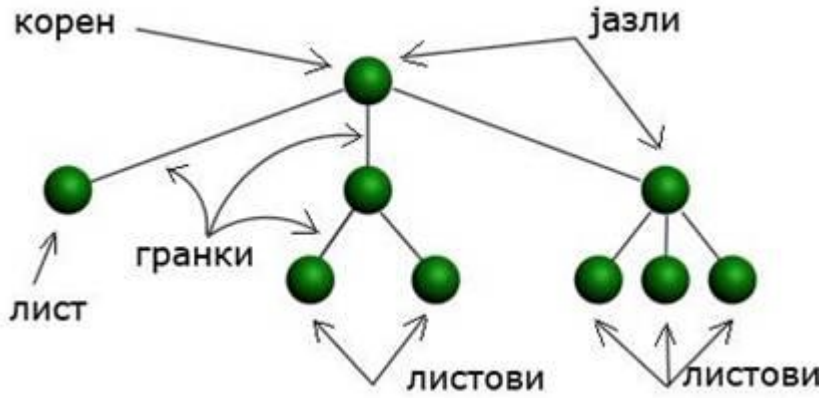
Дрва

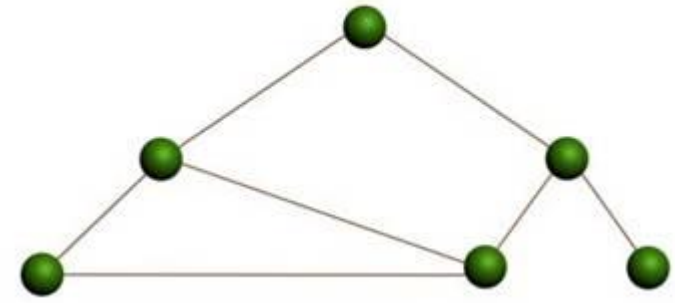
Вовед

Структурата на дрвата е навистина пробив во организацијата на податоците бидејќи тие дозволуваат да имплементираме алгоритми кои се многу побрзи отколку кога користиме линеарна податочна структура. Дрвата се основна податочна структура за зачувување на податоците користена во програмирањето. Тие даваат многу поголема предност во однос на податочните структури што ги видовме досега(низи,куп,ред,шпил). Релациите во дрвото се хиерархиски подредени со некои објекти "над" и некој "под" другите. Главната терминологија за податочната структура е "родител" – "дете".

Што е Дрво?

Дрво е апстрактна податочен тип кој ги чува елементите хиерархиски. Со исклучок на најгорниот елемент, сите имаат елемент "родител" и ниедно или повеќе "деца". Најгорниот елемент го викаме корен на дрвото и тој се црта како највисок елемент.





Зошто да користиме дрва?

Дрва би користеле затоа што ни даваат комбинирани предности на две други структури : подредена низа и поврзана листа. Во дрвото можеме да пребаруваме многу брзо исто како во подредена низа, и можеме брзо да внесуваме и бришеме елементи како во поврзана листа.

Споро внесување на елементи во подредена низа

Замислете си низа каде што елементите се подредени по ред т.е. подредена низа. За ваква низа пребарувањето е многу брзо за некоја вредност, како на пример користејќи binary search. Прво пребаруваме во центарот на низата, ако вредноста на елементот што го бараме е поголема од онаа

вредност на центарот, тогаш продолжуваме со десната страна на низата инаку со пребарувањето продолжуваме со левата половина на низата. Ваквото пребарување ни овозможува брзина од O(log n).

Од друга страна, ако сакаме да внесуваме објект во подредена низа прво треба да ја најдеме позицијата каде што треба да го внесеме елементот, и потоа да ги поместиме сите објекти на десно што се со поголема вредност за да направиме место за нашиот објект. Овие повеќекратни поместувања ни одземаат од процесорското време (N/2 поместувања). Истото важи и за бришење на елемент од подредената низа.

Споро пребарување во поврзана листа

Од друга страна пак, внесувањето и бришењето на елементите во поврзаната листа се одвива многу брзо. Овие операции имаат брзина од O(1) време на извршување. За жал пак, ова наогање на елемент во поврзаната листа се одвива многу споро.

Секогаш мора да се почнува од почеток и да се посети секој елемент додека не се најде оној што го бараме. Така треба да посетиме по просек од N/2 елементи споредувајќи го секој клуч на елементот со посакуваниот клуч на елементот. Ова е споро, има време на извршување од O(N).

Некои би помислиле дека би можеле да ги забрзаат работите користејќи подредена поврзана листа каде што елементите се подредени по ред, но ова не би им помогнало. Сепак мора да се почнува од почеток и да се посетат сите елементи во листата, бидејќи не може да се пристапи до одреден елемент без да се следи синџирот од референци до него.

Предноста на дрвата

Би било убаво ако постои податочна структура со брзо внесување и бришење на елементите како што е поврзаната листа, и исто така брзо пребарување како што е во подредена низа. Дрвото ги овозможува овие две карактеристики.

Апстрактна податочна структура – Дрво

Дрвото T е множество од јазли кој ги зачувува елементите во релација родител – дете со следниве можности :

- T има специјален јазол r, наречен корен.
- Секој јазол v од T различен од r има јазол родител n.

Во компјутерските програми јазлите најчесто ни претставуваат такви ентитети како што се луѓе, делови на коли,авионски резервации, итн, со други зборови кажано, вредности што ги зачувуваме во било која податочна структура.

Гранките се тие коишто ги поврзуваат јазлите. Едно дрво велиме дека е подредено ако има линеарно подредување дефинирано на децата. Во секој јазол, односно дека може да биде прво, второ, трето итн. Подредувањето е одредено според тоа како сакаме ние, обично се подредуваат од лево на десно.

Карактеристики на двата

Пат
Низата што е добиена како резултат на движење низ дрвото од еден јазол кон друг (гранките) се нарекува пат.

Корен
Јазолот којшто се наоѓа на врвот на дрвото се вика корен. Постои само еден корен во дрвото. За една колекција од дрва и гранки што го дефинираат едно дрво, мора да постои еден (и само еден!) пат од коренот до секој останат јазол.

Родител
Секој јазол (освен коренот) има точно една гранка којашто оди нагоре до друг јазол.Јазолот над тој јазол се вика јазол – родител.

Дете
Било кој јазол може да биде поврзан со еден или повеќе јазли надолу по дрвото. Јазлите под даден јазол се викаат негови деца.

Лист (надворешен јазол)
Јазол којшто нема ниту едно едете се вика лист или надворешен јазол. Во едно дрво може да има само еден корен, но повеќе листови.

Поддрво
Поддрво T1 од даденото дрво T со корен во јазолот v е дрво кое се состои од сите елементи под v во T1.
Линиите што се графички претставени кои ги поврзуваат јазлите всушност претставуваат нивните сродни врски.

Поминувања на дрва
Да се помине едно дрво значи да се посетат сите негови јазли по специфичен редослед. Подоцна ќе се запознаеме со поминувања во дрва.

Имплементација на дрва во Pascal

```
type pokazuvac = ^jazol;

jazol = record

Levo, desno: = pokazuvac;

vrednost := char;

end;
```

Алгоритми со дрва

Длабочина на јазол v

Нека v е јазол на дрво T. Длабочина на v е бројот на предци на v вклучувајќи го и него самиот.Длабочината на дрво T може да се дефинира рекурзивно на следниов начин :
Ако v е корен тогаш длабочината на v е 0,
Инаку, длабочината на v е еден плус длабочината на родителот на v

Висина на јазол v

Висина на јазол v исто така се дефинира рекурзивно
Ако v е надворешен јазол, тогаш висината е 0. Инаку, висината на v е 1 + максималната висина на

дете на v.
Висината на дрво T е еднаква на максималната длабочина на надворешен јазол на T.

Видови на дрво

Постојат различни видови на дрва и тоа :

- Бинарни дрва
- АВЛ дрва
- Multi way дрва
- 2 - 4 дрва
- B - дрва

Бинарни дрва

Бинарно дрво е подредено дрво кадешто секој јазол може да има најмногу две деца.
Јазол во едно бинарно дрво не мора да значи дека има точно две деца, може да има само едно лево дете,само едно десно дете или може воопшто да нема деца (во тој сучај станува збор за надворешен јазол). Во правилно бинарно дрво секој внатрешен јазол има точно две деца или ниедно дете. Овие деца се подредени така што левото дете доаѓа пред десното дете.

Можности на бинарни дрва

Бинарните дрва имаат интересни можности во справувањето со врските меѓу висината и бројот на јазли. За множество од сите јазли од дрво T на иста длабочина d велиме дека е ниво d на T. Во бинарно дрво нивото 0 има еден јазол, нивото еден има 2 јазли, нивото два има четири јазли итн. Општо нивото d има 2^d јазли. Максималниот број на јазли на нивоата од бинарното дрво расте експоненцијално надолу по дрвото.
Нека T е правилно бинарно дрво со n јазли и нека h е висина на T.Тогаш T ги има следниве можности:

- Бројот на надворешни јазли во T е најмалку h+1, а најмногу 2^h
- Бројот на внатрешни јазли во T е најмалку h, и најмногу 2^h-1
- Точниот број на јазли во T е најмалку 2*h+1 и најмногу 2^(h+1)-1
- Висината на T е најмалку log(n+1)-1 и најмногу (n-1)/2

Дефиниција на податочна структура на бинарни дрва во Pascal

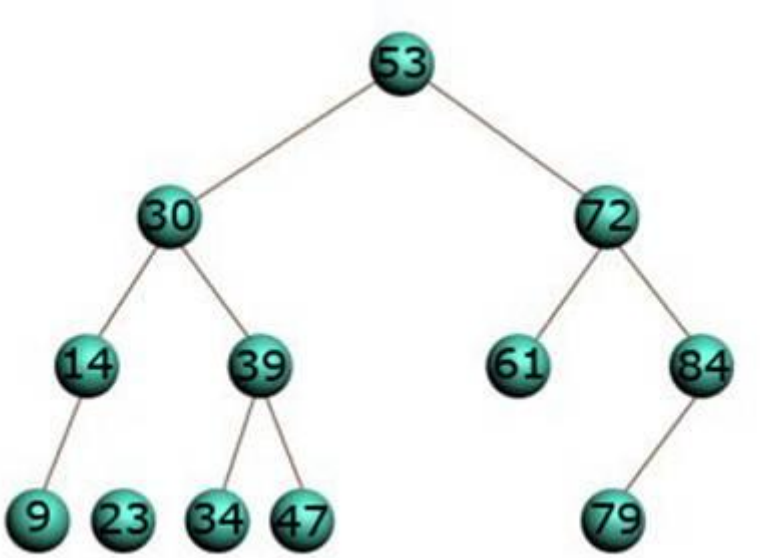
```
type pokazuvac = ^jazol;

jazol = record

vrednost : char;

levo, desno : pokazuvac;

end;
```

PreOrder премин на дрво

PreOrder премин на дрво е кога ќе ги поминеме сите јазли од дрвото по следниов редослед: корен, лево поддрво, десно поддрво. Овој вид на премин на дрвото е многу корисен кога пишуваме програми што анализира алгебарски изрази.

Бинарни пребарувачки дрва

Пребарувачки бинарни дрва се оние дрва кај кои нумеричките вредности на јазлите во левото поддрво на даден јазол се помали од него, а сите нумерички вредности на јазлите во десното поддрво на даден јазол се поголеми од него. На овој начин не мора да се шета низ целото дрво за да се провери дали постои даден елемент во дрвото.



Графички приказ на бинарно пребарувачко дрво

Алгоритмот за пребарување во бинарно дрво е

```
    procedure baraj( kluc : char; koren : pokazuvac );
begin
if koren=nil then {*** Не е најден ***}
        begin
           {* notfound( kluc )   *}
        end
else if koren^.vrednost = kluc then {*** Најден ***}
        begin
           {* notfound( kluc )   *}
        end
else if koren^.vrednost < kluc then baraj( kluc, koren^.vrednost )
     else                          baraj( kluc, koren^.vrednost )
end;
```

Алгоритмот за внесување на елемент во бинарно дрво е

```
procedure vmetnijazol(novjazol:pokazuvac; var koren:pokazuvac);
 begin
      if koren=nil
         then
             begin
                  koren:=novjazol;
                  novjazol^.levo:=nil;
                  novjazol^.desno:=nil
             end
         else
             if novjazol^.vrednost<koren^.vrednost
                then
                    vmetnijazol(novjazol,koren^.levo)
                else
                    vmetnijazol(novjazol,koren^.desno);
 end;
```

Поминувања на дрва

Поминување на дрво значи да се помине секој јазол од дрвото точно еднаш. Имаме 3 видови на поминување на бинарни дрва и тоа :

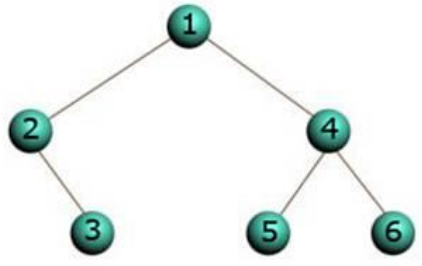
- PreOrder премин
- InOrder премин
- PostOrder премин

Во главно, рекурзивната структурна формула за поминување на непразно бинарно дрво е следната : На јазолот N мора да се направат следниве три работи : ·

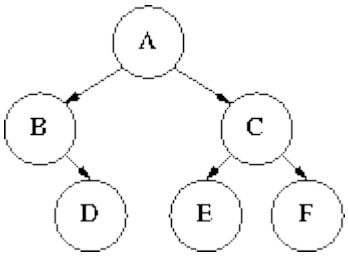
- Рекурзивно помини го левото поддрво. Кога овој чекор е завршен се наоѓате на јазолот N.
- Рекурзивно помини го десното поддрво. Кога овој чекор е завршен се наоѓате на јазолот N.
- Процесирај го јазолот N.

PreOrder премин на дрво

PreOrder премин на дрво е кога ќе ги поминеме сите јазли од дрвото по следниов редослед: корен, лево поддрво, десно поддрво.Овој вид на премин на дрвото е многу корисен кога пишуваме програми што анализира алгебарски изрази.



Графички приказ на PreOrder премин на дрво

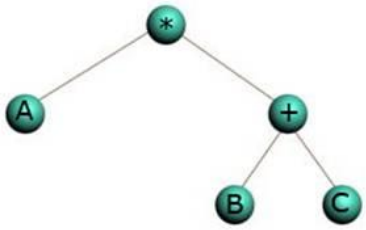


Се чита A-B-D-C-E-F

Имплементација на PreOrder во Pascal

```
procedure preorder(koren:pokazuvac);
begin
     if koren<>nil
        then
            begin
                 write('pritisnete <ENTER> za ispis na vrednosta na sledniot jazol');
                 readln;
                 pisijazol(koren^);
                 inorder(koren^.levo);
                 inorder(koren^.desno);
            end;
end;
```

Бинарно дрво (не бинарно пребарувачко дрво) може да се користи како претставување на аритметички израз вклучувајќи ги бинарните аритметички операции +,-,*,/. Коренот на дрвото содржи оператор и секој од неговите корени претставуваат или име на некоја променлива (како A,B,C итн) или друг израз.



Аритметички израз претставен со дрво

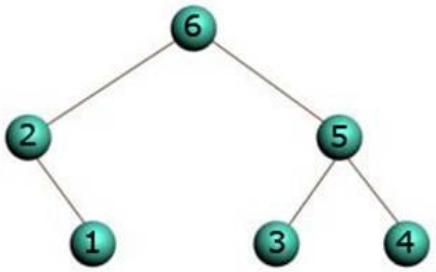
Користејќи го PreOrder преминот на дрво ќе ја добиеме префиксната нотација *A+BC

PostOrder премин на дрво

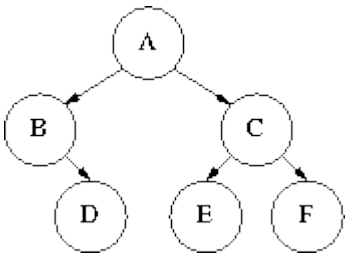
PostOrder премин на дрво е кога ќе ги поминеме сите јазли од дрвото по следниов редослед: лево поддрво, десно поддрво, корен. Посетувајќи го дрвото со PostOrder премин на сите јазли ќе ни ја генерира следниов аритметички израз ABC+*

Имплементација на PostOrder во Pascal

```
procedure postorder(koren:pokazuvac);
begin
     if koren<>nil
        then
            begin
                 inorder(koren^.levo);
                 inorder(koren^.desno);
                 write('pritisnete <ENTER> za ispis na vrednosta na sledniot jazol');
                 readln;
                 pisijazol(koren^);
            end;
end;
```



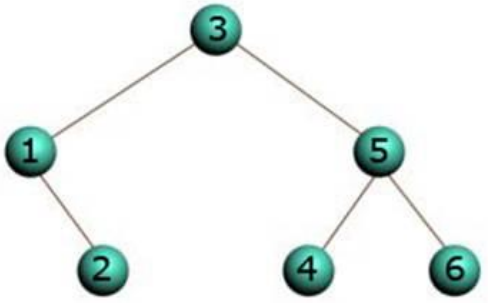
Графички приказ на PostOrder премин на дрво



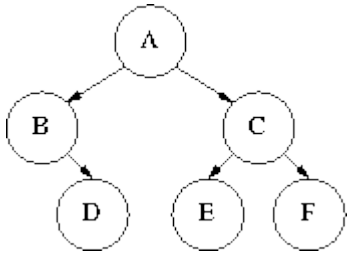
Се чита D-B-E-F-C-A

InOrder премин на дрво

InOrder премин на дрво е кога ќе ги поминеме сите јазли од дрвото по следниов редослед: лево поддрво, корен, десно поддрво.



Графички приказ на InOrder премин на дрво



Се чита B-D-A-E-C-F

Имплементација на Inorder во Pascal

```
procedure inorder(koren:pokazuvac);
begin
     if koren<>nil
        then
            begin
                 inorder(koren^.levo);
                 write('pritisnete <ENTER> za ispis na vrednosta na sledniot jazol');
                 readln;
                 pisijazol(koren^);
                 inorder(koren^.desno);
            end;
end;
```

AVL дрва

Проблемот со постигнувањена логаритамско време на извршување на операциите се разрешува со тоа што во дефиницијата за бинарно пребарувачко дрво секогаш ќе гледаме да се добие логаритамска висина за дрвото. Правило на кое ќе сметаме е балансирање на висината, што ја карактеризира структурата на бинарно пребарувачко дрво T со висина на неговите внатрешни јазли.

Својство за балансирање на висината : За секој внатрешен јазол v на T,висините на децата од v може да се разликуваат најмногу за 1.

Било кое бинарно пребарувачко дрво T кое го задоволува ова својство се нарекува AVL дрво, и е наречено по пронаоѓачите Adelson-Velski и Landis.

Последица од својството за балансирање е тоа дека поддрвото на AVL дрво е AVL дрво .
Својство: Висината на AVL дрво T кое има n членови е O(log n).
Според ова својство операцијата за наоѓање на елементи има време на извршување O(log n).

Внесување на елементи во AVL дрво:

Се внесува нов член во јазолот w во T кој претходно бил надворешен јазол и правиме w да стане внатрешен јазол со додавање на две надворешни "деца" на w. Оваа акција може да влијае на балансирање на висината, за некои јазли се зголемува висината за еден. Затоа треба да го реструктуираме T за да се блалансира висината.

За дадено бинарно пребарувачко дрво T, велиме дека јазолот v од T е балансиран ако апсолутната вредност на разликата меѓу висините на децата на v е најмногу 1 и велиме дека е небалансиран во друг случај. Значи, својството за балансирање на висината која го карактеизира AVL дрвото е исто со тоа секој внатрешен јазол да е балансиран.
Нека T го задоволува својството за балансирање на висината, по внесување нов член на T, висините на некои јазли вклучувајќи го и w, се зголемуваат. Сите такви јазли се на патеката од коренот,и тие се единствените јазли кои не се балансирани. За T да биде AVL дрво треба да ја поправиме оваа небалансираност.

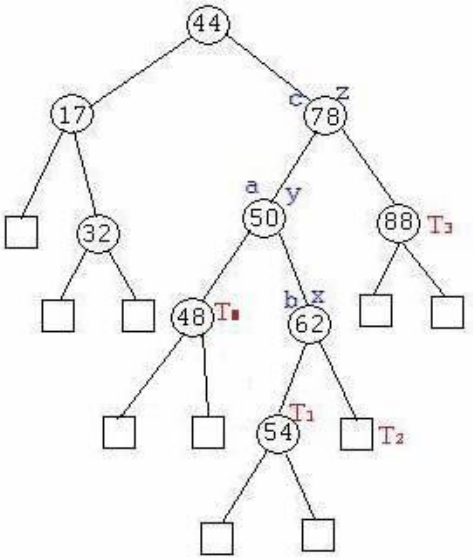
Ја регулираме балансираноста на јазлите од бинарно пребарувачко дрво T со едноставна стратегија "најди и поправи". Нека z е првиот јазол кој сме го вброиле одејќи нагоре од w до коренот на T кој што е небалансиран. Исто така y е дете на z со поголема висина. Нека x е дете на y со поголема висина, притоа x може да биде еднаков на w и x е внук на z. Бидејќи z е небалансиран поради внесување во дрвото со корен во y, висината на y е за 2 поголема од висината на оној што е на исто ниво со него. Сега го ребалансираме поддрвото со корен во z , со тоа што привремено ги преименуваме x,y и z како a,b и c, така што a претходи на b и b на c кога T се поминува inorder.

Имаме четири можни начини на мапирање на x,y и z во a,b и c. Реструктуирањето на трите јазли ги заменува z со јазол b, и негови деца ги прави a и c и прави децац на a и c да бидат претходните четири деца на x,y, и z, при тоа ги задржува inorder врските на сите јазли во T.

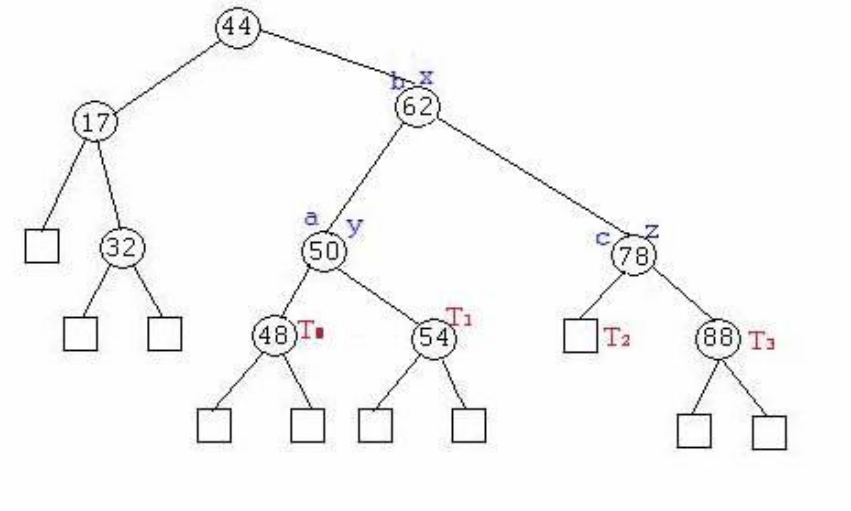
Алгоритам за реструктуирање .

Влез: јазол x од бинарно пребарувачко дрво T што има родител y и надродител z

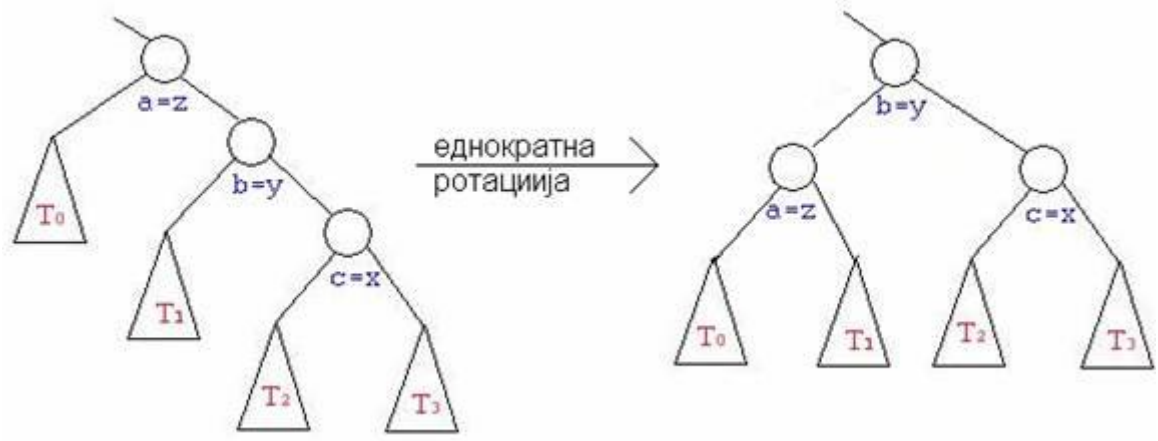
1. Нека (a,b,c) се inorder од лево кон десно листани јазлите x,y,z и нека (T0,T1,T2,T3) се од лево кон десно поддрва на x,y,z
2. Заменете го поддрвото со корен во z со новото поддрво со корен во b
3. Нека a е левото дете на b и нека T0 и T1 се левото и десното поддрво на a
4. Нека c е десното дете на b и нека T2 иT3 се левото и десното поддрво на c

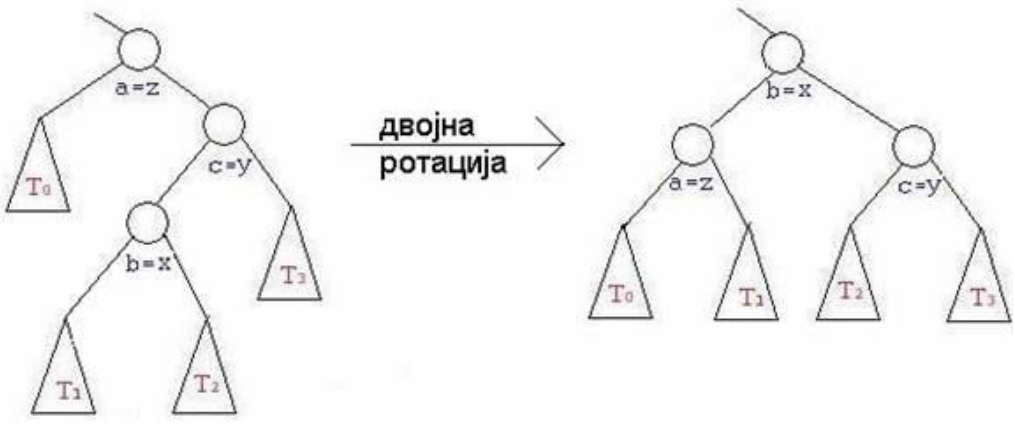


небалансирано бинарно пребарувачко дрво



Модификација на дрвото T предизвикана со операцијата за реструктуирање се нарекува ротација. Ако b=y тогаш методот на реструктуирање се нарекува еднократна ротација, тоа може да биде визуелизирано како ротирање на y околу z . Ако b=хоперацијата за реструктуирање се нарекува двојна ротација , и може да се визуелизира како ротирање на x околу y и потоа околу z.





Имплементација на ротација во Pascal:

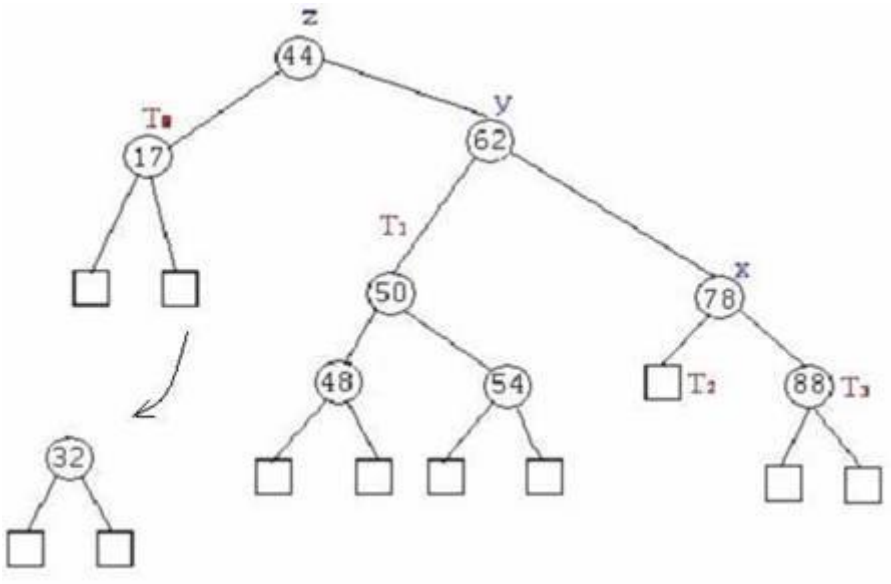
```
procedure lrot( var t : tree );
     var     temp : tree;
     begin
          temp := t;
          t := t^. right;
          temp^. right := t^. left;
          t^. left := temp;
          t^. weight := temp^. weight;
          temp^. weight := wt(temp^. left) + wt(temp^. right);
     end;

procedure rrot( var t : tree );
     var temp : tree;
     begin
          temp := t;
          t := t^. left;
          temp^. left := t^. right;
          t^. right := temp;
          t^. weight := temp^. weight;
          temp^. weight := wt(temp^. left) + wt(temp^. right);
     end;
```

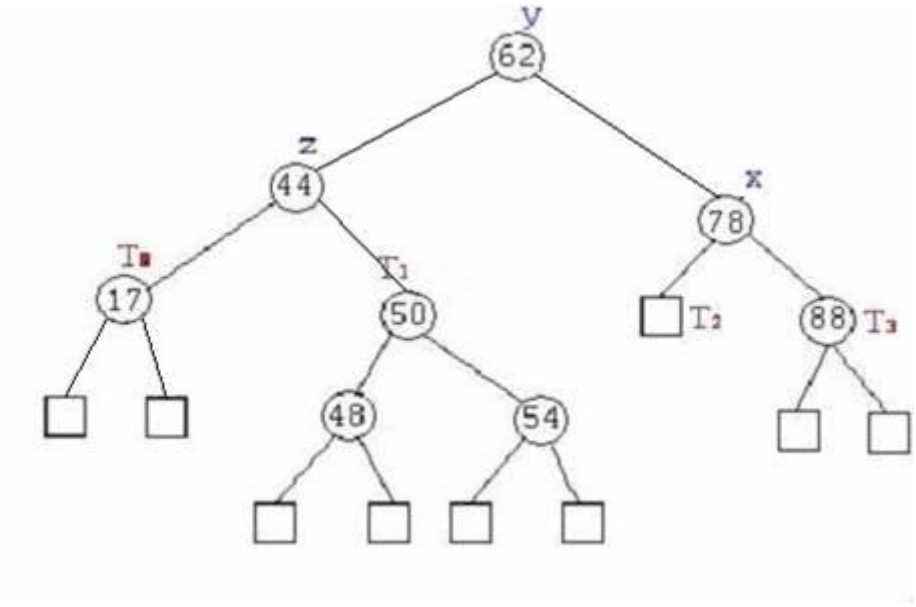
Имплементација на внесување во Pascal:

```
function insert( key : typekey; var t : tree ) : integer;
var     incr : integer;
begin
insert := 0;
if t = nil then begin
               t := NewNode( key, nil, nil );
               t^. bal := 0;
               insert := 1
               end
else if t^. k = key then
               Error {*** Клучот е во  табелата ***}
else    with t^ do     begin
        if k < key then  incr := insert( key, right )
               else     incr := -insert( key, left );
        bal := bal + incr;
        if (incr <> 0) and (bal <> 0) then
               if bal < -1 then
                       {*** левото поддрво е превисоко: потребна е десна ротација ***}
                       if left^. bal < 0 then rrot( t )
                              else    begin   lrot( left ); rrot( t ) end
               else    if bal > 1 then
                       {***  десното поддрво е превисоко: потребна е лева ротација ***}
                       if right^. bal > 0 then        lrot( t )
                              else    begin   rrot( right ); lrot( t ) end
               else    insert := 1;
        end
end;
```

Бришење на елементи на AVL дрво

При бришење на внатрешен јазол и подигање на едно од неговите деца на негово место, може да има небалансиран јазол во T на патот од родителот w на отргнатиот јазол до коренот на T.





Како и при внесувањето користиме реструктуирање за да се регулира балансот на дрвото T. Нека z е првиот небалансиран јазол вброен одејќи нагоре од w до коренот на T. Нека y е дете на z со поголема висина (јазолот y е дете на z кој не е предок на w), и нека x е дете на y со најголема висина. Изборот на x не мора да биде единствен, бидејќи поддрвата на y може да имаат иста висина. Во било кој случај кога ја изведуваме операцијата за реструктуирање, која ја регулира балансира носта на висината локално, поддрвото кое имало корен во z сега има корен во b.

Неформално, ваквото реструктуирање може да ја намали висината на поддрвото со корен во b за 1, што може да предизвика потомците на b да станат небаласирани. Значи еднократното реструктуирање не мора да ја промени балансираноста глобално по бришењето. Па, по ребалансирањето на z, продолжуваме да проверуваме каде во T има небалансирани јазли. Ако најдеме друг, ја изведуваме операцијата за реструктуирање за да се регулира балансираноста , и продолжуваме по T барајќи други се до коренот. Бидејќи висината на T е O(log n), каде n е бројот на членови, значи со ова реструктуирање сме го задоволиле својството за балансираност.

Имплементација на бришење во Pascal:

```
procedure delete( key : typekey; var t : tree );

    begin
    if t = nil then Error {*** клучот не е најден ***}
    else begin
        {*** бараме клуч кој ќе го избришеме ***}
        if      t^. k < key then delete( key, t^. right )
        else if t^. k > key then delete( key, t^. left )

        {*** клучот е најден, избриши го ако нема потомци ***}
        else if t^. left  = nil then t := t^. right
        else if t^. right = nil then t := t^. left

        else if wt( t^. left ) > wt( t^. right ) then
             begin rrot( t );  delete( key, t^. right ) end
        else begin lrot( t );  delete( key, t^. left ) end;

        if t <> nil then begin
            t^. weight := wt( t^. left ) + wt( t^. right );
            checkrots( t )
            end
        end
    end;
```

Multi-way пребарувачки дрва (силно разгранети)

Multi-way пребарувачки дрва се оние со внатрешни јазли кои имаааат две или повеќе деца, членовите кои ги складираме во пребарувачкото дрво се парови од форма (к,х) каде к е клуч, а х е членот кој е поврзан со клучот.
Нека v е јазол на подредено дрво. Велиме дека v е d-јазол ако има d деца. Дефинираме Multi-way пребарувачко дрво да биде подредено дрво Т кое ги има следниве својства.

1. секој внатрешен јазол од Т има најмалку две деца. Значи, секој внатрешен јазол е (к,х) каде к е клуч, а х елемент, каде d ?2.
2. секој внатрешен јазол од Т складира колекција од членови од форма (к,х) каде к е клуч, а х е елемент.
3. секој d-јазол v од Т, со деца $v_1 \ldots . v_d$ складира d-1 членови $(к_1,x_1) \ldots (к_{d-1},x_{d-1})$,каде $к_1 < \ldots < к_{d-1}$
4. конвенционално дефинираме $к_0 = -\infty$ и $кd = +\infty$. За секој член (к,х) во јазол на поддрвото од v со корен во v_i i=1…d, имаме дека $к_{i-1} \leq к \leq к_i$.

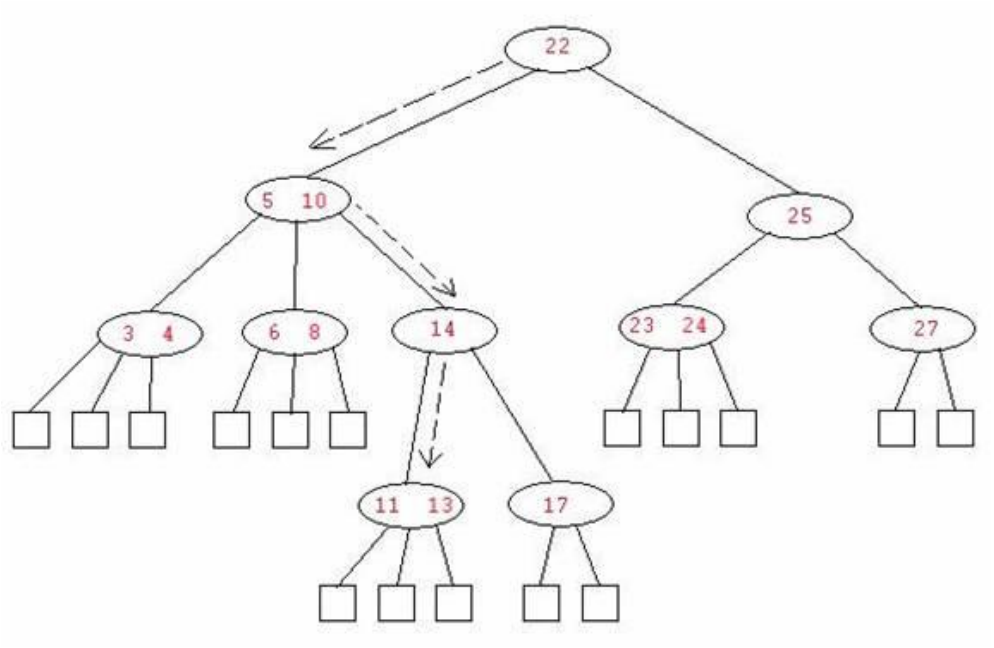
Ако во множеството од клучеви во v ги има и специјалните клучеви $к_0 = -\infty$ и $кd = +\infty$, тогаш клуч к зачуван во поддрвото на Т со корен во јазолот дете v_i мора да биде ”измеѓу” два клуча кои ги има во v. Овој поглед укажува на правилото дека јазол со d деца има d-1 клучеви и исто така ги формира основите на алгоритамот за пребарување во multi-way пребарувачките дрва.

Од претходната дефиниција , надворешните јазли на Multi-way дрвата не зачувуваат членови и служат како ” држачи на место” . Multi-way пребарувачкото дрво може да има само еден внатрешен јазол кој ги зачувува сите членови. Додека надворешните јазли можат да бидат null или референци до објектот Null-Node, но земаме дека тие се јазли кои не чуваат ништо.

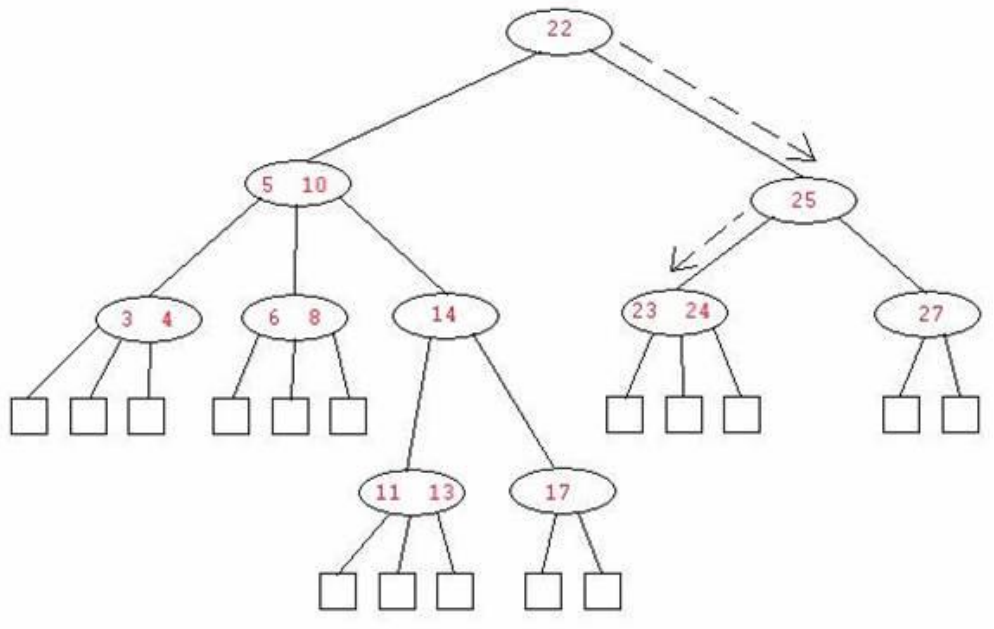
Без разлика дали внатрешните јазли имаат две деца или повеќе, има врска меѓу бројот на членови и бројот на надворешни јазли.
Својство: Multi-way пребарувачко дрво кое има n членови има n+1 надворешни јазли.

Пребарување во Multi-way дрво

Пребарување за елемент со клуч к во Multi-way пребарувачко дрво T, е едноставно. Изведуваме такво пребарување поминувајќи патека во T која започнува во коренот. Кога сме во d –јазолот го споредуваме клучот к со клучевите $к_1 < \ldots < к_{d-1}$ од v . Ако $к = к_i$ за некое i , потрагата е успешно завршена . Инаку, продолжуваме со потрагата во детето vi на v такво што $к_{i-1} \leq к \leq к_i$ (притоа $к_0 = -\infty$ и $кd = +\infty$). Ако дојдеме до надворешен јазол, тогаш знаеме дека нема член со клуч к во T и потрагата завршува неуспешно.



неуспешна потрага на 12



успешна потрага на 24

(2 , 4) ДРВА

Multi-way пребарувачко дрво кое ги исполнува следниве две својства се вика (2,4) или (2,3,4) дрво.

Својство за големина: Секој јазол има најмногу четири деца .

Својство за длабочина: Сите надворешни јазли имаат иста длабочина.

Земаме дека надворешните јазли се празни во опишувањето на методите за пребарување и измена ,земаме дака тие се вистински јазли.

Својство: висината на (2,4) дрво кое зачувува n членови е O(log n) .

Ова својство докажува дека својствата за големина и длабочина се важни за одржување на Multi-way дрвото балансирано и имплицира дека изведувањето на пребарување во (2,4) дрво завзема O(log n) време.

Внесување на елементи во (2,4) дрво

Методот за внесување го засегнува својството за длабочина, бидејќи додаваме нов надворешен јазол на исто ниво како и надворешните јазли, а тоа може да го засегне и својството за големина. Ако јазолот v е прво јазол со 4 члена , потоа може да стане 5-јазол , што предизвикува T веќе да не е (2,4) дрво. Нека v1… v5 се деца на v, и нека к1. . . к4 се клучевите зачувани во v . За да го разрешиме преполнувањето во јазолот v , изведуваме операција за поделба на v како што следи:

1. го заменуваме v со два јазли v’ и v’’, каде

1. 1 v’ е 3-јазол со деца v1, v2, v3 со клучеви к1 и к2

1. 2 v’’ е 2-јазол со деца v4, v5 со клуч к3

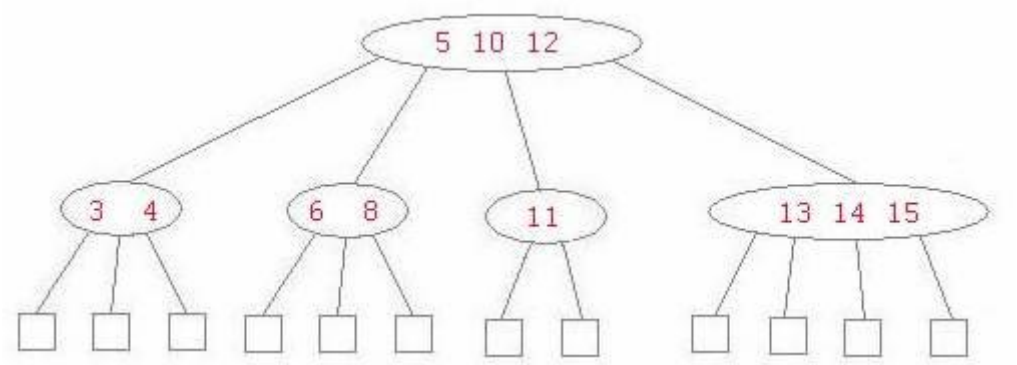
2. ако v е коренот на T, креираме нов корен , јазол u; инаку нека u е родител на v .

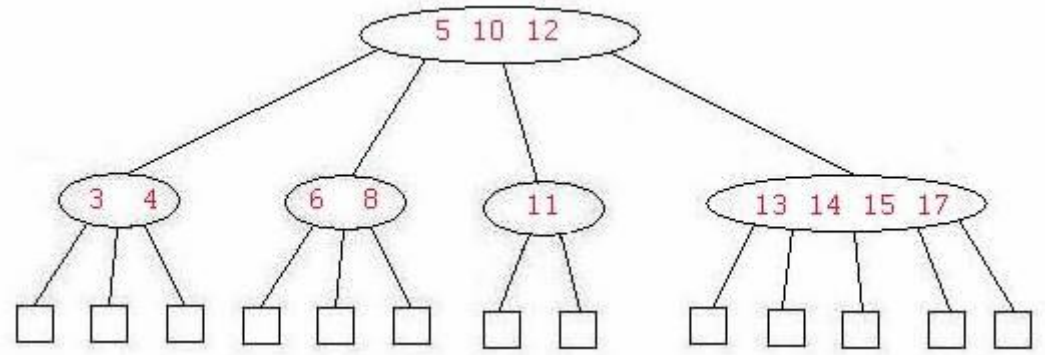
3. внесуваме клуч к3 во u и v’ и v’’ ги правиме деца на u ,така што ако v беше i-то дете на u, тогаш v’ и v’’ стануваат i-то иi+1 дете на u.

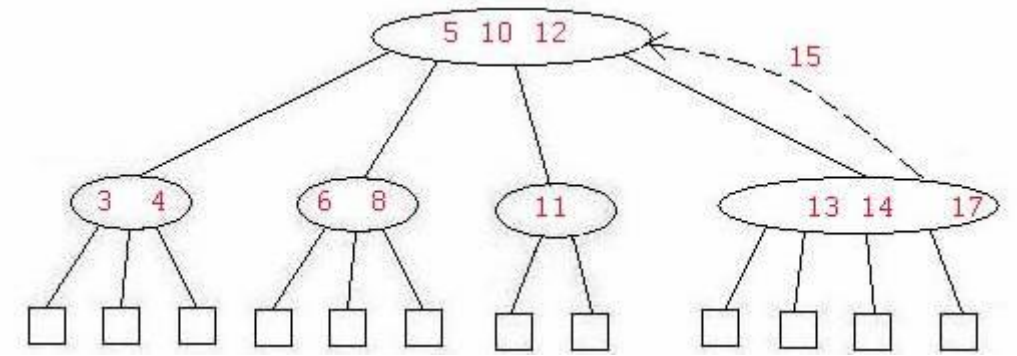
Анализа на внесувањата во (2,4) дрво

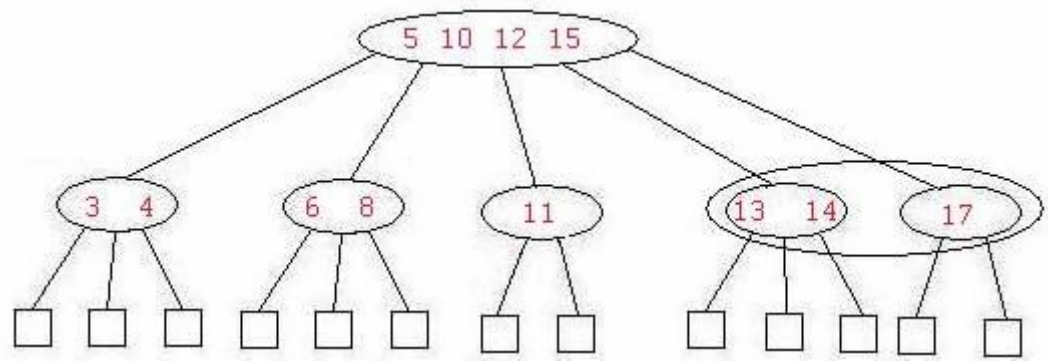
Операцијата на поделба влијае на константниот број на јазли на дрвото и O(1) членовите зачувани во таквите јазли. Значи, може да биде имплементирано за да се изврши во O(1) време.

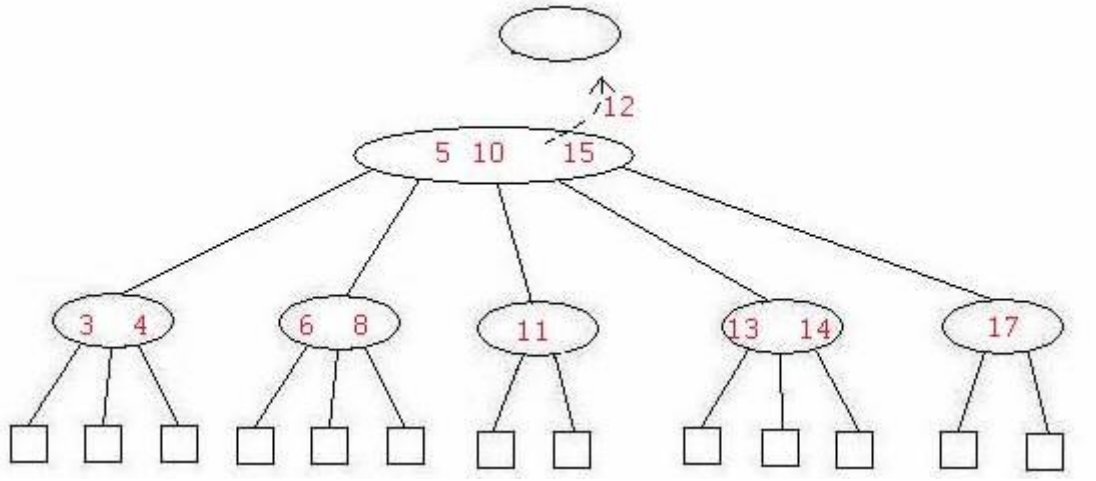
Како последица на операцијата на поделба на јазол v, ново преполнување може да се појави во родителот u на v . Ако се појави такво преполнување , тоа повлекува поделба на јазол u. Операцијата на поделба или го елиминира преполнувањето или го пролонгира во родителот на тој јазол. Бројот на операции на поделба е ограничен со висината на дрвото , кој е O(logn). Затоа целосно време на изведување на внесувањето во (2,4) дрво е O(log n).

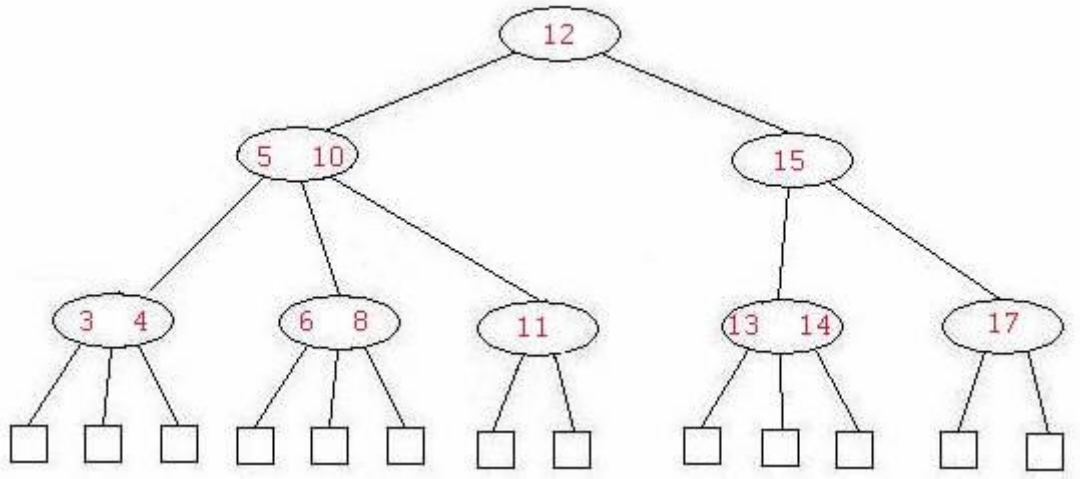

5 10 12
3 4
6 8
11
13 14 15


5 10 12
3 4
6 8
11
13 14 15 17


5 10 12
15
3 4
6 8
11
13 14 17


5 10 12 15
3 4
6 8
11
13 14
17





Бришење на елементи во (2,4) дрво

Почнуваме операција на бришење со изведување на потрага во T за член со клуч к. Бришењето на таков член од (2,4) дрво може секогаш да се редуцира до случајот каде членот кој треба да се избрише е зачуван во јазол v чии деца се надворешни јазли. Да претпоставиме дека членот со клуч к кој сакаме да го избришеме е зачуван во i-тиот член (кi,хi) во јазолот z кој има единствен внатрешен јазол. Во овој случај, го заменуваме членот (кi,хi) со соодветен член кој е зачуван во јазолот v со надворешен јазол:

1. го наоѓаме најдесниот внатрешен јазол v во поддрвото со корен во i-тото дете на z ,забележуваме дека децата на јазолот vсе надворешни јазли

2. го заменуваме членот (кi,хi) на z со последниот член на v.

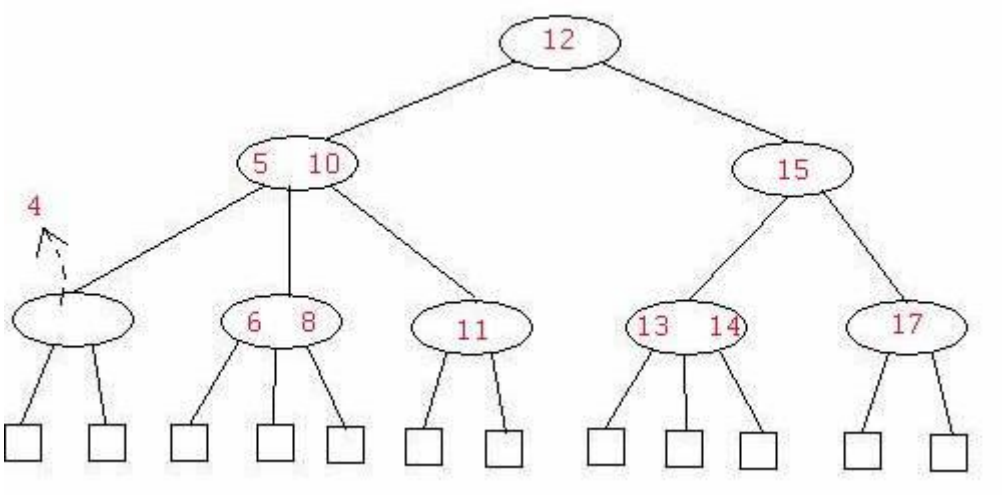
3. кога сме сигурни дека членот кој сакаме да го избришеме од v има деца кои се само надворешни јазли –деца, едноставно го бришеме членот од v и го бришеме i-тиот надворешен јазол од v.

Бришењето на член (и дете) од јазолот v го засега својството за длабочина, затоа секогаш го бришеме детето кое е надворешен јазол од јазолот v со децата надворешни јазли. Во отстранувањето на таков надворешен јазол можеме да влијаеме на својството за големина во v. Ако v бил претходно јазол со два клуча , потоа станува 1-јазол без членови за бришење, што не е дозволено во (2,4) дрво. Овој тип на влијаење на својството за големина се нарекува underflow во v. За да го поправиме ова , проверуваме дали јазолот на исто ниво со v е 3-јазол или 4-јазол. Ако најдеме таков јазол w, тогаш

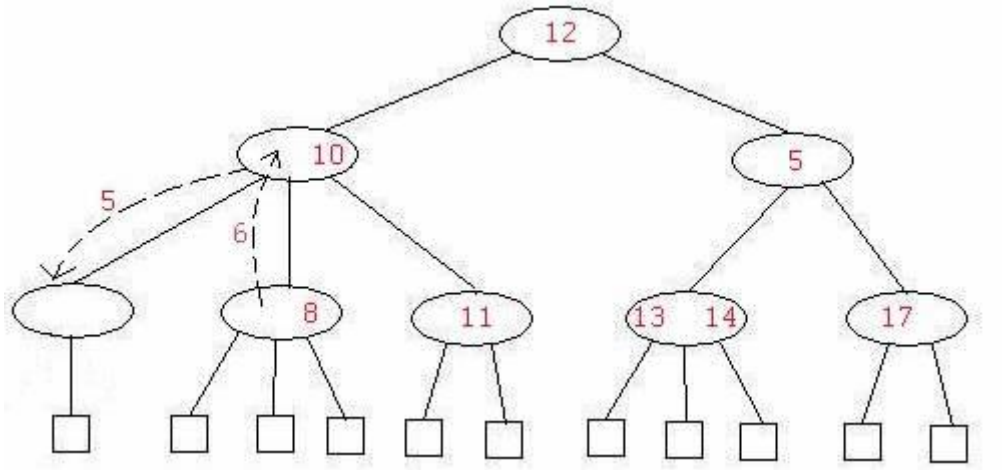
изведуваме операција на трансфер во која ги поместуваме децата од w во v, клучевите од w во родителот u на v и w, и клучот на u во v. Ако v нема „sibling“, или ако и двата се 2-јазли, тогаш изведуваме операција на фузија, во која ги спојуваме v со својот „sibling“, креирајќи нов јазол v’ , и го поместуваме клучот од родителот u на v во v’.

Операцијата на фузија во јазолот v може да предизвика нов underflow во родителот u на v, што повлекува трансфер или фузија во u. Бројот на операции на фузија е ограничен со висината на дрвото, која е O(log n). Ако се пролонгира „underflow“ по целиот пат до коренот тогаш тој се брише.

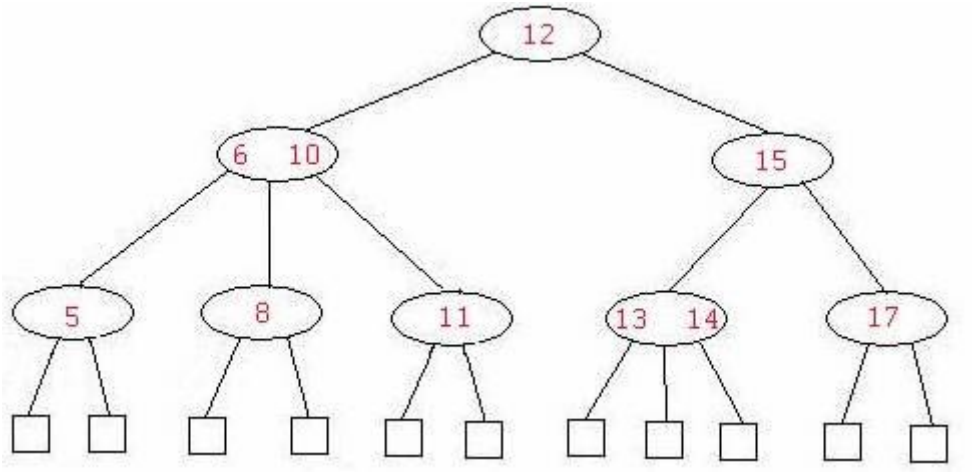
Примери за бришење:



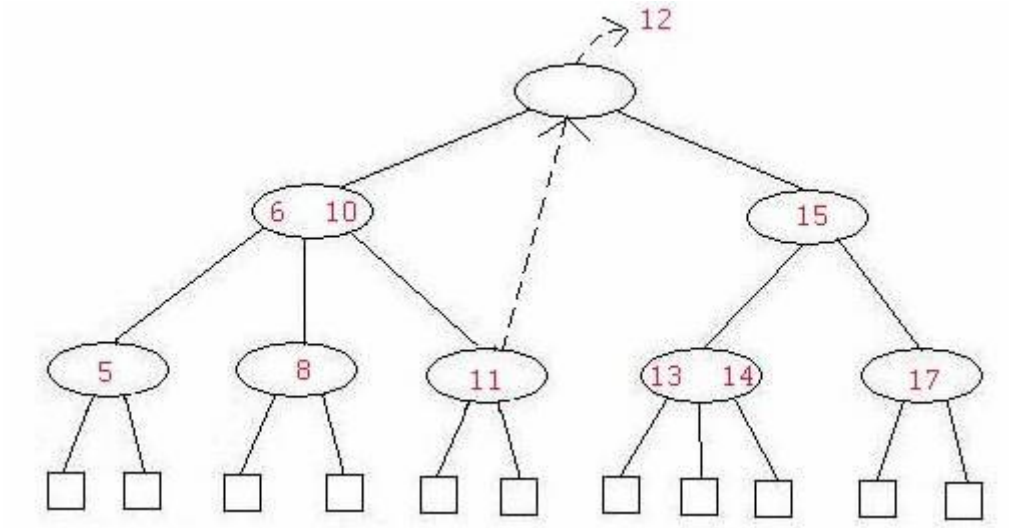
бришење на 4



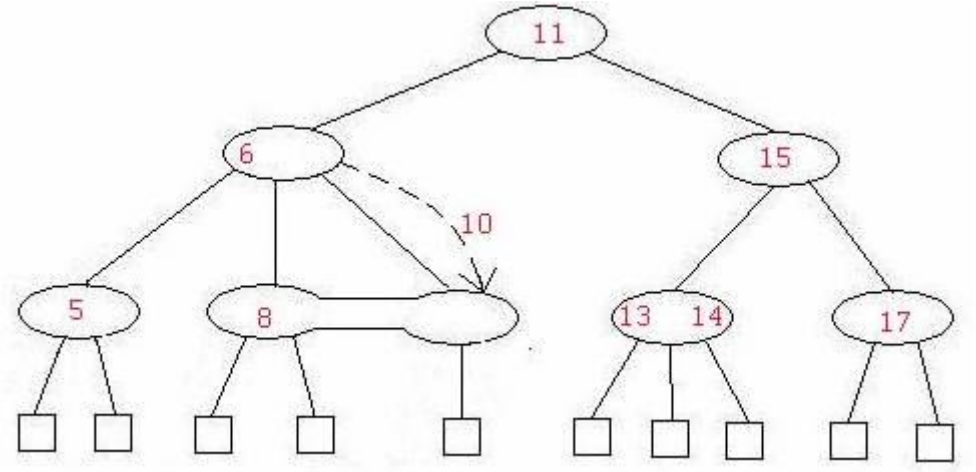
операција на трансфер



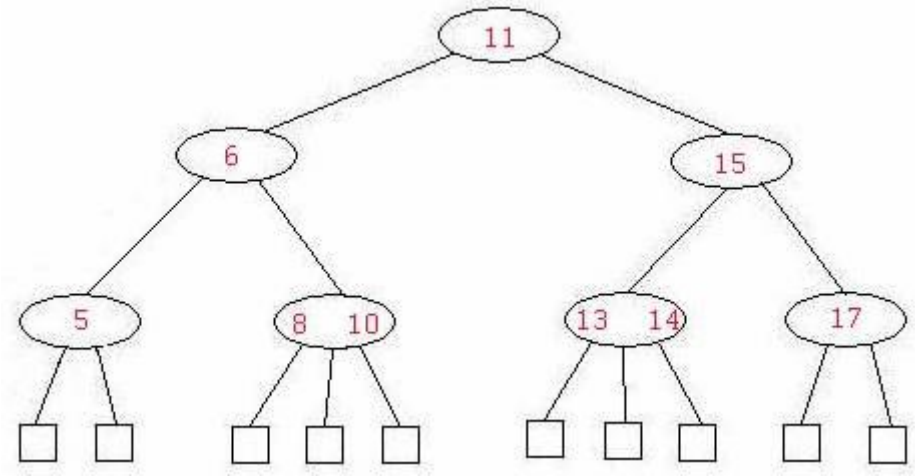
по трансфер операцијата



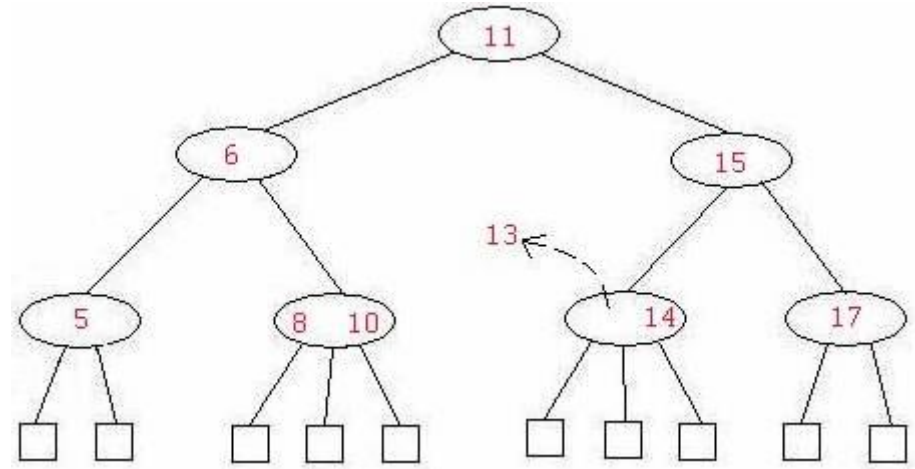
бришење на 12



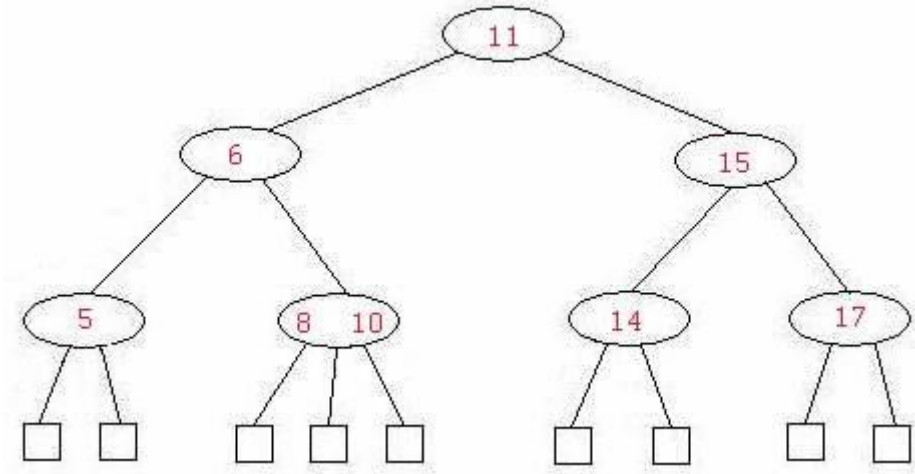
операција на фузија



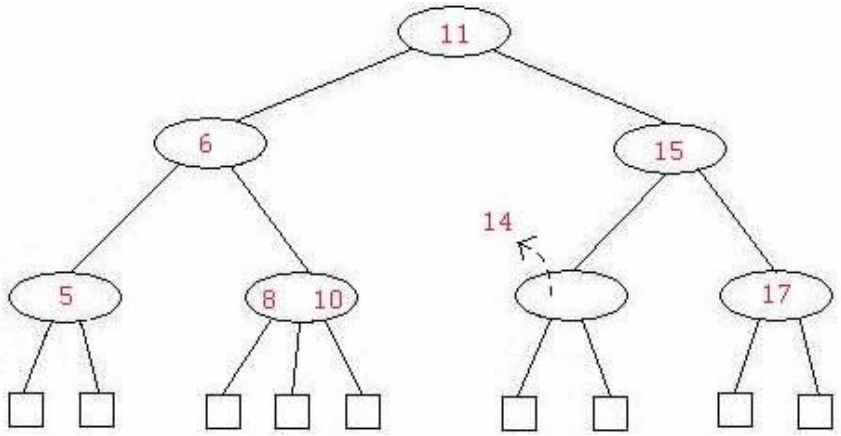
по фузија



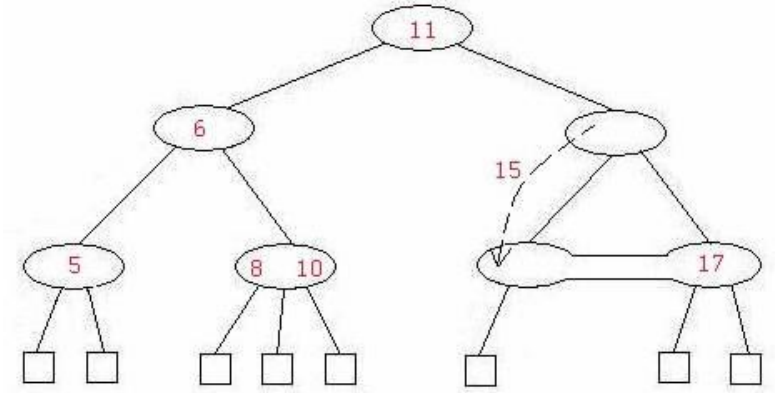
бришење на 13

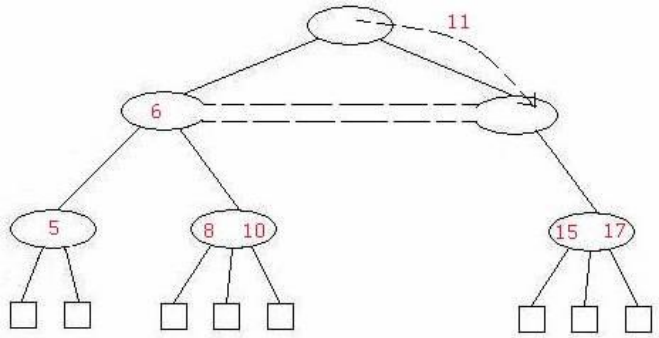


по бришењето



бришење на 14





фузија