DARAXTSIMON TUZILMALAR

Daraxt ko’rinishidagi ma’lumotlar tuzilmasi haqida umumiy tushunchalar.

Uzellar (elementlar) va ularning munosabatlaridan iborat elementlar to’plamining ierarxik tuzilmasiga daraxtsimon ma’lumotlar tuzilmasi deyiladi.

Daraxt – bu shunday chiziqsiz bog’langan ma’lumotlar tuzilmasiki, u quyidagi belgilari bilan tavsiflanadi:

- daraxtda shunday bitta element borki, unga boshqa elementlardan murojaat yo’q. Bu element daraxt ildizi deyiladi;
- daraxtda ixtiyoriy element chekli sondagi ko’rsatkichlar yordamida boshqa tugunlarga murojaat qilishi mumkin;
- daraxtning har bir elementi faqatgina o’zidan oldingi kelgan bitta element bilan bog’langan.

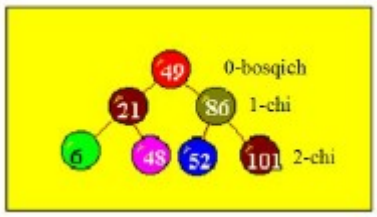
Binar daraxtlarni qurish

Binar daraxtda har bir tugun-elementdan ko’pi bilan 2 ta shox chiqadi. Daraxtlarni xotirada tasvirlashda uning ildizini ko’rsatuvchi ko’rsatkich berilishi kerak. Daraxtlarni kompyuter xotirasida tasvirlanishiga ko’ra har bir element (binar daraxt tuguni) to’rtta maydonga ega yozuv shaklida bo’ladi, ya’ni kalit maydon, informatsion maydon, ushbu elementni o’ngida va chapida joylashgan elementlarning xotiradagi adreslari saqlanadigan maydonlar.

Shuni esda tutish lozimki, daraxt hosil qilinayotganda, otaga nisbatan chap tomondagi o’g’il qiymati kichik kalitga, o’ng tomondagi o’g’il esa katta qiymatli kalitga ega bo’ladi. Har safar daraxtga yangi element kelib qo’shilayotganda u avvalambor daraxt ildizi bilan solishtiriladi. Agar element ildiz kalit qiymatidan kichik bo’lsa, uning chap shoxiga, aks holda o’ng shoxiga o’tiladi. Agar o’tib ketilgan shoxda tugun mavjud bo’lsa, ushbu tugun bilan ham solishtirish amalga oshiriladi, aks holda, ya’ni u shoxda tugun mavjud bo’lmasa, bu element shu tugunga joylashtiriladi.

Masalan, daraxt tugunlari quyidagi qiymatlarga ega 6, 21, 48, 49, 52, 86, 101.

U holda binar daraxt ko’rinishi quyidagi 4.1-rasmdagidek bo’ladi:



4.1-rasm. Binar daraxt ko’rinishi

Natijada, o’ng va chap qism daraxtlari bir xil bosqichli tartiblangan binar

daraxt hosil qildik. Agar daraxtning o’ng va chap qism daraxtlari bosqichlarining

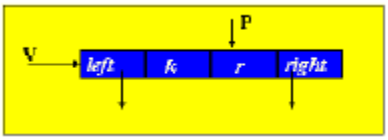
farqi birdan kichik bo’lsa, bunday daraxt ideal muvozanatlangan daraxt deyiladi.

Yuqorida hosil qilgan binar daraxtimiz ideal muvozanatlangan daraxtga misol bo’ladi. Daraxtni muvozanatlash algoritmini sal keyinroq ko’rib chiqamiz. Undan oldin binar daraxtni yaratish algoritmini o’rganamiz.

Binar daraxt yaratish funksiyasi

Binar daraxtni hosil qilish uchun kompyuter xotirasida elementlar quyidagi

4.2-rasmdagidek toifada bo’lishi lozim.



4.2-rasm. Binar daraxt elementining tuzilishi p – yangi element ko’rsatkichi

next, last – ishchi ko’rsatkichlar, ya’ni joriy elementdan keyingi va oldingi

elementlar ko’rsatkichlari

r=rec – element haqidagi birorta ma’lumot yoziladigan maydon

k=key – elementning unikal kalit maydoni

left=NULL – joriy elementning chap tomonida joylashgan element adresi

right=NULL – joriy elementning o’ng tomonida joylashgan element adresi.

Dastlab yangi element hosil qilinayotganda bu ikkala maydonning qiymati 0

ga teng bo’ladi.

tree – daraxt ildizi ko’rsatkichi

n – daraxtdagi elementlar soni

Boshida birinchi kalit qiymat va yozuv maydoni ma’lumotlari kiritiladi,

element hosil qilinadi va u daraxt ildiziga joylashadi, ya’ni tree ga o’zlashtiriladi.

Har bir hosil qilingan yangi elementning left va right maydonlari qiymati 0 ga

tenglashtiriladi. Chunki bu element daraxtga terminal tugun sifatida joylashtiriladi,

hali uning farzand tugunlari mavjud emas. Qolgan elementlar ham shu kabi hosilqilinib, kerakli joyga joylashtiriladi. Ya'ni kalit qiymati ildiz kalit qiymatidan

kichik bo'lgan elementlar chap shoxga, katta elementlar o'ng tomonga

joylashtiriladi. Bunda agar yangi element birorta elementning u yoki bu tomoniga

joylashishi kerak bo'lsa, mos ravishda left yoki right maydonlarga yangi element

adresi yozib qo'yiladi.

Binar daraxtni hosil qilishda har bir element yuqorida ko'rsatilgan toifada

bo'lishi kerak. Lekin hozir biz o'zlashtirish osonroq va tushunarli bo'lishi uchun

key va rec maydonlarni bitta qilib info maydon deb ishlatamiz.

4.3-rasm. Binar daraxt elementining tuzilishi

Ushbu toifada element hosil qilish uchun oldin bu toifani yaratib olishimiz

kerak. Uni turli usullar bilan amalga oshirish mumkin. Masalan, node nomli yangi

toifa yaratamiz:

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

Endi yuqoridagi belgilashlarda keltirilgan ko'rsatkichlarni shu toifada

yaratib olamiz.

```
  node *tree=NULL;
  node *next=NULL;
  int n,key; cout<<"n=";cin>>n;
```

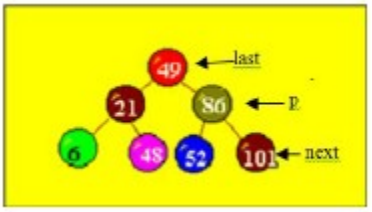
Nechta element (n) kiritilishini aniqlab oldik va endi har bir element

qiymatini kiritib, binar daraxt tuzishni boshlaymiz.

```
     for(int i=0;i<n;i++){
node *p=new node;
```

```
        node *last=new node;
        cin>>key;
    p->info=key;
 p->left=NULL;
        p->right=NULL;
            if(i==0){ tree=p; next=tree;sontinue;}
            next=tree;
                  while(1){    last=next;
                      if(p->info<next->info) next=next->left; else next=next->right;
                      if(next==NULL) break;
                      }
            if(p->info<last->info) last->left=p; else last->right=p;
            }
```

Bu yerda p hali aytganimizdek, kiritilgan kalitga mos hosil qilingan yangi element ko'rsatkichi, next yangi element joylashishi kerak bo'lgan joyga olib boradigan shox adresi ko'rsatkichi, ya'ni u har doim p dan bitta qadam oldinda yuradi, last esa ko'rilayotgan element kimning avlodi ekanligini bildiradi, ya'ni u har doim p dan bir qadam orqada yuradi (4.4-rasm).



4.4-rasm. Binar daraxt elementlarini belgilash

Shunday qilib binar daraxtini ham yaratib oldik. Endigi masala uni ekranda tasvirlash kerak, ya'ni u ko'rikdan o'tkaziladi yoki vizuallashtirsa ham bo'ladi. Binar daraxtlari ko'rigini uchta tamoyili mavjud. Ularni berilgan daraxt misolida ko'rib chiqaylik:

1) Yuqoridan pastga ko'rik (daraxt ildizini qism daraxtlarga nisbatan oldinroq ko'rikdan o'tkaziladi): A, B, C ;

2) Chapdan o'ngga: B, A, C ;

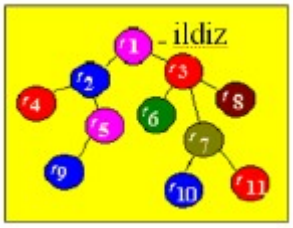
3) Quyidan yuqoriga (ildiz qism daraxtlardan keyin ko'riladi): B, C, A .

Daraxt ko’rigi ko’pincha ikkinchi usul bilan, ya’ni tugunlarga kirish ularning kalit qiymatlarini o’sish tartibida amalga oshiriladi.

Daraxt ko’rigining rekursiv funksiyalari

```
1.  int pretrave(node *tree){
        if(tree!=NULL) {int a=0,b=0;
                    if(tree->left!=NULL) a=tree->left->info;
                    if(tree->right!=NULL) b=tree->right->info;
                    cout<<tree->info<<"   - chapida "<<a<<"   - o’ngida
"<<b<<"   \n";
                    pretrave(tree->left);
                pretrave(tree->right);
                    }
        return 0;  };
2.  int intrave(node *tree){
        if(tree!=NULL) {
                    intrave(tree->left);
                    cout<<tree->info;
                    intrave(tree->right);
                    }
        return 0;
        };
3.  int postrave(node *tree){
        if(tree!=NULL) {
                    postrave(tree->left);
                    postrave(tree->right);
                    cout<<tree->info;
                    }
        return 0;    };
```

Daraxtning har bir tuguni 4.6-rasmdagidek oraliq (2, 3, 5, 7 elementlar) yoki



terminal (daraxt “barg”i) (4, 9, 10, 11, 8, 6 elementlar) bo’lishi mumkin.

4.6-rasm. Daraxtsimon tuzilma

1. Agar tugunning otasi yo’q bo’lsa, bu tugun ildiz hisoblanadi. Buni aniqlash uchun dastur kodini keltiramiz. Dasturda p izlanayotgan tugun. if(p==tree) cout<<”bu tugun ildiz ekan”;

else cout<<”bu tugun ildiz emas”;

2. Biz izlayotgan element daraxtda oraliq tugun ekanligini tekshirish uchun uning yoki o’ng shoxi, yoki chap shoxi, yoki ikkalasiyam mavjudligini tekshirish kerak. Agar ikkala shoxi NULL dan farqli bo’lsa, bu 2 ta farzandga ega oraliq tugun hisoblanadi, yoki ikkalasidan bittasi NULL ga teng bo’lsa, bu tugun 1 ta farzandga ega oraliq tugun hisoblanadi. Berilgan p element daraxtning oraliq tugun ekanligini aniqlash dastur kodini keltiramiz.

```
if(p!=tree){
if((p->left!=NULL)&&(p->right!=NULL)) cout<<”bu tugun 2 ta farzandga
ega oraliq tugun”;
else if((p->left!=NULL)||(p->right!=NULL) cout<<”bu 1 ta farzandga ega
oraliq tugun”;
} else cout<<”bu tugun oraliq tugun emas”;
```

3. Biz izlayotgan tugun terminal tugunligini tekshirishni ko’rib chiqamiz. Agar tugunning har ikkala shoxi NULL ga teng bo’lsa, bu terminal tugun hisoblanadi. Dastur kodini keltiramiz.

```
if((p->left==NULL)&&(p->right==NULL)) cout<<”bu tugun terminal
tugun”;
else cout<<”bu terminal tugun emas”;
```

Binar daraxt bo’yicha qidiruv funksiyasi

Mazkur funksiyaning vazifasi shundan iboratki, u berilgan kalit bo’yicha

daraxt tuguni qidiruvini amalga oshiradi. Qidiruv operatsiyasining davomiyligi daraxt tuzilishiga bog'liq bo'ladi. Haqiqatdan, agar elementlar daraxtga kalit qiymatlari o'sish (kamayish) tartibida kelib tushgan bo'lsa, u holda daraxt 4.7-rasmdagidek bir tomonga yo'nalgan ro'yhat hosil qiladi (chiqish darajasi bir bo'ladi, ya'ni yagona shoxga ega), masalan: Bu holda daraxtda qidiruv vaqti, bir tomonlama yo'naltirilgan ro'yhatdagi kabi bo'lib, o'rtacha qarab chiqishlar soni N/2 bo'ladi. Agar daraxt muvozanatlangan bo'lsa, u holda qidiruv eng samarali natija beradi. Bu holda qidiruv log2 N dan ko'p bo'lmagan elementlarni ko'rib chiqadi.

Qidiruv funksiyasini ko'rib chiqamiz. search fuksiyasi daraxtdan key kalitga mos elementning adresini aniqlaydi.

```
int search(node *tree, int key){
node *next; next=tree;
while(next!=NULL)
   {       if (next->info==key){cout<<"Binar daraxtda "<<key<<" mavjud";
return next; }
        if (next->info>key) next=next->left;
           else next=next->right;
    }
cout<<"tuzilmada izlangan element yo'q!!!"<<endl;
return 0;
}
```

Daraxtga yangi element qo'shish funksiyasi

Daraxtga biror bir elementni qo'shishdan oldin daraxtda berilgan kalit bo'yicha qidiruvni amalga oshirish lozim bo'ladi. Agar berilgan kalitga teng kalit mavjud bo'lsa, u holda dastur o'z ishini yakunlaydi, aks holda daraxtga element qo'shish amalga oshiriladi. Daraxtga yangi yozuvni kiritish uchun, avvalo daraxtning shunday tugunini topish lozimki, natijada mazkur tugunga yangi element qo'shish mumkin bo'lsin. Kerakli tugunni qidirish algoritmi ham xuddi berilgan kalit bo'yicha tugunni topish algoritmi kabi bo'ladi.

Daraxtda qo’shilayotgan element kalitiga teng kalitli element yo’q bo’lgan holda elementni tuzilmaga qo’shish funksiyasini keltirib o’tamiz.

```
Node *q=NULL;
Node *p=tree;
while(p!=NULL){
    q=p;
    if(key==p->key){
       search=p;
       return 0;
       }
    If(key<p->key)  p=p->left;
      else p=p->right;
  }
```

Berilgan kalitga teng tugun topilmadi, element qo’shish talab qilinadi. Ota bo’lishi mumkin tugunga q ko’rsatkich beriladi, elementning o’zi esa yangi nomli ko’rsatkichi bilan beriladi.

```
node *q=new node;
```

Qo’yilayotgan yangi element chap yoki o’ng o’g’il bo’lishini aniqlash lozim.

```
 If(key<q->key)   q->left=yangi;
    else q->right=yangi;
search=yangi;
return 0;
```

Binar daraxtdan elementni o’chirish funksiyasi

Tugunni o’chirib tashlash natijasida daraxtning tartiblanganligi buzilmasligi lozim. Tugun daraxtda o’chirilayotganda 3 xil variant bo’lishi mumkin:

1) Topilgan tugun terminal (barg). Bu holatda tugun otasining qaysi tomonida turgan bo’lsa, otasining o’sha tomonidagi shoxi o’chiriladi va tugunning xotirada joylashgan sohasi tozalanadi.

2) Topilgan tugun faqatgina bitta o’g’ilga ega. U holda o’g’il ota o’rniga joylashtiriladi.

3) O’chirilayotgan tugun ikkita o’g’ilga ega. Bunday holatda shunday qism daraxtlar zvenosini topish lozimki, uni o’chirilayotgan tugun o’rniga qo’yish mumkin bo’lsin. Bunday zveno har doim mavjud bo’ladi:

- bu yoki chap qism daraxtning eng o’ng tomondagi elementi (ushbu zvenoga erishish uchun keyingi uchiga chap shox orqali o’tib, navbatdagi uchlariga esa, murojaat NULL bo’lmaguncha, faqatgina o’ng shoxlari orqali o’tish zarur);
- yoki o’ng qism daraxtning eng chap elementi (ushbu zvenoga erishish uchun keyingi uchiga o’ng shox orqali o’tib, navbatdagi uchlariga esa, murojaat NULL bo’lmaguncha, faqatgina chap shoxlari orqali o’tish zarur).

O’chirlayotgan element chap qism daraxtining eng o’ngidagi element o’chirilayotgan element uchun merosxo’r bo’ladi (12 uchun – 11 bo’ladi). Merosxo’r esa o’ng qism daraxtning eng chapidagi tuguni (12 uchun - 13).

Merosxo’rni topish algoritmini ishlab chiqaylik (4.8-rasmga qarang).

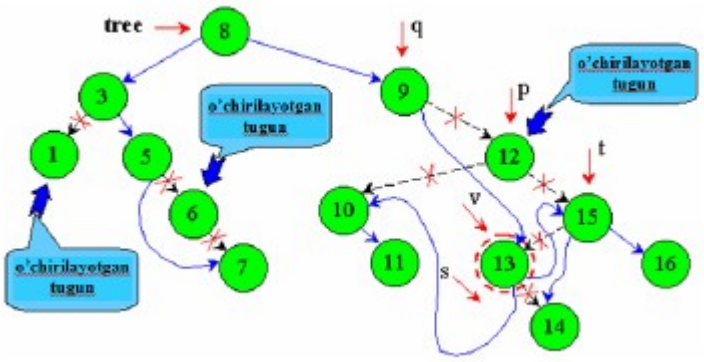
p – ishchi ko’rsatkich;

q - p dan bir qadam orqadagi ko’rsatkich;

v – o’chirilayotgan tugun merosxo’rini ko’rsatadi;

t – v dan bir qadam orqada yuradi;

s - v dan bir qadam oldinda yuradi (chap o’g’ilni yoki bo’sh joyni ko’rsatib boradi).



4.8-rasm. Binar daraxtdan oraliq tugunni o’chirich tartibi

Yuqoridagi daraxt bo’yicha qaraydigan bo’lsak, oxir oqibatda, v ko’rsatkich 13 tugunni, s esa bo’sh joyni ko’rsatishi lozim.

1) Elementni qidirish funksiyasi orqali o’chirilayotgan elementni topamiz. p

ko’rsatkich o’chirilayotgan elementni ko’rsatadi.

2) O’chiriladigan elementning o’rniga qo’yiluvchi tugunga v ko’rsatkich qo’yamiz.

```
node *del(node *tree,int key){
node *p=new node;
node *next=tree;
node *q=NULL;
while(next!=NULL)
{     if (next->info==key){cout<<"Binar daraxtda "<<key<<"
Mavjud"<<endl;   p=next;break; }
   if (next->info>key){ q=next; next=next->left; }
      else  {q=next;next=next->right;}
}
   if(next==NULL) cout<<"tuzilmada izlangan element yo’q!!!"<<endl;
   node *v=NULL,*t=NULL,*s=NULL;
   if(p->left==NULL)v=p->right;
   else  if(p->right==NULL)   v=p->left;
   if((p->left!=NULL)&&(p->right!=NULL)){t=p; v=p->right; s=v->left;}
   while(s!=NULL){
         t=v;
         v=s;
         s=v->left;
         }
   if((t!=NULL)&&(t!=p)){
     t->left=v->right;
     v->right=p->right;
     v->left=p->left;
     }
    if(t==p)    v->left=p->left;
    if(q==NULL){
```

```
      cout<<v->info<<" ildiz\n";
      tree=v;
      delete(p);
      return tree;
      }
   if(p==q->left)
       q->left=v;
   else q->right=v;
   delete(p); // o'chirilgan element joylashgan xotira yacheykasini tozalash
  return tree;
}
```

Daraxtni muvozanatlash algoritmi

Binar daraxt muvozanatlangan yoki AVL-muvozanatlangan bo'lishi mumkin. Daraxt AVL-muvozanatlangan (1962 yil sovet olimlari Adelson, Velsk Georgiy Maksimovich va Landis Yevgeniya Mihaylovichlar tomonidan taklif qilingan) deyiladi, agar daraxtdagi har bir tugunning chap va o'ng qismdaraxtlari balandliklari farqi 1 tadan ko'p bo'lmasa.

Berilgan butun sonlar – kalitlar ketma-ketligidan binar daraxt yaratib olamiz va uni muvozanatlaymiz. Daraxtni muvozanatlashdan maqsad, bunday daraxtga yangi element kiritish va daraxtdan element izlash algoritmi samaradorligini oshirishdan iborat, ya'ni bu amallarni bajarishdagi solishtirishlar soni kamayadi.

Binar daraxtni muvozanatlash algoritmi quyidagicha bo'ladi.

Algoritm

1. Binar daraxtni yaratib olamiz.

2. Binar daraxtni chapdan o'ngga ko'rikdan o'tkazamiz va tugunlarning info maydonlaridan a[..] massiv hosil qilamiz. Tabiiyki, massiv o'sish bo'yicha tartiblangan bo'ladi.

Muvozanatlangan daraxtning tugunlarini belgilash uchun massivni ko'riladigan oralig'ini belgilab olamiz, ya'ni start=0 va end=n-1.

3. Massivning ko'rilayotgan oralig'i o'rtasida joylashgan elementni, ya'ni

mid=(start+end)/2 va a[mid] ni muvozanatlangan daraxtning tuguni qilib olinadi.

Agar ko'rilayotgan oraliqda bitta ham element qolmagan bo'lsa, ya'ni start>end bo'lsa, bajarilish joriy seansdan keyingisiga uzatiladi.

4. Ko'rilayotgan tugunning chap qismdaraxtini hosil qilish uchun massivning ko'rilayotgan oralig'ining 1-yarmini olamiz, ya'ni start=0 va end=mid-1. 3-5 qadamlarni takrorlaymiz.

5. Ko'rilayotgan tugunning o'ng qismdaraxtini hosil qilish uchun massivning ko'rilayotgan oralig'ining 2-yarmini olamiz, ya'ni start=mid+1 va end=end (oldingi qadamdagi end). 3-5 qadamlarni takrorlaymiz.

Datur kodi

```
node *new_tree(int *arr, int start, int end)
{
   if(start>end) return NULL;
   else  {  int mid=(start+end)/2;
      node *tree=new node;
      tree->info=arr[mid];
      tree->left=new_tree(arr,start,mid-1);
      tree->right=new_tree(arr,mid+1,end);
      return tree;
   }
}
```