ADS

JSPM

Page No.: 1

* Assignment NO-: 02 *.

• Title-:

program for constructing a binary search tree and its operation.

• objective-:

To study and implemented the binary search tree.

• problem statement-:

Beginning with empty binary tree: construct BST by inserting the values in the order given after constructing a binary tree.

i) Insert new node.
ii) find number of node in longest path.
iii) Minimum data value found in the tree.
iv) change a tree so that the role of the left & right pointer.
v) search a value.

• Outcome :-

1) input-:

• value for BST, insert Data for left child & Right child • searching a key with given value • find max & min value.

2) Output-:

After constructing BST, display the insert new node & show min & max value & longest path in BST.

- Hardware Requirement :- 8 GB Ram, Dual core processor
- software Requirement :-

  fedora OS, gedit s/w & terminal.

- Theory :-
  - Binary Search Tree (BST) :-

- Def^n :- A Binary Search tree is a binary tree, which is either empty or in which each node contains a key that satisfies the following conditions.

1) All keys are distinct
2) for Every node, x in the tree, the value of all the keys in its left subtree are smallest than the keys value in x.
3) for every node, x in the tree the value of all the keys in its right subtree are larger than the key value in x.

Binary search tree finds its application in searching for e.g. :-

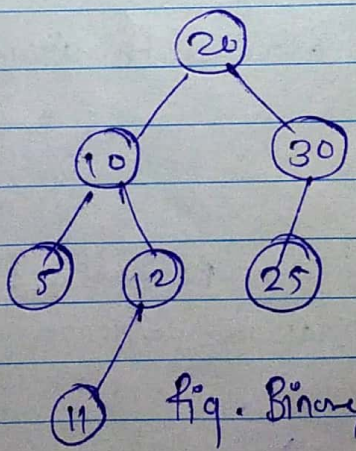


fig. Binary Search tree.

• Operations on Binary Search tree :-

1) Initialise.
2) find.
3) Makempty
4) Insert
5) Delete
6) Create
7) findmin
8) findmax.

Structure of node of binary search tree -

```
Struct BSTnode
{
    int data;
    BSTnode *left, *Right.
};
```

Begining with an empty binary tree, construct binary search tree by inserting the values in the order given.

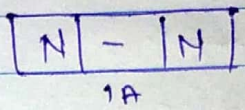
i) Initially the tree is empty and hence the reference pointer root should be set to NULL.

```
BstNode * Initialize()
{
    return (NULL);
}
```

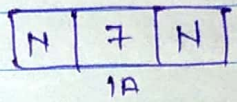
Example-:

Construct BST by inserting the value is given 7, 2, 9, 0, 5, 6, 8, 1.

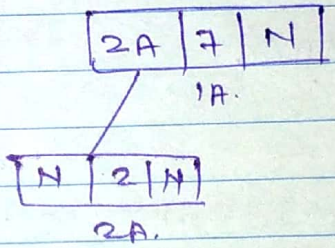
① Step1-: Empty tree-:



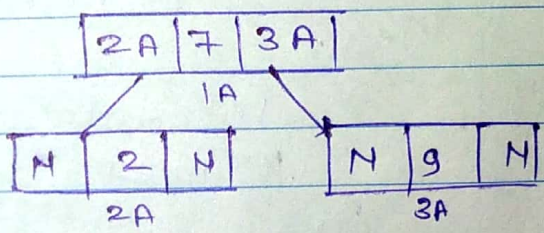
② step2-: insert 7



③ step3-: insert 2

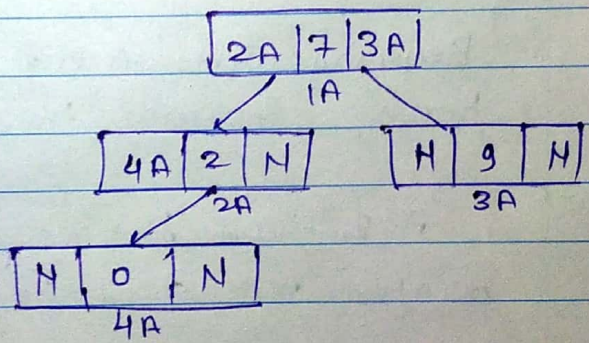


④ step4-: insert 9

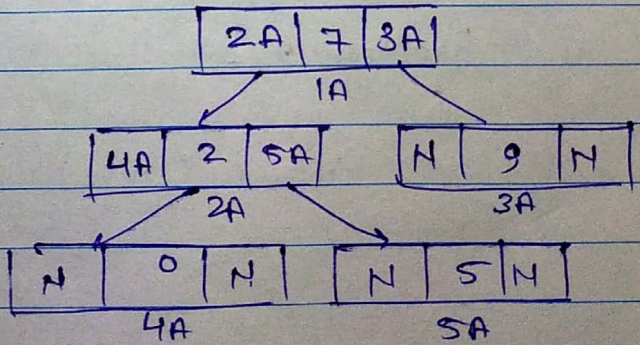


⑤ steps-:

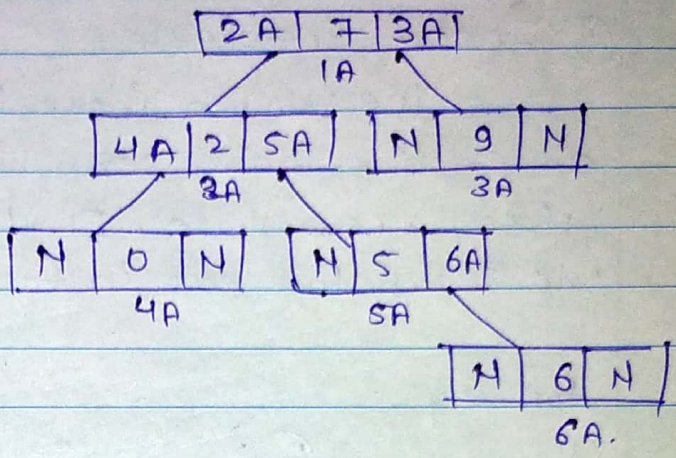
insert 0



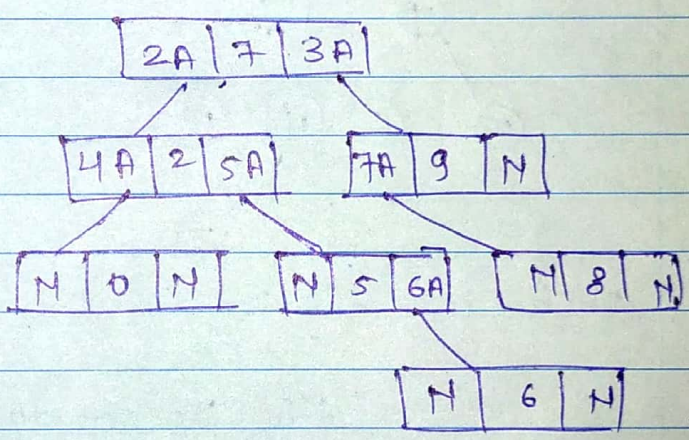
⑥ insert 5:



• Step 7 :- Insert 6 :-

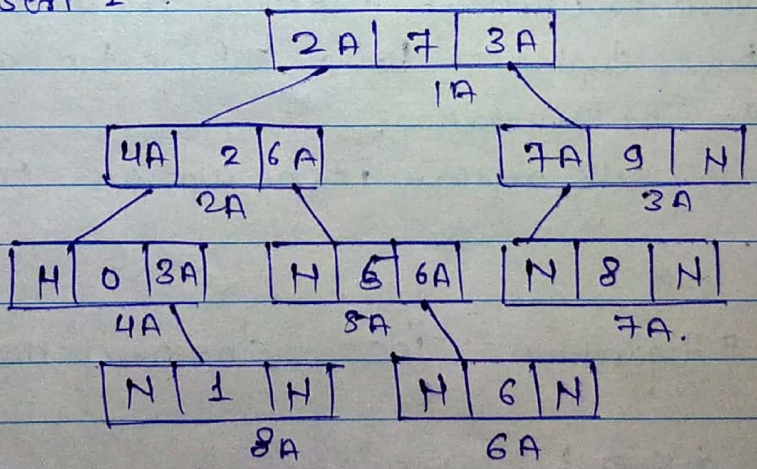


• Step 8 :- Insert 8



• Step 9 :-

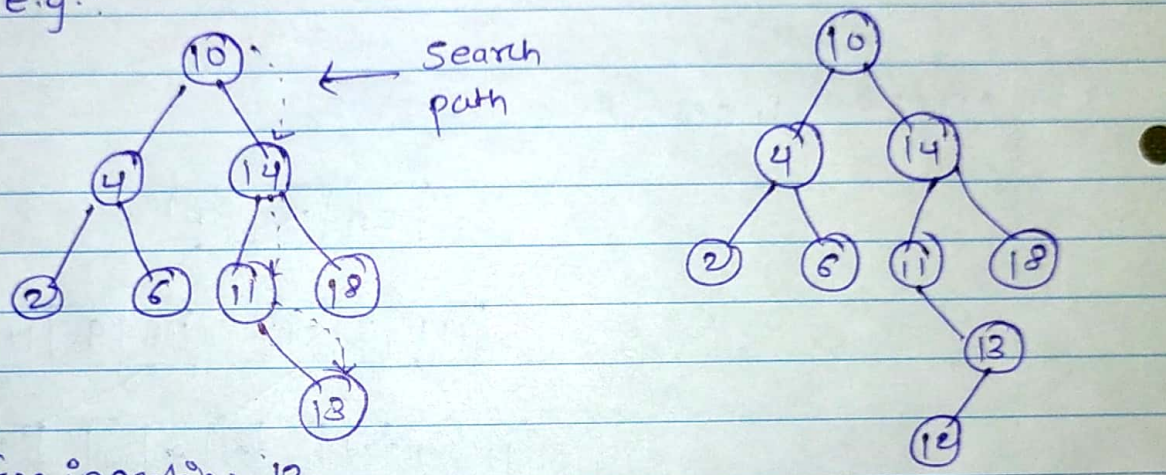
Insert 1 :-



① Insert a new node :-

The function insert (T, X) adds elements X to an existing binary serch tree. T is tested for NULL if so, a new node, If the tree is not empty then we search for X as in find() operations.

for. e.g.:



• before inserting 12 • After inserting 12.

Fig. Insert operation.

If X is already there in the then insert() operation transmits without insertion. as a BST is not allowed to have duplicate keys. If we find a NULL pointer during the find() Operation.

We replace it by a pointer to a new node holding X.

• Algorithm :- insert the new node in BST.

* Step 1 :- Start :

Step 2 :- firstly, we check the gi tree is empty or not,

if (T==NULL) then,
we allocate the memory for node, And in node we insert the data.

```
T=new BSTnode;
T→data=x;
T→left=NULL;
T→right=NULL;
return(T);
```

- Step2-: Then, we insert in a node in right subtree

```
if(x>T→data)
{
  T→right=insert(T→right,x);
  return(T);
}
```

Step3-: Also, we can insert a node in left subtree then, we check same above condition

```
if(x>T→data)
{
  T→Right=insert(T→Right,x);
  return T;
}
else
{
  T→left=insert(T→left,x);
  return T;
}
```

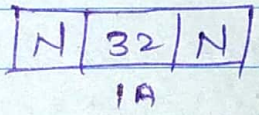
• Minimum data value found in the tree :-

This function Returns to address of the node with smallest value in the tree.

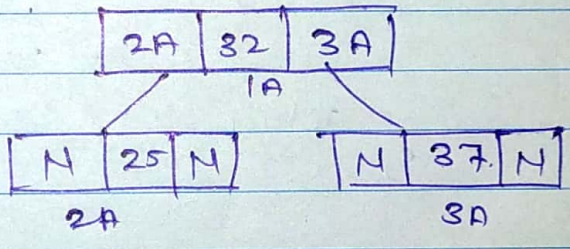
In BST, the smallest value or element as compared with root node is inserted at a the left subtree.

for e.g.: 32, 25, 37, 26

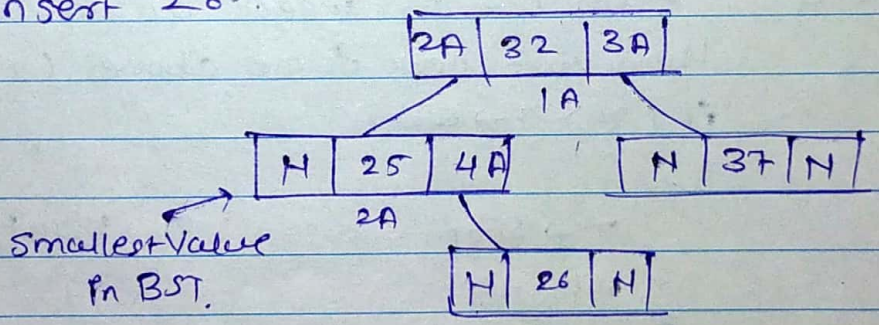
step1 :- insert 32



step2 :- insert 25
37



step3 : insert 26 :-



In above e.g. 25 is less than the 32 so we insert at the left side subtree.

Algorithm :-

```
Step1 :- BSTnode *findmin(BSTNode *T)
         {
            while(T->left != NULL)
            {
               T = T->left;
            }
            return(T);
         }
```

- Maximum data value found in tree :-

- pseudocode :-

```
BSTNode *findmax(BSTNode *T)
{
   while(T->Right != NULL)
   {
      T = T->right;
   }
   return(T);
}
```

* Search a value in BST :-

Recursive algorithm for find :-

```
BSTNode *find(BSTNode *root, int H)
{
   if((root == NULL))
      return(NULL);
   if(root->data == H)
      return(root);
   if(H > root->data)
```

return(find(root→right),H);
return(1+find(root→left),H));
}.

Change a tree so that the role of left & right pointer are swaped at every node.

- pseudocode :

```
Void mirror(node *r)
{
   node *p;
  p1=r→left;
  r→lc=r→rc;
  r→rc=p1;
  if(r→lc!=NULL)
    mirror(r→lc);
 if(r→rc!=NULL)
    mirror(r→rc);
}.
```

* Find Height of the tree : (Number of nodes in logest path)

We are starting from the root and then we cheak left subtree and right subtree of the root, if this is null we return the height. else we traverse left and right again.

We cheak the height of left subtree is greater or less than right subtree and after that we return the greater height of tree.

```
static int i=1, d=1; //Data member of Bst

height (root, 1);
cout<<"\n The no. of nodes (Height) in logest path
is"<<d;

void height(node *r, int l) //fun" Definition
{
    if(r!=NULL)
    {
        if(r->lc==NULL && r->rc==NULL)
        {
            if(l>d)
            d=l;
        }
        else
        {
            height(r->lc, l+1);
            height(r->rc, l+1);
        }
    }
}
```

- **Conclusion -:**

Hence, we studies the implementation of BST.