ENGINEERING METHOD

PARTICIPANTS

CAMILO CAMPAZ JIMÉNEZ

DANIEL ESTEBAN JARABA GAVIRIA

JOHAN STIVEN RICARDO SIBAJA

TEACHER

URAM ANIBAL SOSA

ENGINEERING METHOD

PROBLEM CONTEXT:

ICESI university students have noticed that lately public transport takes a long time to get from their homes to the university or to any point in the city, some think that this is because they only know a few routes that make the travel they need.

For this reason, one of them proposes to seek help to solve this situation in the best way without this implying spending more money than they already spend so far, for this they contact a friend who is studying systems engineering and he he asks if there is any way to find out the most convenient MIO route for each of his friends in terms of taking less time to travel from their homes to where they want; his friend tells him that he is coincidentally working on an application that seeks precisely that, that given a station or initial stop he can know which buses or routes he should take to get there in the fastest way even taking into account possible traffic delays in different times of the day depending on the time and dates.

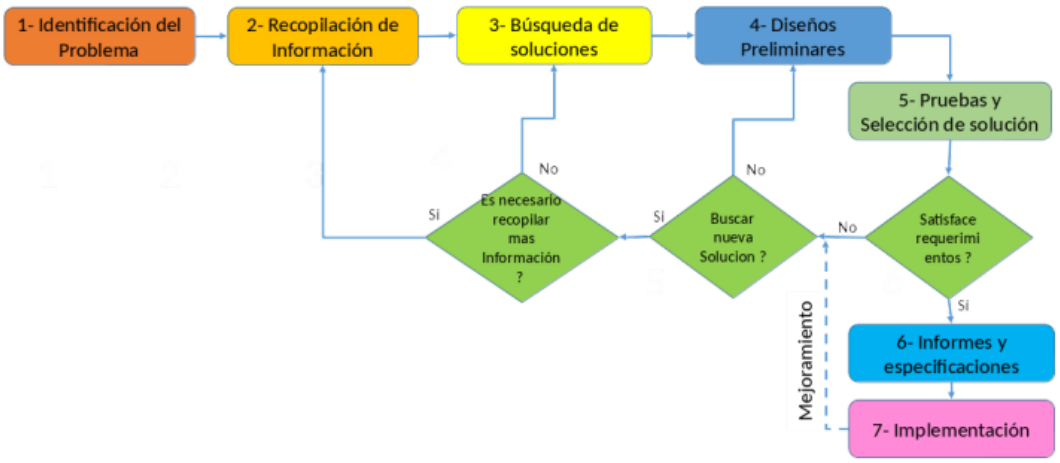
By getting down to work, the student decides to help get as many routes and stops so that his engineer friend can model in the best way the application that will make life easier for him and his other friends in a certain way.

SOLUTION DEVELOPMENT:

To solve the previous situation, the Engineering Method was chosen to develop the solution following a systematic approach and in accordance with the problematic situation posed.

Based on the description of the Engineering Method in the book “Introduction to Engineering” by Paul Wright.

The following flow chart was defined, the steps of which we will follow in the development of the solution.



IDENTIFICATION OF THE PROBLEM:

In this section, it is important to have a good definition of what the solution to the problem that is being addressed is necessary to contain.

Identification of needs:

- Tool which allows to simulate the operation of the buses of the Mass Transportation System for passengers operated by articulated buses.
- The tool must be able to receive test data to make the respective demonstration of the operation of the service taking into account random external factors that may affect it.
- The tool needs to provide information about the best route from any starting point.
- The tool must use in its system an algorithm capable of determining the best path from one station to any adjacent station.

Identification of the problem:

The students require the development of a software module able to determine the best path from one station to any station, that is, it is not expensive in terms of travel time.

INFORMATION GATHERING:

In this section the concepts that are essential for understanding the solution of the specific problem are specified.

Algorithm:

An ordered set of systematic operations that allows making a calculation and finding the solution to a type of problem.

Data structures:

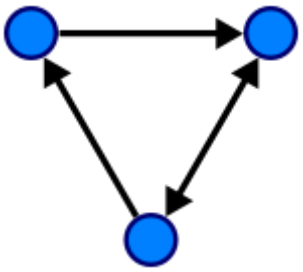
In computer science, a data structure is a particular way of organizing data in a computer so that it can be used efficiently. Different types of data structures are suitable for different types of applications, and some are highly specialized for specific tasks.

Graph:

A graph is an abstract data type that is meant to implement the undirected graph and directed graph concepts from the field of graph theory within mathematics.

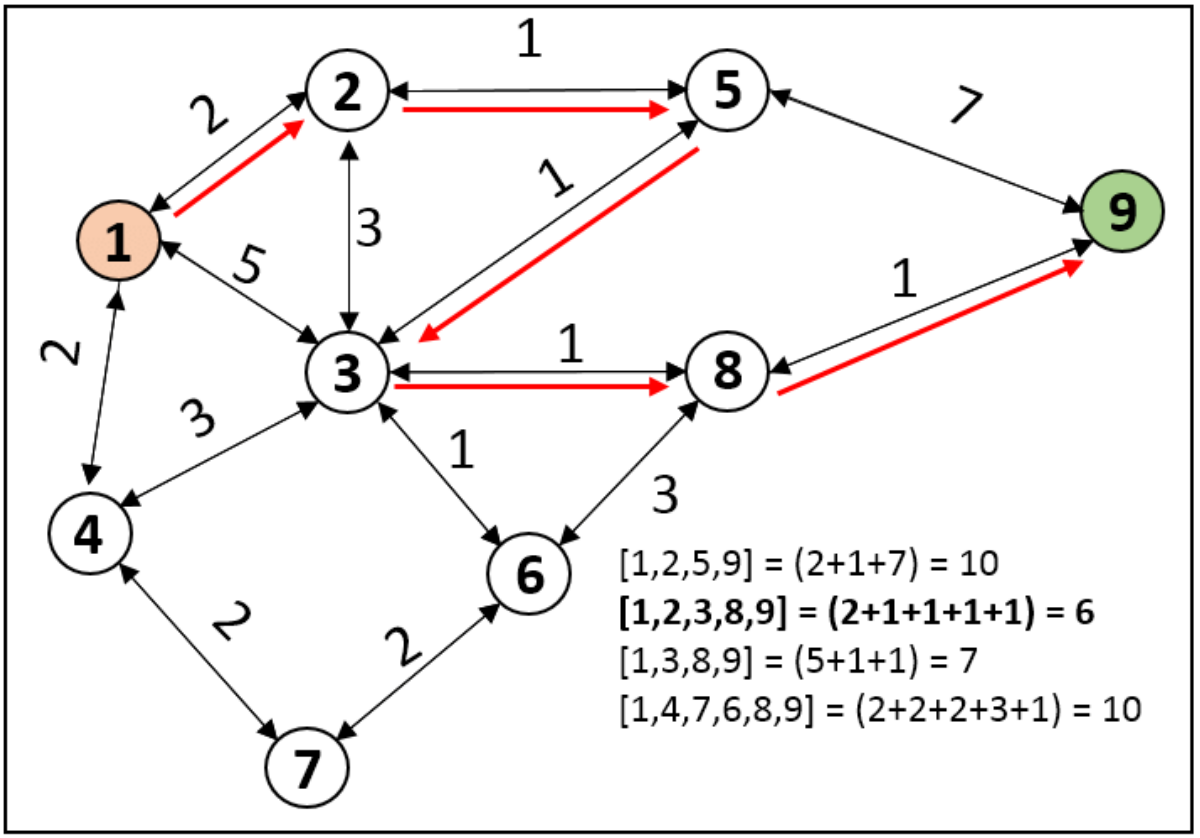
A graph data structure consists of a finite (and possibly mutable) set of vertices (also called nodes or points), together with a set of unordered pairs of these vertices for an undirected graph or a set of ordered pairs for a directed graph. These pairs are known as edges (also called links or lines), and for a directed graph are also known as edges but also sometimes arrows or arcs. The vertices may be part of the graph structure, or may be external entities represented by integer indices or references.

A graph data structure may also associate to each edge some edge value, such as a symbolic label or a numeric attribute (cost, capacity, length, etc.)



Dijkstra's shortest path algorithm

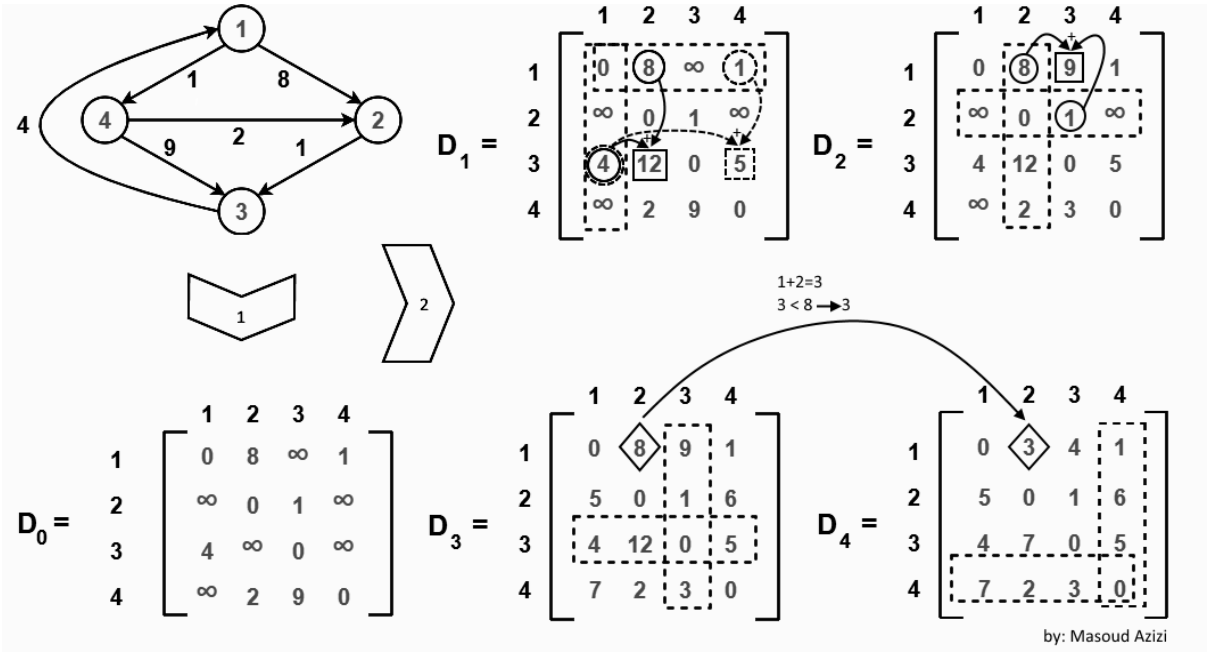
Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a SPT (shortest path tree) with a given source as a root. We maintain two sets, one set contains vertices included in the shortest-path tree, other set includes vertices not yet included in the shortest-path tree. At every step of the algorithm, we find a vertex that is in the other set (set of not yet included) and has a minimum distance from the source.



Floyd–Warshall algorithm

The Floyd–Warshall algorithm (also known as Floyd's algorithm, the Roy–Warshall algorithm, the Roy–Floyd algorithm, or the WFI algorithm) is an algorithm for finding shortest paths in a directed weighted graph with positive or negative edge weights (but with no negative cycles) A single execution of the algorithm will find the lengths (summed

weights) of shortest paths between all pairs of vertices. Although it does not return details of the paths themselves, it is possible to reconstruct the paths with simple modifications to the algorithm.



MIO

The Masivo Integrado de Occidente (MIO) is the integrated mass transportation system (SITM) of the Colombian city of Cali. The system is operated by articulated, standard and complementary buses, which move through trunk, pre-trunk and complementary corridors covering trunk, pre-trunk and feeder routes. It was inaugurated on November 15, 2008 in a test phase. As of March 1, 2009, it began operating firmly.

SEARCH FOR CREATIVE SOLUTIONS

This section is going to introduce the ideas that can be optimal to solve the problem that is proposed.

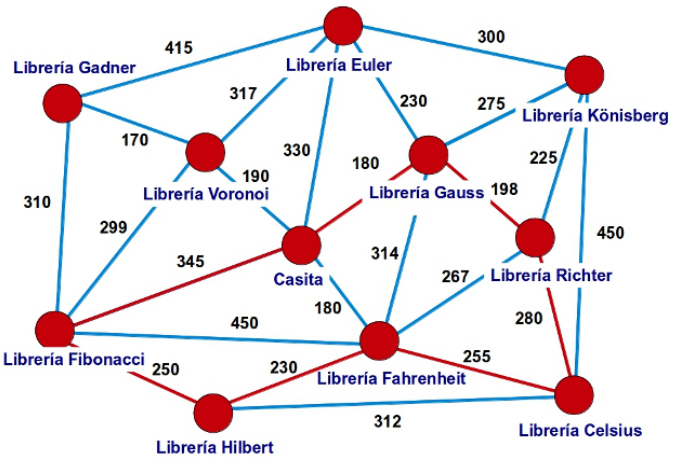
Proposal 1: Simulation of the MIO routes and the cost of each travel using graphs

One way to find an optimal solution for this problem is to simulate the context of the MIO stations and the distance from one station to another taking into account that the traffic could increase the cost of that route (when we talk about cost it is in terms of time of traveling). Graphs will represent a good relation between the stations and the cost of the travel, because all the stations are connected by a specific route and it can be seen as vertices and edges of a graph.

Proposal 2; Notification of the first route using google maps

This proposal seeks to use the tool of google maps in order to get information of the MIO routes provided by the app, then we can manage the information and notify the user of the first route found to get to his destination.

TRANSITION FROM FORMULATION OF IDEAS TO PRELIMINARY DESIGNS



In this case it is considered that the best option is the proposal 1, however it is important to see why the other proposal is not that efficient.

As we can see, the principal difference between the proposals are the cost of the travels (in terms of time) that both solutions imply are quite different and the gap between the results that both solutions clearly show which proposal is the most effective. This is so, because the second proposal does not take into account the cost of moving from one route to another and ignoring that factor would cause us to find a very expensive route in terms of time, which would not be an appropriate solution to the problem we are facing. In this order of ideas, if we analyze proposal 1 it can be seen that a representation by means of graphs of the routes and stations can bring efficient solutions to the problem because if we model that context with graphs we can make use of algorithms that allow us to find the best way from one station to another, which would be the least expensive way in terms of time.

EVALUATION AND SELECTION OF THE BEST SOLUTION

Is time to select the final choice, so it is important to define evaluation criteria that let us prove that the solution we select is optimal (for us) for this problem, in particular thinking about it generally.

Criteria A: Solve the problem correctly

- [2] Yes
- [1] No

Criteria B: The solution provides routes that are inexpensive in terms of time(it means to don't recommend routes that will represent to the user a travel of a long time)

- [3] Yes
- [2] Sometimes
- [1] No

Criteria C: The solution takes into account external factors that may afect the cost of the travel

- [3] Yes
- [2] More or less
- [1] no

	Criteria A	***Criteria B***	***Criteria C***	***Total***
Proposal 1	2	3	3	8
Proposal 2	2	2	2	6

REPORT PREPARATION AND SPECIFICATIONS

Don't forget the limitations of the solution, so it is important to explain which are the limitations of the solution or considerations for the implementation.

In this case is needed show how will be the solution that is proposed

Considerations:

1. For the optimal work in the data structures it might be all the data in the correct format
2. All the data structures are might be defined properly
3. To create properly the objects all the data that is given might be correct

DESIGN IMPLEMENTATION

List of tasks to implement:

a. Create a graph
b. Calculate the shortest path using Dijkstra's algorithm
c. Calculate the shortest path using Floyd-Warshall's algorithm

Specifications and subroutines:

A:

Name:	Graph
Description:	Create a new Graph with its vertices and edges
Input:	- v: the number of vertices
Output:	

```
Graph g = new Graph(n);
```

B:

Name:	dijkstra
Description:	Calculates the shortest path from one vertex to others in a graph
Input:	- g: the graph - n: the size of the list of distances - s: the size of the list of predecessors
Output:	a list with of vertices that represent the shortest path from one vertex to others in the graph

```java
public static void dijkstra(Graph g, int n, int s) {
        int[] dist = new int[n];
        int[] prev = new int[n];
        Distance[] d = new Distance[n];
        PriorityQueue<Distance> q = new PriorityQueue(n, new DistanceComparator());
        dist[s] = 0;
        for (int i = 0; i < n; i++) {
            if (i != s) {
                dist[i] = Integer.MAX_VALUE;
            }
            prev[i] = -1;
            d[i] = new Distance(i, dist[i]);
            q.add(new Distance(i, dist[i]));
        }
        soutDist(dist);
        soutPrev(prev);
        System.out.println(Arrays.toString(q.toArray()) + "\n");

        while (!q.isEmpty()) {
            int u = q.remove().getI();
            for (int i = 0; i < g.getAdj()[u].size(); i++) {
                int alt = dist[u] + g.getAdj()[u].get(i).getW();
                if (alt < dist[g.getAdj()[u].get(i).getD()]) {
                    Object[] distAr = q.toArray();
                    Distance[] distArr = new Distance[q.size()];
                    for (int j = 0; j < distArr.length; j++) {
                        distArr[j] = (Distance) distAr[j];
                    }
                    Distance temp = new Distance(-1, -1);
                    for (int j = 0; j < distArr.length; j++) {
                        if (distArr[j].getI() == g.getAdj()[u].get(i).getD()) {
                            temp = distArr[j];
                        }
                    }
                    q.remove(temp);
                    dist[g.getAdj()[u].get(i).getD()] = alt;
                    prev[g.getAdj()[u].get(i).getD()] = u;
                    q.add(new Distance(g.getAdj()[u].get(i).getD(), alt));
                }
            }

            soutDist(dist);
            soutPrev(prev);
            System.out.println(Arrays.toString(q.toArray()) + "\n");
        }
    }
```

C:

Name:	floydWarshall
Description:	Calculates the shortest for each pair of vertices of a graph
Input:	- g: the graph - n: the size of the matrix of the weight matrix
Output:	a matrix with the shortest for each pair of vertices of the graph

```java
public static void floydWarshall(Graph g, int n) {
       int[][] arr = new int[n][n];
       for (int i = 0; i < arr.length; i++) {
           for (int j = 0; j < arr.length; j++) {
               arr[i][j] = Integer.MAX_VALUE;
           }
       }

       for (int i = 0; i < arr.length; i++) {
           for (int j = 0; j < arr.length; j++) {
               if (i == j) {
                   arr[i][j] = 0;
               }
           }
       }

       LinkedList<Edge> adj[] = g.getAdj();
       for (int i = 0; i < adj.length; i++) {
           for (int j = 0; j < adj[i].size(); j++) {
               arr[adj[i].get(j).getS()][adj[i].get(j).getD()] = adj[i].get(j).getW();
           }
       }
       soutMatrix(arr);

       for (int k = 0; k < arr.length; k++) {
           for (int i = 0; i < arr.length; i++) {
               for (int j = 0; j < arr.length; j++) {
                   if (arr[i][j] > (arr[i][k] + arr[k][j]) && ((arr[i][k]) != Integer.MAX_VALUE && ((
arr[k][j]) != Integer.MAX_VALUE))) {
                       arr[i][j] = (arr[i][k] + arr[k][j]);
                   }
               }
           }
           soutMatrix(arr);
       }
   }
```