26/02/24

BFS & DFS

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_SIZE 100
struct Node {
int data;
struct Node* next;
};
struct Graph {
int numVertices;
struct Node** adjLists;
int* visited;
};
struct Node* createNode(int data) {
struct Node* newNode=(struct Node*) malloc(sizeof(struct Node));
newNode->data =data;
newNode->next = NULL;
return newNode;
}
struct Graph* createGraph(int numVertices)
{
struct Graph* graph =(struct Graph*)malloc(sizeof(struct
Graph));
graph->numVertices = numVertices;
graph->adjLists =(struct Node*) malloc(numVertices * sizeof
(struct Node));
graph->visited =(int*)malloc(numVertices* sizeof(int));
for(int i = 0; i<numVertices; 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 BFS(struct Graph* graph, int startVertex)
{
  int queue[MAX_SIZE];
  int front=-1, rear=-1;
  graph->visited[startVertex] = 1;
  queue[++rear] = startVertex;
  while(front != rear)
  {
    int currentVertex = queue[++front];
    printf("%d", currentVertex);
    struct Node* temp = graph->adjLists[currentVertex];
    while(temp)
    {
      int adjVertex = temp->data;
      if(graph->visited[adjVertex] == 0)
      {
        graph->visited[adjVertex] = 1;
        queue[++rear] = adjVertex;
      }
      temp = temp->next;
    }
  }
}
Void DFS(struct Graph* graph, int vertex)
{
```

```
graph->visited[vertex] = 1;
printf("%d", vertex);
struct Node* temp = graph->adjLists[vertex];
while(temp)
{
  int adjVertex = temp->data;
  if(graph->visited[adjVertex] == 0)
  {
    DFS(graph, adjVertex);
  }
  temp = temp->next;
}
}
int main()
{
struct Graph* graph = createGraph(4);
addEdge(graph, 0, 1);
addEdge(graph, 0, 2);
addEdge(graph, 1, 2);
addEdge(graph, 2, 3);
printf("BFS\n");
BFS(graph, 0);
for(int i = 0; i < graph->numVertices; i++)
{
  graph->visited[i] = 0;
}
printf("\n DFS\n");
DFS(graph, 0);
return 0;
}
```

O/P:-

BFS: 0 2 1 3

DFS: 0 2 3 1

```
BFS
0 2 1 3
DFS
0 2 3 1
Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

26/02/24

→ Delete a node in BST.

```
struct TreeNode* minValueNode(struct TreeNode* node)
{
  struct TreeNode* current = node;
  while(current && current->left != NULL)
    current = current->left;
  return current;
}
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;
    }
  struct TreeNode* temp = minValueNode(root->right);
  root->val = temp->val;
  root->right = deleteNode(root->right, temp->val);
}
return root;
}
```

leetcode.com/problems/delete-node-in-a-bst/submissions/

Problem List

Description Editorial Solutions Submissions

All Submissions

Editorial Solution

Accepted

user0435PL submitted at Feb 26, 2024 10:31

Runtime

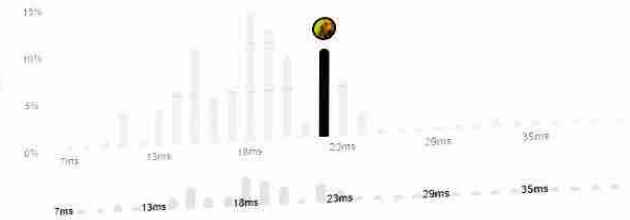
22 ms

Beats 28.62% of users with C

Memory

13.78 MB

Beats 96.27% 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

Code

C Auto

```
48  free(root);
49  return temp;
50  }
51
52
53  struct TreeNode* temp = minValueNode(root->right);
54
55
56  root->val = temp->val;
57
58
59  root->right = deleteNode(root->right, temp->val);
60  }
61  return root;
62  }
63
```

Ln 63, Col 1

Saved to local

Testcase >_ Test Result

Accepted Runtime: 5 ms

- Case 1
- Case 2
- **Case 3**

Input

root =

[]

key =

0

Output

[]

Expected

[]

26/02/24

→ Bottom Left Tree Value

```
void findBottomLeft(struct TreeNode* node, int depth, int* maxDepth,
int *leftmostvalue) {
if (node == NULL)
return;
if (depth > maxDepth)
{
  *maxDepth = depth;
  *leftmostvalue = node->val;
}
findBottomLeft(node->left, depth+1, maxDepth, leftmostvalue);
findBottomLeft(node->right, depth+1, maxDepth, leftmostvalue);
}
int findBottomLeftValues(struct TreeNode* root)
{
  int maxDepth = 0;
  int leftmostvalue = root->val;
  findBottomLeft(root, 1, &maxDepth, &leftmostvalue);
  return leftmostvalue;
}
```

Accepted

user0435PL submitted at Feb 26, 2024 10:33

Editorial | Solution

Runtime

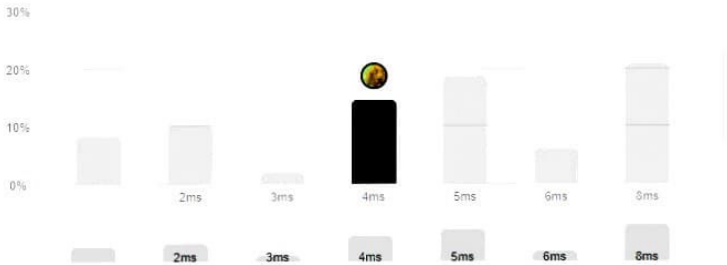
4 ms

Beats 79.17% of users with C

Memory

8.78 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

- 872. Leaf-Similar Trees
- 897. Increasing Order Search Tree
- 655. Print Binary Tree

Code

C | Auto

```
19 findBottomLeft(node->right, depth + 1, maxDepth, leftmostValue);
20 }
21
22 int findBottomLeftValue(struct TreeNode* root) {
23 int maxDepth = 0;
24 int leftmostValue = root->val;
25
26 findBottomLeft(root, 1, &maxDepth, &leftmostValue);
27
28 return leftmostValue;
29 }
30
```

Saved to local

Ln 30, Col 1

Testcase | Test Result

Accepted Runtime: 4 ms

- Case 1
- Case 2

Input

root =

[1,2,3,4,null,5,6,null,null,7]

Output

7

Expected

7

Contribute a testcase