Data Structures and Algorithms

Mohammad GANJTABESH

Department of Computer Science,
School of Mathematics, Statistics and Computer Science,
University of Tehran,
Tehran, Iran.

mgtabesh@ut.ac.ir

Dept. of Computer Science
University of Tehran

Binary Search Tree

Data Structures and Algorithms
Undergraduate course

Mohammad GANJTABESH
mgtabesh@ut.ac.ir

Dept. of Computer Science
University of Tehran

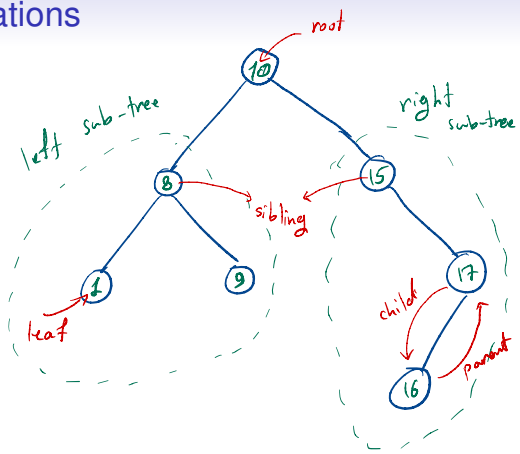
Motivations

$\forall$ node v,

$$key(w) < key(v) < key(u)$$

$w \in$ left sub-tree of root v

$u \in$ right sub-tree of root v



BST

Basic Definitions

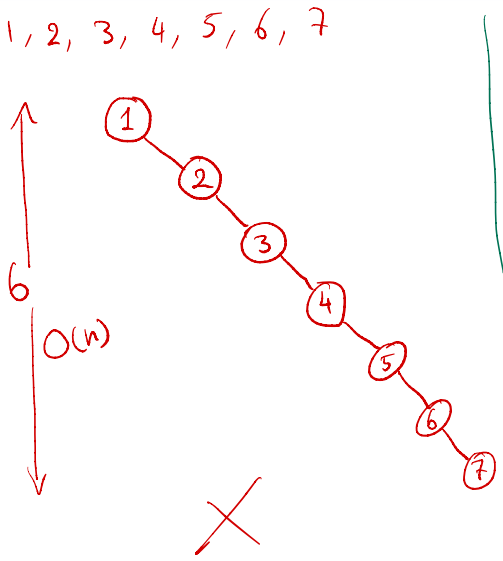
- یک درخت جستجوی دودویی یک درخت دودویی است که در آن خاصیت زیر برای تمامی گره‌ها برقرار است:

$key(w) < key(v) < key(u)$

$w \in$ Left sub-tree

$u \in$ right sub-tree.

Basic Definitions



BST Representations

Using Pointers


```
struct Node {
    int key;
    Node *left;
    Node *right;
    Node (int k){
        key ← k;
        left ← right ← null;
    }
}
```

```
class BST {
    Node * root;
    int size;
    BST (){
        root ← null;
        size ← 0
    }
    // methods
}
```

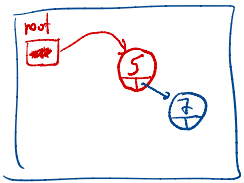
BST Implementation

اضافه کردن یک عنصر به درخت:

```
public Add (int k){
    if (root = null)
        root ← new Node(k);
    else
        Add (k, root);
    size ++;
}
```

```
Private Add (int k, Node * node){
    if (k > node.key){ //go to the right
        if (node.right = null){
            node.right ← new Node(k);
            size ++; return;
        }
        Add (k, node.right);
    }
    if (k < node.key){ //go to the left
        if (node.left = null){
            node.left ← new Node(k);
            size ++;
            return;
        }
        Add (k, node.left);
    }
}
```

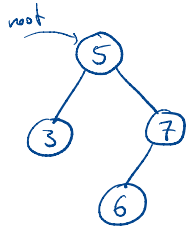
BST



Add (5)
Add (7)
Add (10)

BST Implementation

```
Public bool Contains(int k){
    return (Contains(k, root));
}
Private bool Contains(int k, Node * node){
    if (node = null) return(false);
    if(node.key = k) return(true);
    if (k < node.key)
        return (Contains (node.left));
    else
        return( Contains(node.right));
}
```

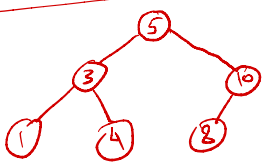


$O(\log n) \leq$ Time complexity $\leq O(n)$

BST Implementation

```
Public Preorder (){
    preorder (root);
}
private preorder (Node *node){
    if (node = null) return;
    visit (node.key);
    preorder (node.left);
    preorder (node.right);
}
```

```
Public Inorder (){
    Inorder (root);
}
private InOrder (Node *node){
    if (node = null) return;
    Inorder (node.left);
    visit (node.key);
    Inorder (node.right);
}
```



Preorder : 5, 3, 1, 4, 10, 8

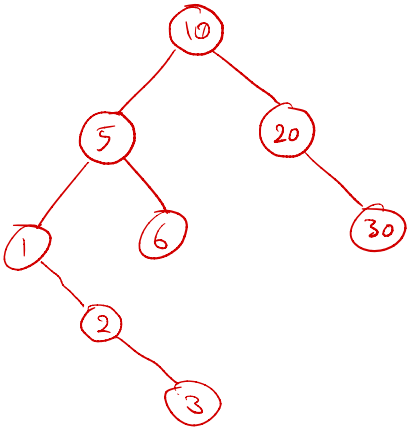
Inorder : 1, 3, 4, 5, 8, 10

Postorder : 1, 4, 3, 8, 10, 5

BST Implementation

پیدا کردن کوچکترین و بزرگترین عنصر در درخت :

```
Node *  FindMin ( Node *node) {
        Node * cur;
        while ( cur . left != null)
                cur ← cur. left ;
        return ( cur) ;
}
Node *  FindMax ( Node  *node) {
        Node *cur ;
        while ( cur. right != null)
                cur ← cur. right ;
        return ( cur) ;
}
```



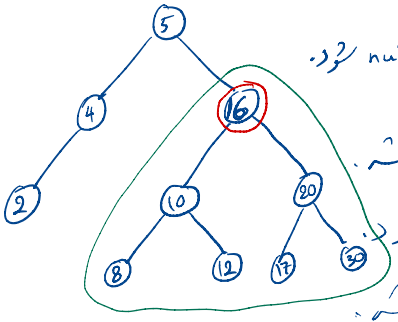
BST Implementation

حذف یک عنصر از درخت:

1) اگر آن عنصر برگ باشد.
کافیست اشاره‌گر پدر آن گره null شود.

2) اگر آن عنصر دارای یک فرزند باشد.
کافیست فرزند جایگزین آن عنصر شود.

3) اگر آن عنصر دارای دو فرزند باشد.



In-order : 8, 10, 12, 16, 17, 20, 30

predecesor

succesor

BST Implementation

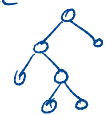
```
Predecessor (Node * node) {
    return (FindMax (node.left));
}
Successor (Node * node) {
    return (FindMin (node.right));
}
```

Variations of (Binary) Search Tree

Binary Tree

- Binary Search Tree : possibly unbalanced . X
- Strictly Balanced : AVL-Tree

- Almost Balanced : Red-Black Tree



k-ary tree

- B-Tree
- B^+-Tree