**CHANDIGARH UNIVERSITY**

**UNIVERSITY INSTITUTE OF ENGINEERING**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**



|  |  |
| --- | --- |
| **Submitted By:** Sahil Kaundal  **Submitted To:** Neha Dutta | |
| **Subject Name** | Design and Analysis of Algorithm Lab |
| **Subject Code** | 20CSP-312 |
| **Branch** | Computer Science Engineering |
| **Semester** | 5th |

**Experiment 8**

**Student Name:** Sahil Kaundal **UID:** 21BCS8197

**Branch:** BE CSE (Lateral Entry) **Section/Group:** 616/A

**Semester:** 5th **Date of Performance:** 10/11/2022

**Subject Name:** DAA Lab **Subject Code:** 21-CSP-312

# Aim/Overview of the practical:

Code and analyze to do a depth-first search (DFS) on an undirected graph. Implementing an application of DFS such as

1. To find the topological sort of a directed acyclic graph.
2. To find a path from source to goal in a maze.

# Task to be done/ Which logistics used:

1. To find the topological sort of a directed acyclic graph.
2. To find a path from source to goal in a maze.

# Requirements:

Laptop or PC.

Operation system (Mac, Windows, Linux, or any)

Vs-Code with MinGw or any C++ Compiler

1. **Algorithm/Flowchart (For programming-based labs):**

**Topological Sort:**

* Create a stack to store the nodes.
* Initialize visited array of size N to keep the record of visited nodes.
* Run a loop from 0 till N
* if the node is not marked True in visited array
* Call the recursive function for topological sort and perform the following steps.
* Mark the current node as True in the visited array.
* Run a loop on all the nodes which has a directed edge to the current node
* if the node is not marked True in the visited array:
* Recursively call the topological sort function on the node
* Push the current node in the stack.
* Print all the elements in the stack.

# Path from source to goal:

* Mark node as visited.
* Add node to the path vector as it can be a possible path.
* If node == goal node then save this path in result and return.
* Then call dfs function on adjacent node if not visited.
* Print result vector

# Steps for experiment/practical/Code:

**Topological sort:**

#include <bits/stdc++.h> using namespace std;

void dfs(int node, vector<bool> &visited, stack<int> &s, unordered\_map<int, list<int>> &adj)

{

visited[node] = 1;

for (auto neighbour : adj[node])

{

if (!visited[neighbour]) dfs(neighbour, visited, s, adj);

}

s.push(node);

}

void topologicalSort(vector<vector<int>> &edges, int n, int e)

{

unordered\_map<int, list<int>> adj; for (int i = 0; i < e; i++)

{

int u = edges[i][0]; int v = edges[i][1];

adj[u].push\_back(v);

}

vector<bool> visited(n + 1, false); stack<int> s;

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

{

if (!visited[i])

dfs(i, visited, s, adj);

}

cout << "Topological Sort: "; while (!s.empty())

{

cout << s.top() << " "; s.pop();

}

cout << endl;

}

int main()

{

2}};

}

int n = 6, e = 6;

vector<vector<int>> edges = {{5, 0}, {4, 0}, {4, 1}, {3, 1}, {2, 3}, {5,

topologicalSort(edges, n, edges.size()); return 0;

# Path from source to goal:

#include <bits/stdc++.h> using namespace std;

void dfs(int node, vector<bool> &visited, vector<int> path, vector<int> &result, unordered\_map<int,

list<int>> &adj, int src, int goal)

{

visited[node] = 1; path.push\_back(node); if (node == goal)

{

result = path; return;

}

for (auto neighbour : adj[node])

{

if (!visited[neighbour])

dfs(neighbour, visited, path, result, adj, neighbour, goal);

}

}

void pathFinder(vector<vector<int>> &edges, int n, int e, int src, int goal)

{

unordered\_map<int, list<int>> adj; for (int i = 0; i < e; i++)

{

int u = edges[i][0]; int v = edges[i][1]; adj[u].push\_back(v); adj[v].push\_back(u);

}

vector<bool> visited(n + 1, false); vector<int> result;

vector<int> path;

dfs(src, visited, path, result, adj, src, goal);

cout << "Path from " << src << " (source) node to " << goal << " (goal) node: ";

for (auto it : result)

{

cout << it << " ";

}

cout << endl;

}

int main()

{

int n, e;

int src, goal;

// Undirected Graph cout << "No of nodes: "; cin >> n;

cout << "No of edges: "; cin >> e; vector<vector<int>> edges; for (int i = 0; i < e; i++)

{

int u, v; cin >> u; cin >> v;

edges.push\_back({u, v});

}

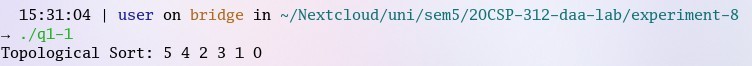
cout << "Enter source node: "; cin >> src;

cout << "Enter goal node: "; cin >> goal;

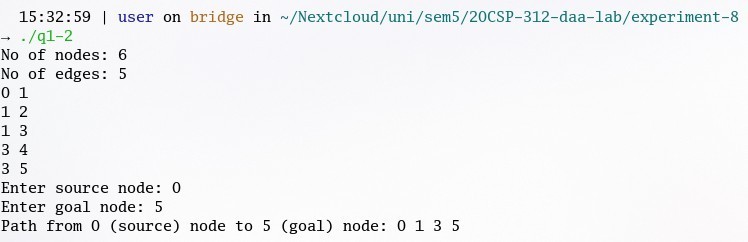
pathFinder(edges, n, edges.size(), src, goal); return 0;}

# Result/Output/Writing Summary:

Topological sort:



Path from source to goal:



**Learning outcomes (What I have learnt):**

Use of dynamic programming Solve knapsack problem

**Evaluation Grid (To be created as per the SOP and Assessment guidelines by the faculty):**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr. No. | Parameters | Marks Obtained | Maximum Marks |
| 1. |  |  |  |
| 2. |  |  |  |
| 3. |  |  |  |