DSAL Practical’s

Consider the telephone book database of N clients. Make use of a hash table

implementation to quickly look up a client’s telephone number. Make use oftwo collision handling techniques and compare them using number of

comparisons required to find a set of telephone numbers

Experiment No. 6

Represent a given graph using adjacency matrix/list to perform DFS and using adjacency list to perform BFS. Use the map of the area around the college as the graph. Identify the prominent landmarks as nodes and perform DFS and BFS on that.

#include<iostream>

#include<stack>

#include<queue>

#include<vector>

#include<map>

#include<iomanip>

using namespace std;

class Graph {

    private:

        int vertices;

        map<string, int> landmarkIndex;

        map<int, string> indexLandmark;

        vector<vector<int>> adjMatrix;

        map<int, vector<int>> adjList;

    public:

        Graph(int v){

            vertices = v;

            adjMatrix.resize(vertices, vector<int>(vertices, 0));

        }

        void addLandmark(string name, int index){

            landmarkIndex[name] = index;

            indexLandmark[index] = name;

        }

        void addEdge(string src, string dest){

            if(landmarkIndex.find(src) == landmarkIndex.end() || landmarkIndex.find(dest ) == landmarkIndex.end()){

                cout << "Invalid landmarks entered!\n";

                return;

            }

            int u = landmarkIndex[src];

            int v = landmarkIndex[dest];

            adjMatrix[u][v] = 1;

            adjMatrix[v][u] = 1; // For undirected graph

            adjList[u].push\_back(v);

            adjList[v].push\_back(u); // For undirected graph

        }

        void dfs()

        {

            cout << "Enter starting landmark for DFS: ";

            string startLandmark;

            getline(cin, startLandmark);

            if (landmarkIndex.find(startLandmark) == landmarkIndex.end()) {

                cout << "Invalid starting landmark!" << endl;

                return;

            }

            int start = landmarkIndex[startLandmark];

            vector<bool> visited(vertices, false);

            stack<int> s;

            s.push(start);

            visited[start] = true;

            cout << "DFS Traversal: ";

            while(!s.empty())

            {

                int node = s.top();

                s.pop();

                cout << indexLandmark[node] << " ";

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

                {

                    if(adjMatrix[node][i] != 0 && !visited[i])

                    {

                        s.push(i);

                        visited[i] = true;

                    }

                }

            }

        }

        void bfs()

        {

            cout << "Enter starting landmark for BFS: ";

            string startLandmark;

            getline(cin, startLandmark);

            if (landmarkIndex.find(startLandmark) == landmarkIndex.end()) {

                cout << "Invalid starting landmark!" << endl;

                return;

            }

            int start = landmarkIndex[startLandmark];

            vector<bool> visited(vertices, false);

            queue<int> q;

            q.push(start);

            visited[start] = true;

            cout << "BFS Traversal: ";

            while(!q.empty())

            {

                int node = q.front();

                q.pop();

                cout << indexLandmark[node] << " ";

                for( int neighbor : adjList[node])

                {

                    if(!visited[neighbor]){

                        visited[neighbor] = true;

                        q.push(neighbor);

                    }

                }

            }

        }

       void displayGraph()

       {

         int maxwidth = 0;

         for( int i = 0; i < vertices; i++){

            if(indexLandmark[i].length() > maxwidth){

                maxwidth = indexLandmark[i].length();

            }

         }

         maxwidth += 2; // For padding

         cout << "Graph Adjacency Matrix:" << endl;

         cout << setw(maxwidth) << " ";

         for( int i = 0; i < vertices; i++){

            cout << setw(maxwidth) << indexLandmark[i];

         }

         cout << endl;

         for ( int i = 0; i < vertices; i++){

            cout << setw(maxwidth) << indexLandmark[i];

            for( int j = 0; j < vertices; j++){

                cout << setw(maxwidth) << adjMatrix[i][j];

            }

            cout << endl;

         }

         cout << "Graph Adjacency List:" << endl;

         for( auto &pair : adjList){

            cout << setw(maxwidth) << indexLandmark[pair.first] << "-> ";

            for( int neighbor : pair.second){

                cout << indexLandmark[neighbor] << " ";

            }

            cout << endl;

         }

        }

};

int main()

{

    int num\_landmarks;

    int choice;

    cout << "Enter the number of landmarks: ";

    cin >> num\_landmarks;

    cin.ignore();

    Graph g(num\_landmarks);

    cout << "Enter the landmarks (name): " << endl;

    for (int i = 0; i< num\_landmarks; i++){

        string name;

        getline(cin, name);

        g.addLandmark(name, i);

    }

    //menu driven

    do{

        cout<< endl << "Menu: " << endl;

        cout<< "1. Add Edge" << endl;

        cout<< "2. Display Graph" << endl;

        cout<< "3. DFS Traversal" << endl;

        cout<< "4. BFS Traversal" << endl;

        cout<< "5. Exit" << endl;

        cout<< "Enter your choice: ";

        cin >> choice;

        cin.ignore(); // To ignore the newline character after the integer input

        switch (choice) {

            case 1: {

                string src, dest;

                cout << "Enter source landmark: ";

                getline(cin, src);

                cout << "Enter destination landmark: ";

                getline(cin, dest);

                g.addEdge(src, dest);

                break;

            }

            case 2:

                g.displayGraph();

                break;

            case 3:

                g.bfs();

                break;

            case 4:

                g.dfs();

                break;

            case 5:

                cout << "Exiting program.\n";

                return 0;

            default:

                cout << "Invalid choice! Try again.\n";

        }

    }while(choice != 5);

    return 0;

}

Experiment No. 7

You have a business with several offices; you want to lease phone lines to connect them up with each other; and the phone company charges different amounts of money to connect different pairs of cities. You want a set of lines that connects all your offices with a minimum total cost. Solve the problem by suggesting appropriate data structures.