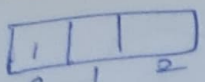


SURYA Gold
Date Page

26/02/24

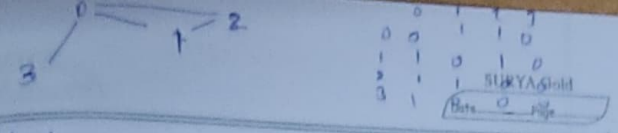
BFS

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_NODES 100
int queue[MaxNodes];
int front = -1, rear = -1;
void enqueue(int node) {
    if (rear == MAX_NODES - 1) {
        printf("Queue is full");
    } else {
        if (front == -1) {
            front = 0;
        }
        rear++;
        queue[rear] = node;
    }
}
int dequeue() {
    int node;
    if (front == -1) {
        printf("Queue is empty");
        return -1;
    } else {
        node = queue[front];
        front++;
        if (front > rear) {
            front = rear = -1;
        }
        return node;
    }
}
```

```
bool isEmpty() {
    return front == -1;
}
void BFS (int adjMat[max_nodes], int node, int
startnode){
    bool visited[max_nodes] = {false};
    visited[startNode] = true;
    enqueue(startnode);
    while (!isEmpty()) {
        int currentNode = dequeue();
        printf("%d", currentNode);

        for(int i=0; i<nodes; i++) {
        if (adjMatrix[currentnode][i] == 1 && !visited[i])
            visited[i] = true;
            enqueue(i);
        }
    }
}
```

Output

Enter the number of nodes: 3
Enter the adjacency matrix:
0 0 1 1
1 0 1
0 1 1
Enter the starting node for BFS: 1
BFS Traversal starting from node 1 : 102



Enter the number of nodes: 4
Enter the adjacency matrix:
0 1 1 0
1 0 1 0
1 1 0 0
1 0 0 0
Enter the starting node for BFS: 2
BFS Traversal starting from node 1 = 2 0 1 3

DFS

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

```
Enter the number of nodes: 4

Enter the adjacency matrix:
0 1 1 0
1 0 1 1
1 1 0 1
0 1 1 0
The nodes visited from 0: 0 1 2 3
The nodes visited from 1: 1 0 2 3
The nodes visited from 2: 2 0 1 3
The nodes visited from 3: 3 1 2 0
```

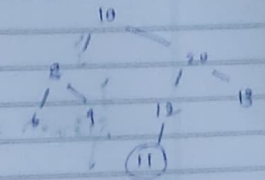
```
}
printf("DFS traversal starts from node 1:\n");
dfs(1);
for (i=1; i<=n; i++){
    if (s[i]) {
        count++;
    }
}
if (count==n){
    printf("\n Graph is connected\n");
}
else{
    printf("\n Graph is not connected\n");
}
return 0;
}
```

Output

```
Enter number of vertices: 3
Enter the adjacency matrix:
0 1 1
1 0 1
1 1 0
DFS traversal starting from node 1: 1 2 3
Graph is connected
```

513 Find Bottom left tree value

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



Output

11

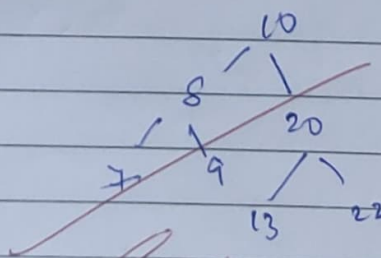
SURYA Gold
Date Page

28/02/24

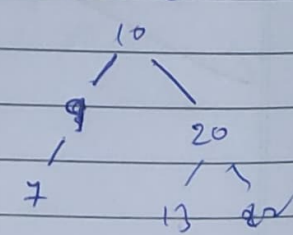
450 Delete node in BST.

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

Output



Delete 8



26.02.24

```
Enter number of vertices: 4
Enter the adjacency matrix:
0 1 1 0
1 0 0 1
1 0 0 1
0 1 1 0
Graph is connected
```

```c
#include <stdio.h>

void bfs(int a[10][10], int n, int u) {
int f = 0, r = -1, q[10] = {0}, v, s[10] = {0};
printf("The nodes visited from %d: ", u);

q[++r] = u;
s[u] = 1;
printf("%d ", u);

while (f <= r) {
u = q[f++];

for (v = 0; v < n; v++) {
if (a[u][v] == 1 && s[v] == 0) {
printf("%d ", v);
s[v] = 1;
q[++r] = v;
}
}
}
printf("\n");
}

int main() {
int n, a[10][10], source, i, j;
printf("\nEnter the number of nodes: ");
scanf("%d", &n);
printf("\nEnter the adjacency matrix:\n");

for (i = 0; i < n; i++) {
for (j = 0; j < n; j++) {
scanf("%d", &a[i][j]);
}
}
```

```
 q[++r] = u;
 s[u] = 1;
 printf("%d ", u);

while (f <= r) {
 u = q[f++];

for (v = 0; v < n; v++) {
if (a[u][v] == 1 && s[v] == 0) {
 printf("%d ", v);
 s[v] = 1;
 q[++r] = v;
}
}
}
 printf("\n");
}

int main() {
 int n, a[10][10], source, i, j;
 printf("\nEnter the number of nodes: ");
 scanf("%d", &n);
 printf("\nEnter the adjacency matrix:\n");

for (i = 0; i < n; i++) {
for (j = 0; j < n; j++) {
 scanf("%d", &a[i][j]);
}
}

for (source = 0; source < n; source++) {
 bfs(a, n, source);
}
 return 0;
}
```

```c
#define MAX_SIZE 100

int n;
int a[MAX_SIZE][MAX_SIZE];
int s[MAX_SIZE];

void dfs(int v) {
s[v] = 1;
for (int i = 1; i <= n; i++) {
if (a[v][i] && !s[i]) {
dfs(i);
}
}
}

int main() {
int i, j, count = 0;

printf("\nEnter number of vertices: ");
scanf("%d", &n);
for (i = 1; i <= n; i++) {
s[i] = 0;
for (j = 1; j <= n; j++) {
a[i][j] = 0;
}
}
printf("Enter the adjacency matrix:\n");
for (i = 1; i <= n; i++) {
for (j = 1; j <= n; j++) {
scanf("%d", &a[i][j]);
}
}
dfs(1);
for (i = 1; i <= n; i++) {
if (s[i]) {
count++;
}
```

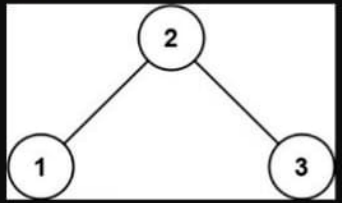
```
if (a[v][i] && !s[i]) {
dfs(i);
}
}
}

int main() {
int i, j, count = 0;

printf("\nEnter number of vertices: ");
scanf("%d", &n);
for (i = 1; i <= n; i++) {
s[i] = 0;
for (j = 1; j <= n; j++) {
a[i][j] = 0;
}
}
printf("Enter the adjacency matrix:\n");
for (i = 1; i <= n; i++) {
for (j = 1; j <= n; j++) {
scanf("%d", &a[i][j]);
}
}
dfs(1);
for (i = 1; i <= n; i++) {
if (s[i]) {
count++;
}
}
if (count == n) {
printf("Graph is connected\n");
} else {
printf("Graph is not connected\n");
}
return 0;
}
```

513. Find Bottom Left Tree Value

Medium | Topics | Companies

Given the `root` of a binary tree, return the leftmost value in the last row of the tree.

Example 1:




```
Input: root = [2,1,3]
Output: 1
```

Example 2:

C++ • Auto

```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
int findBottomLeftValue(struct TreeNode* root) {
    int value=root->val;
    int mdepth=0;
void transverse(struct TreeNode* p,int depth){
    if(!p)
    return;
    if(depth>mdepth){
    mdepth=depth;
    value=p->val;
    }
    transverse(p->left,depth+1);
    transverse(p->right,depth+1);
}
transverse(root,0);
return value;
```

Saved to local

Ln 25, Col 20

Testcase | >_ Test Result

Accepted Runtime: 3 ms

• Case 1 • Case 2

Input

root =

[2,1,3]

Output

1

Expected

1

Testcase | >_ Test Result

Accepted Runtime: 3 ms

- Case 1
- Case 2

Input

root =

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

Output

```
7
```

Expected

```
7
```

Description | Editorial | Solutions | Submissions

450. Delete Node in a BST

Medium | Topics | Companies

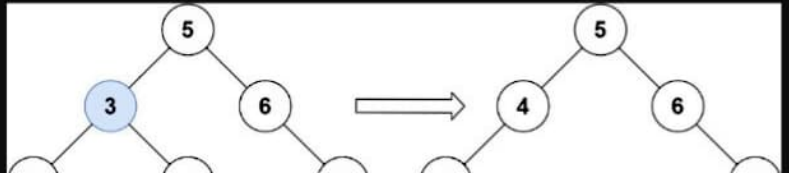
Given a root node reference of a BST and a key, delete the node with the given key in the BST. Return ***the root node reference*** *(possibly updated) of the BST.*

Basically, the deletion can be divided into two stages:

1. Search for a node to remove.

2. If the node is found, delete the node.

Example 1:



Code

C++ | Auto

```
 */
struct TreeNode* deleteNode(struct TreeNode* root, int key) {
if (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 && !root->right)
    return NULL;
    if (!root->left || !root->right)
    return root->left ? root->left : root->right;
struct TreeNode* temp = root->left;
while (temp->right != NULL)
    temp = temp->right;
    root->val = temp->val;
    root->left = deleteNode(root->left, temp->val);
}
}
return root;
}
```

Testcase | Test Result

Accepted Runtime: 6 ms

• Case 1 • Case 2 • Case 3

Input

root =
[5,3,6,2,4,null,7]

key =
3

Output

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

Expected

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

Testcase | Test Result

Accepted Runtime: 6 ms

• Case 1 • Case 2 • Case 3

Input

root =
[5,3,6,2,4,null,7]

key =
0

Output

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

Expected

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

Testcase | >_ Test Result

Accepted Runtime: 6 ms

• Case 1 • Case 2 • Case 3

Input

root =

[]

key =

0

Output

[]

Expected

[]