East West University
Department of Computer Science and Engineering
Fall '24

Project Report

Course Code: CSE246
Course Title: Algorithms
Section: 1

Project Title: Travel Planner Using Graphs

Name	Student ID
1. M. Nura Alam Naim	2022-3-60-123

Date of Submission: 30 January 2025

Title: Travel Planner Using Graphs.

Abstract:

This project presents a program that calculates the minimum distance, time, and cost between two nodes in a weighted graph. The graph is generated randomly with N nodes and less than or equal to N*(N-1)/2 edges, each representing different aspects of a travel network. The program uses Dijkstra's algorithm to compute the shortest paths for each criterion: distance, time, and cost. The implementation also allows the user to interactively input source and target nodes to compute the optimal paths.

Introduction:

This project aims to simulate a travel network where each edge between nodes has three associated properties: distance, cost, and time. The objective is to calculate the optimal route between any two nodes based on one of the following criteria:

- **Minimum Distance**
- **Minimum Time**
- **Minimum Cost**

The program is designed using Dijkstra's algorithm, which is well-suited for solving shortest path problems in weighted graphs. This solution can be applied in various real-life scenarios such as travel planning, logistics optimization, and route selection for transportation systems.

Problem Statement:

Given a travel network with a set of nodes and weighted edges, calculate the optimal path between two nodes based on the following criteria:

1. The minimum distance between the source and target nodes.
2. The minimum time required to travel from the source to the target node.
3. The minimum cost incurred to travel between the two nodes.

The graph consists of 10 nodes and 45 edges, where each edge has an associated distance, cost, and time.

Methodology:

The program follows these steps to solve the problem:

1. **Graph Representation:**
    - A graph is represented as three separate adjacency matrices for distance, cost, and time. Each entry in the matrix indicates the respective weight between two nodes.
2. **Graph Initialization:**
    - The `initial()` function generates a random graph with 10 nodes and 45 edges. The edges have random values for distance, cost, and time, ensuring a variety of weights.

3. **Dijkstra’s Algorithm:**
    - The program uses Dijkstra's algorithm to calculate the shortest path between the source and target nodes. The algorithm works by iteratively selecting the node with the smallest tentative distance and updating the distances of its neighbors.
4. **User Interaction:**
    - The program allows the user to choose between three modes of optimization:
        - **Minimum Distance**
        - **Minimum Time**
        - **Minimum Cost**
    - The user is prompted to input the source and target nodes, and the program computes the optimal path based on the selected criterion.
5. **Results Calculation:**
    - Once the optimal path is found, the program calculates the total distance, total time, and total cost for the selected route.

Results:

Sample Output:

```
Minimum Distance From: 3 to 9 is: 53
The route: 3 -> 8 -> 5 -> 1 -> 2 -> 9
Total Cost: 1622
Total Required Time: 415

Minimum Required time From: 1 to 7 is: 109
The route: The route: 1 -> 2 -> 9 -> 7
Total Cost: 1133
Total Distance: 103

Minimum Required Cost From: 8 to 3 is: 107
The route: The route: 8 -> 7 -> 3
Total Reqired: 178
Total Distance: 27
```

Explanation:

- The program calculates the minimum distance, total cost, and total time based on the selected graph's parameters. Each of the three criteria is calculated independently using the respective adjacency matrix.

Code Explanation:

1. **Graph Initialization (`initial()`):**
    - Randomly generates a graph with distance, cost, and time for each edge between nodes.
2. **Dijkstra's Algorithm (`dijstra()`):**
    - Implements Dijkstra's algorithm to find the shortest path in a weighted graph. It selects the node with the smallest tentative distance and updates the distances of neighboring nodes.
3. **Main Menu (`main()`):**
    - The user is presented with a menu to choose the criterion (distance, time, or cost) for path optimization. The program performs the corresponding calculations based on the user's choice.
4. **Result Calculation (`calc()`):**
    - After computing the optimal route, the program calculates the total distance, time, and cost by traversing the route.

Conclusion:

This project successfully calculates the optimal route between two nodes in a travel network based on three criteria: distance, time, and cost. By utilizing Dijkstra's algorithm, the program efficiently finds the shortest paths for each criterion. The random generation of the graph ensures that the program can be used to simulate various travel networks. This approach can be applied to real-world scenarios such as travel planning, logistics, and transportation optimization.

Future improvements can include:

- Handling more complex graphs with additional constraints (e.g., traffic conditions, multiple routes).
- Improving the user interface for better interaction.
- Extending the graph size for larger networks.

Future Work:

- Implementing more sophisticated algorithms such as A* for heuristic-based optimization.
- Allowing users to input custom graphs for personalized calculations.
- Integrating a web interface to allow users to visualize the travel routes.

Source Code:

```
//|ADMIRAL_AUDITORE|//
#include <bits/stdc++.h>
using namespace std;
typedef long long int ll;
typedef vector<ll> vl;
#define nl "\n"
#define sp ' '
#define yes cout << "YES" << nl
#define no cout << "NO" << nl
#define all(v) v.begin(), v.end()
#define pb push_back
#define ff first
#define ss second
const ll N = 10;                          // Nodes
const ll M = 45;                          // Edges: max: (N*(N-1))/2
vector<vl> dis(N + 1, vl(N + 1, 0));
vector<vl> cst(N + 1, vl(N + 1, 0));
vector<vl> tim(N + 1, vl(N + 1, 0));
ll cnt = 0;
void initial()
{
   srand(time(0));
   while (cnt < M)
   {
      ll u = 1 + rand() % N;
      ll v = 1 + rand() % N;

      if (u == v or dis[u][v])
         continue;

      ll d = 2 + rand() % 99;
      ll c = 5 + rand() % 499;
      ll t = 2 + rand() % 149;

      dis[u][v] = d;
      cst[u][v] = c;
      tim[u][v] = t;

      dis[v][u] = d;
      cst[v][u] = c;
      tim[v][u] = t;

      cnt++;
   }
}
```

```
void print_route(vl &route)
{
   cout << "The route: ";
   for (ll i = 0; i < (ll)route.size(); i++)
   {
      cout << route[i];
      if (i == (ll)route.size() - 1)
         cout << nl;
      else
         cout << " -> ";
   }
}
void print()
{
   cout << "Distance Graph: " << nl;
   for (ll i = 1; i <= N; i++)
   {
      for (ll j = 1; j <= N; j++)
         cout << dis[i][j] << sp;
      cout << nl;
   }
   cout << nl;
   cout << "Cost Graph: " << nl;
   for (ll i = 1; i <= N; i++)
   {
      for (ll j = 1; j <= N; j++)
         cout << cst[i][j] << sp;
      cout << nl;
   }
   cout << nl;
   cout << "Time Graph: " << nl;
   for (ll i = 1; i <= N; i++)
   {
      for (ll j = 1; j <= N; j++)
         cout << tim[i][j] << sp;
      cout << nl;
   }
   cout << nl << nl;
}
ll minFind(vector<bool> &visited, vl &dist)
{
   ll mn = LLONG_MAX;
   ll node = -1;
   for (ll i = 1; i <= N; i++)
   {
      if (!visited[i] and dist[i] <= mn)
      {
         mn = dist[i];
```

```
            node = i;
        }
    }
    return node;
}
vl dijstra(ll src, ll target, ll &req, vector<vl> &graph)
{
    vl dist(N + 1, LLONG_MAX);
    vector<bool> visited(N + 1, false);
    vl parent(N + 1, 0);
    dist[src] = 0;
    parent[src] = src;
    for (ll i = 1; i <= N; i++)
    {
        ll u = minFind(visited, dist);
        if (u == -1)
            break;
        visited[u] = true;
        for (ll v = 1; v <= N; v++)
            if (!visited[v] and graph[u][v] and dist[u] != LLONG_MAX and dist[u] + graph[u][v] < dist[v])
            {
                dist[v] = dist[u] + graph[u][v];
                parent[v] = u;
            }
    }
    ll t = target;
    vl route;
    if (!parent[t])
    {
        req = -1;
        return route;
    }
    route.pb(target);
    while (t != src)
    {
        if (!parent[t])
        {
            req = -1;
            return route;
        }
        route.pb(parent[t]);
        t = parent[t];
    }
    req = dist[target];
    reverse(all(route));
    return route;
}
void calc(vl &route, ll &tot_dis, ll &tot_tm, ll &tot_cst)
```

```
{
   ll u = route[0];
   for (ll i = 1; i < (ll)route.size(); i++)
   {
      tot_dis += dis[u][route[i]];
      tot_tm += tim[u][route[i]];
      tot_cst += cst[u][route[i]];
      u = route[i];
   }
}
void shortest_distance()
{
   cout << "\n# Minimum Distance from source to target #" << nl;
   ll src, target;
   src = 1;
   target = 5;
   cout << "Enter your Source Node (Between 1 to " << N << " ): ";
   cin >> src;
   cout << "Enter your Target Node (Between 1 to " << N << " ): ";
   cin >> target;
   ll req = -1;
   vl route = dijstra(src, target, req, dis);
   if (req == -1)
   {
      cout << "No Path Exixts" << nl;
      return;
   }
   ll tot_dis = 0, tot_tm = 0, tot_cst = 0;
   calc(route, tot_dis, tot_tm, tot_cst);
   cout << "Minimum Distance From: " << src << " to " << target << " is: " << tot_dis << nl;
   print_route(route);
   cout << "Total Cost: " << tot_cst << nl;
   cout << "Total Required Time: " << tot_tm << nl;
   cout << nl;
}
void minimum_time()
{
   cout << "\n# Miminum Required Time from source to target #" << nl;
   ll src, target;
   src = 1;
   target = 5;
   cout << "Enter your Source Node (Between 1 to " << N << " ): ";
   cin >> src;
   cout << "Enter your Target Node (Between 1 to " << N << " ): ";
   cin >> target;
   ll req = -1;
   vl route = dijstra(src, target, req, tim);
   if (req == -1)
```

```
    {
        cout << "No Path Exixts" << nl;
        return;
    }
    ll tot_dis = 0, tot_tm = 0, tot_cst = 0;
    calc(route, tot_dis, tot_tm, tot_cst);
    cout << "Minimum Required time From: " << src << " to " << target << " is: " << tot_tm << nl;
    cout << "The route: ";
    print_route(route);
    cout << "Total Cost: " << tot_cst << nl;
    cout << "Total Distance: " << tot_dis << nl;
    cout << nl;
}
void minimum_cost()
{
    cout << "\n# Miminum Required Cost from source to target #" << nl;
    ll src, target;
    src = 1;
    target = 5;
    cout << "Enter your Source Node (Between 1 to " << N << " ): ";
    cin >> src;
    cout << "Enter your Target Node (Between 1 to " << N << " ): ";
    cin >> target;
    ll req = -1;
    vl route = dijstra(src, target, req, cst);
    if (req == -1)
    {
        cout << "No Path Exixts" << nl;
        return;
    }
    ll tot_dis = 0, tot_tm = 0, tot_cst = 0;
    calc(route, tot_dis, tot_tm, tot_cst);
    cout << "Minimum Required Cost From: " << src << " to " << target << " is: " << tot_cst << nl;
    cout << "The route: ";
    print_route(route);
    cout << "Total Reqired: " << tot_tm << nl;
    cout << "Total Distance: " << tot_dis << nl;
    cout << nl;
}
int main()
{
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    initial();
    print();
    ll inp = 5;
    while (inp != 4)
    {
```

```
        cout << "#######################################################" << nl;
        cout << "1 -> Minimum Distance from source to target." << nl;
        cout << "2 -> Miminum Required Time from source to target." << nl;
        cout << "3 -> Miminum Required Cost from source to target." << nl;
        cout << "4 -> EXIT." << nl;
        cout << "*Enter Choice: ";
        cin >> inp;
        if (inp == 1)
            shortest_distance();
        else if (inp == 2)
            minimum_time();
        else if (inp == 3)
            minimum_cost();
        else if (inp == 4)
            exit(1);
        else
            cout << "Wrong Input" << nl;
    }
}
```