Binary Tree - Array

ALGORITHM

I. int searchkey (int i, int key)

1. START
2. if (arr [i] == key)
    1. return i.
3. else if (2 * i <= len(arr)) // len (arr) ← returns leng
    1. x = searchkey (2* i, key)
4. else
    1. return 0;
5. End If
6. If (x == 0 && (2 * i + 1 <= len (arr))
    1. x = searchkey (2* i + 1, key);
7. ~~End If~~ Else
    1. return -1
8. ~~STOP~~.
8. return x
9. STOP

II. void bt_delleaf (int node) {

1. START
2. i = searchkey (1, node)
3. if (i == -1) ~~do~~ Then
    1. print ("Node does not exists")
4. Else if (arr[i*2] == 0 && arr[2*i+1] == 0)
    1. print (arr[i] + " is removed")
    2. arr[i] = 0;
    ~~3. }~~
5. Else
    1. print(arr[i] + " is not a leaf node")
6. STOP

III void build_tree (int i, int item)

1. START
2. arr[i] = item, L = 0, R = 0
3. print(" does " + item + " have a left node")
4. scanf ("%d", &L)
5. if (L != 0)
    1. print "Left child val:"
    2. scanf ("%d", &x)
    3. build_tree (2*i, x);

6. End If
7. print ("does "+ item + "have a right node (0/1):")
8. scanf ("%d", &R).
9. If (R != 0)
    1. print("right child value:")
    2. scanf ("%d", &x)
    3. build_tree (2*i+1, x);
10. End If
11. STOP

IV void bt_insert (int node)

1. START
2. i = searchkey (1, node)
3. if (len (arr) == 0)
    1 build_tree (1, node)
    2. return
4. Else if (i == -1)
    1. print (" node + "is not a btree element")
5. Else if (arr[2*i] != 0 && arr[2*i+1] != 0)
    1. printf (node " has a children")
6. Else
    1. print ("enter val to insert")

2. scanf ("%d", &item)
3. if (arr[2*i] == 0)
    1. print ("add to the left of" + arr[i])
    2. scanf ("%d", &l);
    3. if (l != 0)
        1. arr[2*i] = item;
        2. print (arr[2*i] + "was successfully added")
    4. End If
4. If (arr[2*i+1] == 0 && l == 0)
    1. print ("add to the right of" + arr[i])
    2. scanf ("%d", &r)
    3. if (r != 0)
        1. arr[2*i+1] = item;
        2. print (item + "was successfully added")
    4. End If
5. End If
6. STOP

V void dispinorder (int i)
1 START
2. If (i < len(arr) && arr[i] != 0)
    1. dispinorder (2*i)
    2. print ("\t" + arr[i])

3. j++
4. dispinorder (2*i+1)

3. End If

4. STOP

VI void dispreorder (int i)

1. START
2. If (j<=len(arr) && arr[i]!=0)
   1. j++
   2. print ("\t" + arr[i])
   3. dispreorder (2*i)
   4. dispreorder (2*i+1)
3. End If
4. STOP

VII void dispostorder (int i)

1. START
2. If (j<=len(arr) && arr[i]!=0)
   1. dispostorder (2*i)
   2. dispostorder (2*i+1)
   3. printf ("\t" + arr[i]
   4. j++
3. End If
4. STOP

VIII

int main()

1. START
2. build_tree (1, root)
3. Insert, delete and display traversal as per the user's choice
4. STOP.

```
#include <stdio.h>

#define n 65

int arr[n],stack[n/2],root,tmp=0,i,x=0,sz=0,last=0;
int R=0,L=0;

//searchkey without recurrsion
int searchkey2(int j,int key){
    int top=0,i=-1;
    tmp=arr[j];
    while(i!=-1 || j!=-1){
        if(tmp==key)
            return j;
        if(i>=0 && arr[2*j]==0 && arr[2*j+1]==0){
            j=stack[i];
            j=2*j+1;
            tmp=arr[j];
            --i;
        }
        else if(arr[2*j]==0 && arr[2*j+1]!=0){
            j=2*j+1;
            tmp=arr[j];
        }
        else if(arr[2*j+1]==0 && arr[2*j]!=0){
            j=2*j;
            tmp=arr[j];
        }
        else if(arr[2*j]!=0 && arr[2*j+1]!=0){
            i++;
            stack[i]=j;
            j=2*j;
            tmp=arr[j];
        }
        else
            j=-1;
        //printf("%d,%d\n",j,i);
    }
    return j;
}

// searckey using recurrsion
int searchkey(int i,int key){
   // if(j<=sz && arr[i]!=0){
        if(arr[i]==key)
            return i;

        if(2*i<=sz)
            x=searchkey(2*i,key);
        else
```

```
            return 0;

        if ((x==0) && (2*i+1<= sz))
            x=searchkey(2*i+1,key);
        else
            return -1;
        return x;
//    }
}


void bt_deleaf(int node){
    i=searchkey2(1,node);
    if(i==-1)
        printf("Node does not exist\n");
    else if(arr[2*i]==0 && arr[2*i+1]==0){
        printf("%d is removed\n",arr[i]);
        arr[i]=0;
        sz--;
    }
    else
        printf("%d is not a leaf node\n",arr[i]);
}

void build_tree(int i, int item){
    sz++;
    arr[i]=item;
    R=0;
    L=0;
    printf("does [%d] have a left node (y-1/n-0) : ",item);
    scanf("%d",&L);
    if(L!=0){
        printf("left child value : ");
        scanf("%d",&x);
        build_tree(2*i,x);
    }

    printf("does [%d] have a right node (y-1/n-0) : ",item);
    scanf("%d",&R);
    if(R!=0){
        printf("right child value : ");
        scanf("%d",&x);
        build_tree(2*i+1,x);
    }
}

void bt_insert(int node){
    i=searchkey2(1,node);
    int l=0,r=0,item;
    if(sz==0){
```

```
        build_tree(1,node);
        return ;
    }
    else if(i==-1)
        printf("%d is not an element of Btree\n",node);
    else if(arr[2*i]!=0 && arr[2*i+1]!=0)
        printf("%d has two children so insertion not possible\n", node);
    else{
        printf("enter the val to insert : ");
        scanf("%d",&item);
        if(arr[2*i]==0){

            printf("\nadd to the left of [%d] (1-yes/0-no) :",arr[i]);
            scanf("%d",&l);
            if(l!=0) {//&& 2*i<=sz){
                arr[2*i]=item;
                printf("%d was succesfully added to left\n",arr[2*i]);
            }
        }
        if(arr[2*i+1]==0 && l==0){

            printf("\nadd to the right of [%d] (1-yes/0-no) :",arr[i]);
            scanf("%d",&r);
            if(r!=0) {//&& 2*i+1<=sz)
                arr[2*i+1]=item;
                printf("%d was succesfully added to right\n",item);
            }
        }
    sz++;
    }
}

int j=0;
void dispinorder(int i){
    if(j<=sz && arr[i]!=0){
        dispinorder(2*i);
        printf("%d\t",arr[i]);
        j++;
        dispinorder(2*i+1);
    }
}
void dispreorder(int i){
    if(j<=sz && arr[i]!=0){
        j++;
        printf("%d\t",arr[i]);
        dispreorder(2*i);
        dispreorder(2*i+1);
    }
}
void dispostorder(int i){
```

```
    if(j<=sz && arr[i]!=0){
        dispostorder(2*i);
        dispostorder(2*i+1);
        printf("%d\t",arr[i]);
        j++;
    }
}
int main() {

    printf("lets build a tree\nstart with the root : ");
    scanf("%d",&root);

    build_tree(1,root);


    int choice, pos, node, item;
    printf("\n1...display\n");
    printf("2...insert_a_child\n");
    printf("3...delete_val\n");
    printf("4...quit\n");
    int quit=1;
    while(quit!=0){
        printf("\nOption : ");
        scanf("%d",&choice);
        switch(choice){
            case 1: if(sz!=0){
                    j=0;
                    printf("preorder : \t");
                    dispreorder(1);
                    j=0;
                    printf("\ninorder : \t");
                    dispinorder(1);
                    j=0;
                    printf("\npostorder : ");
                    dispostorder(1);
                    printf("\n");
                }
                else
                    printf("Empty tree\n");
                break;
            case 2: printf("enter the parent val : ");
                scanf("%d",&node);

                bt_insert(node);
                break;
            case 3: printf("enter the node val to delete : ");
                scanf("%d",&node);

                bt_deleaf(node);
                break;
```

```
            case 4: quit=0;
                break;
            default:
                printf("\n1...display\n");
                printf("2...insert_a_child\n");
                printf("3...delete_val\n");
                printf("4...quit\n");
        }
    }

    return 0;
}
```

```
lets build a tree
start with the root : 10
does [10] have a left node(y-1/n-0):1
left child value : 5
does [5] have a left node(y-1/n-0):1
left child value : 3
does [3] have a left node(y-1/n-0):0
does [3] have a right node(y-1/n-0):0
does [5] have a right node(y-1/n-0):1
right child value : 4
does [4] have a left node(y-1/n-0):0
does [4] have a right node(y-1/n-0):0
does [10] have a right node(y-1/n-0):1
right child value : 20
does [20] have a left node(y-1/n-0):0
0does [20] have a right node(y-1/n-0):

1...display
2...insert_a_child
3...delete_val
4...quit

Option : 1
preorder :  10  5   3   4   20
inorder :   3   5   4   10  20
postorder : 3   4   5   20  10
sz=5

Option : 2
enter the parent val : 20
enter the val to insert : 55

add to the left of [20] (1-yes/0-no) :1
55 was succesfully added to left
```

```
add to the left of [20] (1-yes/0-no) :1
55 was succesfully added to left

Option : 1
preorder :  10  5   3   4   20  55
inorder :   3   5   4   10  55  20
postorder : 3   4   5   55  20  10
sz=6

Option : 3
enter the node val to delete : 4
4 is removed

Option : 1
preorder :  10  5   3   20  55
inorder :   3   5   10  55  20
postorder : 3   5   55  20  10
sz=5

Option : 3
enter the node val to delete : 3
3 is removed

Option : 1
preorder :  10  5   20  55
inorder :   5   10  55  20
postorder : 5   55  20  10
sz=4
```