Mini-Project

Team 19

International institute of Information Technology

Data Structures and Algorithms – CS1.201

Team members:

Srujana Vanka (2020102005)

Shreyash Jain (2020101006)

Karthik Prasanna N (2020115007)

Gorre Sashidhar Reddy (20171113)

Souvik Karfa (2020102051)

ROUTING

Aim: Improving Google Maps shortest path algorithm by implementing Dijkstra Algorithm with revised weightages to consider indeterminable events like marriage processions while travelling.

Data Structures:

- Array
- Structs: Used to store the data related to Graph, Vertex and Edges.
- Linked lists
- Adjacency List: An array of linked lists to store the outgoing edges of each vertex.
- Min-heaps
- Priority Queue using min-heaps
- Implemented Graph.

Algorithms:

- getmap() function: complexity of O($V + E$)
- CreateGraph() function: complexity of O(V)
- Insert Edge implemented with time complexity of O(1)
- Delete Edge implemented with time complexity of O(E)
- Dijkstra Algorithm: complexity of O($(V + E)logV$)
- GetWeight() function: complexity of O(1)
- PrintPath() function: complexity of O(V)
- BuildHeap() function: complexity of O(V)
- DownHeap() function: complexity of O($logV$)
- ExtractMin() function: complexity of O($logV$), as it uses DownHeap() function to again heapify the min-heap
- RelaxEdge() function: complexity of O($logV$), as this also uses DownHeap() function to heapify the min-heap

General Idea:

We read the inputs from a file using the getmap() function and implement a weighted directed graph such that the intersections as nodes(V) of the graph and the roads as edges(E) of the graph.
In our case the edge weights are time and are not fixed but depends on the current time. The weight for each edge is calculated whenever required using the GetWeight() function.
Then we used the Dijkstra Algorithm to find the quickest path. Since our edge weights are in time, we get the path with shortest time to travel from source to destination.

Datasets:

- Intersections (nodes)
- Roads (edges)
- Marriage hall data at each node
- Length of the road
- Traffic Density of the road
- Speed limits (maximum speed to travel on a particular road)

Calculating weights of edges:

The general idea of calculating the edge weights is to increase the usual traffic density of a road by a certain proportion, that includes the traffic congestion due to marriage processions

$$T_N = T_u + V_c \left(\frac{t_S}{|E_S|} + K \right) L_f$$

$$weight\ (time) = \frac{D_E T_N}{S_L T_u} + C_c \frac{\sum_D incT_i}{P}$$

- $\boldsymbol{T_u}$ = Usual traffic density of a road
- $\boldsymbol{t_S}$ = Number of marriage halls at current node
- $\boldsymbol{|E_S|}$ = Number of outgoing edges at current node
- $\boldsymbol{K}$ = a constant < 1, that makes sure that every road has a little possibility of a marriage procession going on even if $t_S = 0$
- $\boldsymbol{L_f}$ = a factor that depends on the time of crossing the node.
- $\boldsymbol{V_c}$ = A constant that gives dimension to the dimensionless quantity
- $\boldsymbol{T_N}$ = New traffic density
- $\boldsymbol{\sum_D incT_i}$ = Total incoming traffic density at the destination node
- $\boldsymbol{P}$ = A constant that takes into account that only a certain proportion of incoming traffic can enter the road
- $\boldsymbol{C_c}$ = A time constant that denotes the average time taken to cross a intersection

Explanation:

Say if there is a marriage happening at a node, then it may exit through any outgoing edge with a probability of $\frac{\mathbf{1}}{|\boldsymbol{E_S}|}$ for each edge. So, if there are $\boldsymbol{t_S}$ marriage halls at each node, then assuming there is a marriage happening at each hall, the probability of a marriage procession happening through each outgoing edge becomes $\frac{\boldsymbol{t_S}}{|\boldsymbol{E_S}|}$. We add a constant $\boldsymbol{K}$ to take into account of a marriage procession incoming from another vertex.

This is value is multiplied with a factor $\boldsymbol{L_f}$. $\boldsymbol{L_f}$ takes into account of the fact that the probability of a marriage procession happening will be much higher during certain hours of the day, as compared to the rest of the day.

$\boldsymbol{V_c}$ increases the weight of the value and gives unit to the value. It is a general estimate of the excess traffic density only due to a marriage procession.

By adding $\boldsymbol{V_c}\left(\frac{\boldsymbol{t_S}}{|\boldsymbol{E_S}|} + \boldsymbol{K}\right)\boldsymbol{L_f}$ to the usual traffic density of the road $\boldsymbol{T_u}$ we get our new traffic density $\boldsymbol{T_N}$.

As we can guess, if the traffic density is more on a road, then the travelling speed on that road will decrease. And considering we can travel at maximum with the speed limit with usual traffic density of the road.

$$T_u \propto \frac{1}{s_L}$$

$$\text{Therefore,} \quad s_T = \frac{S_2 T_u}{T_N}, \quad s_T = \text{travelling speed}$$

We can say that the time to cover the road = $\frac{D_E}{S_T}$

We also have to also consider the time taken to pass the intersection at destination vertex. Assuming that the time to pass through each intersection will be proportional to the total incoming traffic at the node, we get

$$\text{Time to pass a intersection} \propto \sum_D incT_i$$

Only a certain fraction $\boldsymbol{P}$ of the total incoming traffic will enter the intersection.

$$\text{Time to pass a intersection} = C_c \frac{\sum_D incT_i}{P}$$

We add the time to cross the road and the time to cross the destination vertex to calculate the final time to cross each edge.

Work Distribution:

We conducted frequent meets to come up with a solution to improve google shortest path and to calculate weights and finally we have agreed on using time as weights and come up with the above formulae to calculate weights.

- **Souvik:** Worked on the weights function, Dijkstra Algorithm, and debugging
- **Karthik:** Worked on the creating graph, Dijkstra Algorithm, and debugging
- **Shreyash:** Worked on main function, min heaps, and debugging
- **Srujana:** Implemented getmap() function, input text file, min heaps, and debugging
- **Shashidhar:** Worked on implementing functions related to Graph and debugging