8a) Write a program

a)To construct a binary Search tree.

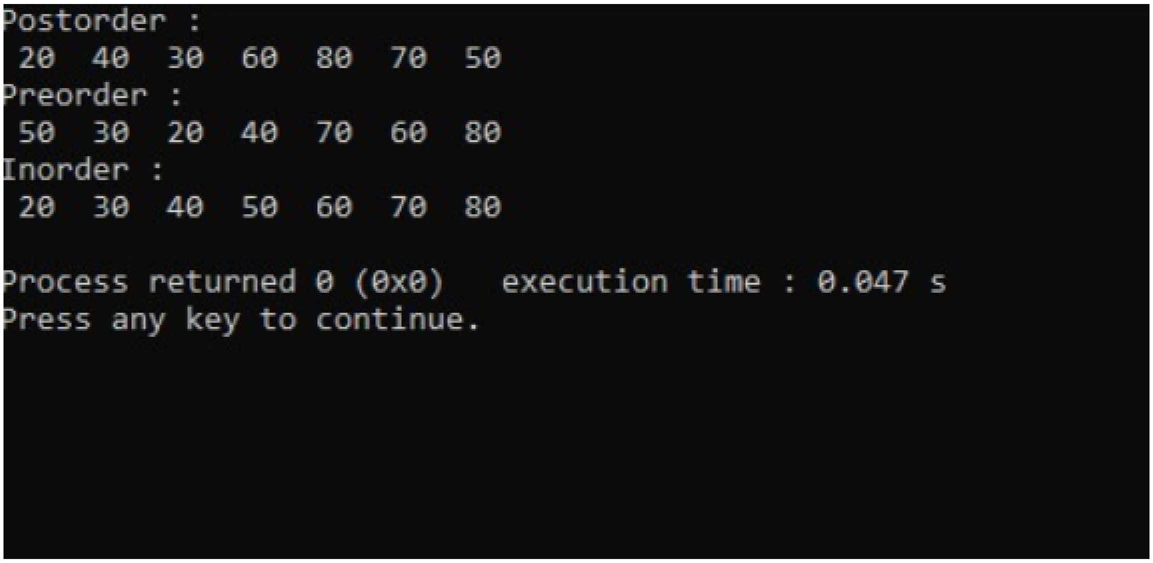
b)To traverse the tree using all the methods i.e., in-order, preorder and post order

To display the elements in the tree.

Code:

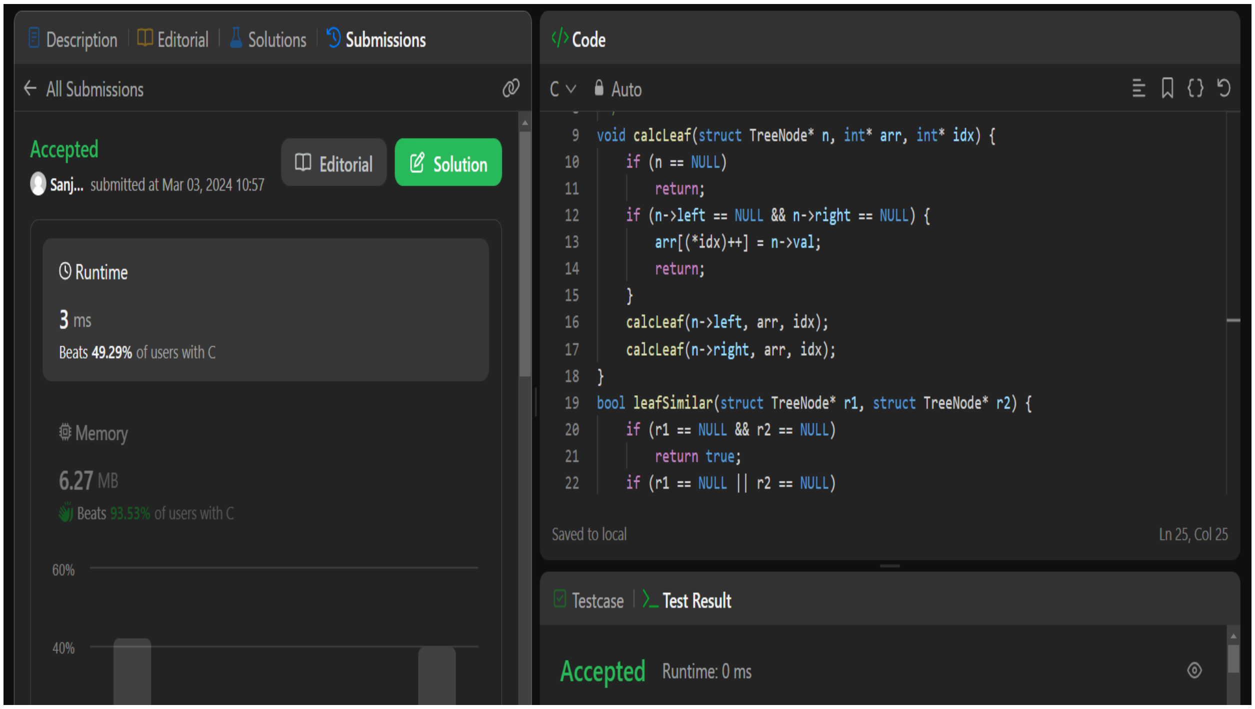
```
#include <stdio.h>
#include <stdlib.h>
struct Node{
 int data;
 struct Node *left, *right;
};
struct Node* newnode(int value)
{
 struct Node* temp= (struct Node*)malloc(sizeof(struct Node));
 temp->data = value;
 temp->left = temp->right = NULL;
 return temp;
}
struct Node* insertNode(struct Node* node, int value)
{
 if (node == NULL) {
 return newnode(value);
 }
 if (value < node->data) {

 node->left = insertNode(node->left, value);
 }
 else if (value > node->data) {
 node->right = insertNode(node->right, value);
 }
 return node;
}
void postOrder(struct Node* root)
{
 if (root != NULL) {
 postOrder(root->left);
 postOrder(root->right);
 printf(" %d ", root->data);
 }
}
void inOrder(struct Node* root)
{
 if (root != NULL) {
 inOrder(root->left);
 printf(" %d ", root->data);
 inOrder(root->right);
 }
}

void preOrder(struct Node* root)
{
 if (root != NULL) {
 printf(" %d ", root->data);
 preOrder(root->left);
 preOrder(root->right);
 }
}
int main()
{
 struct Node* root = NULL;
 root = insertNode(root, 50);
 insertNode(root, 30);
 insertNode(root, 20);
 insertNode(root, 40);
 insertNode(root, 70);
 insertNode(root, 60);
 insertNode(root, 80);
 printf("Postorder :\n");
 postOrder(root);
 printf("\n");
 printf("Preorder :\n");

 preOrder(root);

 printf("\n");

 printf("Inorder :\n");

 inOrder(root);

 printf("\n");

 return 0;

}
```

Output:



```
Postorder :
 20  40  30  60  80  70  50
Preorder :
 50  30  20  40  70  60  80
Inorder :
 20  30  40  50  60  70  80

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

8b) Program - Leetcode platform - Leaf-Similar Trees



Accepted
Sanj... submitted at Mar 03, 2024 10:57

Runtime
3 ms
Beats 49.29% of users with C

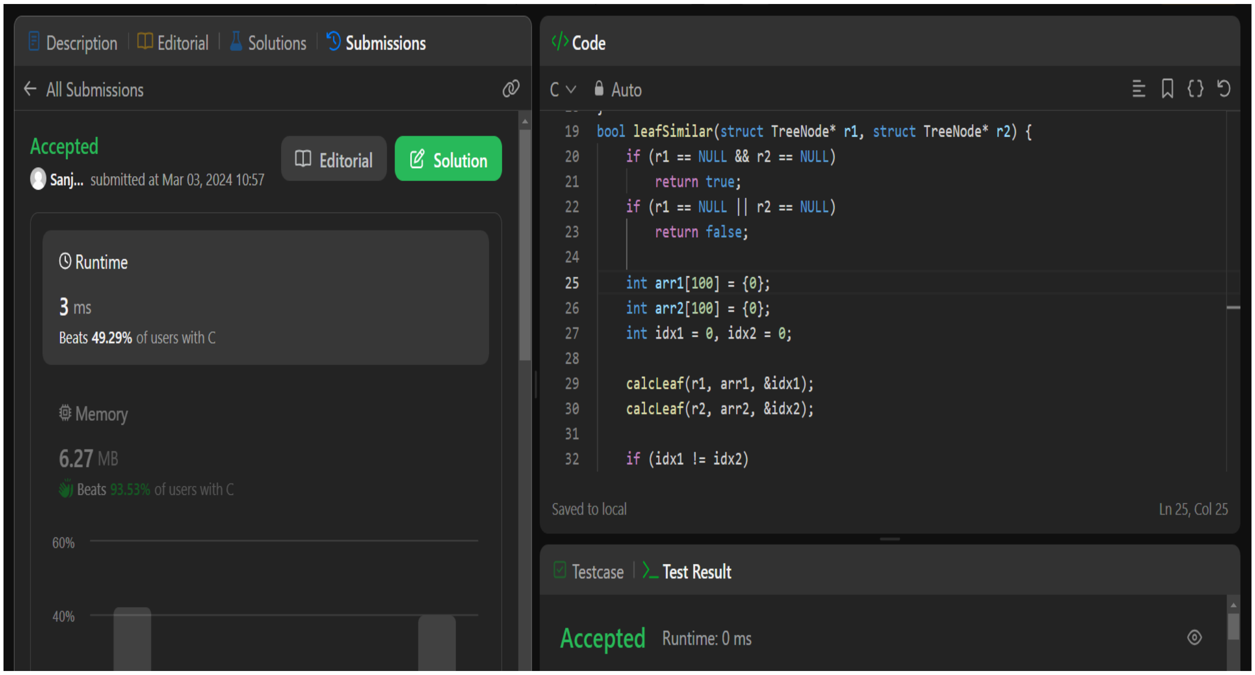
Memory
6.27 MB
Beats 93.53% of users with C

```c
void calcLeaf(struct TreeNode* n, int* arr, int* idx) {
    if (n == NULL)
        return;
    if (n->left == NULL && n->right == NULL) {
        arr[(*idx)++] = n->val;
        return;
    }
    calcLeaf(n->left, arr, idx);
    calcLeaf(n->right, arr, idx);
}
bool leafSimilar(struct TreeNode* r1, struct TreeNode* r2) {
    if (r1 == NULL && r2 == NULL)
        return true;
    if (r1 == NULL || r2 == NULL)
```

Saved to local
Ln 25, Col 25

Testcase | Test Result

Accepted Runtime: 0 ms



Accepted
Sanj... submitted at Mar 03, 2024 10:57

Runtime
3 ms
Beats 49.29% of users with C

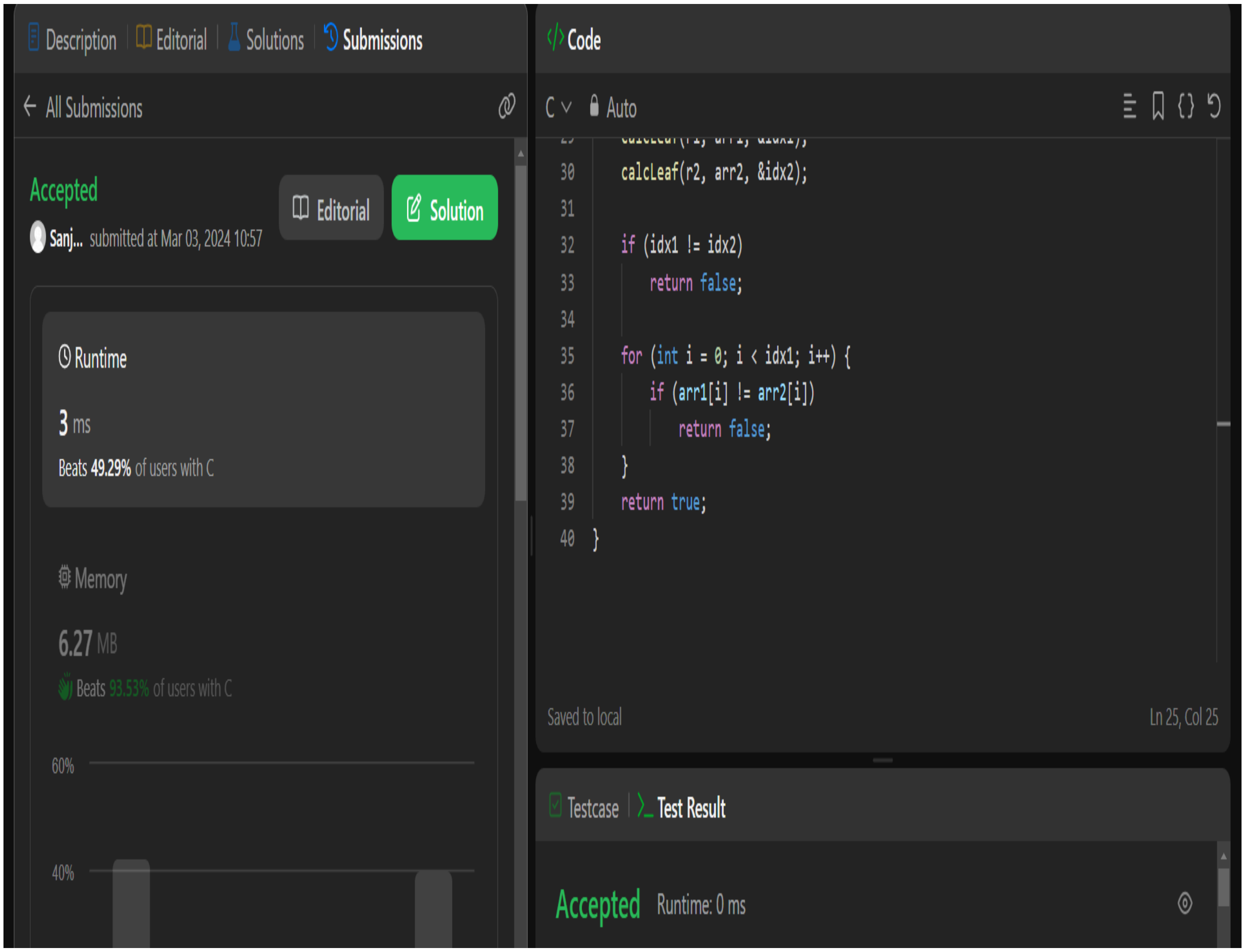
Memory
6.27 MB
Beats 93.53% of users with C

```c
bool leafSimilar(struct TreeNode* r1, struct TreeNode* r2) {
    if (r1 == NULL && r2 == NULL)
        return true;
    if (r1 == NULL || r2 == NULL)
        return false;

    int arr1[100] = {0};
    int arr2[100] = {0};
    int idx1 = 0, idx2 = 0;

    calcLeaf(r1, arr1, &idx1);
    calcLeaf(r2, arr2, &idx2);

    if (idx1 != idx2)
```

Saved to local
Ln 25, Col 25

Testcase | Test Result

Accepted Runtime: 0 ms



Accepted

Sanj... submitted at Mar 03, 2024 10:57

Runtime

3 ms

Beats 49.29% of users with C

Memory

6.27 MB

Beats 93.53% of users with C

```
    calcLeaf(r1, arr1, &idx1);
    calcLeaf(r2, arr2, &idx2);

    if (idx1 != idx2)
        return false;

    for (int i = 0; i < idx1; i++) {
        if (arr1[i] != arr2[i])
            return false;
    }
    return true;
}
```

Saved to local

Ln 25, Col 25

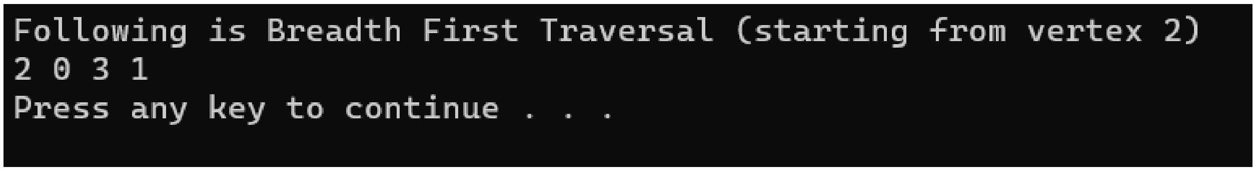
Accepted Runtime: 0 ms

9a) Write a program to traverse a graph using BFS method.

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#define MAX_VERTICES 50
typedef struct Graph_t
{
   int V;
   bool adj[MAX_VERTICES][MAX_VERTICES];
} Graph;
Graph* Graph_create(int V)
{
   Graph* g = malloc(sizeof(Graph));
   g->V = V;

   for (int i = 0; i < V; i++)
   {
      for (int j = 0; j < V; j++)
      {
         g->adj[i][j] = false;
      }
   }
   return g;
}
void Graph_destroy(Graph* g)
{
   free(g);
}
```

```
void Graph_addEdge(Graph* g, int v, int w)
{
    g->adj[v][w] = true;
}

void Graph_BFS(Graph* g, int s)
{
    bool visited[MAX_VERTICES];
    for (int i = 0; i < g->V; i++)
    {
        visited[i] = false;
    }

    int queue[MAX_VERTICES];
    int front = 0, rear = 0;

    visited[s] = true;
    queue[rear++] = s;

    while (front != rear)
    {
        s = queue[front++];
        printf("%d ", s);

        for (int adjacent = 0; adjacent < g->V;
                adjacent++)
        {
            if (g->adj[s][adjacent] && !visited[adjacent])
            {
                visited[adjacent] = true;
        queue[rear++] = adjacent;
      }
    }
  }
}

int main()
{

  Graph* g = Graph_create(4);
  Graph_addEdge(g, 0, 1);
  Graph_addEdge(g, 0, 2);
  Graph_addEdge(g, 1, 2);
  Graph_addEdge(g, 2, 0);
  Graph_addEdge(g, 2, 3);
  Graph_addEdge(g, 3, 3);

 printf("Following is Breadth First Traversal (starting from vertex 2) \n");
  Graph_BFS(g, 2);
  Graph_destroy(g);

  return 0;
}
```

Output:



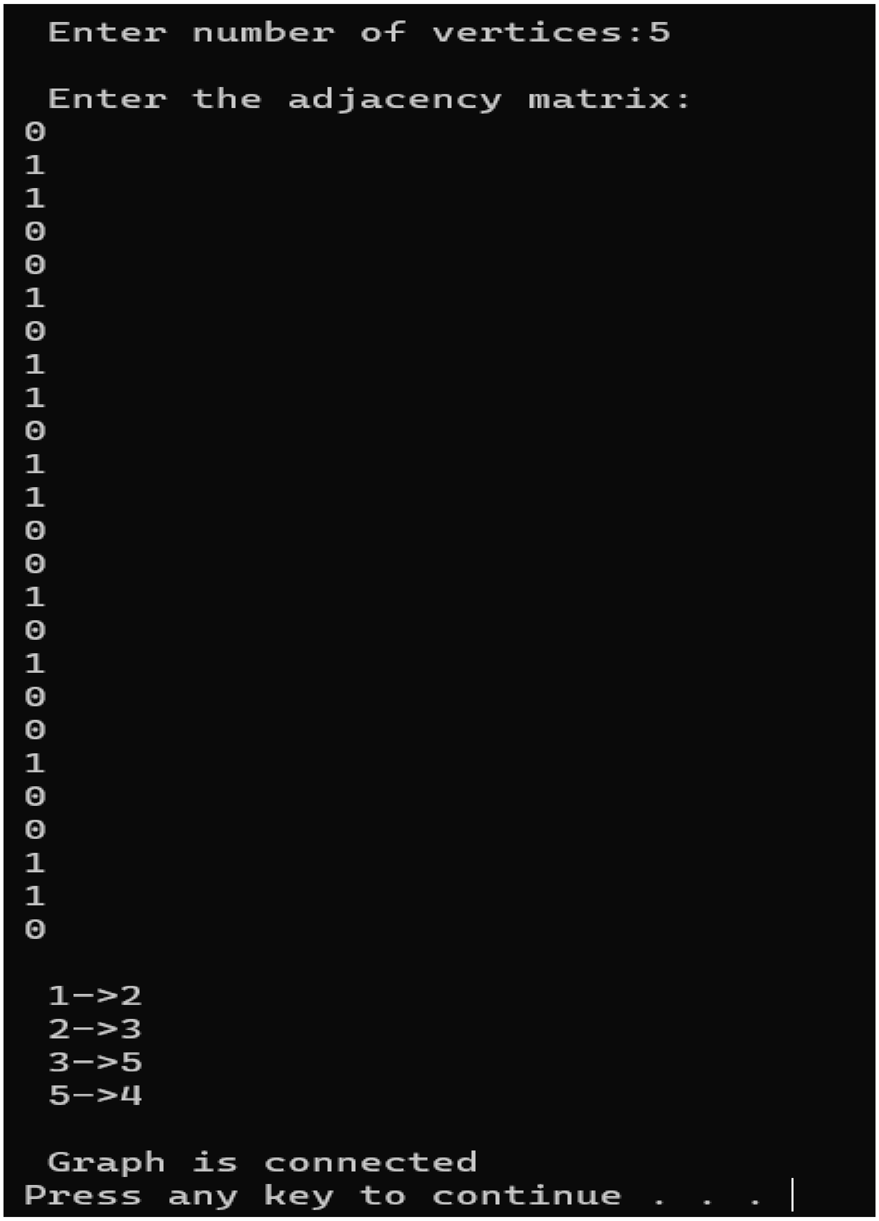
```
Following is Breadth First Traversal (starting from vertex 2)
2 0 3 1
Press any key to continue . . .
```

9b) Write a program to check whether given graph is connected or not using DFS method

```
#include<stdio.h>
int a[20][20], reach[20], n;
void dfs(int v) {
    int i;
    reach[v] = 1;
    for (i = 1; i <= n; i++)
        if (a[v][i] && !reach[i]) {
            printf("\n %d->%d", v, i);
            dfs(i);
        }
}
int main() {
    int i, j, count = 0;
    printf("\n Enter number of vertices:");
    scanf("%d", &n);
    for (i = 1; i <= n; i++) {
        reach[i] = 0;
        for (j = 1; j <= n; j++)
            a[i][j] = 0;
    }
    printf("\n Enter the adjacency matrix:\n");
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            scanf("%d", &a[i][j]);
    dfs(1);
    printf("\n");
    for (i = 1; i <= n; i++) {
        if (reach[i])
            count++;
    }
    if (count == n)
```

```
        printf("\n Graph is connected");
    else
        printf("\n Graph is not connected");
    return 0;
}
```

Output:



```
 Enter number of vertices:5

 Enter the adjacency matrix:
0
1
1
0
0
1
0
1
1
0
1
1
0
0
1
0
1
0
0
1
0
0
1
1
0

 1->2
 2->3
 3->5
 5->4

 Graph is connected
Press any key to continue . . .
```