CS141, Programming V Report **Due: 23:59, Tue 11/22, 2022**

Name: Majd Kawak / NetID: mkawa025 / CodeForces ID: mkawa025 / Student ID: 862273310

A. Single Source Shortest Paths

Submission#: 181838976

In this program, firstly I declared an *int* that would hold number of cities is in the map(graph), an *int* for roads(edges) number, an *int* for city s (Source vertex), an *int* for city t (Destination vertex), and lastly a $2D$ *vector* (Matrix) to model our map(Vertices and Edges).

After inputing all the above, I declare 2 $vectors$; one to hold distance value from city s to all other cities in map and the other to check if a city $visited$, both with size num_Cities and initialize all its elements to INT_MAX and $false$ respectively (Because distance is unknown at this point).

Using a for loop, I iterate through all cities (other than s) computing tentative distance using Dijkstra's Algorithm. Extracting city with minimum distance from current(using another for loop) at each iteration then visiting that city, saving tentative distance to my $distance\ vector$. Lastly, after iterating through all cities I end up with *distance vector* holding distance value at each *index* representing minimum time to travel from city s to city i at index i. Display value at index t.

I implemented Dijkstra's Algorithm using *priority queue*, Total time complexity $= O(n^2)$.

B. Minimum Spanning Tree

Submission#: 181837325

In this program, firstly I declared an *int* that would hold number of devices to connect in LAN(graph), an *int* for connections(edges) number, an *int* for minimum cost to connect all devices *min_Cost* and a *pair* of 3 *ints* to represent edges $< connection\_Cost, device_1, device_2 >$.

After inputing all the above, I *sort* all edges based on their *connection_Cost* from cheapest to most costly. Then I declare a *vector root* to hold each device(vertex) parent. At first, all devices root are initialized to itself (Meaning root of device i when 3 is 3).

Computing *min_Cost* using Kruskal's Algorithm. Using a for loop, I iterate through all connections starting from cheapest to most costly. At each iteration I check if parent of device u is the same for v using *parent* function, if not; I do *union_Set* operation for u and v update *root* vector, add *connection_Cost* edge to *min_Cost* and continue.

Lastly, after iterating through all connection edges I end up with *min_Cost* representing minimum value to connect all devices together.

I implemented Kruskal's Algorithm using $union - find$, Total time complexity $= O(n^2 + m\ log\ n)$.

D. Add Oil

Submission#: 181837983

This Program is exactly the same as B with 1 line changed

In this program, firstly I declared an *int* that would hold number of cities is in the map(graph), an *int* for roads(edges) number, an *int* for minimum gas tank capacity *min_Capacity* and a *pair* of 3 *ints* to represent edges $< value\_Distance, city_1, city_2 >$.

After inputing all the above, I *sort* all road based on their *value_Distance* from shortest to largest. Then I declare a *vector root* to hold each city(vertex) parent. At first, all cities root are initialized to itself (Meaning root of city i when 3 is 3).

Computing *min_Capacity* using Kruskal's Algorithm. Using a for loop, I iterate through all roads starting from shortest to largest. At each iteration I check if parent of city u is the same for city v using *parent* function, if not; I do *union_Set* operation for u and v update *root* vector, assign *route_Distance* to *min_Capacity* and continue.

Lastly, after iterating through all roads(edges) I end up with *min_Capacity* pointing to last route picked from roads to span our map cities, this would represent minimum gas tank capacity to travel form city x to y which is the longest route.

I implemented Kruskal's Algorithm using $union\_find$, Total time complexity = $O(n^2 + m\ log\ n)$.

References

[1] geeksforgeeks *Dijkstra's Shortest Path Algorithm* `https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/`

[2] geeksforgeeks *Kruskal's Algorithm (Simple Implementation for Adjacency Matrix)* `https://www.geeksforgeeks.org/kruskals-algorithm-simple-implementation-for-adjacency-matrix/`