LAB – 05

Task –01:

Q1. BFS Implementation:

Code:

```
#include <bits/stdc++.h>
using namespace std;
void bfs(int node,vector<vector<int>>&adj,vector<bool>&visited){
    queue<int>q;
    q.push(node);
    visited[node] = true;
    while(!q.empty()){
        int curr = q.front();
        q.pop();
        cout<<curr<<" ";
        for(auto i:adj[curr]){
            if(!visited[i]){
                q.push(i);
                visited[i] = true;
            } } }}
int main(){
    vector<vector<int>>adj(10);
    adj[1] = {2,3};adj[2] = {1,3,4,5};adj[3] = {1,2,6};adj[4] = {2,5};
    adj[5] = {2,4,6,7};adj[6] = {3,5,8,9};adj[7] = {5,8};adj[8] = {6,7};adj[9] = {6};
    vector<bool>visited(10,false);
    int i;
    cout<<"Enter Node to start: ";
    cin>>i;
    bfs(i,adj,visited);
}
```

```
PS C:\Users\Ritik gupta\Desktop\Lab\DAA\Lab 5> cd "c:\Users\Ritik gupta\Desktop\L
 }
Enter Node to start: 2
2 1 3 4 5 6 7 8 9
```

Q2.DFS Implementation:

Code:

```
#include <bits/stdc++.h>
using namespace std;
void dfs(int node,vector<vector<int>>&adj,vector<bool>&visited){
    cout<<node<<" ";
    visited[node] = true;
    for(auto i : adj[node]){
        if(!visited[i]){
            dfs(i,adj,visited);
        }}}
void dfsStack(int start, vector<vector<int>>& adj, vector<bool>& visited) {
    stack<int> st;
    st.push(start);
    while (!st.empty()) {
```

```
        int node = st.top();
        st.pop();
        if (!visited[node]) {
            cout << node << " ";
            visited[node] = true;}
        for (auto neighbor : adj[node]) {
            if (!visited[neighbor]) {
                st.push(neighbor);}} }
}
int main(){
    vector<vector<int>>adj(10);
    adj[1] = {2,3};adj[2] = {1,3,4,5};adj[3] = {1,2,6};adj[4] = {2,5};
    adj[5] = {2,4,6,7};adj[6] = {3,5,8,9};adj[7] = {5,8};adj[8] = {6,7};adj[9] = {6};
    vector<bool>visited(10,false);
    int i;
    cout<<"Enter Node to start: ";
    cin>>i;
    cout<<"Dfs using recussion: ";
    dfs(i,adj,visited);
    vector<bool>visited2(10,false);
    cout<<"\nDfs using stack: ";
    dfsStack(i,adj,visited2);
}
```

```
PS C:\Users\Ritik gupta\Desktop\Lab\DAA\Lab 5> cd "c:\Users\Ritik gupta\Desktop\Lab\DAA\Lab 5\" ; if ($?) { g++ 2
 }
Enter Node to start: 2
Dfs using recussion: 2 1 3 6 5 4 7 8 9
Dfs using stack: 2 5 7 8 6 9 3 1 4
```

Q3.Prim's Algorithm:

Code:

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

struct Edge {
    int destination;
    int weight;
};
class Graph {
private:
    int numVertices;
    vector<list<Edge>> adjList;
public:
    Graph(int vertices) {
        numVertices = vertices;
        adjList.resize(vertices);
    }
    void addEdge(int source, int destination, int weight) {
        if (source >= 0 && source < numVertices && destination >= 0 && destination <
numVertices) {
            adjList[source].push_back({destination, weight});
            adjList[destination].push_back({source, weight});
        }
```

```
    }
    void displayGraph() {
        for (int i = 0; i < numVertices; i++) {
            cout << "Vertex " << i << " is connected to: ";
            for (const auto& edge : adjList[i]) {
                cout << edge.destination << " (weight: " << edge.weight << ") ";
            }
            cout << endl;
        }
    }
    int getNumVertices() {
        return numVertices;
    }
    const list<Edge>& getAdjList(int vertex) {
        if (vertex >= 0 && vertex < numVertices) {
            return adjList[vertex];
        }
        static list<Edge> emptyList;
        return emptyList;
    }
    void primAlgorithm() {
        vector<int> parent(numVertices, -1);
        vector<int> key(numVertices, INT_MAX);
        vector<bool> inMST(numVertices, false);
        priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>>
pq;
        key[0] = 0;
        pq.push({0, 0});
        while (!pq.empty()) {
            int u = pq.top().second;
            pq.pop();
            if (inMST[u]) continue;
            inMST[u] = true;
            for (const Edge& edge : adjList[u]) {
                int v = edge.destination;
                int weight = edge.weight;
                if (!inMST[v] && weight < key[v]) {
                    key[v] = weight;
                    parent[v] = u;
                    pq.push({key[v], v});
                }
            }
        }
        cout << "\nMinimum Spanning Tree (MST) using Prim's Algorithm:\n";
        for (int i = 1; i < numVertices; i++) {
            cout << "Edge: " << parent[i] << " - " << i << " (Weight: " << key[i] <<
")\n";
        }}};
int main() {
    Graph graph(7);
    graph.addEdge(0,1,28);graph.addEdge(0,5,10);
graph.addEdge(5,4,25);graph.addEdge(4,3,22);
```

```
graph.addEdge(3,2,12);graph.addEdge(2,1,16);graph.addEdge(6,3,18);graph.addEdge(6,4,24);gr
aph.addEdge(6,1,14);
    cout << "Original graph:" << endl;
    graph.displayGraph();
    graph.primAlgorithm();
    return 0;
}
```

```
Minimum Spanning Tree (MST) using Prim's Algorithm:
Edge: 2 - 1 (Weight: 16)
Edge: 3 - 2 (Weight: 12)
Edge: 4 - 3 (Weight: 22)
Edge: 5 - 4 (Weight: 25)
Edge: 0 - 5 (Weight: 10)
Edge: 1 - 6 (Weight: 14)
PS C:\Users\Ritik gupta\Desktop\Lab\DAA\Lab 5>
```

Task- 02

Q1.Topological sort:

```
#include <bits/stdc++.h>
using namespace std;
void dfs(int node, vector<vector<int>>& adj, vector<bool>& visited, stack<int>& st) {
    visited[node] = true;
    for (int neighbor : adj[node]) {
        if (!visited[neighbor]) {
            dfs(neighbor, adj, visited, st);
        }
    }
    st.push(node);
}
void topologicalSort(int n, vector<vector<int>>& adj) {
    vector<bool> visited(n, false);
    stack<int> st;
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            dfs(i, adj, visited, st);
        }
    }
    while (!st.empty()) {
        cout<<st.top()<<" ";
        st.pop();
    }}
int main() {
    int v=6;
    vector<vector<int>> adj(v);
    adj ={{},{},{3},{1},{0,1},{0,2}};
    cout << "Topological Sort Order: ";
    topologicalSort(v, adj);
    return 0;}
```

```
PS C:\Users\Ritik gupta\Desktop\Lab\DAA\Lab 5> cd "c:\Users\Ritik gupta\Desktop\Lab\
 }
Topological Sort Order: 5 4 2 3 1 0
PS C:\Users\Ritik gupta\Desktop\Lab\DAA\Lab 5>
```

Leetcode:

Q1.Construct smallest number from DI string:

2375. Construct Smallest Number From DI String

Solved

Medium | Topics | Companies | Hint

You are given a **0-indexed** string `pattern` of length `n` consisting of the characters `'I'` meaning **increasing** and `'D'` meaning **decreasing**.

A **0-indexed** string `num` of length `n + 1` is created using the following conditions:

- `num` consists of the digits `'1'` to `'9'`, where each digit is used **at most** once.
- If `pattern[i] == 'I'`, then `num[i] < num[i + 1]`.
- If `pattern[i] == 'D'`, then `num[i] > num[i + 1]`.

Return *the lexicographically **smallest** possible string* `num` *that meets the conditions.*

C++ Auto

```cpp
class Solution {
public:
    string smallestNumber(string pattern) {
        stack<int>st;
        string ans ="";

    for(int i=0;i<=pattern.length();i++){
        st.push(i+1);
        if(i==pattern.length() || pattern[i] == 'I'){
            while(!st.empty()){
                ans = ans + to_string(st.top());
                st.pop();
            }
        }
    }
    return ans;
    }
};
```

Accepted 104 / 104 testcases passed

RITIK GUPTA submitted at Feb 18, 2025 20:29

Editorial | Solution

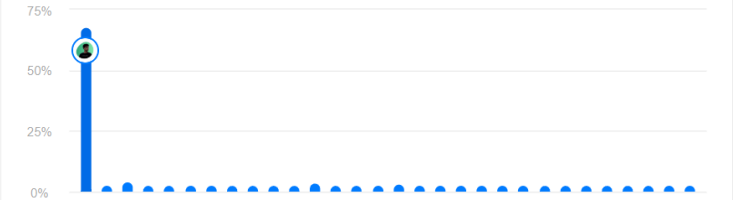
Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

8.54 MB | Beats 18.33%



Q2.Letter Tile Possibilities

1079. Letter Tile Possibilities

Solved

Medium | Topics | Companies | Hint

You have `n` `tiles`, where each tile has one letter `tiles[i]` printed on it.

Return *the number of possible non-empty sequences of letters* you can make using the letters printed on those `tiles`.

Example 1:

```
Input: tiles = "AAB"
Output: 8
Explanation: The possible sequences are "A", "B", "AA", "AB", "BA",
"AAB", "ABA", "BAA".
```

C++ Auto

```cpp
class Solution {
public:
int x=0;
    void count(string ans,unordered_map<char,int> &map){
        for(auto &pair:map){
            if(pair.second>0){
                map[pair.first]--;
                x++;
            count(ans+pair.first,map);
                map[pair.first]++; }}}
    int numTilePossibilities(string tiles) {
        string ans;
        unordered_map<char,int>map;
        for(auto i:tiles){
            map[i]++;}
        count(ans,map);
     return x;
    }};
```

Accepted 86 / 86 testcases passed

RITIK GUPTA submitted at Feb 18, 2025 00:48

Editorial | Solution

Runtime

2 ms | Beats 79.17%

Analyze Complexity

Memory

9.08 MB | Beats 50.10%