NAME:- PARAM RASIKLAL KANADA

ADMISSION NO. :- U24AI047

SUBJECT:- AI202 ARTIFICIAL INTELLIFENCE

ASSIGNMENT:- LAB 1

Q1 Consider a map given in image.

Implement Breadth First Search(BFS) and Depth First Search (DFS) algorithm to find all possible step cost between Syracuse to Chicago.

SOLUTION:

```
#include <bits/stdc++.h>

using namespace std;


map <string,vector<pair<string, int>>> graph = {

   {"Chicago", {{"Detroit", 283}, {"Cleveland", 345}, {"Indianapolis",182}}},

   {"Indianapolis", {{"Chicago",182}, {"Columbus",176}}},
```

```
    {"Columbus", {{"Indianapolis",176}, {"Cleveland",144}, {"Pittsburgh",185}}},
    {"Cleveland", {{"Chicago",345}, {"Detroit",169}, {"Columbus",144},
{"Pittsburgh",134}}},
    {"Detroit", {{"Chicago",283}, {"Cleveland",169}, {"Buffalo",256}}},
    {"Buffalo", {{"Detroit",256}, {"Cleveland",189}, {"Pittsburgh",215},
{"Syracuse",150}}},
    {"Pittsburgh", {{"Cleveland",134}, {"Columbus",185}, {"Buffalo",215},
{"Philadelphia",305}, {"Baltimore",247}}},
    {"Syracuse", {{"Buffalo",150}, {"New York",254}, {"Boston",312}}},
    {"New York", {{"Syracuse",254}, {"Philadelphia",97}, {"Boston",215},
{"Providence",181}}},
    {"Philadelphia", {{"New York",97}, {"Pittsburgh",305}, {"Baltimore",101}}},
    {"Baltimore", {{"Philadelphia",101}, {"Pittsburgh",247}}},
    {"Boston", {{"Syracuse",312}, {"New York",215}, {"Providence",50},
{"Portland",107}}},
    {"Providence", {{"Boston",50}, {"New York",181}}},
    {"Portland", {{"Boston",107}}}};

void BFS (string start, string goal){
    queue<pair<string, int>> q;
    map<string, bool> visited;
    map<string, string> parent;

    q.push({start, 0});
    visited[start] = true;

    while(!q.empty()){
        // auto[city,cost]=q.front();
```

```
pair<string,int> p=q.front();
string city = p.first;
int cost = p.second;
q.pop();

//terminating condition
if(city==goal){
    vector<string> path;
    string cur = goal;
    int totalCost = cost;

    while(cur!= ""){
        path.push_back(cur);
        cur = parent[cur];
    }
    reverse(path.begin(), path.end());
    //reversed to get correct order

    cout << "\nBFS Path: ";
    for (int i = 0; i < path.size(); i++) {
        cout << path[i] << "-->";
    }
    cout << "END";

    cout << "\nTotal Step Cost: " << totalCost << " miles\n";
```

```
            return;
        }

        for (auto &neighbor : graph[city]) {
            if (!visited[neighbor.first]) {
                visited[neighbor.first] = true;
                parent[neighbor.first] = city;
                q.push({neighbor.first, cost + neighbor.second});
            }
        }
    }
}

void DFS(string start, string goal) {
    stack<pair<string, int>> st;
    map<string, bool> visited;
    map<string, string> parent;

    st.push({start, 0});

    while (!st.empty()) {
        // auto [city, cost] = st.top();
        pair<string,int> p=st.top();
        string city = p.first;
        int cost = p.second;
```

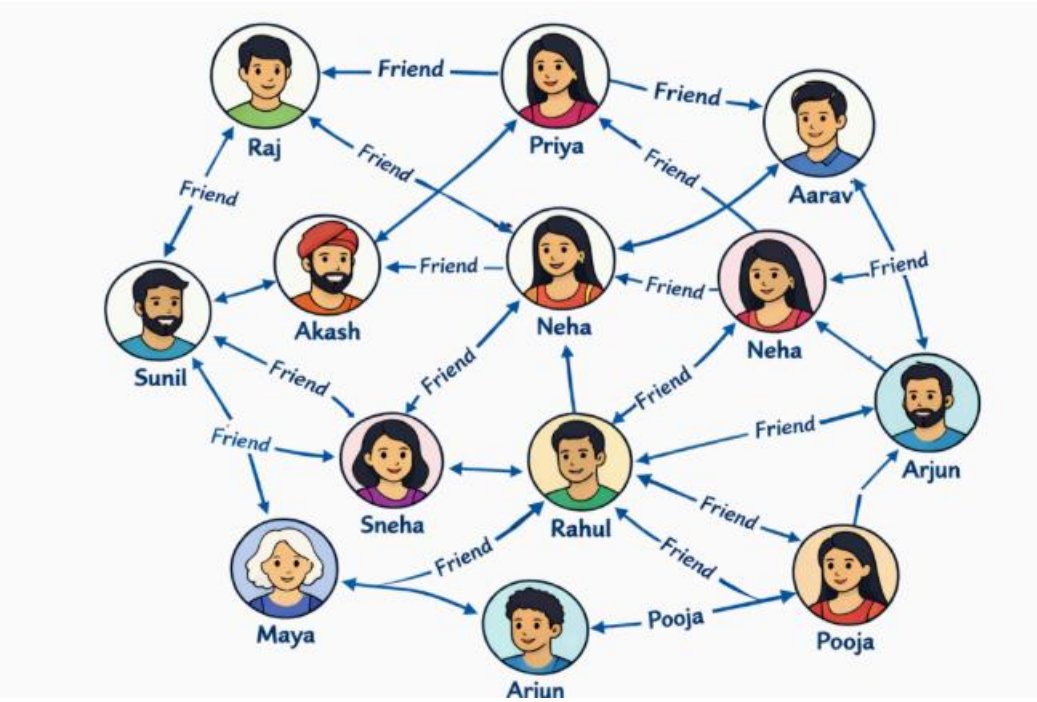
```
st.pop();

if (visited[city]) continue;
visited[city] = true;

if (city == goal) {
    vector<string> path;
    string cur = goal;

    while (cur != "") {
        path.push_back(cur);
        cur = parent[cur];
    }
    reverse(path.begin(), path.end());

    // Calculate actual path cost
    int totalCost = 0;
    for (int i = 0; i < path.size() - 1; i++) {
        for (auto &neighbor : graph[path[i]]) {
            if (neighbor.first == path[i + 1]) {
                totalCost += neighbor.second;
                break;
            }
        }
    }
```

```
            cout << "\nDFS Path: ";
            for (int i = 0; i < path.size(); i++) {
                cout << path[i] << "-->";
            }
            cout << "END";

            cout << "\nTotal Step Cost: " << totalCost << " miles\n";

            return;
        }

        for (auto &neighbor : graph[city]) {
            if (!visited[neighbor.first]) {
                parent[neighbor.first] = city;
                st.push({neighbor.first, cost + neighbor.second});
            }
        }
    }
}



    int main(){
        string startCity="Chicago";
        string goalCity = "New York";
```

```
    cout<< "Starting City: " << startCity <<endl;
    cout<<"Goal City: " << goalCity <<endl;

    BFS(startCity, goalCity);
    DFS(startCity, goalCity);

  return 0;

  }
```

```
Starting City: Chicago
Goal City: New York

BFS Path: Chicago-->Detroit-->Buffalo-->Syracuse-->New York-->END
Total Step Cost: 943 miles

DFS Path: Chicago-->Indianapolis-->Columbus-->Pittsburgh-->Baltimore-->Philadelphia-->New York-->END
Total Step Cost: 988 miles
```

Q2 Consider a social media network as given below. Find BFS and DFS tree.



SOLUTION:

```
#include <iostream>
```

```
#include<bits/stdc++.h>

using namespace std;

class SocialNetwork {
   //This will give me adjacent nodes (direct neighbours/ friends)
   map<string, vector<string>> adj;

public:
   //fxn to add u follows these v
   void addEdge(string u, string v) {
      adj[u].push_back(v);
   }

   // Helper to sort neighbors for consistent output
   void sortNeighbors() {
      for (auto& pair : adj) {
         sort(pair.second.begin(), pair.second.end());
      }
   }

   void BFS(string startNode) {
      set<string> visited;
      queue<string> q;

      visited.insert(startNode);
```

```
    q.push(startNode);

    cout << "BFS Traversal: ";
    while (!q.empty()) {
        string current = q.front();
        q.pop();
        cout << current << " -> ";

        for (string neighbor : adj[current]) {
            if (visited.find(neighbor) == visited.end()) {
                visited.insert(neighbor);
                q.push(neighbor);
            }
        }
    }
}

void DFSUtil(string node, set<string>& visited) {
    visited.insert(node);
    cout << node << " -> ";

    for (string neighbor : adj[node]) {
        if (visited.find(neighbor) == visited.end()) {
            DFSUtil(neighbor, visited);
        }
    }
```

```
    }

    void DFS(string startNode) {
        set<string> visited;
        cout << "DFS Traversal: ";
        DFSUtil(startNode, visited);
        cout << "End (Other nodes unreachable)" << endl;
    }
};

int main() {
    SocialNetwork g;

    // --- Building the DIRECTED Graph ---

    // Sunil's Outgoing
    g.addEdge("Sunil", "Raj");
    g.addEdge("Sunil", "Sneha");
    g.addEdge("Sunil", "Akash");
    g.addEdge("Sunil", "Maya");

    // Raj's Outgoing
    g.addEdge("Raj", "Akash");
    g.addEdge("Raj", "Neha");

    // Akash's Outgoing
```

```
g.addEdge("Akash", "Sunil");
g.addEdge("Akash", "Priya");
// g.addEdge("Akash", "Neha (Center)");

// Sneha's Outgoing
g.addEdge("Sneha", "Akash");
g.addEdge("Sneha", "Rahul");
g.addEdge("Sneha", "Neha");
g.addEdge("Sneha", "Maya");

// Neha (Center)'s Outgoing
g.addEdge("Neha (Center)", "Sneha");
g.addEdge("Neha (Center)", "Akash");
g.addEdge("Neha (Center)", "Aarav");
g.addEdge("Neha (Center)", "Sneha");


g.addEdge("Priya", "Raj");
g.addEdge("Priya", "Akash");
// g.addEdge("Priya", "Neha (Center)");
g.addEdge("Priya", "Aarav");

g.addEdge("Rahul", "Sneha");
g.addEdge("Rahul", "Neha (Center)");
g.addEdge("Rahul", "Neha (Right)");
g.addEdge("Rahul", "Maya");
```

```
g.addEdge("Rahul", "Pooja");
g.addEdge("Rahul", "Arjun (Right)");

g.addEdge("Neha (Right)", "Neha (Center)");
g.addEdge("Neha (Right)", "Aarav");
g.addEdge("Neha (Right)", "Priya");
g.addEdge("Neha (Right)", "Arjun (Right)");
g.addEdge("Neha (Right)", "Rahul");

g.addEdge("Aarav", "Arjun (Right)");
g.addEdge("Aarav", "Neha (Right)");
g.addEdge("Aarav", "Neha (Center)");

// g.addEdge("Arjun (Right)", "Pooja");
g.addEdge("Arjun (Right)", "Rahul");
g.addEdge("Arjun (Right)", "Aarav");
g.addEdge("Arjun (Right)", "Neha (Right)");

g.addEdge("Pooja", "Arjun (Bottom)");
g.addEdge("Pooja", "Arjun (Right)");
g.addEdge("Pooja", "Rahul");

g.addEdge("Maya", "Arjun (Bottom)");
g.addEdge("Maya", "Rahul");
g.addEdge("Maya", "Sneha");
g.addEdge("Maya", "Sunil");
```

```
    g.addEdge("Arjun (Bottom)", "Rahul");
    g.addEdge("Arjun (Bottom)", "Maya");
    g.addEdge("Arjun (Bottom)", "Pooja");

    // Sort for deterministic output
    g.sortNeighbors();

    cout << "Starting Directed Traversals from 'Sunil':\n" << endl;

    g.BFS("Sunil");
    cout << endl;
    g.DFS("Sunil");

    return 0;
}
```

```
$ ./A.EXE
Starting Directed Traversals from 'Sunil':

BFS Traversal: Sunil -> Akash -> Maya -> Raj -> Sneha -> Priya -> Arjun (Bottom) -> Rahul -> Neha -> Aarav -> Pooja -> Arjun (Right) ->
Neha (Center) -> Neha (Right) ->
DFS Traversal: Sunil -> Akash -> Priya -> Aarav -> Arjun (Right) -> Neha (Right) -> Neha (Center) -> Sneha -> Maya -> Arjun (Bottom) ->
Pooja -> Rahul -> Neha -> Raj -> End (Other nodes unreachable)
```