$
D $BT = \{K, L, R\}$

24. Oddiy sozlangan ma'lumotlar turlari (atomlar)ga quyidagilar kiradi:
A *mantiqiy, butun, haqiqiy, belgili, ko'rsatkichli tur
B massiv, yozuv, rekursiv turlar, to'plam
C jadval, stek, navbat, ruyxat, dek
D daraxtlar, graflar

25. Sozlangan tuzilmaviy MT (molekulalar) ga quyidagilar kiradi:
A *massiv, yozuv, rekursiv turlar, to'plam
B jadval, stek, navbat, ruyxat, dek
C daraxtlar, graflar
D mantiqiy, butun, haqiqiy, belgili, ko'rsatkichli tur

26. ENUM kalit so'zi yordamida qanday tuzilma yaratiladi?
A *Birlashma
B Yozuv
C Matritsa
D Standart toifa

27. C++ tilida ko'rsatkichni to'g'ri e'lon qilingan variantni ko'rsating
A *int *x
B int & x
C int x
D int [N

28. Xotirani dinamik ajratish uchun kalit so'zini ko'rsating
A *new
B create
C make
D value

29. Dinamik xotirani bo'shatish uchun kalit so'zini ko'rsating
A *delete
B clear
C free
D cls

30. int mas[10]; ko'rinishida massiv e'lon qilinganda, uning yettinchi elementiga murojaat qanday amalga oshiriladi?
A *mas[6];
B mas[7];
C mas(7);
D mas(6);

int function (char x1, float x2, double x3); funksiya qiymatining qaytarish turini ko'rsating

31.
A *int
B char
C float
D double

32. Qaysi turlarni keltirishda ma'lumotnng qisman yo'qotish bilan oshiriladi?
A *float to int
B char to float
C char to int
D int to float

33. Taqqoslash amalning qaysi biri noto'g'ri berilgan?
A *=!
B !=
C <=
D >=

34. Yuqori prioritetga ega bo'lgan amalni ko'rsating.
A *( )
B /
C .
D _

35. Ma'lumotlarning turlarni keltirishda to'g'ri javobini toping
A *(char) a
B to(char, a)
C a (char)
D char : a

36. char a; o'zgaruvchisi e'lon qilingan. Keltirilgan ifodalarning qaysi biri noto'g'ri?

A *a = "3";
B a = '3';
C a = #3;
D a = 3;

37. Ma'lumotlar tuzilmalari bog'lanishiga ko'ra quyidagilarga klassifikasiyalanadi
A *Bog'lamli va bog'lamsiz
B Statik, yarimstatik va dinamik
C Chiziqli va chiziqsiz
D Oddiy va murakkab

38. Ma'lumotlar tuzilmalari vaqt o'zgaruvchanligi yoki dastur bajarilishi jarayoniga ko'ra quyidagilarga klassifikasiyalanadi
A *Statik, yarimstatik va dinamik
B Chiziqli va chiziqsiz
C Bog'lamli va bog'lamsiz
D Oddiy va murakkab

39. Ma'lumotlar tuzilmalari tartibiga ko'ra quyidagilarga klassifikasiyalanadi.
A *Chiziqli va chiziqsiz
B Statik, yarimstatik va dinamik
C Bog'lamli va bog'lamsiz
D Oddiy va murakkab

40. Ma'lumotlar tuzilmalari uchun xotira ajratish amali qanday nomlanadi?
A *yaratish
B yo'qotish
C tanlash (ruxsat)
D yangilash

41. Ma'lumotlar tuzilmalari uchun ajratilgan xotirani o'chirish amali qanday nomlanadi?
A *yo'qotish
B yaratish
C tanlash (ruxsat)
D yangilash

42. Ma'lumotlar tuzilmalari qiymatini o'zgartirish amali qanday nomlanadi?
A *yangilash
B yo'qotish
C yaratish
D tanlash (ruxsat)

43. Ma'lumotlar tuzilmalariga ruxsat olish amali qanday nomlanadi?
A *tanlash (ruxsat)
B yangilash
C yo'qotish
D yaratish

44. C++ tilida o'zgaruvchilarni e'lon qilinganlardan qaysi biri massiv tuzilmasini anglatadi?
A *int A[100];
B
```
struct {
int P1, P2;
float P3;
} A;
```
C
```
struct {
int P1, P2;
float P3;
} A[100];
```
D int A;

45. C++ tilida o'zgaruvchilarni e'lon qilinganlardan qaysi biri yozuv tuzilmasini

anglatadi?

A
```
struct {
int P1,P2;
float P3;
} A;
```

B
```
int A[100];
```

C
```
struct {
int P1, P2;
float P3;
} A[100];
```

D
```
int A;
```

46. C++ tilida o'zgaruvchilarni e'lon qilinganlardan qaysi biri jadval tuzilmasini anglatadi?

A
```
struct {
int P1, P2;
float P3;
} A[100];
```

B
```
int A[100];
```

C
```
struct {
int P1,P2;
float P3;
} A;
```

D
```
int A;
```

47. X=3.1415; haqiqiy sonning mantissasi nimaga teng bo'ladi?

A *31415
B 1415
C 3
D 51413

48. Dastur bajarilish jarayonida xotira hajmi bir xil bo'lgan oddiy va asosiy tuzilma to'plamlariga ... deyiladi.

A *Statik ma'lumotlar tuzilmasi
B Dinamik ma'lumotlar tuzilmasi
C Yarimstatik ma'lumotlar tuzilmasi
D Rekursiv ma'lumotlar tuzilmasi

49. 12 & 10 ifodaning bitlar ustidagi amal natijasini aniqlang.

A *8
B 6
C 14
D 1

50. 12 | 10 ifodaning bitlar ustidagi amal natijasini aniqlang.

A *14
B 6
C 8
D 1

51. 12 ^10 ifodaning bitlar ustidagi amal natijasini aniqlang.

A *6
B 14
C 8
D 1

52. !12 ifodaning bitlar ustidagi amal natijasini aniqlang.

A *3
B 0
C 21
D 1

53. ~12 ifodaning bitlar ustidagi amal natijasini aniqlang.

C 13
D 12

A *-13
B -12

54. `char *a; a = new char[20];` berilgan. Egallab turgan xotirani qanday to'g'ri o'chirish mumkin?

A
```
*delete []a;
```

B
```
delete a[];
```

C
```
delete a;
```

D
```
a = NULL;
```

55. Dastur fragmentining natijasini aniqlang:

```
!((1 || 0) & & 0)
```

A *1
B 0
C NaN
D ERROR

56. Dastur fragmentining natijasini aniqlang:

```
(!(1 & &1) || 0)
```

A *0
B 1
C NaN
D ERROR

57. Dastur fragmentining natijasini aniqlang:

```
cout << (5 << 3);
```

A *40
B 53
C 35
D 0

58. Dastur fragmentining natijasini aniqlang:

```
cout << (5 >> 3);
```

A *0
B 53
C 35
D 40

59. Dastur fragmentining natijasini aniqlang:

```
1000 / 100 % 7 * 2
```

A *6
B 10
C 0
D 250

60. Dastur fragmentining natijasini aniqlang:

```
1000 / (100 % 7) * 2
```

A *1000
B 10
C 0
D 250

61. Dastur fragmentining natijasini aniqlang:

```
float X = 12.54;
cout << ceil (X) <<
" "<< floor (X);
```

A *13 12
B 12 13
C 12 12
D 13 13

62. Dastur fragmentining natijasini aniqlang:

```
x = y = 5;
z = ++x + y++;
cout << x << y << z;
```

A *6 6 11
B 6 6 12
C 6 5 11
D 5 6 11

63. Dastur fragmentining natijasini aniqlang:

```
cout << 22 / 5 * 3;
```

A *12
B 13,2
C 1,47
D 1

64. Dastur fragmentining natijasini aniqlang:

```
cout << 22.0 / 5.0 * 3;
```

A *13,2
B 12
C 1,47
D 1

65. Dastur fragment nimani anglatadi?

```
#define PI 3.14
```

A *dastur kodida Pini 3.14ga almashtirish qoidasi
B yangi tur PI kiritadi
C dastur kodida 3 satrni o'chirish
D dastur kodida 3 va 14 satrlarini o'zaro almashtirish

66. Dastur fragment nimani anglatadi?

```
Typedef unsigned char COD;
```

A *COD yangi tur kiritadi
B dastur kodida CODni almashtirish qoidasi
C dastur kodida COD so'zlarini o'chirish
D dastur tugatilishini aniqlaydi

Statik va yarimstatik ma'lumotlar tuzilmalari

67. Ikkita satrni o'zaro taqqoslash funksiyasini ko'rsating
A *strcmp();
B stringcompare();
C compare();
D cmp();

68. Birinchi satrning davomida ikkinchi satrni qo'shish funksiyasini ko'rsating

A *strcat ();
B stringadd ();
C append ();
D insert ();

69. Stek tuzilmasida qanday hizmat ko'rsatish turi qo'llaniladi?
A *LIFO
B FIFO
C FILO
D LILO

70. Navbat tuzilmasida qanday hizmat ko'rsatish turi qo'llaniladi?
A *FIFO
B LIFO
C FILO
D LILO

71. Stekga yangi element qushish funksiyasi qanday belgilanadi?
A *Push
B Pop
C Top
D Empty

72. Stekdan yuqori elementini o'chirish funksiyasi qanday belgilanadi?
A *Pop
B Push
C Top
D Empty

73. Stekdan yuqori elementini o'qitib olish funksiyasi qanday belgilanadi?
A *Top
B Pop
C Push
D Empty

74. Yarimstatik ma'lumotlar tuzilmasiga nimalar kiradi?
A *Stek, Dek, Navbat
B Stek, Massiv
C Graf. Vektor
D Yozuv, Jadval

75. Ro'yxatni massivdan ustunligini ko'rsating
A *ro'yxatni uzunligiga chegara belgilanmaydi
B Ular orasida sezilarli farq yo'q
C Ro'yxat elementlari turli tipda bo'lishi mumkin
D Ro'yxat elementlari butun tipda bo'lishi kerak

76. Dastur bajarilish jarayonida xotira hajmi statik belgilanadi va deskriptor-ko'rsatkich orqali foydalanilgan tuzilma to'plamlariga ... deyiladi.
A *Yarimstatik ma'lumotlar tuzilmasi
B Statik ma'lumotlar tuzilmasi
C Dinamik ma'lumotlar tuzilmasi
D Rekursiv ma'lumotlar tuzilmasi

77. C++ tilida standart andozalar kutubxonasi yordamida stekni qanday e'lon qilish mumkin?
A * stack < int > S;
B queue < int > S;

C deque < int > S;

D list < int > S;

78. C++ tilida standart andozalar kutubxonasi yordamida navbatni qanday e'lon qilish mumkin?

A *queue < int > S;

B stack < int > S;

C deque < int > S;

D list < int > S;

79. C++ tilida standart andozalar kutubxonasi yordamida dekni qanday e'lon qilish mumkin?

A *deque < int > S;

B queue < int > S;

C stack < int > S;

D list < int > S;

80. Funksiyalarning qaysi biri kiritish oqimidan 100 belgini x satrga o'qitadi?

A *cin.getline(x,100);

B gets(x,100);

C getline(cin, x,100);

D getch(x,100);

81. Stek bu ...

A *chiziqli ma'lumotlar tuzilmasi bo'lib, ma'lumotlarni kiritish va chiqarish uning bir tomonidan amalga oshiriladi

B shunday tuzilmaki, u yelementlar qo'shilishi bilan kengayib boradi va elementlarni faqatgina bir tomondan qabul qiladi

C chiziqli ma'lumotlar tuzilmasi bo'lib, ma'lumotlarni kiritish va chiqarish uning ikki tomonlama amalga oshiriladi

D chiziqli ma'lumotlar tuzilmasi bo'lib, ma'lumotlarni kiritish va chiqarish uning faqat o'rtasiga amalga oshiriladi

82. Navbat bu...

A *shunday tuzilmaki, u yelementlar qo'shilishi bilan kengayib boradi va elementlarni faqatgina bir tomondan qabul qiladi

B chiziqli ma'lumotlar tuzilmasi bo'lib, ma'lumotlarni kiritish va chiqarish uning bir tomonidan amalga oshiriladi

C chiziqli ma'lumotlar tuzilmasi bo'lib, ma'lumotlarni kiritish va chiqarish uning ikki tomonlama amalga oshiriladi

D chiziqli ma'lumotlar tuzilmasi bo'lib, ma'lumotlarni kiritish va chiqarish uning faqat o'rtasiga amalga oshiriladi

83. Dek bu ...

A * chiziqli ma'lumotlar tuzilmasi bo'lib, ma'lumotlarni kiritish va chiqarish uning ikki tomonlama amalga oshiriladi

B shunday tuzilmaki, u yelementlar qo'shilishi bilan kengayib boradi va elementlarni faqatgina bir tomondan qabul qiladi

C chiziqli ma'lumotlar tuzilmasi bo'lib, ma'lumotlarni kiritish va chiqarish uning bir tomonidan amalga oshiriladi

D chiziqli ma'lumotlar tuzilmasi bo'lib, ma'lumotlarni kiritish va chiqarish uning faqat o'rtasiga amalga oshiriladi

Chiziqli va rekursiv ma'lumotlar tuzilmasi
Chiziqli ro'yxat. Dinamik tuzilmalar.

84. Qanday kalit so'zi yordamida nol havola (bo'sh manzil) belgilanadi?

A *NULL

B NaN

C ERROR

D EMP

85. Bir bog'lamli ro'yxatda nechta ko'rsatkichdan foydalaniladi?

A *1

B 2

C 3

D 4

86. Ikki bog'lamli ro'yxatda nechta ko'rsatkichdan foydalaniladi?

A *2

B 1

C 3

D 4

87. Dastur bajarilish jarayonida xotira hajmi aniqlanadigan yoki ularning soni ma'lum bo'ladigan tuzilmaga ... deyiladi.

A *Dinamik ma'lumotlar tuzilmasi

B Statik ma'lumotlar tuzilmasi

C Yarimstatik ma'lumotlar tuzilmasi

D Rekursiv ma'lumotlar tuzilmasi

88. C++ tilida standart andozalar kutubxonasi yordamida ro'yxatni qanday e'lon qilish mumkin?

A *list < int > S;

B queue < int > S;

C deque < int > S;

D stack < int > S;

```
struct List
{ int Data;
List * Next;
```

89. Bir bog'lamli ro'yxatlarda Next ko'rsatkichi nima uchun ishlatiladi?

A *Keyingi elementni ko'rsatish uchun

B Oldingi elementni ko'rsatish uchun

C Ro'yxatning boshini ko'rsatish uchun

D Ro'yxatning oxirini ko'rsatish uchun

```
struct List
{ int Data;
List * Next, * Prev;
};
```

90. Ikki bog'lamli ro'yxatlarda Next va Prev ko'rsatkichlari nima uchun ishlatiladi?
A *Keyingi va oldingi elementlarini ko'rsatish uchun
B Faqat oldingi va undan keyingi elementlarini ko'rsatish uchun
C Ro'yxatning boshini ko'rsatish uchun
D Ro'yxatning oxirini ko'rsatish uchun

Halqasimon ro'yxatlar

91. Halqasimon ro'yxatdan element o'chirilganda ...
A *ro'yxat bitta elementga qisqaradi
B ro'yxatda teshik hosil bo'ladi
C ro'yxat uziladi
D chiziqli ro'yxat hosil bo'ladi

92. Halqasimon ikki yo'nalishli ro'yxatda qaysi yo'nalishlar bo'yicha harakatlanish mumkin?
A *ikkala
B chapga
C o'nga
D ro'yxat oxiriga

93. Ro'yxat elementlarning ro'yxatlar bo'lishi mumkin tuzilma qanday nomlanadi
A *Lug'at
B Daraxt
C Graf
D Ro'yxat

Rekursiya

94. ... - obyektni mazkur obyektga murojaat qilish orqali aniqlashdir.
A *Rekursiya
B Algoritm
C Dastur
D Tuzilma

95. Ma'lumotlar tuzilmasi, tashkil qiluvchi elementlari qaysining o'xshash elementlar bo'lsa, u holda ... deyiladi.
A *Rekursiv ma'lumotlar tuzilmasi
B Dinamik ma'lumotlar tuzilmasi
C Yarimstatik ma'lumotlar tuzilmasi
D Statik ma'lumotlar tuzilmasi

96. Rekursiv funksiyalar apparati kim tomondan kashf qilingan?
A *A.Chyorch
B B.Mandelbrot
C A.Landis
D V.Velson

97. Rekursiya masalasini hal qiluvchi bosqichlari qanday nomlanadi?
A *Rekursiv triada
B Rekursiv algoritm
C Rekursiv munosabat
D Rekursiv obyekt

98. Rekursiv triada qaysi bosqichlardan iborat?
A *parametrizatsiya, rekursiya bazasi va dekompozitsiya
B aniqlash, chaqiruv, o'zgartirish
C oson, o'rta, qiyin
D qo'shish, ayirish, ko'paytirish

99. Rekursiv triadaning qaysi bosqichida masala shartini tasniflash va uni hal etish uchun parametrlar aniqlanadi?
A *parametrizatsiya
B rekursiya bazasi
C dekompozitsiya
D chaqiruv

100. Rekursiv triadaning qaysi bosqichida masala yechimi aniq bo'lgan trivial holat aniqlanadi, ya'ni bu holatda funksiyani o'ziga murojaat qilishi talab etilmaydi?
A *rekursiya bazasi
B dekompozitsiya
C parametrizatsiya
D chaqiruv

101. Rekursiv triadaning qaysi bosqichida umumiy holatni nisbatan ancha oddiy bo'lgan o'zgargan parametrli qism masalalar orqali ifodalaydi?
A *dekompozitsiya
B rekursiya bazasi
C parametrizatsiya
D chaqiruv

Chiziqsiz ma'lumotlar tuzilmasi
Daraxtsimon tuzilmalar

102. Daraxtsimon tuzilmadagi shunday elementga murojaat yo'qki, u... tugun hisoblanadi.
A *ildiz
B oraliq
C so'ngi
D ildiz bo'lmagan

103. Daraxtsimon tuzilmada boshqa elementlarga murojaat bo'lmasa, u... tugun hisoblanadi.
A *barg
B oraliq
C ildiz
D terminal

104. Qachon daraxt muvozanatlangan hisoblanadi?
A *agar uning chap va o'ng qism daraxtlari balandligi farqi 1tadan ko'p bo'lmasa
B agar uning chap va o'ng qism daraxtlari kengligi farqlanmasa
C agar uning chap va o'ng qism daraxtlari barglari teng sonli bo'lsa
D Agar uning oraliq tugunlari juft qiymatli bo'lsa

105. Chiziqsiz ma'lumotlar tuzilmasiga nimalar kiradi?
A *Daraxt, graf
B Stek, Dek, Navbat
C Yozuv, Jadval
D Graf. Vektor

106. Daraxt balandligi – bu ...
A *daraxt bosqichlari soni
B tugunlar soni
C oraliq elementlari soni
D barglar soni

107. Daraxt darajasi – bu ...
A *Daraxtga tegishli tugunning munosabatlar sonining maksimal qiymati
B Daraxtga tegishli tugunning munosabatlar sonining minimal qiymati
C Daraxt bosqichlari soni
D Tugunlar soni

108. Quyidagilarning qaysi biri minimal balandlikka ega daraxt hisoblanadi?
A *HEAP TREE
B BINARY TREE
C Red Black Tree
D 2-3 TREE

109. A C Binar daraxt uchun to'g'ri (yuqoridan pastga) ko'ruv amalining natijasini ko'rsating
A *BAC
B ACB
C ABC
D CAB

110. A C Binar daraxt uchun teskari (pastdan yuqoriga) ko'ruv amalining natijasini ko'rsating
A *ACB
B BAC
C ABC
D CAB

111. A C Binar daraxt uchun simmetrik (chapdan o'nga) ko'ruv amalining natijasini ko'rsating
A *ABC
B ACB
C BAC
D CAB

112. Daraxt qanday nomlanadi, agar uning chiqish darajasi ikkidan oshmasa?
A *Binar
B Ternar
C Tetradli
D Ko'pqatlamli

113. Qidiruv daraxtda nechta va qaysilar ko'ruv amallarini ifodalaydi?
A *Uchta (to'g'ri, teskari, simmetrik)
B Ikkita (eniga va tubiga)
C Ikkita (eniga va uzunasiga)
D Uchta (to'g'ri, teskari, akslanuvchi)

114. Kompyuter xotirasida binar daraxtni qanday ko'rinishda tasvirlash qulay?
A *bog'langan chiziqsiz ro'yxatlar
B massivlar
C jadvallar
D bog'langan chiziqli ro'yxatlar

115. Daraxt uzunligi – bu ...
A * tugunlar soni
B daraxt bosqichlari soni
C oraliq elementlari soni
D barglar soni

116. Chiziqsiz iyerarxik bog'langan ma'lumotlar tuzilmasi – bu ...
A *Daraxt
B Graf
C Lug'at
D Ro'yxat

117. Daraxt tugunlar ketma-ketligini tartiblangan holda chiqarish nima deyiladi?
A *Ko'ruv amali
B Daraxt uzunligi
C Daraxt balandligi
D Daraxt kengligi

118. Agar daraxtni tashkil etuvchi element (tugun)lardan faqat ikkita tugun bilan bog'langan bo'lsa, u holda bunday binar daraxt ... deyiladi.
A * to'liq
B Ikkilik
C minimal balandlikka ega daraxt
D muvozanatlangan

119. 56,34,60,23,40,65 sonlaridan hosil bo'lgan binar daraxt muvozanatlanganmi yoki yo'qmi?
A *xa
B yo'q
C har ikkalasi ham bo'lishi mumkin
D o'rtacha muvozanatlangan

120. Agar elementlar soni 100 ta bo'lsa, u holda minimal balandga ega daraxt balandligi nechiga teng bo'ladi?
A *7
B 8
C 9
D 10

121. Agar minimal balandga ega daraxt balandligi 10 ga teng bo'lsa, u holda maksimal elementlar soni nechiga teng bo'ladi?
A *1023
B 1024
C 2047
D 2048

122. Agar elementlar soni 10 ta bo'lsa, u holda minimal balandga ega daraxt balandligi nechiga teng bo'ladi?
A *4
B 1
C 3
D 2

123. 10,7, 12, 2, 5, 3, 11, 14 sonlaridan hosil qilingan binar daraxtda nechta oraliq tugun mavjud?

A *4
B 2
C 5
D 8

124. 10,7, 12, 2, 5, 3, 11, 14 sonlaridan hosil qilingan binar daraxtda nechta barg tugun mavjud?
A *3
B 2
C 5
D 8

125. 10,7, 12, 2, 5, 3, 11, 14 sonlaridan hosil qilingan binar daraxt balandligi nechiga teng?
A *5
B 3
C 4
D 8

126. 35, 27, 5,78, 29, 43 sonlaridan hosil qilingan binar daraxtda nechta barg tugun mavjud?
A *3
B 4
C 5
D 6

127. 35, 27, 5,78, 29, 43 sonlardan hosil qilingan binar daraxtda nechta oraliq tugun mavjud?
A *2
B 3
C 4
D 6

128. 35, 27, 5,78, 29, 43 sonlardan hosil qilingan binar daraxt balandligi nechiga teng?
A *3
B 4
C 2
D 1

Graf va uning turlari

129. Murrakab obyektlarning xususiyati va munosabatlarini aks ettiruvchi chiziqsiz ko'p bog'lamli dinamik tuzilma quyidagilardan qaysi biri hisoblanadi?
A *Graf
B Lug'at
C Daraxt
D Ro'yxat

130. Graf tuzilmasini matematik qanday ifodalash mumkin?
A *$G = \{V, E\}$
B $S = \{D, R\}$
C $A = \{D(1..n)\}$
D $HT = \{K, L, R\}$

131. Agar grafning munosabatlarini tasvirlashda qirralardan foydalanilsa, u holda graf ... deyiladi.
A *Yo'naltirilmagan
B Yo'naltirilgan
C Aralash
D Vaznga ega

132. Agar grafning munosabatlarini tasvirlashda yoylardan foydalanilsa, u holda graf ... deyiladi.
A *Yo'naltirilgan
B Yo'naltirilmagan
C Aralash
D Vaznga ega

133. Agar grafning munosabatlarini tasvirlashda yoy va qirralardan foydalanilsa, u holda graf ... deyiladi
A *Aralash
B Yo'naltirilmagan
C Yo'naltirilgan
D Vaznga ega

134. Agar grafning munosabatlariga og'irlik qiymati belgilansa, u holda graf ... deyiladi.
A *Vaznga ega
B Yo'naltirilmagan
C Yo'naltirilgan
D Aralash

135. Grafning tartibi nimaga teng?
A *Uchlar soniga
B Qirralar soniga
C Qirra va uchlar soniga
D Ilmoqlar soniga

136. Grafning o'lchami nimaga teng?
A *Qirralar soniga
B Uchlar soniga
C Qirra va uchlar soniga
D Ilmoqlar soniga

137. Graflarda tugun darajasi bu...
A *undan chiquvchi qirralar soni hisoblanadi
B undan chiquvchi tugunlar soni hisoblanadi
C undan chiquvchi qirralar o'rta arifmetik soni hisoblanadi
D undan chiquvchi qirralar o'rta geometrik soni hisoblanadi

138. Grafda nechta va quyidagilardan qaysilari ko'ruv amallarini ifodalaydi?
A *Ikkita (eniga va tubiga)
B Ikkita (eniga va uzunligiga)
C Uchta (to'g'ri, teskari, akslanuvchi)
D Uchta (to'g'ri, teskari, simmetrik)

139. Qanday konteyner yordamida grafda tubiga qarab ko'rishda qo'llaniladi?
A *stek
B navbat
C ro'yxat
D dek

140. Qanday konteyner yordamida grafda eniga qarab ko'rishda qo'llaniladi?
A *navbat
B stek
C ro'yxat
D dek

Kim tomondan va qaysi yilda graf tushunchasini kiritgan?
A *D.Kenig, 1936
B D.Ritchi, 1976
C A.Lovli, 1966
D Ch.Bebidj, 1946

141. Agar grafda boshi va oxiri bitta tugunda tutashadigan qirra mavjud bo'lsa, unga ... deyiladi.
A *Ilmoq
B Halqa
C Yo'l
D Daraja

142. Bironta tugundan boshqa bir tugungacha bo'lgan yonma-yon joylashgan tugunlar ketma-ketligidir, bu - ... deyiladi.
A *Yo'l
B Halqa
C Ilmoq
D Daraja

143. ... – bu boshi va oxiri tutashuvchi tugundan iborat yo'l.
A *Halqa
B Yo'l
C Ilmoq
D Daraja

144. Agar grafning to'yinganlik darajasi D>0.5 bo'lsa, u holda graf ... hsoblanadi.
A *To'yingan
B Siyrak
C Ikkilamchi
D To'liq

145. Agar grafning to'yinganlik darajasi D<0.5 bo'lsa, u holda graf ... hisoblanadi.
A *Siyrak
B To'yingan
C Ikkilamchi
D To'liq

146. Agar grafning to'yinganlil darajasi D=1 bo'lsa, u holda graf ... hisoblanadi.
A *To'liq
B Siyrak
C To'yingan
D Ikkilamchi

147. G grafni aks etishda *N* o'lchamli *A* kvadrat Matritsasi qanday nomlanadi?
A *Qo'shma Matritsa
B Munosabat Matritsasi
C Qo'shnilik ro'yxati
D Qirralar ro'yxati

148. G grafni aks etishda *NxM* o'lchamli *B* Matritsasi qanday nomlanadi?
A *Munosabat Matritsasi
B Qo'shma Matritsa
C Qo'shnilik ro'yxati
D Qirralar ro'yxati

149. G grafni aks etishda A[n] massiv bo'lib, massivning har bir elementi tugun bilan qo'shni tugunlar ro'yxati qanday nomlanadi?
A *Qo'shnilik ro'yxati
B Qo'shma Matritsa
C Munosabat Matritsasi
D Qirralar ro'yxati

150. G grafni aks etishda qo'shni tugunlar qirralaridan iborat chiziqli ro'yxat qanday nomlanadi?
A *Qirralar ro'yxati
B Qo'shnilik ro'yxati
C Qo'shma Matritsa
D Munosabat Matritsasi

151. Berilgan tugundan boshlab barcha tugunlarni ko'rib chiqish protsedurasi qanday nomlanadi?
A *ko'rikdan o'tkazish
B halqa
C yo'l
D daraja

152. Grafning D to'yinganlik darajasi nimaga teng?
A $*D = \frac{2m}{n(n-1)}$
B $D = \frac{n(n-1)}{2m}$
C $D = \frac{n}{m}$
D $D = \frac{m}{n}$

153. To'liq grafning qirralar soni qanday formula orqali hisoblanadi?
A $*m = \frac{n(n-1)}{2}$
B $m = n^2$
C $m = n!$
D $m = \sqrt{n}$

154. Yo'naltirilmagan grafning qo'shma Matritsasi to'g'ri berilgan javobini tanlang
A *
$$\begin{vmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{vmatrix}$$
B
$$\begin{vmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{vmatrix}$$
C
$$\begin{vmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{vmatrix}$$
D

155.
$$\begin{vmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{vmatrix}$$
yo'naltirilmagan grafning qo'shma Matritsasi berilgan. Grafning tartibi nechiga teng?
A *5
B 4
C 7
D 6

156.
$$\begin{vmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{vmatrix}$$
yo'naltirilmagan grafning qo'shma Matritsasi berilgan. Grafning o'lchami nechiga teng?
A *7
B 4
C 5
D 6

157. $\begin{vmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{vmatrix}$ yo'naltirilmagan grafning qo'shma Matritsasi berilgan.

Grafning to'yinganlik darajasi D ning qiymati nechiga teng?

A *0,7
B 0,3
C 1
D 0

158. $\begin{vmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{vmatrix}$ yo'naltirilmagan grafning qo'shma Matritsasi berilgan.

Grafning tartibi nechiga teng?

A *4
B 5
C 7
D 6

159. $\begin{vmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{vmatrix}$ yo'naltirilmagan grafning qo'shma Matritsasi berilgan.

Grafning o'lchami nechiga teng?

A *4
B 7
C 5
D 6

160. $\begin{vmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{vmatrix}$ yo'naltirilmagan grafning qo'shma Matritsasi berilgan.

Grafning to'yinganlik darajasi D nechiga teng?

A *0,66
B 0,33
C 1
D 0,5

Tuzilma ustida amal bajarish algoritmlari
Qidiruv algoritmlari

161. Qidiruvni vazifasi nimadan iborat?

A *berilgan argumentga mos keluvchi ma'lumotlarni massiv ichidan topish
B massivda ma'lumot yo'qligini aniqlash
C ma'lumotlar yordamida argumentni topish
D ma'lumot yordamida eng kichik elementni topish

162. Berilgan argumentga mos keluvchi ma'lumotlarni massiv ichidan topish amali qanday ataladi?

A *Qidiruv
B Saralash
C Algoritmlash
D Uslubiyot

163. Jadvalning tuzilmasiga qarab nechta qidiruv usullari mavjud?

A *4
B 5
C 6
D 7

164. Chiziqli qidiruv g'oyasi nimadan iborat?

A *har bir element ketma-ket ko'rib chiqiladi
B elementlar ketma-ket jadval o'rtasidan boshlab ko'rib chiqiladi
C elementlarni ko'rib chiqish ketma-ket ravishda boshidan oxirigacha va aksincha, 2 ta element tashlab qaraladi
D binar daraxt barcha tugunlari ko'rib chiqiladi

165. Transpozitsiya usulining ma'nosi nima?

A *Topilgan element o'zidan oldinda turgan element bilan almashtirila di.
B Topilgan element o'zidan keyingi turgan element bilan almashtirila di.
C Topilgan element tuzilmaning 1-elementi bilan almashtirila di.
D Topilgan element tuzilmaning oxirgi elementi bilan almashtirila di.

166. O'rinlashtirish usulini ma'nosi nimadan iborat?

A *topilgan element ro'yxat boshiga joylashtiriladi
B topilgan element ro'yxat oxiriga joylashtiriladi
C topilgan element o'zidan keyingi element bilan o'rin almashtiriladi
D qo'shni elementlar o'rni almashtiriladi

167. Noyob kalit nima?

A *agar jadvalda kaliti mazkur kalitga teng ma'lumot yagona bo'lsa
B agar ikkita ma'lumot qiymatlari yig'indisi kalitga teng bo'lsa
C agar jadvalda bunday kalitli element mavjud bo'lmasa
D agar ikkita ma'lumot qiymatlari farqi kalitga teng bo'lsa

168. Katta O notatsiyada belgilangan chiziqli qidiruv samaradorligini ko'rsating

A *O(N)
B $O(\log_2(N))$
C O(1)
D $O(\sqrt{N})$

169. Katta O notatsiyada belgilangan binar qidiruv samaradorligini ko'rsating

A *$O(\log_2(N))$
B O(N)
C O(1)
D $O(\sqrt{N})$

170. Katta O notatsiyada belgilangan indeksli ketma-ket qidiruv algoritmi samaradorligini ko'rsating

A *$O(\sqrt{N})$
B O(N)
C O(1)
D $O(\log_2(N))$

171. Katta O notatsiyada belgilangan xeshlash va rexeshlash qidiruv samaradorligini ko'rsating
A *O(1)
B O(N)
C $O(\log_2(N))$
D $O(\sqrt{N})$

172. Ketma-ket qidiruv algoritmi tartibi qanday?
A *Chiziqli
B Logarifmik
C Konstantali
D Eksponensial

173. Binar qidiruv algoritm tartibi qanday?
A *Logarifmik
B Chiziqli
C Konstantali
D Eksponensial

174. Xeshlashtirish algoritm tartibi qanday?
A *Konstantali
B Chiziqli
C Logarifmik
D Eksponensial

175. Chiziqli qidiruv qachon samarali hisoblanadi?
A *massiv va ro'yxatda
B dekda
C daraxtda
D navbatda

176. Ketma-ket yoki chiziqli qidiruv – bu ...
A *Ma'lumotlar butun jadval bo'yicha operativ xotirada kichik adresdan boshlab, to katta adresgacha ketma-ket qarab chiqiladi
B Indekslar jadvalidan guruh topiladi, va unda ko'rsatilgan mos chegaralarda chiziqli algoritm oshiriladi
C Berilgan massiv o'rtasidagi element olinadi, ya'ni $m = (L+R)/2$, va u qidiruv argumenti bilan taqqoslanadi. Topilmasa chegaralar mos ravishda o'zgartiriladi
D Funksiya yerdamida xesh-jadval to'ldiriladi va undan qidiriladi

177. Indeksli-ketma-ket qidiruv – bu ...
A *Indekslar jadvalidan guruh topiladi, va unda ko'rsatilgan mos chegaralarda chiziqli algoritm oshiriladi
B Ma'lumotlar butun jadval bo'yicha operativ xotirada kichik adresdan boshlab, to katta adresgacha ketma-ket qarab chiqiladi
C Berilgan massiv o'rtasidagi element olinadi, ya'ni $m = (L+R)/2$, va u qidiruv argumenti bilan taqqoslanadi. Topilmasa chegaralar mos ravishda o'zgartiriladi
D Funksiya yerdamida xesh-jadval to'ldiriladi va undan qidiriladi

178. Binar qidiruv – bu ...
A *Berilgan massiv o'rtasidagi element olinadi, ya'ni $m = (L+R)/2$, va u qidiruv argumenti bilan taqqoslanadi. Topilmasa chegaralar mos ravishda o'zgartiriladi
B Ma'lumotlar butun jadval bo'yicha operativ xotirada kichik adresdan boshlab, to katta adresgacha ketma-ket qarab chiqiladi
C Indekslar jadvalidan guruh topiladi, va unda ko'rsatilgan mos chegaralarda chiziqli algoritm oshiriladi
D Funksiya yerdamida xesh-jadval to'ldiriladi va undan qidiriladi

179. Xeshlash – bu ...
A *Funksiya yerdamida xesh-jadval to'ldiriladi va undan qidiriladi
B Ma'lumotlar butun jadval bo'yicha operativ xotirada kichik adresdan boshlab, to katta adresgacha ketma-ket qarab chiqiladi
C Berilgan massiv o'rtasidagi element olinadi, ya'ni $m = (L+R)/2$, va u qidiruv argumenti bilan taqqoslanadi. Topilmasa chegaralar mos ravishda o'zgartiriladi
D Indekslar jadvalidan guruh topiladi, va unda ko'rsatilgan mos chegaralarda chiziqli algoritm oshiriladi

Saralash algoritmlari

180. Operativ xotirada bajariladigan saralash qanday ataladi?
A *ichki saralash
B to'liq saralash
C qo'shish orqali saralash
D adreslar jadvalini saralash

181. Saralash usullari orasidan noto'g'risini toping.
A *dinamik
B yaxshilangan
C logarifmik
D qat'iy

182. Saralashning qaysi usullari $\Theta(N^2)$ kalitlarni taqqoslash tartibiga ega?
A *qat'iy
B binar
C yaxshilangan
D logarifmik

183. Berilgan to'plam elementlarini biror bir tartibda joylashtirish jarayoni nima deyiladi?
A *Saralash
B Qidiruv
C Algoritmlash
D Uslubiyot

184. Saralash usuli ... deyiladi, agar saralash jarayonida bir xil kalitli elementlar nisbiy joylashuvi o'zgarmasa.
A *Turg'un (stable)
B Murakkab (difficult)
C Oddiy (typical)
D Turg'un emas (unstable)

185. Qo'yish orqali saralash g'oyasini ko'rsating.
A *Obyektlar hayolan tayyor a(1),...,a(i-1) va boshlang'ich ketma-ketliklarga bo'linadi. Har bir qadamda (i=2 dan boshlab) boshlang'ich ketma-ketlikdan i-chi element ajratib olinib tayyor ketma-ketlikning kerakli joyiga qo'shiladi.
B Berilgan obyektlar ichidan eng kichik kalitga ega element tanlanadi.Ushbu element boshlang'ich ketma-ketlikdagi birinchi element bilan o'rin almashadi. Undan keyin ushbu jarayon qolgan elementlarda amalga oshiriladi.

n - 1 marta massivda quyidan yuqoriga qarab yurib kalitlar jufti-jufti bilan taqqoslanadi. Agar pastki kalit qiymati yuqoridagi jufti kalitidan kichik bo'lsa, u holda ular o'rni almashtiriladi.

D Boshlang'ich ketma-ketlikning har r o'rinda joylashgan elementlari guruhlanib, har bir guruh alohida qo'shish usuli orqali saralanadi.

186. Tanlash orqali saralash g'oyasini ko'rsating.

A *Berilgan obyektlar ichidan eng kichik kalitga ega element tanlanadi.Ushbu element boshlang'ich ketma-ketlikdagi birinchi element bilan o'rin almashadi. Undan keyin ushbu jarayon qolgan elementlarda amalga oshiriladi.

B n - 1 marta massivda quyidan yuqoriga qarab yurib kalitlar jufti-jufti bilan taqqoslanadi. Agar pastki kalit qiymati yuqoridagi jufti kalitidan kichik bo'lsa, u holda ular o'rni almashtiriladi.

C Boshlang'ich ketma-ketlikning har r o'rinda joylashgan elementlari guruhlanib, har bir guruh alohida qo'shish usuli orqali saralanadi.

D Obyektlar hayolan tayyor a(1),...,a(i-1) va boshlang'ich ketma-ketliklarga bo'linadi. Har bir qadamda (i=2 dan boshlab) boshlang'ich ketma-ketlikdan i-chi element ajratib olinib tayyor ketma-ketlikning kerakli joyiga qo'shiladi.

187. Almashtirish orqali saralash g'oyasini ko'rsating.

A *n - 1 marta massivda quyidan yuqoriga qarab yurib kalitlar jufti-jufti bilan taqqoslanadi. Agar pastki kalit qiymati yuqoridagi jufti kalitidan kichik bo'lsa, u holda ular o'rni almashtiriladi.

B Obyektlar hayolan tayyor a(1),...,a(i-1) va boshlang'ich ketma-ketliklarga bo'linadi. Har bir qadamda (i=2 dan boshlab) boshlang'ich ketma-ketlikdan i-chi element ajratib olinib tayyor ketma-ketlikning kerakli joyiga qo'shiladi.

C Berilgan obyektlar ichidan eng kichik kalitga ega element tanlanadi.Ushbu element boshlang'ich ketma-ketlikdagi birinchi element bilan o'rin almashadi. Undan keyin ushbu jarayon qolgan elementlarda amalga oshiriladi.

D Boshlang'ich ketma-ketlikning har r o'rinda joylashgan elementlari guruhlanib, har bir guruh alohida qo'shish usuli orqali saralanadi.

188. QuickSort usulining algoritm tartibini ko'rsating.

A *Logarifmik
B Chiziqli
C Kvadratik
D Differensial

189. Qat'iy usullarning algoritmlar tartibini ko'rsating.

A *Kvadratik
B Kubik
C Logarifmik
D Differensial

190. Saralash samaradorligini qaysi mezonlar yordamida aniqlanadi?

A *taqqoslashlar va almashtirishlar soni
B dastur yozishga ketgan vaqt
C ishlatilayotgan identifikatorlar soni va turlari
D amallar soni

191. Qanday saralash usullari qat'iy usullar hisoblanadi?

A *to'g'ridan-to'g'ri qo'shish; to'g'ridan-to'g'ri tanlash; to'g'ridan-to'g'ri almashtirish.
B Tez saralash; Shella saralashi; Birlashtirish saralashi.
C Birlashtirish saralashi; to'g'ridan-to'g'ri tanlash; to'g'ridan-to'g'ri almashtirish.
D Tez saralash, to'g'ridan-to'g'ri tanlash; to'g'ridan-to'g'ri almashtirish.

192. Qanday saralash usullari yaxshilangan usullar deb belgilangan?

A *Tez saralash; Shella saralashi; Birlashtirish saralashi.
B to'g'ridan-to'g'ri qo'shish; to'g'ridan-to'g'ri tanlash; to'g'ridan-to'g'ri almashtirish.
C Birlashtirish saralashi; to'g'ridan-to'g'ri tanlash; to'g'ridan-to'g'ri almashtirish.
D Tez saralash, to'g'ridan-to'g'ri tanlash; to'g'ridan-to'g'ri almashtirish.

Tashqi xotira ustida amal bajarish algoritmlari

193. Bu ismga ega obyekt bo'lib, shu ism orqali ichidagi ma'lumotlar bilan ishlovchi obyektdir.

A *Fayl
B Katalog
C Xotira
D Ma'lumot tashuvchi

194. Faylni aniqlovchi bir nechta ketma-ket bilgilar faylning ... deyiladi.

A *Ismi
B Kengaytmasi
C Atributi
D Yo'li

195. Faylning ma'lumotlar tarkibini va dasturiy ta`minotini aniqlovchi ko'rsatkichni ko'rsating.

A *Kengaytma
B Ism
C Atribut
D Yo'l

196. C++ tilida qanday klass yordamida faylga yozish jarayoni boshqariladi?

A *ofstream
B ifstream
C input_file
D output_file

197. Qanday klass yordamida fayldan o'qish jarayoni boshqariladi?

A *ifstream
B ofstream
C input_file
D output_file

198. Dasturda fayl ma'lumotlari ustida amal bajarishda unga qanday murojaat qilish mumkin?

A *ko'rsatkichlar yordamida
B Yangi yaratilgan nostandart tipdagi o'zgaruvchi orqali
C Faylni o'zi bevosita ekranda ochilib amal bajariladi

D Dasturda fayl oobyektlariga murojaat yo'q

199. C++ tilining qaysi kutubxonasida faylga kiritish/chiqarish oqimlari amalga oshiriladi?

A *fstream
B filestream
C streamfile
D iostream

MUNDARIJA

Ergashev A.Q., Akbaraliyev B.B., Yusupova Z.Dj.

MA'LUMOTLAR TUZILMASI VA ALGORITMLAR

Muhammad Al-Xorazmiy nomidagi Toshkent axborot texnologiyalari universiteti tomonidan o'quv qo'llanma sifatida tavsiya etilgan

Toshkent - "METODIST NASHRIYOTI" - 2024

Muharrir: Bakirov Nurmuhammad

Texnik muharrir: Tashatov Farrux
Musahhih: Saidova Nurshoda
Dizayner: Ochilova Zarnigor

Bosishga 17.05.2024.da ruxsat etildi.
Bichimi 60x90. "Times New Roman" garniturasi.
Ofset bosma usulida bosildi.
Shartli bosma tabog'i 12. Nashr bosma tabog'i 11.75.
Adadi 300 nusxa.

"METODIST NASHRIYOTI" MCHJ matbaa bo'limida chop etildi.
Manzil: Toshkent shahri, Shota Rustaveli 2-vagon tor ko'chasi, 1-uy.

+99893 552-11-21

ISBN 978-9910-03-142-7