Data Structures & Algorithms
Mini Project
Routing

Data Structures used

Structures

In our code, we have used the following structs:-

```
typedef struct vertex
{
        char vertexName[50];
        int vertexId;

} vertex;
```

CONTENT	INFORMATION
VERTEX NAME	Name given to a vertex.
VERTEX ID	Special integer ID for a given vertex.

```
typedef struct Graph{
        int numVertices; //Number of vertices
        struct ListNode  adjacencyList;
        struct ListNode path;
}Graph;
```

This struct is used for represent the city map.

CONTENT	**INFORMATION**
NUMBER OF VERTICES	Number of vertices in the graph.
ADJACENCY LIST	Contains information about linkage of nodes in the graph.
PATH	To print the shortest path (used in the dijkstra algorithm)

```
struct StreetData{
        int length;
        int traffic;
        int numLanes;
        int num_cars;/
        int num_accidents;
        int speed_limit;/
        int weight;
        int safety_value;
};
```

This is structure is used to store the data/parameters about every street in the city map.

CONTENT	INFORMATION
LENGTH	Gives us information about length of the route.
TRAFFIC	Value of average traffic for a given time period.
NUMBER OF LANES	Number of lanes present on the path between two travelling nodes. Range 1-4
NUMBER OF CARS	The average number of cars present on that road. Range 10-300
NUMBER OF ACCIDENTS	Number of accidents that took place during a time interval. Range 1-10
SPEED LIMIT	Represents the max. speed at which user can drive. Range 10-100
WEIGHT	Mathematical function designed to differentiate between more time taking route and less time taking route, taking all factors into account. It also gives the final weightage/congestion index of the street.
SAFETY VALUE	Mathematical function designed for calculating the safety index for a given route. Based on the probability of an accident.

```
typedef struct ListNode{
        vertex dest;
        struct ListNode* next;
        struct StreetData SD;
}ListNode;
```

CONTENT	**INFORMATION**
DEST	A structure containing name and ID of the node.
NEXT	Contains address of the next node.
STREETDATA SD	Contains detailed information about the street.

Linked list:-

We have used the linked list structure in creating the adjacency list to represent the city map.
Adjacency list is a array of pointers to “listNodes”.
It contains information about linking of different vertices in the map.
The index of the array(adjacency List) represents a vertex and each element in its linked list represents the other vertices that form an edge with the vertex.

Queue:-

In a Queue data structure, we maintain two pointers, front and rear. The front points the first item of queue and rear points to last item.
In our program, we have used queues in the dijkstra algorithm.

Graph:-

We are storing the city map as a graph where each node represents a place and each edge represents the street between two nodes using adjacency list representation.

Algorithms used and their complexity

Dijkstra Algorithm

Dijkstra algorithm is a very famous greedy algorithm.
It is used for solving the single source shortest path problem.
It computes the shortest path from one particular node to all other remaining nodes.
Dijkstra algorithm works only with connected graphs, directed/undirected edges with positive weights.
It provides the value or cost of the shortest paths and by making minor modifications in the actual algorithm, the shortest paths can be obtained.

In our program, we use the algorithm to find the fastest (with minimum congestion) and the safest (with maximum safety value and minimum probability of accidents) path between the source and the destination entered by the user.
The corresponding function gives us the congestion value, safety value and the complete path.

Working -
The dijkstra code we used in this program runs with queues. We run breadth first search and then enqueue every element into the queue and checked for their neighbours to find the least of them

and continued the process until the queue was empty and stored each vertex shortest paths in the path list and finally printed the path list for the shortest path between source and destination entered by the user.

The time complexity for the Dijkstra algorithm is O(NE)
N-Number of nodes/places
E-Number of Edges

Other major functions and their complexity

- CreateGraph - Initialises the adjacency List. O(n)
- AddStreet - Adds a node on the adjacency list (adds a street in the city map) Complexity can vary from O(1) to O(E)
- Update Street - updates the data in a particular node of the adjacency list. Complexity can vary from O(1) to O(E)
- Delete Street - deletes a particular node in the adjacency list Complexity can vary from O(1) to O(E).

Division of work

Harshita Gupta :- Worked on the user interface, major part of the main.c, wrote functions like createGraph, addStreet, testing and debugging, readme file, deciding street parameters.

Sarthak Aggarwal:- Deciding the parameters, formulating definitions of congestion as well as safe routing, writing report and readme files.

Ayush Agrawal:- Worked on the update street, delete street functions, User Interface, test-cases, testing and debugging.

Polavarapu Neeraj:- Wrote the shortest path function (Dijkstra Algorithm) ,wrote functions for queues(init queue,enqueue,dequeue) and Worked on creating test-cases and graphs, debugging.

Samarth Pandey:- made the .h file forming structs required and deciding function input parameters,forming mathematical definitions for program ,worked on readme and report files.

OUR GITHUB REPOSITORY LINK:-
https://github.com/ayushagr2002/DSA_MiniProjectTeam15/tree/main