Experiment Title- 3.1

Student Name: YANA SRIVASTAVA
UID: 20BCS2279
Branch: CSE
Section/Group: 20BCS-WM-906/B
Semester: 5th
Subject Code: 21CSP-312
Subject Name: DAA LAB

Aim:

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

(i) To find the topological sort of a directed acyclic graph.

(ii) To find a path from source to goal in a maze.

Algorithm:

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

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()
{
	int n = 6, e = 6;
	vector<vector<int>> edges = {{5, 0}, {4, 0}, {4, 1}, {3, 1}, {2, 3}, {5, 2}};
	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;
}
```

Output:

Topological sort:

```
  15:31:04 | user on bridge in ~/Nextcloud/uni/sem5/20CSP-312-daa-lab/experiment-8
→ ./q1-1
Topological Sort: 5 4 2 3 1 0
```

Path from source to goal:

```
  15:32:59 | user on bridge in ~/Nextcloud/uni/sem5/20CSP-312-daa-lab/experiment-8
→ ./q1-2
No of nodes: 6
No of edges: 5
0 1
1 2
1 3
3 4
3 5
Enter source node: 0
Enter goal node: 5
Path from 0 (source) node to 5 (goal) node: 0 1 3 5
```

Learning Outcomes:

- Use of dynamic programming.
- Solve knapsack problem.