Worksheet Experiment:- 3.1

Student Name: Pranjal Kumar **UID:** 20BCS3504

Branch: CSE **Section/Group:** 607 /B

Semester: 5th **Date of Performance:** 03/11/2022

Subject Name: Design & Analysis Algorithm **Subject Code:** 20CSP-312

1.AIM

Code and analyze to do a depth-first search (DFS) on an undirected graph. Implementing an application of DFS such as: To find the topological sort of a directed acyclic graph. And To find a path from source to goal in a maze.

2.TASK TO BE DONE

To find the topological sort of a directed acyclic graph.

To find a path from source to goal in a maze.

3.ALGORITHM/FLOWCHART

Topological Sort:

1. Create a stack to store the nodes.
2. Initialize visited array of size N to keep the record of visited nodes.
3. Run a loop from 0 till N.
4. if the node is not marked True in visited array.
5. Call the recursive function for topological sort and perform the following steps.
6. Mark the current node as True in the visited array.
7. Run a loop on all the nodes which has a directed edge to the current node
8. if the node is not marked True in the visited array:
9. Recursively call the topological sort function on the node
10. Push the current node in the stack.

11.Print all the elements in the stack.

Path from source to goal:

1. create M x N matrix.

2. Check if it is possible to go to (x, y) from current position.
3. The function returns false if the cell has value 0 or already visited.
4. if not a valid position, return false.
5. Find Shortest Possible Route in a Matrix mat from source cell (0, 0) to destination cell (x, y).
6. min_dist is passed by reference and stores length of longest path from source to destination found so far dist maintains length of path from source cell to current cell (i, j).
7. if destination is found, update min_dist.
8. set (i, j) cell as visited.
9. go to bottom cell
10. go to right cell
11. go to top cell
12. go to left cell
13. Backtrack - Remove (i, j) from visited matrix
14. Find Shortest Path in Maze

4.STEPS FOR EXPIREMENT/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(){ 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;
```

}

main.cpp

```cpp
#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 = {{6, 0}, {4, 1}, {8, 2}, {3, 5}, {5, 4}, {7, 6}};
    topologicalSort(edges, n, edges.size());
    return 0;
}
```

Path from source to goal:

#include <iostream>

#include <climits>
#include <cstring>
using namespace std;

#define M 10 #define N 10 bool isSafe(int mat[M][N],
int visited[M][N], int x, int y)

{

if (mat[x][y] == 0 || visited[x][y])
return false; return true;

```
}

bool isValid(int x, int y)
{
if (x < M && y < N && x >= 0 && y >= 0)
return true; return false;
}

void findShortestPath(int mat[M][N], int visited[M][N], int i, int j, int
x, int y, int& min_dist, int dist)
{

if (i == x && j == y)
{
min_dist = min(dist, min_dist); return;
}

visited[i][j] = 1;

if (isValid(i + 1, j) && isSafe(mat, visited, i + 1, j))
findShortestPath(mat, visited, i + 1, j, x, y, min_dist, dist + 1);

if (isValid(i, j + 1) && isSafe(mat, visited, i, j + 1))
findShortestPath(mat, visited, i, j + 1, x, y, min_dist, dist + 1);

if (isValid(i - 1, j) && isSafe(mat, visited, i - 1, j))
findShortestPath(mat, visited, i - 1, j, x, y, min_dist, dist + 1);
```

```
if (isValid(i, j - 1) && isSafe(mat, visited, i, j - 1))
findShortestPath(mat, visited, i, j - 1, x, y, min_dist, dist + 1);

visited[i][j] = 0;
}

int main()
{
int mat[M][N] =
{
{ 1, 1, 1, 1, 1, 0, 0, 1, 1, 1 },
{ 0, 1, 1, 1, 1, 1, 0, 1, 0, 1 },
{ 0, 0, 1, 0, 1, 1, 1, 0, 0, 1 },
{ 1, 0, 1, 1, 1, 0, 1, 1, 0, 1 },
{ 0, 0, 0, 1, 0, 0, 0, 1, 0, 1 },
{ 1, 0, 1, 1, 1, 0, 0, 1, 1, 0 },
{ 0, 0, 0, 0, 1, 0, 0, 1, 0, 1 },
{ 0, 1, 1, 1, 1, 1, 1, 1, 0, 0 },
{ 1, 1, 1, 1, 1, 0, 0, 1, 1, 1 },
{ 0, 0, 1, 0, 0, 1, 1, 0, 0, 1 }
};

int visited[M][N];

memset(visited, 0, sizeof visited); int min_dist =
INT_MAX; findShortestPath(mat, visited, 0, 0, 7, 5,
```

min_dist, 0); if (min_dist != INT_MAX) cout << "The
shortest path from source to destination "

"has length " << min_dist;

else cout << "Destination can't be reached from given source";
return 0;

}

```
#include <iostream>
#include <climits>
#include <cstring>
using namespace std;
#define M 10
#define N 10
bool isSafe(int mat[M][N], int visited[M][N], int x, int y)
{
if (mat[x][y] == 0 || visited[x][y])
return false;
return true;
}

bool isValid(int x, int y)
{
if (x < M && y < N && x >= 0 && y >= 0)
return true;
return false;
}

void findShortestPath(int mat[M][N], int visited[M][N], int i, int j,
int x, int y, int& min_dist, int dist)
{

if (i == x && j == y)
{
min_dist = min(dist, min_dist);
return;
}

visited[i][j] = 1;

if (isValid(i + 1, j) && isSafe(mat, visited, i + 1, j))
```

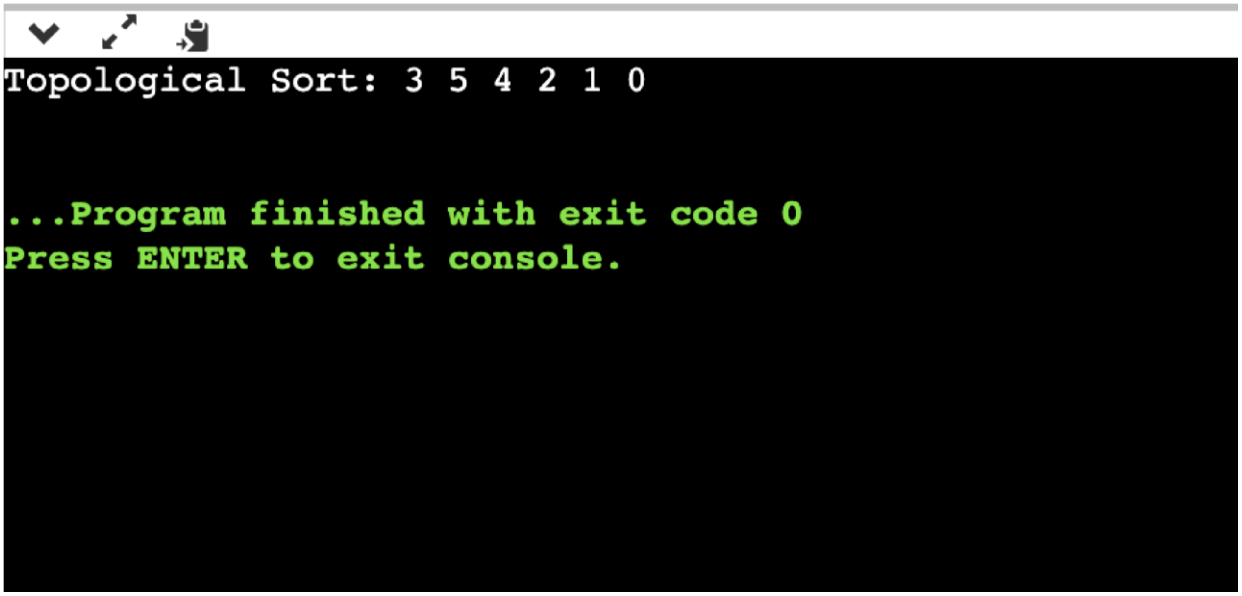
```
findShortestPath(mat, visited, i + 1, j, x, y, min_dist, dist + 1);

if (isValid(i, j + 1) && isSafe(mat, visited, i, j + 1))
findShortestPath(mat, visited, i, j + 1, x, y, min_dist, dist + 1);

if (isValid(i - 1, j) && isSafe(mat, visited, i - 1, j))
findShortestPath(mat, visited, i - 1, j, x, y, min_dist, dist + 1);

if (isValid(i, j - 1) && isSafe(mat, visited, i, j - 1))
findShortestPath(mat, visited, i, j - 1, x, y, min_dist, dist + 1);

visited[i][j] = 0;
}

int main()
{
int mat[M][N] =
{
{ 1, 1, 1, 1, 1, 0, 0, 1, 1, 1 },
{ 0, 1, 1, 1, 1, 1, 0, 1, 0, 1 },
{ 0, 0, 1, 0, 1, 1, 1, 0, 0, 1 },
{ 1, 0, 1, 1, 1, 0, 1, 1, 0, 1 },
{ 0, 0, 0, 1, 0, 0, 0, 1, 0, 1 },
{ 1, 0, 1, 1, 1, 0, 0, 1, 1, 0 },
{ 0, 0, 0, 0, 1, 0, 0, 1, 0, 1 },
{ 0, 1, 1, 1, 1, 1, 1, 1, 0, 0 },
{ 1, 1, 1, 1, 1, 0, 0, 1, 1, 1 },
{ 0, 0, 1, 0, 0, 1, 1, 0, 0, 1 }
};

int visited[M][N];

memset(visited, 0, sizeof visited);
int min_dist = INT_MAX;
findShortestPath(mat, visited, 0, 0, 7, 5, min_dist, 0);
if (min_dist != INT_MAX)
cout << "The shortest path from source to destination "
"has length " << min_dist;
else
cout << "Destination can't be reached from given source";
return 0;
}
```

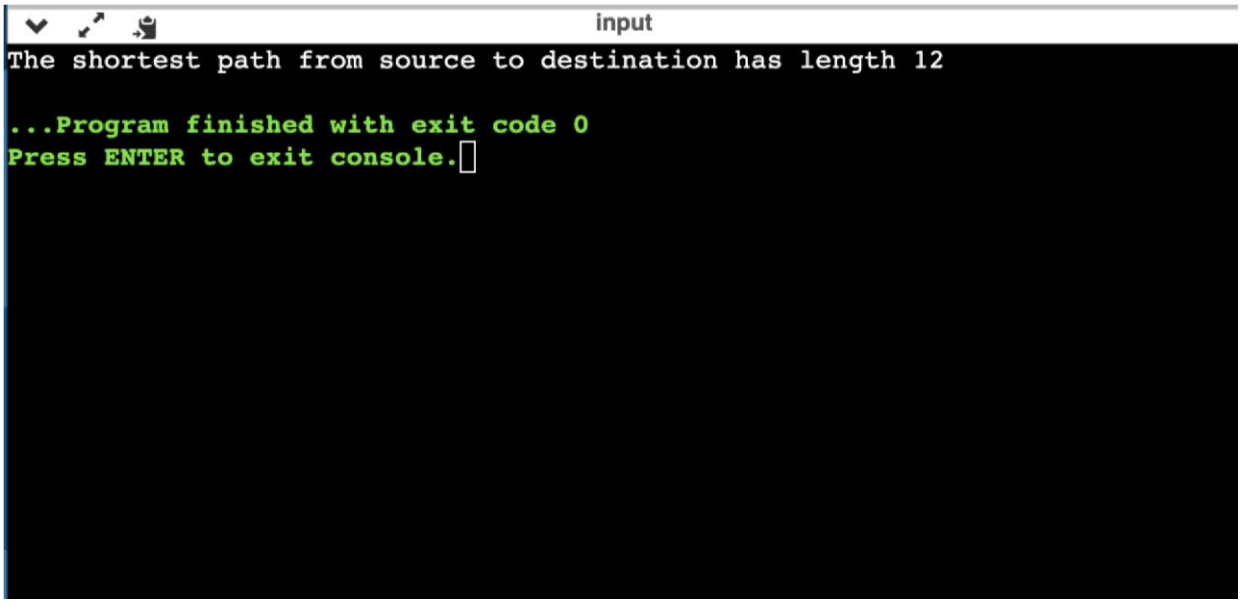
4.OBSERVATIONS/DISCUSSIONS/COMPLEXITY ANALYSIS

Time Complexity: O(V+E)

Space Complexity: O(V)

5.OUTPUT/RESULT





LEARNING OUTCOMES

1. Learnt about DFS and its application like Topological sort and Path from goal to source in a maze.
2. Also, learnt about how to analyze time and space complexity.

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.			