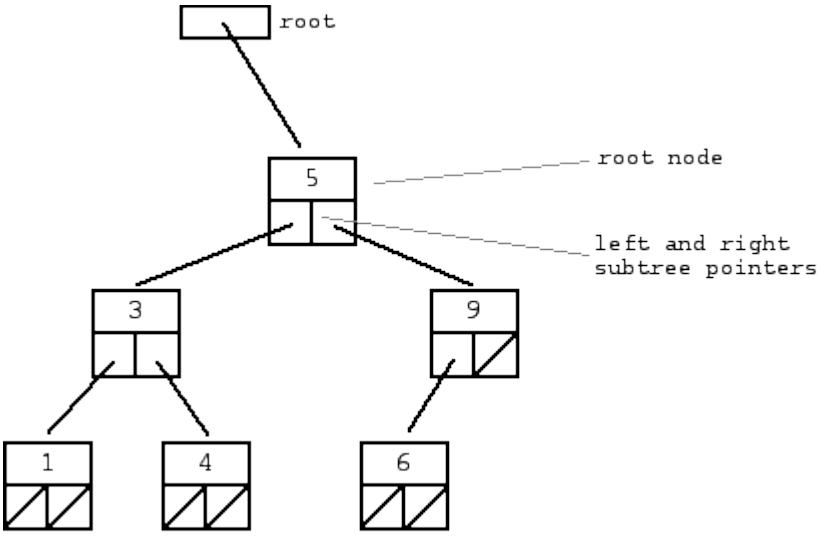
Нелинеарни податочни структури – дрво (tree) – час 2

Бинарно дрво е создадено од јазли каде секој јазол содржи лев, десен покажувач и податок. Корен покажувачот покажува на најгорниот јазол во дрвото. Левиот и десниот покажувач покажува на поддрвата на секоја страна. NULL покажувач претставува бинарно дрво без елементи – празно дрво.



BST (binary search tree) или подредено бинарно дрво е тип на бинарно дрво каде јазлите се подредени по редослед: за секој јазол сите елементи во неговот лево поддрво се помали или еднакви на јазолот, а елементите во десното поддрво се поголеми од јазолот.

Подредените бинарни дрва (**BST)** се брзи за додавање на елемент и изминување. Затоа се добри пример за речници каде се внесуваат и бараат елементи според некој клуч.

Друг вид на дрво е куп (heap). Heap е дрво каде секој јазол - наследник има вредност помала од вредноста на јазолот на кој следи. Корен јазолот има најголема вредност во дрвото, па најголема вредност може да се најде во константно време: едноставно се враќа вредноста на коренот.

Лист (leaf) јазол е јазол кој нема следбеници, а nonleaf се вика внатрешен јазол. Бројот на наследници во дрво со број на јазли x е еднаков на степенот на x. Должината на патеката од коренот r до јазолот x е длабочина на x во дрвото T. Ниво на дрво е составено од сите јазли на иста длабочина. Висина на јазол во дрво е број на рабови на најдолгиот пат во длабочина од јазолот до листот и висина на дрво е еднаква на најголемата патека во длабочина на неговиот корен од било кој јазол на дрвото.

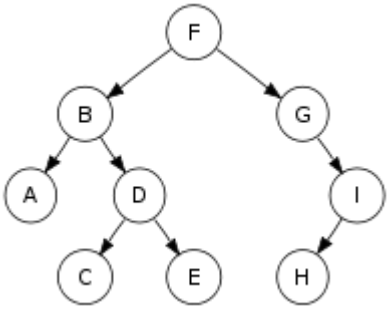
Операции во бинарно дрво:

1. Look Up (lookUp()) – е едноставна и брза и се користи за внесување на податоци.
2. New Node (newTreeNode()) – креира корен јазол.
3. Insert Node (insertTreeNode()) – внесувањето започнува како пребарувањето. Ако коренот не е еднаков на вредноста, се пребарува левото или десното подрво. Ако се стигне до лист,

се додава вредноста како лев или десен наследник во зависност од вредноста. Со други зборови, го испитуваме коренот и рекурзивно го вметнуваме новиот јазол во левото поддрво ако вредноста е помала од коренот, или десното поддрво ако новата вредност е поголема или еднаква на коренот.

4. Бришење на елемент (deleteKey())
   Може да биде:
   - Лев јазол – тогаш е најлесно и смо се брише јазолот
   - Јазолот има лев и десен наследник – тогаш јазолот се заменува со неговиот претходник
   - Јазолот има само еден наследник – наследникот се заменува со претходникот
5. Size (treeSize()) – е вкупен број на јазли на дрвото
6. Maximum/ Minimum Depth (maxDepth()/minDepth()) – е број на јазли на најдолгиот/ најкраткиот пат од коренот до најоддалечениот лев јаол. Најголема длабочина на празно дрво е 0.
7. Is balanced? (osBalanced()) -балансирано дерво се дефинира како бинарно дрво каде на секое ниво двете поддрва на секој јазол не се разликуваат за повеќе од 1.
8. Minimum/Maximum Value (minTree()/maxTree())
9. In Order Successor/Predecessor (succesorInOrder()/predecessorInOrder()) – јазол кој има следен / претходен елемент
   Successor – следбеник на јазол е следниот со најголема вредност во дрвото. Односно, ако сите вредности (keys) се различни, следбеник на јазолот xx е јазол со најмала вредност поголема од x.keyx.key. Структурата на бинарното дрво овозможува да се одреди следбеник на јазол без споредување на вредностите.
10. Степен на јазол е вкупниот број на гранки на тој јазол
11. Шума – колекција од неповрзани дрва

Пр.



На сликава, следбеник на B е C, на E е F, а I нема следбеник.

Наоѓањето следбеник вклучува два различни случаи:

а) Ако јазолот е десен наследник (дете) тогаш следбеник е минимум на помалото дрво (поддрвото). Пр. за да се најде следбеник на B, знаеме дека има десен наследник D, па минимум е C.

б) Ако јазолот нема десен наследник (дете) како јазолот E, треба да се бара наназад во дрвото се додека не се наиде на првото десно „вртење“. Односно, се бара нагоре во дрвото се додека не се

најде јазол што е лев наследник и се зема неговиот претходник.Во примеров, се од од E до D, потоа лево до B, е десно до F.

Креирање бинарно дрво:

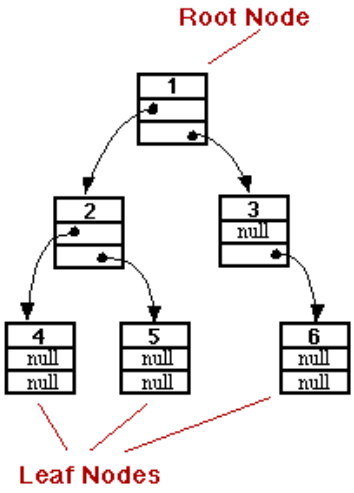
```
struct jazol {
  int x;        // Податокот во јазолот
  jazol *left;   // покажувач на левото поддрво
  jazol *right;  // покажувач на десното поддрво

}
```

Left и right покажувачите во јазолот може да бидат NULL или да покажуваат на други објекти од тип jazol. Јазолот кој покажува на друг јазол се вика „родител“, а јазолот на кој покажува е „дете“. На сликава, јазолот 3 е родител на јазолот 6, а јазлите 4 и 5 се деца на јазолот 2.

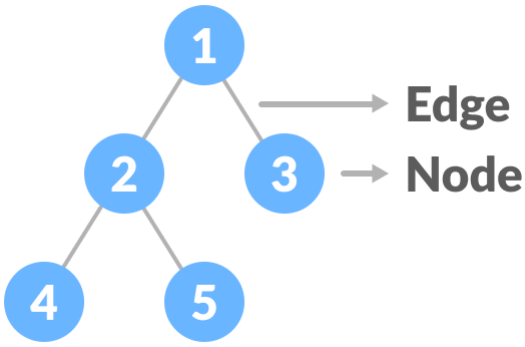
Не секоја поврзана структура создадена од дрва – јазли е бинарно дрво. Бинарно дрво мора да има:

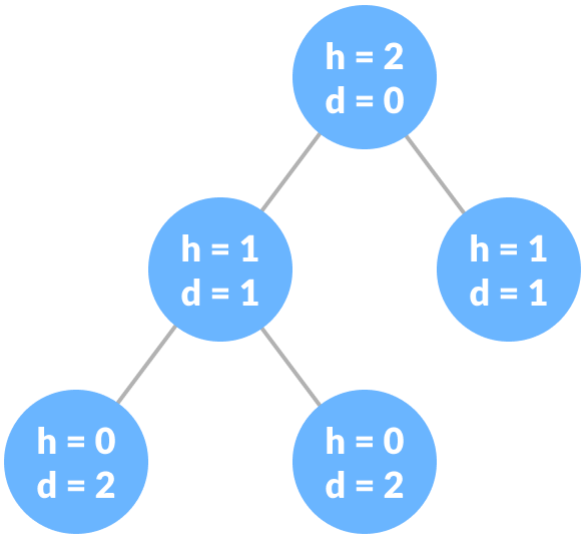
- Само еден јазол кој нема родител. Овој јазол е корен на дрвото.
- Секој друг јазол има само еден родител
- Не смее да има циклуси во бинарно дрво, т.е. не постои пат низ покажувачи со кој се поаѓа од еден јазол и се стигнува во истиот јазол.



Јазолот кој нама деца (наследници) се вика лист. Левиот и десниот покажувач кај листот се NULL.

Да разгледаме јазол со неговите следбеници. Структурата што се формира е бинарно дрво и се нарекува поддрво на дрвото. Пр. на сликата јазликте 2, 4 и 5 формираат поддрво. Ова е лево поддрво на коренот. Слично, јазлите 3 и 6 формираат десно поддрво. Секое непразно бинарно дрво е изградено од корен, лево и десно поддрво. Било кое или двете поддрва може да бидат празни. Ова е рекурзивна дефиниција и често се користи при работата со дрва.

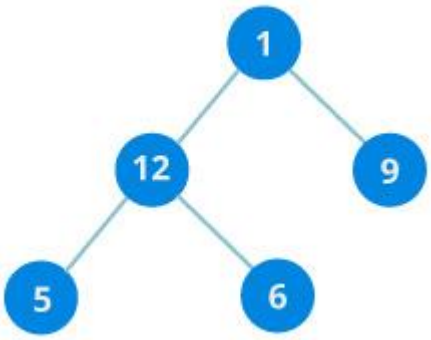




Висина и длабочина на секој јазол на дрвото

Изминување на дрва - inorder, preorder and postorder

Изминувањето на дрва значи посетување на секој јазол во дрвото. Пр. сакаме да најдеме збир на сите вредности во дрвото или да се најде најголемата вредност. За да се извршат овие операции мора да се посетат сите јазли во дрвото. Линеарните податочни структури како низи, стекови, редови и поврзани листи имаат само еден начин да се прочитаат податоците. Но кај дрвата може изминувањето да биде на повеќе начини.



Елементите на сликава почнувајќи од горе, лево, десно се:

1 -> 12 -> 5 -> 6 -> 9

Почнувајќи од долу, лево, десно се:

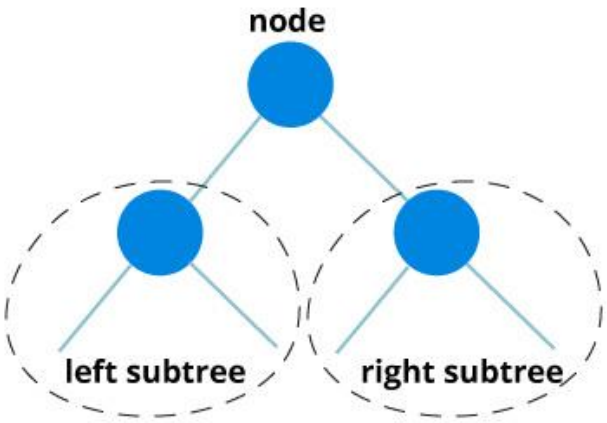
5 -> 6 -> 12 -> 9 -> 1

Иако овој начин можеби е полесен тој не ја запазува хиерархијата на дрвото, туку само длабочината на јазлите.

Затоа се користат методи за изминување кои ја запазуваат основната структура на дрвото т.е.

```
struct jazol {
   int x;
   struct jazol * left;
   struct jazol * right;
}
```

struct jazol има лев и десен покажувач кои може да имаат наследници (деца) кои ги сметаме поддрва, а на подјазли.



Треба да се посети секој јазол, па треба да се посетат сите јазли во левото поддрво, коренот и јазлите во десното поддрво. Има три начини на изминување:

Inorder изминување

Прво се изминуваат сите јазли во левото поддрво

Се посетува коренот

Се изминуваат сите јазли во десното поддрво

inorder(root->left)

display(root->x)

inorder(root->right)

Preorder изминување

Се посетува коренот

Се изминуваат сите јазли во левото поддрво

Се изминуваат сите јазли во десното поддрво

display(root->x)

preorder(root->left)

preorder(root->right)

Postorder изминување

Се изминуваат сите јазли во левото поддрво

Се изминуваат сите јазли во десното поддрво

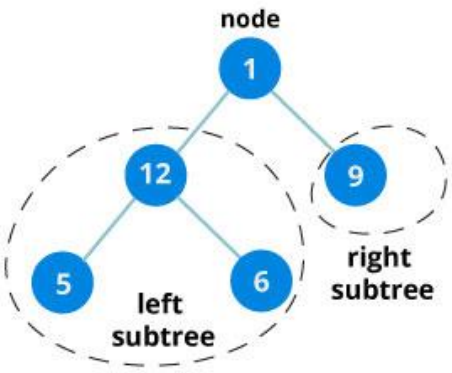
Се посетува коренот

postorder(root->left)

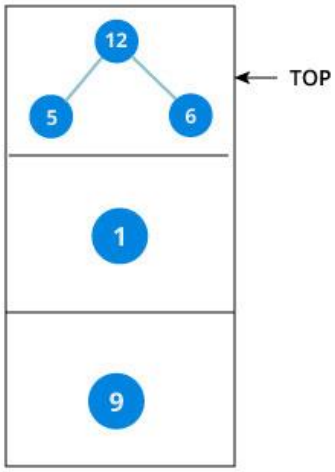
postorder(root->right)

display(root->x)

Inorder изминување



Прво се изминува левото поддрво, коренот па десното поддрво.



Сега треба да се измине поддрвото на кое покажува TOP на стекот (пак лево, корен десно)

Бидејќи 5 нема поддрва, се печати. Потоа се печати неговиот родител 12, а потоа десниот наследник на 12, а тоа е 5.

По целосно изминување на дрвото, се добива

5 -> 12 -> 6 -> 1 -> 9

Preorder изминување

1 ->12 ->5 ->6 ->9

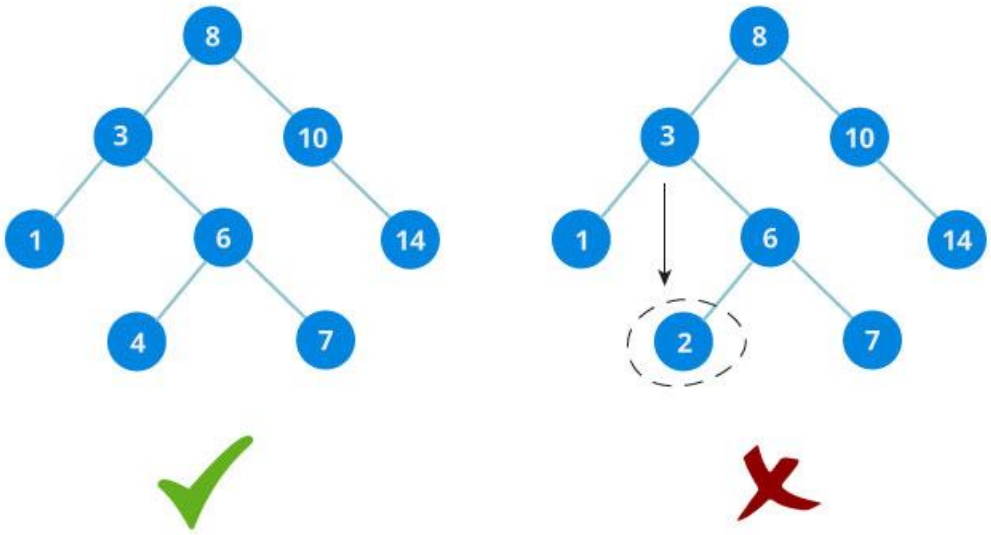
Postorder изминување

5 ->6 ->12 ->9 ->1

Бинарно подредено дрво - Binary Search Tree(BST)

Секој јазол има најмногу два јазли. Разликата со бинарно дрво е:

- Сите јазли на од левото поддрво се помали од коренот
- Сите јазли од десното поддрво се поголеми од коренот
- И левото и десното поддрво се бинарни подредени дрва



1 ->3 ->4 ->6 ->7 ->8 ->10 ->14

Креирање на подредено бинарно дрво

```
#include<iostream>
using namespace std;
class BST
{
   struct jazol
 {
       int k;
       jazol* left;
       jazol* right;
   };
   jazol* root;
```

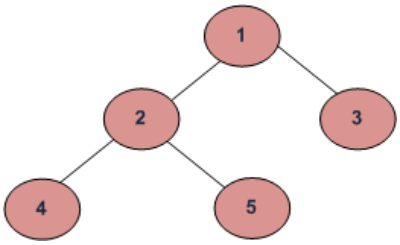
```
   jazol* makeEmpty(jazol* t)
{
      if(t == NULL)
         return NULL;
      {
         makeEmpty(t->left);
         makeEmpty(t->right);
         delete t;
      }
      return NULL;
   }
   jazol* insert(int x, jazol* t)
   {
      if(t == NULL)
      {
         t = new jazol;
         t->k = x;
         t->left = t->right = NULL;
      }
      else if(x < t->k)
         t->left = insert(x, t->left);
      else if(x > t->k)
         t->right = insert(x, t->right);
      return t;
   }
   void inorder(jazol* t)
{
      if(t == NULL)
         return;
      inorder(t->left);
      cout << t->k << " ";
      inorder(t->right);
   }
public:
   BST()
{
      root = NULL;
   }
  void insert(int x)
{
      root = insert(x, root);
   }
   void display()
{
      inorder(root);
      cout << endl;
   }
};
```

```
int main()
{
   BST t;
   int n,i,l;
   cout<<"vnesi broj na elementi"<<endl;
   cin>>n;
   for(i=1;i<=n;i++)
   {
      cout<<"vnesi vrednost"<<endl;
      cin>>l;
      t.insert(l);
   }
      t.display();
   return 0;
}
```

Бришење на дрво

За да се избрише дрво, мора да се изминат сите јазли и еден по еден да се избришат. Треба да се измине дрвото Inorder, Preorder или Postorder. Најсоодветно е Postorder бидејќи пред да се избрише родителот, се бришат сите негови наследници.

За следниов пример, редоследот на бришење е – 4, 5, 2, 3, 1




```
#include<iostream>
using namespace std;
/* binarnoto drvo ima podatok, i pokazuvaci na lev
i desen naslednik */
class jazol
{
   public:
   int x;
   jazol* left;
   jazol* right;

    jazol(int x)
   {
      this->x = x;
      this->left = NULL;
      this->right = NULL;
   }
```

```
};

/* Funkcijata go izminuva drvoto vo posorder
za da go izbrise sekoj jazol od drvoto*/
void deleteTree(jazol* jazol)
{
   if (jazol == NULL) return;

   /* prvo se brisat dvete poddrva */
   deleteTree(jazol->left);
   deleteTree(jazol->right);

   /* potoa se brise jazol */
   cout << "\n Izbrisan jazol: " << jazol->x;
   delete(jazol);
}
int main()
{
   jazol *root = new jazol(1);
   root->left     = new jazol(2);
   root->right     = new jazol(3);
   root->left->left = new jazol(4);
   root->left->right = new jazol(5);
   deleteTree(root);
   root = NULL;
   return 0;
}
```

Пр. Изминување на дрвото од претходниот пример

```
#include <iostream>

using namespace std;

struct jazol

{

   int x;

   struct jazol* left, *right;

   jazol(int x)

   {

      this->x = x;

      left = right = NULL;

   }

};
```

```
void printPostorder(struct jazol* jazol)
{
    if (jazol == NULL)
        return;
    // prvo se izminuva levoto poddrvo
    printPostorder(jazol->left);
    // potoa se izminuva desnoto poddrvo
    printPostorder(jazol->right);
    // se pecati vrednosta
    cout << jazol->x << " ";
}
/*  inorder izminuvanje*/
void printInorder(struct jazol* jazol)
{
    if (jazol == NULL)
        return;
    /* prvo pristapuva do levoto dete */
    printInorder(jazol->left);
    /* pecati vrednost na jazolot */
    cout << jazol->x << " ";
    /* pristapuva do desnoto dete*/
    printInorder(jazol->right);
}
/*  preorder izminuvanje*/
void printPreorder(struct jazol* jazol)
{
    if (jazol == NULL)
        return;
    /* prvo pristapuva do podatokot */
    cout << jazol->x << " ";
```

```
    /* pristapuva do levoto poddrvo */
    printPreorder(jazol->left);
    /* pristapuva do desnoto poddrvo */
    printPreorder(jazol->right);
}
int main()
{
    struct jazol *root = new jazol(1);
    root->left            = new jazol(2);
    root->right         = new jazol(3);
    root->left->left     = new jazol(4);
    root->left->right = new jazol(5);
    cout<<endl;
    cout << "Preorder izminuvanje e"<<endl;
    printPreorder(root);
    cout<<endl;
    cout << "Inorder izminuvanje e"<<endl;
    printInorder(root);
    cout<<endl;
    cout << "Postorder izminuvanje e"<<endl;
    printPostorder(root);
    return 0;
}
```

Пр. максимална широчина на бинарно дрво

```
          1
        /   \
       2     3
     /   \     \
    4     5     8
                /   \
```

6 7

За примерот:

Широчина на ниво 1 е 1

Широчина на ниво 2 е 2

Широчина на ниво 3 е 3

Широчина на ниво 4 е 2

Значи, максимална широчина е 3.

Прашања поврзани со наставните единици може да се испаќаат на email:
anetastojceska@gmail.com