CU CHANDIGARH UNIVERSITY

DEPARTMENT OF

COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment-3.1

Student Name: UTKARSH JOSHI **UID:** 21BCS9158

Branch: CSE **Section/Group:** 21BCS_ST802-A

Semester: 5th **Date of Performance:** 2/11/2023

Subject Name: Design Analysis & Algorithm **Subject Code:** 21CSH-311

Aim:

Develop a program and analyze complexity 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, OR
(ii) to find a path from source to goal in a maze.

Objectives:

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

Input/Apparatus Used:

Laptop / PC & compiler

Algorithm:

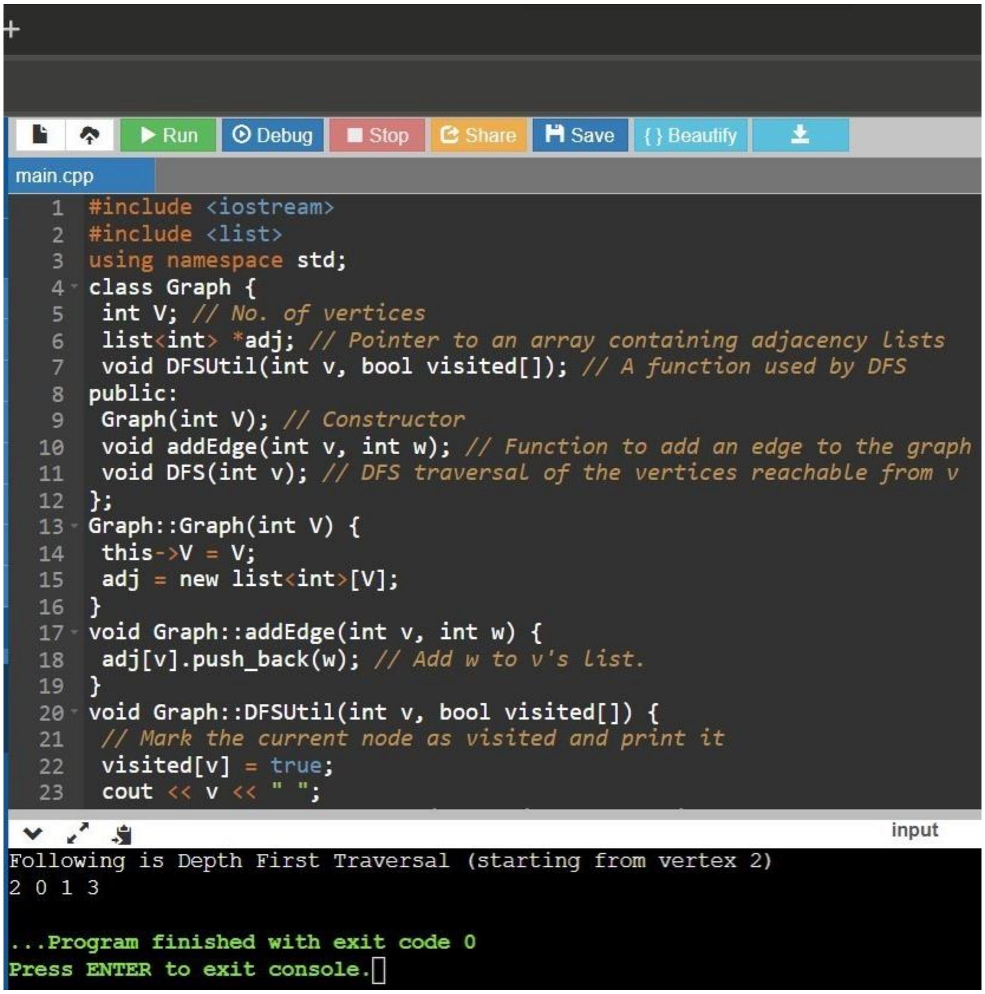
1. Start
2. Create a recursive function that takes the index of the node and a visited array.
3. Mark the current node as visited and print the node.
4. Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.
5. End

Code:

```
#include <iostream> #include
<list>


using namespace std;


class Graph {
    int V;              // No. of vertices

    list<int> *adj;      // Pointer to an array containing adjacency lists

    void DFSUtil(int v, bool visited[]); // A function used by DFS

public:
    Graph(int V);      // Constructor void addEdge(int v, int w); //
    Function to add an edge to the graph void DFS(int v);        // DFS
    traversal of the vertices reachable from v
};

Graph::Graph(int   V)   {
    this->V = V; adj = new
    list<int>[V];
}

void Graph::addEdge(int v, int w) { adj[v].push_back(w);
    // Add w to v's list.
}

void Graph::DFSUtil(int v, bool visited[]) { //
    Mark the current node as visited and print it
    visited[v] = true; cout << v << " ";



    // Recur for all the vertices adjacent to this vertex
    list<int>::iterator i; for (i = adj[v].begin(); i !=
    adj[v].end(); ++i) if (!visited[*i])
            DFSUtil(*i, visited);
}
```

```
      void Graph::DFS(int v) {
        // Mark all the vertices as not visited



        bool *visited = new bool[V]; for
        (int i = 0; i < V; i++)

          visited[i] = false;


        // Call the recursive helper function to print DFS traversal
        DFSUtil(v, visited);
      }

      int main() {
        // Create a graph given in the above diagram
        Graph g(4);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(2, 0);
        g.addEdge(2, 3);
        g.addEdge(3, 3); cout << "Following is Depth First Traversal (starting
        from vertex 2)\n"; g.DFS(2);

        return 0;
}
```

Observations/Outcome:



```
#include <iostream>
#include <list>
using namespace std;
class Graph {
 int V; // No. of vertices
 list<int> *adj; // Pointer to an array containing adjacency lists
 void DFSUtil(int v, bool visited[]); // A function used by DFS
public:
 Graph(int V); // Constructor
 void addEdge(int v, int w); // Function to add an edge to the graph
 void DFS(int v); // DFS traversal of the vertices reachable from v
};
Graph::Graph(int V) {
 this->V = V;
 adj = new list<int>[V];
}
void Graph::addEdge(int v, int w) {
 adj[v].push_back(w); // Add w to v's list.
}
void Graph::DFSUtil(int v, bool visited[]) {
 // Mark the current node as visited and print it
 visited[v] = true;
 cout << v << " ";
```

```
Following is Depth First Traversal (starting from vertex 2)
2 0 1 3

...Program finished with exit code 0
Press ENTER to exit console.
```

Time Complexity:

O(V + E), where V is the number of vertices and E is the number of edges in the graph.

Learning Outcomes :-

- Algorithmic Thinking.
- Graph Theory

- Implement of dfs