26/02/2024.

1. BFS:

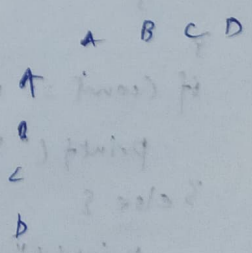
```
#include <stdio.h>
#include <stdbool.h>
#define MAX_NODES 100
int queue[MAX_NODES];
int front = -1, rear = -1;
void enqueue(int node) {
    if (rear == MAX_NODES - 1) {
        printf("Queue is full\n");
    } else {
        if (front == -1) {
            front = 0;
        }
        rear++;
        queue[rear] = node;
    }
}
int dequeue() {
    int node;
    if (front == -1) {
        printf("Queue is empty.\n");
        return -1;
    } else {
        node = queue[front];
        front++;
        if (front > rear) {
            front = rear = -1;
        }
        return node;
    }
}
bool isEmpty() {
    return front == -1;
}
void BFS(int adjMatrix[MAX_NODES][MAX_NODES], int nodes, 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);
        }
    }
}
}
int main() {
    int nodes, adjMatrix[MAX_NODES][MAX_NODES];
    printf("Enter no. of nodes: ");
    scanf("%d", &nodes);
    printf("Enter adjacency matrix:\n");
    for (int i=0; i<nodes; i++) {
        for (int j=0; j<nodes; j++) {
            scanf("%d", &adjMatrix[i][j]);
        }
    }
    int startNode;
    printf("Enter starting node for BFS: ");
    scanf("%d", &startNode);
    printf("BFS Traversal starting from node %d: ", startNode);
    BFS(adjMatrix, nodes, startNode);
    return 0;
}
```

26/12/24

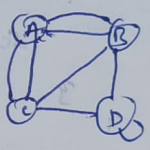
Output:

```
Enter no. of nodes: 7
Enter adjacency matrix:
0 1 1 1 1 0 0
1 0 0 1 0 1 0
1 0 0 0 0 0 1
1 1 0 0 0 1 0
1 0 0 0 0 0 1
0 1 0 1 0 0 0
0 0 1 0 1 0 0
Enter starting node for BFS: 2
BFS Traversal starting from node 2: 2 0 6 1 3 4 5
```



2. 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]);
        }
    }
    printf("DFS Traversal starting from node 1: ");
    dfs(1);
    for (i=1; i<=n; i++) {
        if (s[i]) {
            count++;
        }
    }
    if (count == n) {
        printf("\nGraph is connected\n");
    } else {
        printf("\nGraph is not connected\n");
    }
    return 0;
```



Graph is

Output:

Enter the no. of vertices: 4

Enter the adjacency matrix:

0 1 1 0

1 0 0 1

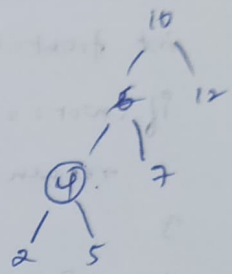
1 0 0 0

0 1 0 1

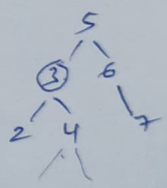
DFS Traversal starting from node 1: 1 2 4 3

Graph is connected.

3. Delete Node in a BST (LeetCode):




```
struct TreeNode* smallest(struct TreeNode* root) {
    struct TreeNode* cur = root;
    while(cur->left != NULL) {
        cur = cur->left;
    }
    return cur;
}
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;
```



NP
26/2/24

```
    struct TreeNode *temp = smallest(root->right);
    root->val = temp->val;
    root->right = deleteNode(root->right, root->val);
  }
  return root;
}
```

4. Bottom left Tree:

```
int findBottomLeftValue(struct TreeNode* root) {
    struct TreeNode* queue[10000];
    int front=0, rear=0, levelSize=0, leftmostvalue=0;
    if(root==NULL){
        return 0;
    }
    queue[rear++]=root;
    while(front<rear){
        levelSize = rear-front;
        for(int i=0; i<levelSize; i++){
            struct TreeNode* curr = queue[front++];
            if(i==0){
                leftMostvalue = curr->val;
            }
            if(curr->left!=NULL){
                queue[rear++]=curr->left;
            }
            if(curr->right!=NULL){
                queue[rear++]=curr->right;
            }
        }
    }
    return leftmostvalue;
}
```

NLP
26/2/24