#include <iostream>

using namespace std;

const int MAX = 10;

// ------------------ Stack Class ------------------

class Stack {

int arr[MAX];

int top;

public:

Stack() { top = -1; }

void push(int val) {

if (top >= MAX - 1)

cout << "Stack Overflow\n";

else

arr[++top] = val;

}

int pop() {

if (top < 0) {

cout << "Stack Underflow\n";

return -1;

} else

return arr[top--];

}

bool isEmpty() { return top == -1; }

};

// ------------------ Queue Class ------------------

class Queue {

int arr[MAX];

int front, rear;

public:

Queue() { front = rear = -1; }

void enqueue(int val) {

if (rear == MAX - 1)

cout << "Queue Overflow\n";

else {

if (front == -1) front = 0;

arr[++rear] = val;

}

}

int dequeue() {

if (front == -1 || front > rear) {

cout << "Queue Underflow\n";

return -1;

} else

return arr[front++];

}

bool isEmpty() { return front == -1 || front > rear; }

};

// ------------------ Graph Class ------------------

class Graph {

int adj[MAX][MAX];

int vertices;

public:

void createInput(int nodes) {

vertices = nodes;

// Initialize all to 0

for (int i = 0; i < vertices; i++)

for (int j = 0; j < vertices; j++)

adj[i][j] = 0;

cout << "Enter upper triangular adjacency matrix (excluding diagonal):\n";

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

for (int j = i + 1; j < vertices; j++) {

cout << "Edge between " << i << " and " << j << " (0 or 1): ";

cin >> adj[i][j];

adj[j][i] = adj[i][j]; // mirror for undirected graph

}

}

}

void DFS(int start) {

bool visited[MAX] = { false };

Stack s;

s.push(start);

cout << "DFS Traversal: ";

while (!s.isEmpty()) {

int node = s.pop();

if (!visited[node]) {

cout << node << " ";

visited[node] = true;

// Push adjacent unvisited nodes (reverse for correct order)

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

if (adj[node][i] && !visited[i])

s.push(i);

}

}

}

cout << endl;

}

void BFS(int start) {

bool visited[MAX] = { false };

Queue q;

q.enqueue(start);

visited[start] = true;

cout << "BFS Traversal: ";

while (!q.isEmpty()) {

int node = q.dequeue();

cout << node << " ";

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

if (adj[node][i] && !visited[i]) {

q.enqueue(i);

visited[i] = true;

}

}

}

cout << endl;

}

};

// ------------------ Main Function ------------------

int main() {

Graph g;

int v;

cout << "Enter number of vertices: ";

cin >> v;

g.createInput(v);

int start;

cout << "Enter starting node for DFS and BFS: ";

cin >> start;

g.DFS(start);

g.BFS(start);

return 0;

}