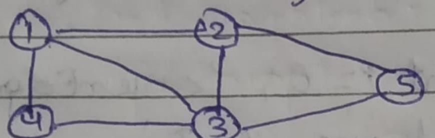
1. Breadth First search

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 100

struct node{
        int data;
        struct node *next;
};

struct graph{
        int numNodes;
        struct node * adjLists[MAX];
        int visited[MAX];
};

st~~r~~uct node * create(int data){
    struct Node * newNode = (struct Node *)malloc
                                  (sizeof(struct Node));
    newNode->data = data;
    newNode->next = NULL;
    return newNode;
}

struct ~~graph~~ * createGraph (int n){     //n = no of nodes
    struct graph *g = (struct graph*)malloc(sizeof
                                      (struct graph));
    ~~grap~~ g->numNodes = n;
    for(int i=0; i<n; i++) {
      ~~grap~~ g->adjLists[i]= NULL;
```

```
        g → visited[i] = 0;
    }
    return g;
}
void addEdge(struct graph *g, int src, int dest){
    struct node * newNode = create(dest);
    newNode → next = g → adjLists[src];
    ~~graph~~ g → adjLists[src] = newNode;

    newNode = create(src);
    newNode → next = g → adjLists[dest];
    g ~~→~~ → adjLists[dest] = newNode;
}

void BFS(struct graph *g, int startNode){
    int queue[MAX];
    int front = 0, rear = 0;
    g → visited[startNode] = 1;
    queue[rear++] = startNode;

    while(front < rear){
        int current = queue[front++];
        print("%d", current);

        struct node * temp = g → adjLists[current];
        while(temp){
            int adjNode = temp → data;
            if(!g → visited[adjNode]){
                g → visited[adjNode] = 1;
                queue[rear++] = adjNode;
            }
            temp = temp → next;
        }
    }
}
```

```
int main(){
    int numNodes;
    printf("Enter the number of nodes:");
    scanf("%d", &numNodes);

    struct graph *g = createGraph(numNodes);

    int numEdges;
    printf("Enter the number of edges: ");
    scanf("%d", &numEdges);

    for(int i=0; i<numEdges; i++){
        int src, dest;
        printf("Enter edge %d(source, dest):", i+1);
        scanf("%d %d", &src, &dest);
        addEdge(g, src, dest);
    }

    int startNode;
    printf("Enter the starting node for BFS
                    traversal:\n");
    scanf("%d", &startNode);
    printf("BFS traversal starting from node %d:",
                                   startNode);

    BFS(g, startNode);

    return 0;
}
//output
Enter the number of nodes: 5
Enter the number of Edges: 7
Enter edge 1 (source destination): 1 2
Enter edge 2 (source destination): 1 3
Enter edge 3 (source destination : 1 4
Enter edge 4 (source destination): 2 3
Enter edge 5 (source destinatination): 2 5
Enter edge 6 (source destination): 3 4
Enter edge 7 (source destination): 3 5

Enter the starting node for BFS traversal: 1
BFS traversal starting from node 1: 1 4 3 2 5
```

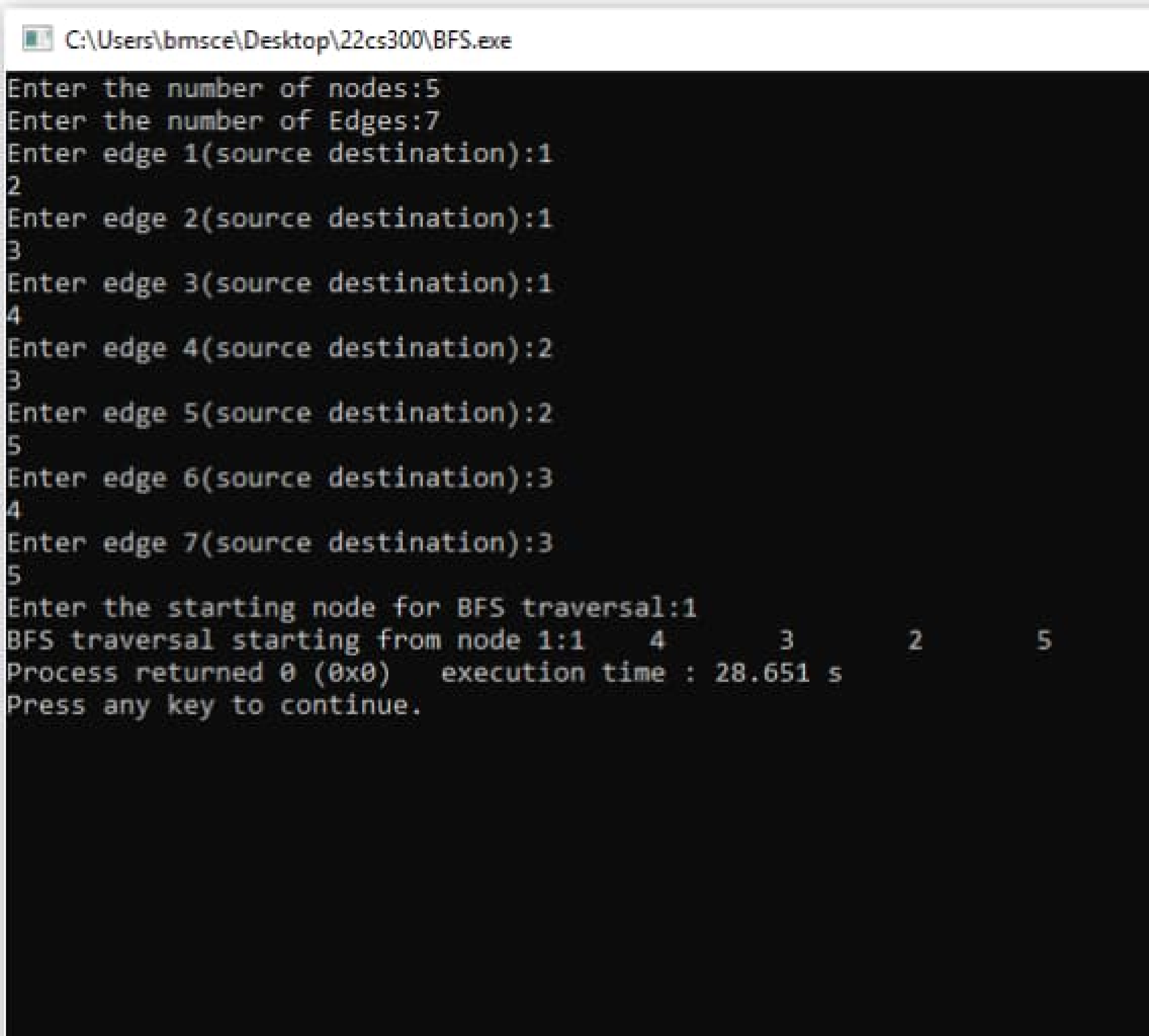


26/2/24

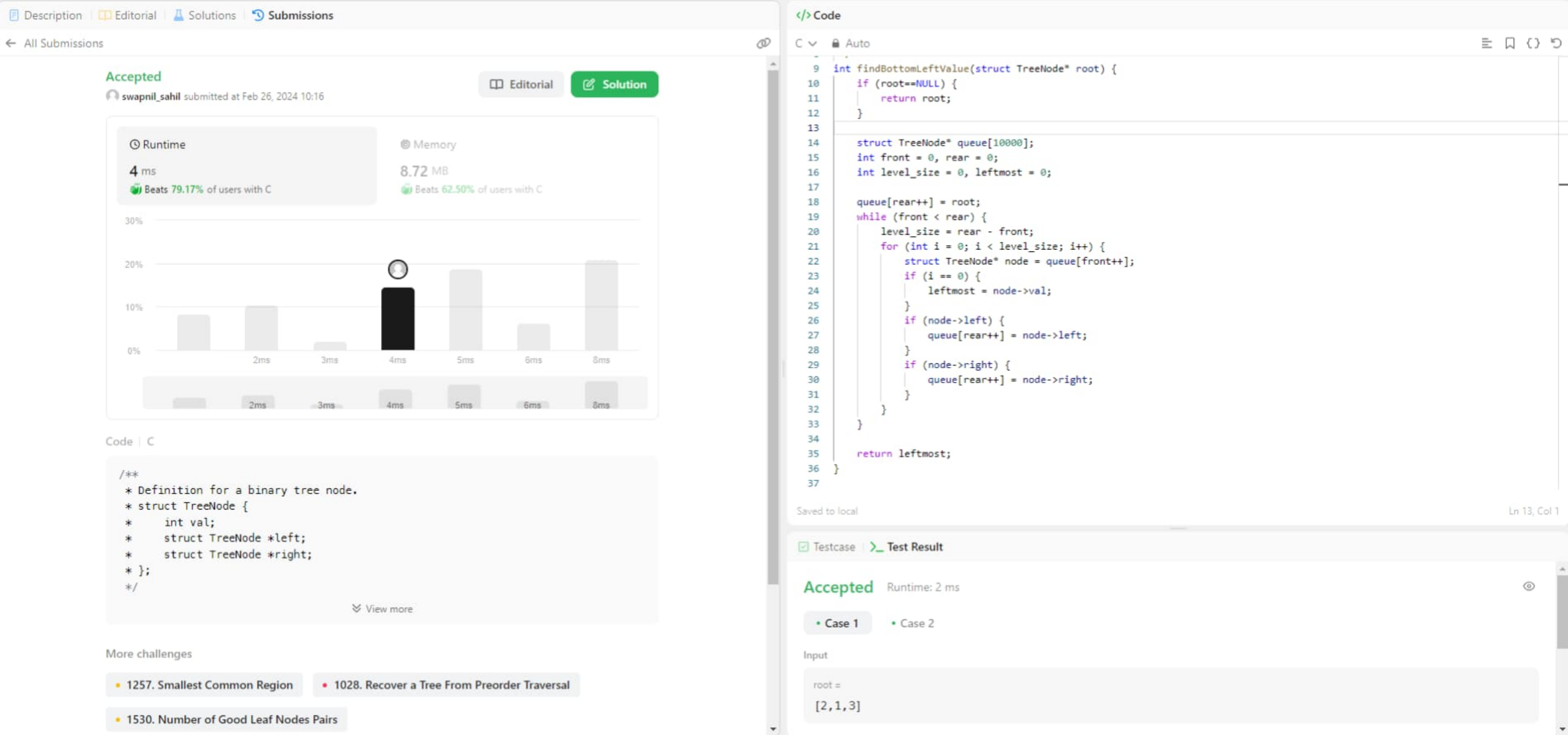
Leetcode 2 : Find Bottom Left Tree value:

```
int findBottomLeftValue (struct TreeNode * root) {
    if (root==NULL) { return root; }
    struct TreeNode * queue[10000]:
    int front = 0, rear = 0;
    int level size = 0, leftmost = 0;
    queue [rear ++] = root;
    while (front < rear) {
        level_size = rear - front;
        for (int i = 0; i< level_size; i++) {
            struct TreeNode * node: queue [front++];
            if (i==0) { leftmost = node->val; }
            if(node->left) { queue [rear++] = node->left;
            } if (node ->right) {
                queue [rear ++] = node->right; }
        }
    }
    return leftmost;
}
```

26/2/24



```
Enter the number of nodes:5
Enter the number of Edges:7
Enter edge 1(source destination):1
2
Enter edge 2(source destination):1
3
Enter edge 3(source destination):1
4
Enter edge 4(source destination):2
3
Enter edge 5(source destination):2
5
Enter edge 6(source destination):3
4
Enter edge 7(source destination):3
5
Enter the starting node for BFS traversal:1
BFS traversal starting from node 1:1    4       3       2       5
Process returned 0 (0x0)   execution time : 28.651 s
Press any key to continue.
```



Accepted

swapnil_sahil submitted at Feb 26, 2024 10:16

Runtime

4 ms

Beats 79.17% of users with C

Memory

8.72 MB

Beats 62.50% of users with C

Code | C

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
```

View more

More challenges

- 1257. Smallest Common Region
- 1028. Recover a Tree From Preorder Traversal
- 1530. Number of Good Leaf Nodes Pairs

```c
int findBottomLeftValue(struct TreeNode* root) {
    if (root==NULL) {
        return root;
    }

    struct TreeNode* queue[10000];
    int front = 0, rear = 0;
    int level_size = 0, leftmost = 0;

    queue[rear++] = root;
    while (front < rear) {
        level_size = rear - front;
        for (int i = 0; i < level_size; i++) {
            struct TreeNode* node = queue[front++];
            if (i == 0) {
                leftmost = node->val;
            }
            if (node->left) {
                queue[rear++] = node->left;
            }
            if (node->right) {
                queue[rear++] = node->right;
            }
        }
    }

    return leftmost;
}
```

Saved to local

Ln 13, Col 1

Accepted Runtime: 2 ms

Case 1 Case 2

Input

root =

[2,1,3]

2. Depth First Search

```
Q. #include <stdio.h>
#inclued <stdlib.h>
#define MAX 100
struct node {
        int data;
        struct node * next;
};
struct Graph{
        int numNodes;
        struct Node * adjLists [MAX];
        int visited [MAX];
 };
struct node * createNode (int data) {
      struct node * newNode = (struct node *) malloc
                              (sizeof (struct node));
      newNode → data = data;
      newNoet → next = NULL;
      return newNode;
}
struct Graph * createGraph (int n) {
     struct Graph *graph = (struct Graph *) malloc
                             (sizeof (struct Graph));
     graph → numNodes = n;
     for (int i = 0; i<n; i++) {
         graph → adjLists [i] = NULL;
         graph → visited [i] = 0;
       }
       return graph;
  }
void addEdge(struct Graph * graph, int src, int dest) {
     struct node * newNode = createNode (dest);
     newNode → next = graph → adjLists [src];
     graph → adjLists [src] = newNode;
```

```
        newNode = createNode(src);
        newNode->next = graph->adjLists[dest];
        graph->adjLists[dest] = newNode;
    }
void DFS(struct Graph *graph, int startNode){
        graph->visited[startNode]=1;
        printf("%d", startNode);

        struct node * temp = graph->adjLists[startNode];
        while(temp){
              int adjNode = temp->data;
              if(!graph->visited[adjNode]){
                    DFS(graph, adjNode);
              }
              temp = temp->next;
          }
    }
  int main(){
        int numNodes;
        printf("Enter the number of nodes:");
        scanf("%d", &numNodes);

        struct Graph *graph = createGraph(numNodes);
        int numEdges;
        printf("Enter the number of edges:");
        scanf("%d", &numEdges);
```

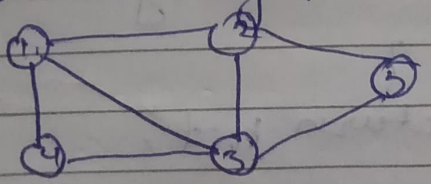
26/2/24

```
        for(int i=0; i<numEdges; i++){
              int src, dest;
              printf("Enter edge %d (src, dest):", i+1);
              scanf("%d %d", &src, &dest);
              addEdge(graph, src, dest);
        }
```

```
    int startNode;
    printf("Enter the starting node for DFS traversal:");
    scanf("%d", &startNode);
    printf("DFS traversal starting from node %d:",
                                          startNode);
    DFS(graph, startNode);

    return 0;
}
```

//Output:

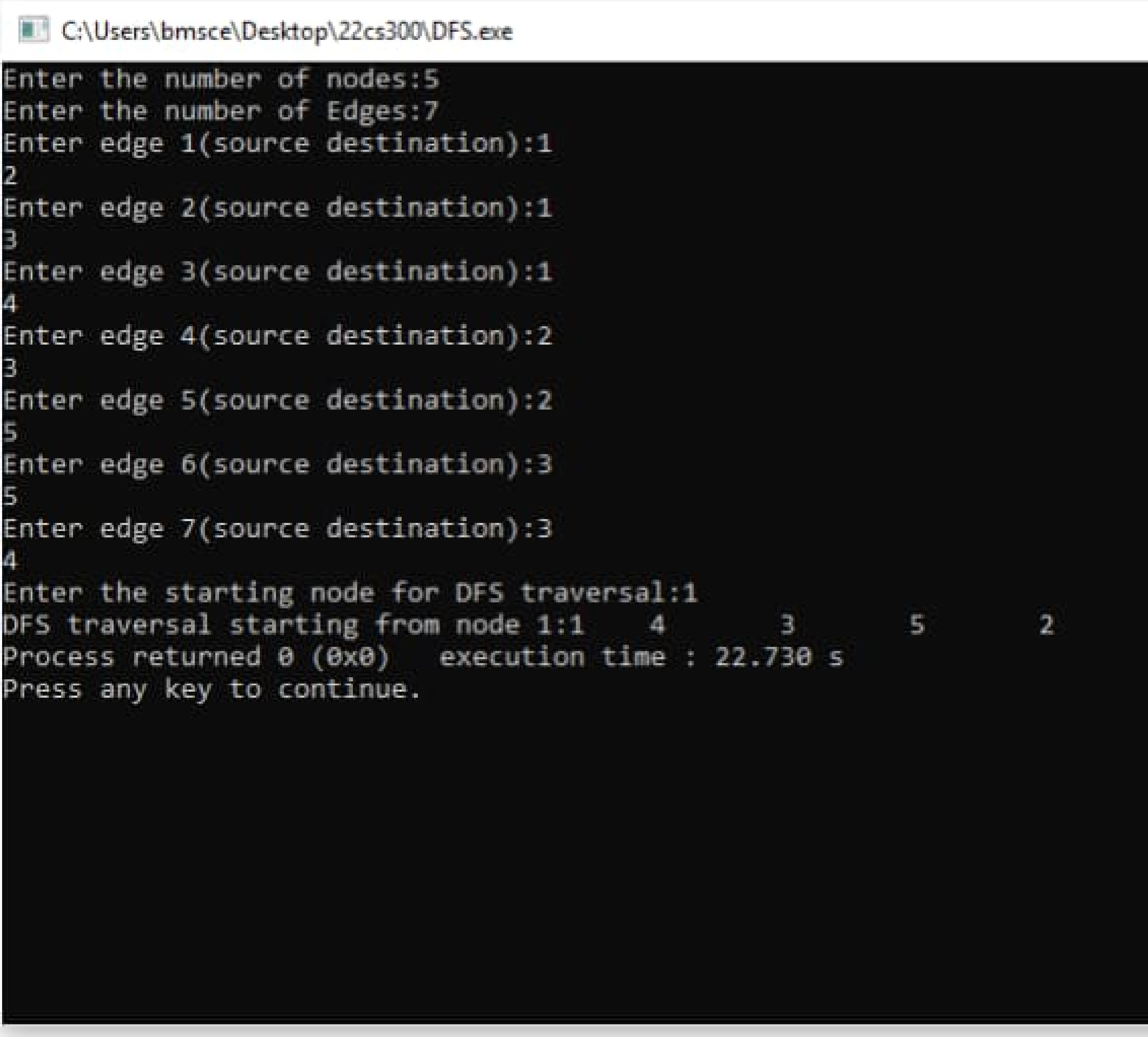
```
Enter the number of nodes: 5
Enter the number of Edges: 7
Enter edge 1 (source destination): 1 2
Enter edge 2 (source destination): 1 3
Enter edge 3 (source destination): 1 4
Enter edge 4 (source destination): 2 3
Enter edge 5 (source destination): 2 5
Enter edge 6 (source destination): 3 4
Enter edge 7 (source destination): 3 5

Enter the starting node for DFS traversal: 1
DFS traversal starting from node 1: 1 4 3 5 2
```



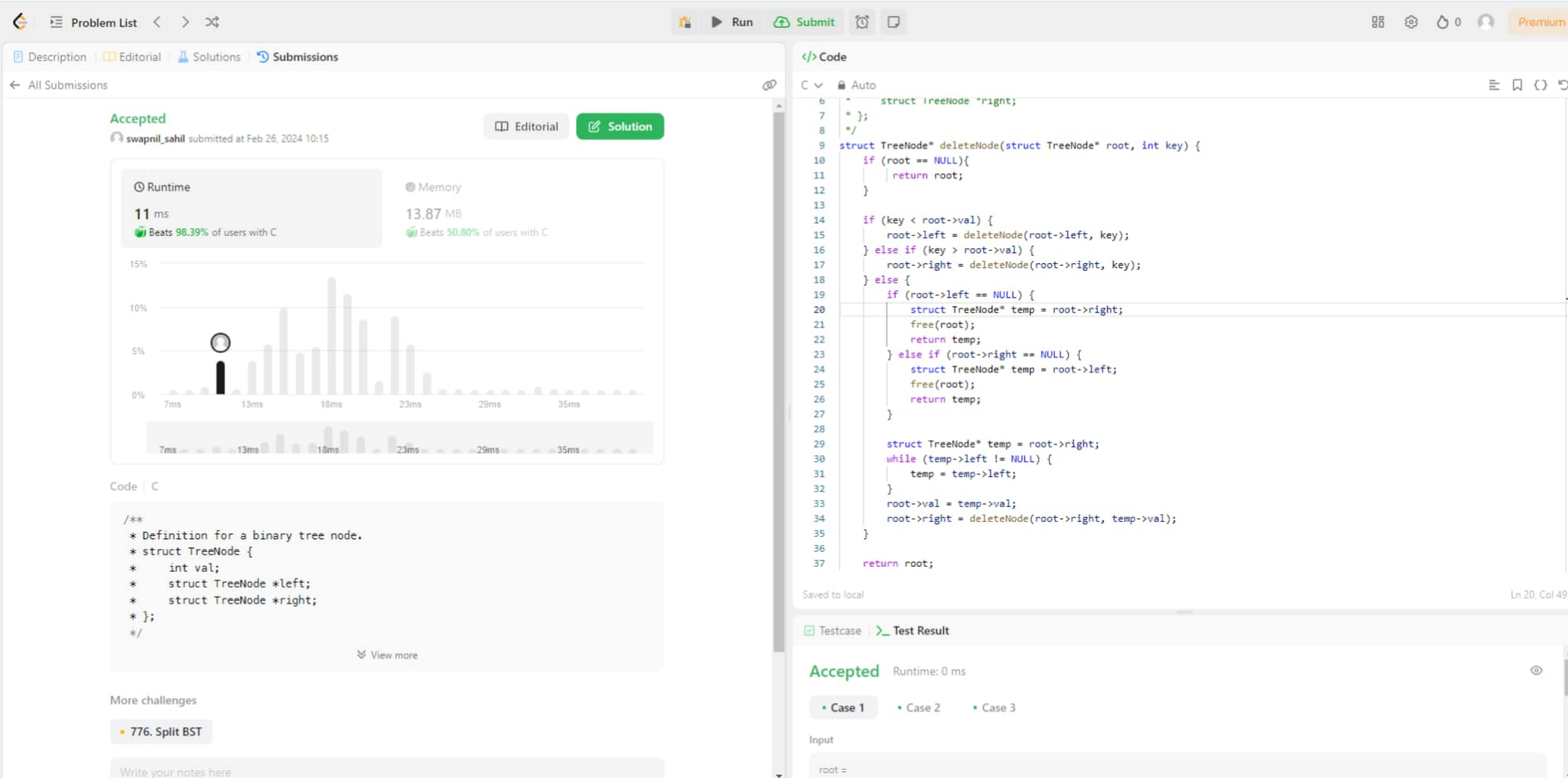
leetcode 1: Delete Node in a BST.

```
struct TreeNode *deleteNode(struct TreeNode *root, int key)
{ if (root == NULL) { return root; }
  if (key < root->val) { root->left = deleteNode(root->left, key)
  } else if (key > root->val)
     { root->right = deleteNode(root->right, key);
     } else {
          if (root->left == NULL) {
               struct TreeNode * temp = root->right;
               free (root);
               return temp;
          } else if (root->right == NULL) {
               struct TreeNode * temp = root->left;
               free (root);
               return temp; }

          struct TreeNode * temp = root->right;
          while (temp->left != NULL) {
                    temp = temp->left;
          }
          root->val = temp->val;
          root->right = deleteNode (root->right,
                                        temp->val);

       }
     return root;
}
```

NP
26/12/24



```
Enter the number of nodes:5
Enter the number of Edges:7
Enter edge 1(source destination):1
2
Enter edge 2(source destination):1
3
Enter edge 3(source destination):1
4
Enter edge 4(source destination):2
3
Enter edge 5(source destination):2
5
Enter edge 6(source destination):3
5
Enter edge 7(source destination):3
4
Enter the starting node for DFS traversal:1
DFS traversal starting from node 1:1    4       3       5       2
Process returned 0 (0x0)   execution time : 22.730 s
Press any key to continue.
```



Accepted

swapnil_sahil submitted at Feb 26, 2024 10:15

Runtime

11 ms

Beats 98.39% of users with C

Memory

13.87 MB

Beats 50.80% of users with C

Code | C

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
```

View more

More challenges

776. Split BST

Write your notes here

```
 *     struct TreeNode *right;
 * };
 */
struct TreeNode* deleteNode(struct TreeNode* root, int key) {
    if (root == NULL){
        return root;
    }

    if (key < root->val) {
        root->left = deleteNode(root->left, key);
    } else if (key > root->val) {
        root->right = deleteNode(root->right, key);
    } else {
        if (root->left == NULL) {
            struct TreeNode* temp = root->right;
            free(root);
            return temp;
        } else if (root->right == NULL) {
            struct TreeNode* temp = root->left;
            free(root);
            return temp;
        }

        struct TreeNode* temp = root->right;
        while (temp->left != NULL) {
            temp = temp->left;
        }
        root->val = temp->val;
        root->right = deleteNode(root->right, temp->val);
    }

    return root;
```

Saved to local

Ln 20, Col 49

Accepted Runtime: 0 ms

Input

root =