15/10/21

Page No.:
Date:
YOUVA

Week - 8

1) Construct a binary search tree & Traverse the tree using all 3 methods and display the elements in the tree.

```
#include <stdio.h>
#include <stdlib.h>
Struct node {
int value;
struct node * left;
Struct node * right;
} * root = NULL, * temp = NULL, *t2, *t1, *tP;

int key;
int s;
int n;
int count;
Void Search (Struct node *t);
Void insert ()
{
int data;
Printf ("Enter data to be inserted");
Scanf ("%d", &data);
temp = (Struct node *) malloc (sizeof(Struct node));
temp -> value = data;
temp -> left = temp -> right = NULL;
if (root == NULL)
root = temp;
else
Search (root);
```

```
void search (struct node *t)
{
  if ((temp -> value > t -> value) && (t -> right != NULL)
      search (t -> right);
  else if ((temp -> value < t -> value) && (t -> right != NULL)
      t -> right = temp;
  else if ((temp -> value < t -> value) && (t -> left != NULL)
      search (t -> left);
  else if ((temp -> value < t -> value) && (t -> left == NULL))
      t -> left = temp;
}

Void inorder (struct node *t)
{
  if (root == NULL)
  {
    Printf ("No elements in the tree \n");
    return;
  }
if (t -> left != NULL)
  inorder (t -> left);
  Printf ("%d" -> ", t -> value);
if (t -> right != NULL)
inorder (t -> right);
}

Void Preorder (struct node *t)
{
  if (root == NULL)
  {
    Printf ("No elements in tree \n");
    return;
```

```
if (t->left != NULL)
  Postorder (t->left);
if (t->right != NULL)
  Postorder (t->right);
Printf ("%d -> ", t->value);
}

struct node *maxvaluenode (struct node *t)
{
  struct node* current = t;
  While (current && current->right != NULL)
    current = current->right;
  return current;
}
int main ()
{
  int choice;
  While (1)
  {
  Printf ("\n*** menu ***\n");
  Printf ("1. insert an Element into tree\n");
  Printf ("2. to get the max value in tree\n");
  Printf ("3. to Print tree Elements in order traversal\n");
  Printf ("4. to Print the tree Elements in Preorder traversal\n");
  Printf ("5. to Print tree elements in Postorder traversal\n");
  Printf ("6. to exit\n");
  Printf ("Enter your choice\n");
  Scanf ("%d", &choice);
  }
```

```
Switch (choice)
{
Case 1 :
  insert ();
  break;
Case 2 :
  tp = maxvalue node (root);
  Printf ("***** Max Value Node *****");
  Printf ("Node with Maximum Value = %d", tp -> value);
  break;
Case 3 :
  Printf ("*** inorder traversal ***\n");
  inorder (root);
  break;

Case 4 :
  Printf ("*** Order traversal ***");
  Preorder (root);
  break;
Case 5 :
  Printf ("*** postorder traversal ***");
  Postorder (root);
  break;

Case 6 :
  exit(0)
Default :
  Printf ("Invalid choice");
  break;
}
break return 0;
}
```

output: *** menu ***
1) insert an Element into tree
2) to get the max value
3) to Print the tree Elements in inorder Traversal
4) to Print the tree Elements in Preorder Traversal
5) to Print the tree Elements in Postorder traversal
6) to Exit

Enter your choice:
1. insert an Element into tree

~~Enter your choice~~.
Enter data to be inserted
20

Enter your choice: 1

Enter the data to be inserted
30

Enter your choice: 1
enter the data to be inserted
10

Enter your choice: 2
Node with max value is = 30

Enter your choice: 3
inorder traversal
10 → 20 → 30 →

Enter your choice : 4

Preorder traversal

20 → 10 → 30 →

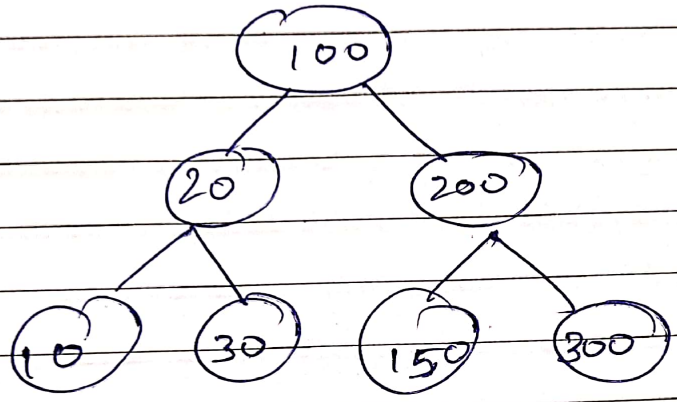
Enter your choice : 5

Postorder traversal

10 → 30 → 20 →

Enter your choice : 6

Exit.



Inorder traversal

left, root, right

10 → 20 → 30 → 100 → 150 → 200 → 300 →

Preorder traversal

~~Root, left, right~~

100 → 20 → 10 → 30 → 200 → 150 → 300 →

Postorder traversal

left, right, root

10 → 30 → 20 → 150 → 300 → 200 → 100 →

15/2/24

Leetcode 3

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */

/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize) {
    int length = 0;
    struct ListNode* current = head;
    while (current != NULL) {
        length++;
        current = current->next;
    }
    int partSize = length / k;
    int extraNodes = length % k;
    struct ListNode** result = (struct ListNode**)malloc(k * sizeof(struct ListNode*));
    current = head;
    for (int i = 0; i < k; i++) {
        int currentPartSize = partSize + (i < extraNodes ? 1 : 0);
```

dividing length by k to determine avg size of each part

extra nodes that needs to be distributed among parts

```
result[i] = current;
for (int j=0; j<currentPartSize-1 && current!=NULL; j++) {
    current = current->next;
}
if (current!=NULL) {
    struct ListNode* nextNode = current->next;
    current->next = NULL;
    current = nextNode;
}
}
*returnSize = k;
return result;
}
```

Output: input = head = [1, 2, 3]

output = [[1], [2], [3], [], []]

[[1], [2], [3], [], []]

```c
struct ListNode** splitListToParts(struct ListNode* head, int k, int* returnSize)
{
    // Calculate the length of the linked list
    int length = 0;
    struct ListNode* current = head;
    while (current != NULL) {
        length++;
        current = current->next;
    }

    // Calculate the size of each part and the number of nodes that will have an
extra node
    int partSize = length / k;
    int extraNodes = length % k;

    // Initialize the array to store the heads of the parts
    struct ListNode** result = (struct ListNode*)malloc(k * sizeof(struct
ListNode));

    // Split the linked list into parts
    current = head;
    for (int i = 0; i < k; i++) {
        // Determine the size of the current part
        int currentPartSize = partSize + (i < extraNodes ? 1 : 0);

        // Store the head of the current part in the result array
        result[i] = current;

        // Move to the end of the current part
        for (int j = 0; j < currentPartSize - 1 && current != NULL; j++) {
            current = current->next;
```

```
    current = head;
    for (int i = 0; i < k; i++) {
        // Determine the size of the current part
        int currentPartSize = partSize + (i < extraNodes ? 1 : 0);

        // Store the head of the current part in the result array
        result[i] = current;

        // Move to the end of the current part
        for (int j = 0; j < currentPartSize - 1 && current != NULL; j++) {
            current = current->next;
        }

        // If there are more nodes, break the link between parts
        if (current != NULL) {
            struct ListNode* nextNode = current->next;
            current->next = NULL;
            current = nextNode;
        }
    }

    // Set the returnSize
    *returnSize = k;

    return result;
}
```

head =

```
[1,2,3]
```

k =

```
5
```

Output

```
[[1],[2],[3],[],[]]
```

Expected

```
[[1],[2],[3],[],[]]
```

```
include<stdio.h>
include<stdlib.h>
truct node{
nt value;
truct node *left;
truct node *right;
*root=NULL,*temp=NULL,*t2,*t1,*tp;

nt key;
nt s;
nt n;
nt count;
oid search(struct node *t);



oid insert()

nt data;
rintf("Enter data to be inserted-");
canf("%d",&data);
 temp=(struct node*)malloc(sizeof(struct node));
 temp->value=data;
 temp->left=temp->right=NULL;
f(root==NULL)

   root=temp;
lse
   search(root);



oid search(struct node*t)

f((temp->value>t->value)&&(t->right!=NULL))
 search(t->right);
lse if((temp->value>t->value)&&(t->right==NULL))
 t->right=temp;
lse if ((temp->value<t->value)&&(t->left!=NULL))
```

```
f((temp->value>t->value)&&(t->right!=NULL))
 search(t->right);
lse if((temp->value>t->value)&&(t->right==NULL))
 t->right=temp;
lse if ((temp->value<t->value)&&(t->left!=NULL))
 search(t->left);
lse if((temp->value<t->value)&&(t->left==NULL))
 t->left=temp;

oid inorder(struct node *t)
 {

f(root==NULL)
   {
rintf("No elements in the tree\n");
eturn;
   }
f(t->left!=NULL)
norder(t->left);
rintf("%d->",t->value);
f(t->right!=NULL)
norder(t->right);
 }
oid preorder(struct node *t)

f(root==NULL)
 {
rintf("No elements in the tree\n");
eturn;
 }
rintf("%d->",t->value);
f(t->left!=NULL)
reorder(t->left);
f(t->right!=NULL)
reorder(t->right);
```

```
oid postorder(struct node *t)


f(root==NULL)
 {
rintf("No elements in the tree\n");
eturn;
 }
f(t->left!=NULL)
ostorder(t->left);
f(t->right!=NULL)
ostorder(t->right);
rintf("%d->",t->value);


truct node* maxvaluenode(struct node* t)

truct node* current = t;

hile (current && current->right != NULL)
        current = current->right;

eturn current;


nt main()


nt choice;
hile(1)
{
rintf("\n***menu**\n");
rintf("1. insert an element into tree\n");
rintf("2. to get the max value in the tree\n");
rintf("3. to print the tree elements in inorder traversal\n");
```

```
printf("4. to print the tree elements in preorder traversal\n");
printf("5. to print the tree elements in postorder traversal\n");
printf("6. to exit\n");
printf("Enter your choice\n");
scanf("%d",&choice);
switch(choice)
   {
case 1:
    insert();
    break;
case 2:
    tp=maxvaluenode(root);
    printf("*****MAX VALUE NODE****");
    printf("Node with Maximum value =%d",tp->value);
    break;
case 3:
    printf("***inorder traversal***\n");
    inorder(root);
    break;
case 4:
    printf("***preorder traversal***\n");
    preorder(root);
    break;
case 5:
    printf("***postorder traversal***\n");
    postorder(root);
    break;

case 6:
    exit(0);
default:
    printf("Invalid choice");
    break;
   }

 }

return 0;
}
```

```
**menu**
. insert an element into tree
. to get the max value in the tree
. to print the tree elements in inorder traversal
. to print the tree elements in preorder traversal
. to print the tree elements in postorder traversal
. to exit
nter your choice

nter data to be inserted-20

**menu**
. insert an element into tree
. to get the max value in the tree
. to print the tree elements in inorder traversal
. to print the tree elements in preorder traversal
. to print the tree elements in postorder traversal
. to exit
nter your choice

nter data to be inserted-30

**menu**
. insert an element into tree
. to get the max value in the tree
. to print the tree elements in inorder traversal
. to print the tree elements in preorder traversal
. to print the tree elements in postorder traversal
. to exit
nter your choice

nter data to be inserted-10

**menu**
. insert an element into tree
. to get the max value in the tree
. to print the tree elements in inorder traversal
. to print the tree elements in preorder traversal
. to print the tree elements in postorder traversal
. to exit
nter your choice

****MAX VALUE NODE****Node with Maximum value =30
**menu**
. insert an element into tree
. to get the max value in the tree
. to print the tree elements in inorder traversal
. to print the tree elements in preorder traversal
. to print the tree elements in postorder traversal
. to exit
nter your choice

**preorder traversal***
0->10->30->
**menu**
. insert an element into tree
. to get the max value in the tree
. to print the tree elements in inorder traversal
. to print the tree elements in preorder traversal
. to print the tree elements in postorder traversal
. to exit
nter your choice
```

```
Enter your choice
1
Enter data to be inserted-10

***menu**
1. insert an element into tree
2. to get the max value in the tree
3. to print the tree elements in inorder traversal
4. to print the tree elements in preorder traversal
5. to print the tree elements in postorder traversal
6. to exit
Enter your choice
2
*****MAX VALUE NODE****Node with Maximum value =30
***menu**
1. insert an element into tree
2. to get the max value in the tree
3. to print the tree elements in inorder traversal
4. to print the tree elements in preorder traversal
5. to print the tree elements in postorder traversal
6. to exit
Enter your choice
4
***preorder traversal***
20->10->30->
***menu**
1. insert an element into tree
2. to get the max value in the tree
3. to print the tree elements in inorder traversal
4. to print the tree elements in preorder traversal
5. to print the tree elements in postorder traversal
6. to exit
Enter your choice
5
***postorder traversal***
10->30->20->
***menu**
1. insert an element into tree
2. to get the max value in the tree
3. to print the tree elements in inorder traversal
4. to print the tree elements in preorder traversal
5. to print the tree elements in postorder traversal
6. to exit
Enter your choice
6

Process returned 0 (0x0)   execution time : 33.196 s
Press any key to continue.
```