**Write a program to traverse a graph using BFS method.**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

void BFS(int adjacency\_matrix[MAX][MAX], int visited[MAX], int n, int start) {

int queue[MAX], front = -1, rear = -1;

visited[start] = 1;

queue[++rear] = start;

while (front != rear) {

int current = queue[++front];

printf("%d ", current + 1);

for (int i = 0; i < n; i++) {

if (adjacency\_matrix[current][i] == 1 && !visited[i]) {

visited[i] = 1;

queue[++rear] = i;

}

}

}

}

int main() {

int adjacency\_matrix[MAX][MAX], visited[MAX], n, i, j;

printf("Enter the number of nodes: ");

scanf("%d", &n);

printf("Enter the adjacency matrix:\n");

for (i = 0; i < n; i++) {

visited[i] = 0; // Initialize visited array

for (j = 0; j < n; j++) {

scanf("%d", &adjacency\_matrix[i][j]);

}

}

int start\_node;

printf("Enter the starting node for BFS traversal (1 to %d): ", n);

scanf("%d", &start\_node);

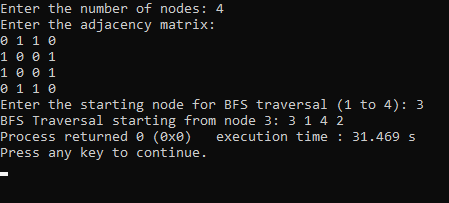
printf("BFS Traversal starting from node %d: ", start\_node);

BFS(adjacency\_matrix, visited, n, start\_node - 1);

return 0;

}

**Output:**

****

**Write a program to check whether given graph is connected or not using DFS method.**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

void DFS(int adjacency\_matrix[MAX][MAX], int visited[MAX], int n, int current) {

visited[current] = 1;

for (int i = 0; i < n; i++) {

if (adjacency\_matrix[current][i] == 1 && !visited[i]) {

DFS(adjacency\_matrix, visited, n, i);

}

}

}

int isConnected(int adjacency\_matrix[MAX][MAX], int visited[MAX], int n) {

for (int i = 0; i < n; i++) {

visited[i] = 0;

}

DFS(adjacency\_matrix, visited, n, 0);

for (int i = 0; i < n; i++) {

if (!visited[i]) {

return 0;

}

}

return 1;

}

int main() {

int adjacency\_matrix[MAX][MAX], visited[MAX], n, i, j;

printf("Enter the number of nodes: ");

scanf("%d", &n);

printf("Enter the adjacency matrix:\n");

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

scanf("%d", &adjacency\_matrix[i][j]);

}

}

if (isConnected(adjacency\_matrix, visited, n)) {

printf("The graph is connected.\n");

} else {

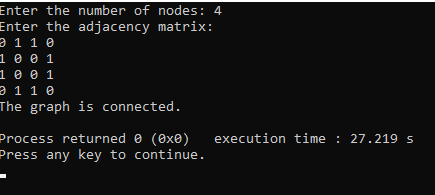
printf("The graph is not connected.\n");

}

return 0;

}

**Output:**



**Hacker Rank on Tree**

#include <assert.h>

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct Node {

    int data;

    struct Node\* left;

    struct Node\* right;

} Node;

Node\* createNode(int data) {

    Node\* newNode = (Node\*)malloc(sizeof(Node));

    newNode->data = data;

    newNode->left = NULL;

    newNode->right = NULL;

    return newNode;

}

void inOrderTraversal(Node\* root, int\* result, int\* index) {

    if (root == NULL) return;

    inOrderTraversal(root->left, result, index);

    result[(\*index)++] = root->data;

    inOrderTraversal(root->right, result, index);

}

void swapAtLevel(Node\* root, int k, int level) {

    if (root == NULL) return;

    if (level % k == 0) {

        Node\* temp = root->left;

        root->left = root->right;

        root->right = temp;

    }

    swapAtLevel(root->left, k, level + 1);

    swapAtLevel(root->right, k, level + 1);

}

int\*\* swapNodes(int indexes\_rows, int indexes\_columns, int\*\* indexes, int queries\_count, int\* queries, int\* result\_rows, int\* result\_columns) {

    Node\*\* nodes = (Node\*\*)malloc((indexes\_rows + 1) \* sizeof(Node\*));

    for (int i = 1; i <= indexes\_rows; i++) {

        nodes[i] = createNode(i);

    }

    for (int i = 0; i < indexes\_rows; i++) {

        int leftIndex = indexes[i][0];

        int rightIndex = indexes[i][1];

        if (leftIndex != -1) nodes[i + 1]->left = nodes[leftIndex];

        if (rightIndex != -1) nodes[i + 1]->right = nodes[rightIndex];

    }

    int\*\* result = (int\*\*)malloc(queries\_count \* sizeof(int\*));

    \*result\_rows = queries\_count;

    \*result\_columns = indexes\_rows;

    for (int i = 0; i < queries\_count; i++) {

        swapAtLevel(nodes[1], queries[i], 1);

        int\* traversalResult = (int\*)malloc(indexes\_rows \* sizeof(int));

        int index = 0;

        inOrderTraversal(nodes[1], traversalResult, &index);

        result[i] = traversalResult;

    }

    free(nodes);

    return result;

}

int main() {

    int n;

    scanf("%d", &n);

    int\*\* indexes = malloc(n \* sizeof(int\*));

    for (int i = 0; i < n; i++) {

        indexes[i] = malloc(2 \* sizeof(int));

        scanf("%d %d", &indexes[i][0], &indexes[i][1]);

    }

    int queries\_count;

    scanf("%d", &queries\_count);

    int\* queries = malloc(queries\_count \* sizeof(int));

    for (int i = 0; i < queries\_count; i++) {

        scanf("%d", &queries[i]);

    }

    int result\_rows;

    int result\_columns;

    int\*\* result = swapNodes(n, 2, indexes, queries\_count, queries, &result\_rows, &result\_columns);

    for (int i = 0; i < result\_rows; i++) {

        for (int j = 0; j < result\_columns; j++) {

            printf("%d ", result[i][j]);

        }

        printf("\n");

        free(result[i]);

    }

    free(result);

    for (int i = 0; i < n; i++) {

        free(indexes[i]);

    }

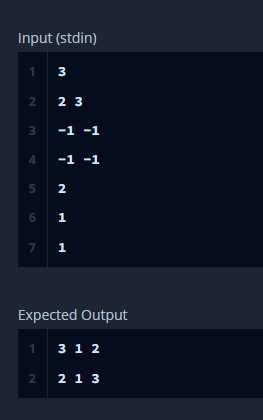
    free(indexes);

    free(queries);

    return 0;

}

**Output:**

****