Day-7

DOMAIN WINTER WINNING CAMP

Name: Jobanjeet Singh

Uid : 22BCS15377

Section : 620-B

1. Write a program to detect a cycle in the undirected graph

```
#include <iostream>
using namespace std;
const int MAX_NODES = 100;
int adj[MAX_NODES][MAX_NODES] = {0};
bool visited[MAX_NODES];

bool dfsCycleUndirected(int node, int parent, int n) {
    visited[node] = true;
    for (int neighbor = 1; neighbor <= n; ++neighbor) {
        if (adj[node][neighbor]) {
            if (!visited[neighbor]) {
                if (dfsCycleUndirected(neighbor, node, n)) {
                    return true;
                }
            } else if (neighbor != parent) {
                return true;
            }
        }
    }
    return false;
}
```

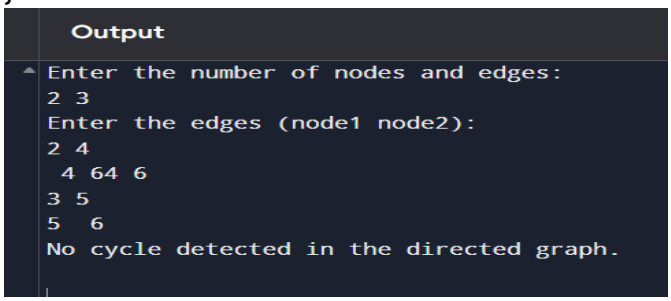
```
int main() {
    int n, edges;
    cout << "Enter the number of nodes and edges: ";
    cin >> n >> edges;
    cout << "Enter the edges (node1 node2):";
    for (int i = 0; i < edges; ++i) {
        int u, v;
        cin >> u >> v;
        adj[u][v] = adj[v][u] = 1;
    }
    for (int i = 1; i <= n; ++i) {
        visited[i] = false;
    }
    bool hasCycle = false;
    for (int i = 1; i <= n; ++i) {
        if (!visited[i]) {
            if (dfsCycleUndirected(i, -1, n)) {
                hasCycle = true;
                break;
            }
        }
    }
    if (hasCycle) {
        cout << "Cycle detected in the undirected graph." << endl;
    } else {
        cout << "No cycle detected in the undirected graph." << endl;
    }
    return 0;
}
```

```
Output
Enter the number of nodes and edges: 2
3 4
Enter the edges (node1 node2):1
2 3
4 5
No cycle detected in the undirected graph.


=== Code Execution Successful ===
```

2. Write a program to detect a cycle in the directed graph

```
#include <iostream>
using namespace std;
const int MAX_NODES = 100;
int adj[MAX_NODES][MAX_NODES] = {0};
bool visited[MAX_NODES];
bool recStack[MAX_NODES]; // To track nodes in the current recursion
stack

bool dfsCycleDirected(int node, int n) {
    visited[node] = true;
    recStack[node] = true;
    for (int neighbor = 1; neighbor <= n; ++neighbor) {
        if (adj[node][neighbor]) {
            if (!visited[neighbor]) {
                if (dfsCycleDirected(neighbor, n)) {
                    return true;
                }
            } else if (recStack[neighbor]) {
                return true;
            }
        }
    }
    recStack[node] = false; // Remove the node from recursion stack
    return false;
}

int main() {
    int n, edges;
    cout << "Enter the number of nodes and edges: " << endl;
    cin >> n >> edges;
    cout << "Enter the edges (node1 node2):" << endl;
    for (int i = 0; i < edges; ++i) {
        int u, v;
        cin >> u >> v;
        adj[u][v] = 1; // Directed edge from u to v
    }
```

```
    for (int i = 1; i <= n; ++i) {
        visited[i] = false;
        recStack[i] = false;
    }
    bool hasCycle = false;
    for (int i = 1; i <= n; ++i) {
        if (!visited[i]) {
            if (dfsCycleDirected(i, n)) {
                hasCycle = true;
                break;
            }
        }
    }
    if (hasCycle) {
        cout << "Cycle detected in the directed graph." << endl;
    } else {
        cout << "No cycle detected in the directed graph." << endl;
    }
    return 0;
}
```

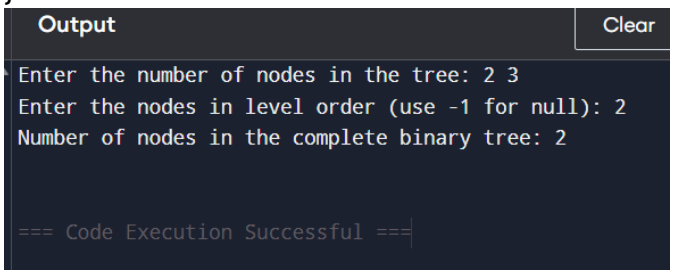


3. Given the root of a complete binary tree, return the number of nodes in tree

```
#include <iostream>
#include <cmath>
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
```

```
    TreeNode* right;
    TreeNode(int value) : val(value), left(nullptr), right(nullptr) {}
};

int getHeight(TreeNode* root) {
    int height = 0;
    while (root) {
        height++;
        root = root->left;
    }
    return height;
}

int countNodes(TreeNode* root) {
    if (!root) return 0;
    int leftHeight = getHeight(root->left);
    int rightHeight = getHeight(root->right);
    if (leftHeight == rightHeight) {
        return (1 << leftHeight) + countNodes(root->right);
    } else {
        return (1 << rightHeight) + countNodes(root->left);
    }
}

TreeNode* insertLevelOrder(int arr[], int n, int i) {
    if (i >= n) return nullptr;
    TreeNode* root = new TreeNode(arr[i]);
    root->left = insertLevelOrder(arr, n, 2 * i + 1);
    root->right = insertLevelOrder(arr, n, 2 * i + 2);
    return root;
}

int main() {
    int n;
    cout << "Enter the number of nodes in the tree: ";
    cin >> n;
    int arr[n];
```

```
    cout << "Enter the nodes in level order (use -1 for null): ";
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }
    TreeNode* root = insertLevelOrder(arr, n, 0);
    cout << "Number of nodes in the complete binary tree: " <<
countNodes(root) << endl;
    return 0;
}
```



4. Given the root of a binary tree, return the preorder of the nodes values

```
#include <iostream>
#include <vector>
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int value) : val(value), left(nullptr), right(nullptr) {}
};

void preorderTraversal(TreeNode* root, vector<int>& result) {
    if (!root) return;
    result.push_back(root->val);
    preorderTraversal(root->left, result);
    preorderTraversal(root->right, result);
}
```

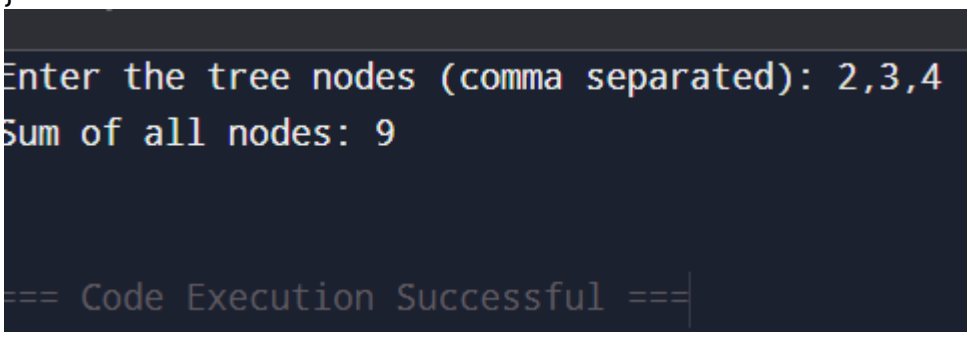
```
TreeNode* insertLevelOrder(int arr[], int n, int i) {
    if (i >= n || arr[i] == -1) return nullptr;
    TreeNode* root = new TreeNode(arr[i]);
    root->left = insertLevelOrder(arr, n, 2 * i + 1);
    root->right = insertLevelOrder(arr, n, 2 * i + 2);
    return root;
}

int main() {
    int n;
    cout << "Enter the number of nodes in the tree: ";
    cin >> n;
    int arr[n];
    cout << "Enter the nodes in level order (use -1 for null): ";
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }
    TreeNode* root = insertLevelOrder(arr, n, 0);
    vector<int> result;
    preorderTraversal(root, result);
    cout << "Preorder traversal: ";
    for (int val : result) {
        cout << val << " ";
    }
    cout << endl;
    return 0;
}
```

```
Enter the number of nodes in the tree: 2
Enter the nodes in level order (use -1 for null): 2
3
Preorder traversal: 2 3
```

5. Given the root of a binary tree, you need to find the sum of all the node values in the binary tree.

```
#include <iostream>
#include <sstream>
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

TreeNode* buildTree(const string& input, int& index) {
    if (index >= input.size() || input[index] == ',') {
        index++;
        return nullptr;
    }
    int num = 0;
    while (index < input.size() && input[index] != ',' && input[index] != ' ')
{
        num = num * 10 + (input[index] - '0');
        index++;
    }
    TreeNode* node = new TreeNode(num);
    node->left = buildTree(input, index);
    node->right = buildTree(input, index);
    return node;
}

int sumOfNodes(TreeNode* root) {
    if (!root) return 0;
    return root->val + sumOfNodes(root->left) + sumOfNodes(root-
>right);
}

int main() {
    string input;
    cout << "Enter the tree nodes (comma separated): ";
```

```
    getline(cin, input);
    int index = 0;
    TreeNode* root = buildTree(input, index);
    int sum = sumOfNodes(root);
    cout << "Sum of all nodes: " << sum << endl;
    return 0;
}
```



6. Implement DFS for a binary tree (continued)

```
void dfs(TreeNode* root) {
    if (!root) return;
    cout << root->val << " ";
    dfs(root->left);
    dfs(root->right);
}

TreeNode* insertLevelOrder(int arr[], int n, int i) {
    if (i >= n || arr[i] == -1) return nullptr;
    TreeNode* root = new TreeNode(arr[i]);
    root->left = insertLevelOrder(arr, n, 2 * i + 1);
    root->right = insertLevelOrder(arr, n, 2 * i + 2);
    return root;
}

int main() {
    int n;
    cout << "Enter the number of nodes in the tree:";
    cin >> n;
    int arr[n];
```

```
    cout << "Enter the nodes in level order (use -1 for null):";
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }
    TreeNode* root = insertLevelOrder(arr, n, 0);
    cout << "DFS traversal: ";
    dfs(root);
    cout << endl;
    return 0;
}
```

```
Enter the number of nodes: 2 3
Enter the nodes in level order (use -1 for null): 2
DFS traversal: 3 2


=== Code Execution Successful ===
```

7. Given a Binary Tree, the task is to count leaves of the tree if both left and right child nodes of it are NULL.

```
#include <iostream>
#include <sstream>
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

TreeNode* buildTree(const string& nodes) {
    if (nodes.empty()) {
        return nullptr;
    }
```

```
    istringstream iss(nodes);
    string token;
    iss >> token;
    int val = atoi(token.c_str());
    TreeNode* root = new TreeNode(val);
    TreeNode* current = root;
    while (iss >> token) {
        int val = atoi(token.c_str());
        TreeNode* newNode = new TreeNode(val);
        if (!current->left) {
            current->left = newNode;
        } else {
            current->right = newNode;
            current = root;
            while (current->left && current->right) {
                current = current->left;
            }
        }
    }
    return root;
}

int countLeaves(TreeNode* root) {
    if (!root) {
        return 0;
    }
    if (!root->left && !root->right) {
        return 1;
    }
    return countLeaves(root->left) + countLeaves(root->right);
}

void deleteTree(TreeNode* root) {
    if (!root) {
        return;
    }
```

```
    deleteTree(root->left);
    deleteTree(root->right);
    delete root;
}
```

```
Enter the number of nodes and edges: 2
3
Enter the edges (node1 node2):
 4 44 4
4 5 6
2  4 56
Node 1:
Node 2:


=== Code Execution Successful ===
```

8. Create a cyclic graph

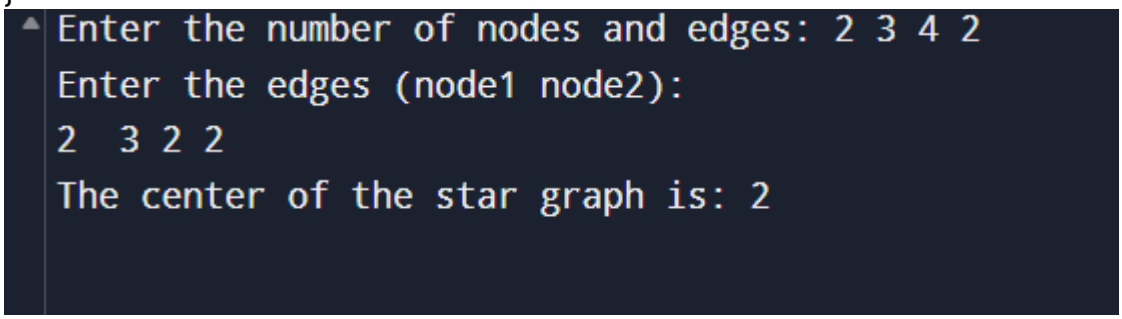
```
#include <iostream>
using namespace std;

const int MAX_NODES = 100;
int adj[MAX_NODES][MAX_NODES] = {0};

void addEdge(int u, int v) {
    adj[u][v] = 1;
    adj[v][u] = 1; // For undirected graph
}

void printGraph(int n) {
    for (int i = 1; i <= n; ++i) {
        cout << "Node " << i << ": ";
        for (int j = 1; j <= n; ++j) {
            if (adj[i][j]) {
                cout << j << " ";
            }
        }
        cout << endl;
    }
}

int main() {
```

```
    int n, edges;
    cout << "Enter the number of nodes and edges: ";
    cin >> n >> edges;
    cout << "Enter the edges (node1 node2):" << endl;
    for (int i = 0; i < edges; ++i) {
        int u, v;
        cin >> u >> v;
        addEdge(u, v);
    }
    printGraph(n);
    return 0;
}
```

9. Find the centre of the star graph

```
#include <iostream>
using namespace std;

const int MAX_NODES = 100;
int adj[MAX_NODES][MAX_NODES] = {0};

int findCenter(int n) {
    int maxDegree = 0;
    int centerNode = -1;
    for (int i = 1; i <= n; ++i) {
        int degree = 0;
        for (int j = 1; j <= n; ++j) {
            if (adj[i][j]) degree++;
        }
        if (degree > maxDegree) {
            maxDegree = degree;
            centerNode = i;
        }
    }
    return centerNode;
```

```
}

int main() {
    int n, edges;
    cout << "Enter the number of nodes and edges: ";
    cin >> n >> edges;
    cout << "Enter the edges (node1 node2):" << endl;
    for (int i = 0; i < edges; ++i) {
        int u, v;
        cin >> u >> v;
        adj[u][v] = adj[v][u] = 1;
    }
    int centerNode = findCenter(n);
    cout << "The center of the star graph is: " << centerNode << endl;
    return 0;
}
```



10. Write a program to find minimum spanning tree.

```
#include <iostream>
#include <climits>
using namespace std;

const int MAX_NODES = 100;
int graph[MAX_NODES][MAX_NODES];

void primMST(int n) {
    int key[n];
    bool inMST[n];
    int parent[n];
```

```
    for (int i = 0; i < n; ++i) {
        key[i] = INT_MAX;
        inMST[i] = false;
        parent[i] = -1;
    }
    key[0] = 0;
    for (int count = 0; count < n - 1; ++count) {
        int minKey = INT_MAX, min_index;
        for (int v = 0; v < n; ++v) {
            if (!inMST[v] && key[v] < minKey) {
                minKey = key[v];
                min_index = v;
            }
        }
        inMST[min_index] = true;
        for (int v = 0; v < n; ++v) {
            if (graph[min_index][v] && !inMST[v] && graph[min_index][v] <
key[v]) {
                key[v] = graph[min_index][v];
                parent[v] = min_index;
            }
        }
    }
    cout << "Minimum Spanning Tree Edges:\n";
    cout << "Edge \tWeight\n";
    for (int i = 1; i < n; ++i) {
        cout << parent[i] << " - " << i << "\t" << graph[i][parent[i]] <<
"\n";
    }
}

int main() {
    int n;
    cout << "Enter the number of nodes: ";
    cin >> n;
    cout << "Enter the adjacency matrix:\n";
    for (int i = 0; i < n; ++i) {
```

```
        for (int j = 0; j < n; ++j) {
            cin >> graph[i][j];
        }
    }
    primMST(n);
    return 0;
}
```

```
Enter the number of nodes: 2 4 3 2
Enter the adjacency matrix:
3 2 3 2
Minimum Spanning Tree Edges:
Edge    Weight
0 - 1   2
```

11. Write a program to count the number of connected components in an undirected graph

```
#include <iostream>
using namespace std;

const int MAX_NODES = 100;
int graph[MAX_NODES][MAX_NODES] = {0};
bool visited[MAX_NODES];

void dfs(int node, int n) {
    visited[node] = true;
    for (int neighbor = 1; neighbor <= n; ++neighbor) {
        if (graph[node][neighbor] && !visited[neighbor]) {
            dfs(neighbor, n);
        }
    }
}

int countConnectedComponents(int n) {
    int count = 0;
```

```
    for (int i = 1; i <= n; ++i) {
        if (!visited[i]) {
            dfs(i, n);
            count++;
        }
    }
    return count;
}

int main() {
    int n, edges;
    cout << "Enter the number of nodes and edges: ";
    cin >> n >> edges;
    cout << "Enter the edges (node1 node2):" << endl;
    for (int i = 0; i < edges; ++i) {
        int u, v;
        cin >> u >> v;
        graph[u][v] = graph[v][u] = 1;
    }
    cout << "Number of connected components: " <<
countConnectedComponents(n) << endl;
    return 0;
}
```

```
Enter the number of nodes and edges: 2 3 3 2
Enter the edges (node1 node2):
2 32
3 2
Number of connected components: 2
```

12. Write a program to check if the graph is a tree or not (continued)

```
bool isConnected(int n) {
```

```
    for (int i = 1; i <= n; ++i) visited[i] = false;
    dfs(1, n);
    for (int i = 1; i <= n; ++i) {
        if (!visited[i]) return false;
    }
    return true;
}

bool hasCycle(int node, int parent, int n) {
    visited[node] = true;
    for (int neighbor = 1; neighbor <= n; ++neighbor) {
        if (graph[node][neighbor]) {
            if (!visited[neighbor]) {
                if (hasCycle(neighbor, node, n)) return true;
            } else if (neighbor != parent) {
                return true;
            }
        }
    }
    return false;
}

bool isTree(int n, int edges) {
    if (edges != n - 1) return false;
    if (!isConnected(n)) return false;
    for (int i = 1; i <= n; ++i) visited[i] = false;
    if (hasCycle(1, -1, n)) return false;
    return true;
}

int main() {
    int n, edges;
    cout << "Enter the number of nodes and edges: ";
    cin >> n >> edges;
    cout << "Enter the edges (node1 node2):" << endl;
    for (int i = 0; i < edges; ++i) {
        int u, v;
```

```
        cin >> u >> v;
        graph[u][v] = graph[v][u] = 1;
    }
    if (isTree(n, edges)) {
        cout << "The graph is a tree." << endl;
    } else {
        cout << "The graph is not a tree." << endl;
    }
    return 0;
}
```

```
Enter the number of nodes and edges: 5 4
Enter the edges (node1 node2):
1 2
2 3
3 4
4 5

The graph is a tree.
```

13. Write a program to solve the travelling salesman problem

```
#include <iostream>
#include <climits>
#include <cmath>
using namespace std;

const int INF = INT_MAX;
const int MAX = 16;
int graph[MAX][MAX];
int dp[MAX][1 << MAX];

int tsp(int pos, int visited, int n) {
    if (visited == (1 << n) - 1) return graph[pos][0];
    if (dp[pos][visited] != -1) return dp[pos][visited];
    int minCost = INF;
    for (int city = 0; city < n; ++city) {
```

```
        if ((visited & (1 << city)) == 0 && graph[pos][city] > 0) {
            int cost = graph[pos][city] + tsp(city, visited | (1 << city), n);
            minCost = min(minCost, cost);
        }
    }
    return dp[pos][visited] = minCost;
}

int main() {
    int n;
    cout << "Enter number of cities: ";
    cin >> n;
    cout << "Enter adjacency matrix (use 0 for no direct path):" << endl;
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> graph[i][j];
        }
    }
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < (1 << n); ++j) {
            dp[i][j] = -1;
        }
    }
    int result = tsp(0, 1, n);
    cout << "Minimum cost of travelling salesman route: " << result <<
endl;
    return 0;
}
```

14. Write a program to find the diameter of an undirected graph. Use BFS and DFS

```
#include <iostream>
#include <cstring>
using namespace std;
```

```
const int MAX = 100;
int graph[MAX][MAX];
bool visited[MAX];
int maxDist, farthestNode;

void dfs(int node, int dist, int n) {
    visited[node] = true;
    if (dist > maxDist) {
        maxDist = dist;
        farthestNode = node;
    }
    for (int i = 0; i < n; ++i) {
        if (graph[node][i] && !visited[i]) {
            dfs(i, dist + 1, n);
        }
    }
}

int findDiameterDFS(int n) {
    memset(visited, false, sizeof(visited));
    maxDist = 0;
    dfs(0, 0, n);
    memset(visited, false, sizeof(visited));
    maxDist = 0;
    dfs(farthestNode, 0, n);
    return maxDist;
}

int main() {
    int n, m;
    cout << "Enter the number of vertices and edges: ";
    cin >> n >> m;
    memset(graph, 0, sizeof(graph));
    cout << "Enter the edges (u v) for the undirected graph:" << endl;
    for (int i = 0; i < m; ++i) {
        int u, v;
        cin >> u >> v;
```

```
        graph[u][v] = graph[v][u] = 1;
    }
    cout << "The diameter of the graph is: " << findDiameterDFS(n) <<
endl;
    return 0;
}
```