Computación y Estructuras Discretas I

Santiago José Barraza Sinning
Jhonatan Castaño
Sebastián López García

Método de Ingeniería

Rutas Más Rápidas del Sistema de Transporte Masivo (MIO)

20 de septiembre de 2022

Problemática

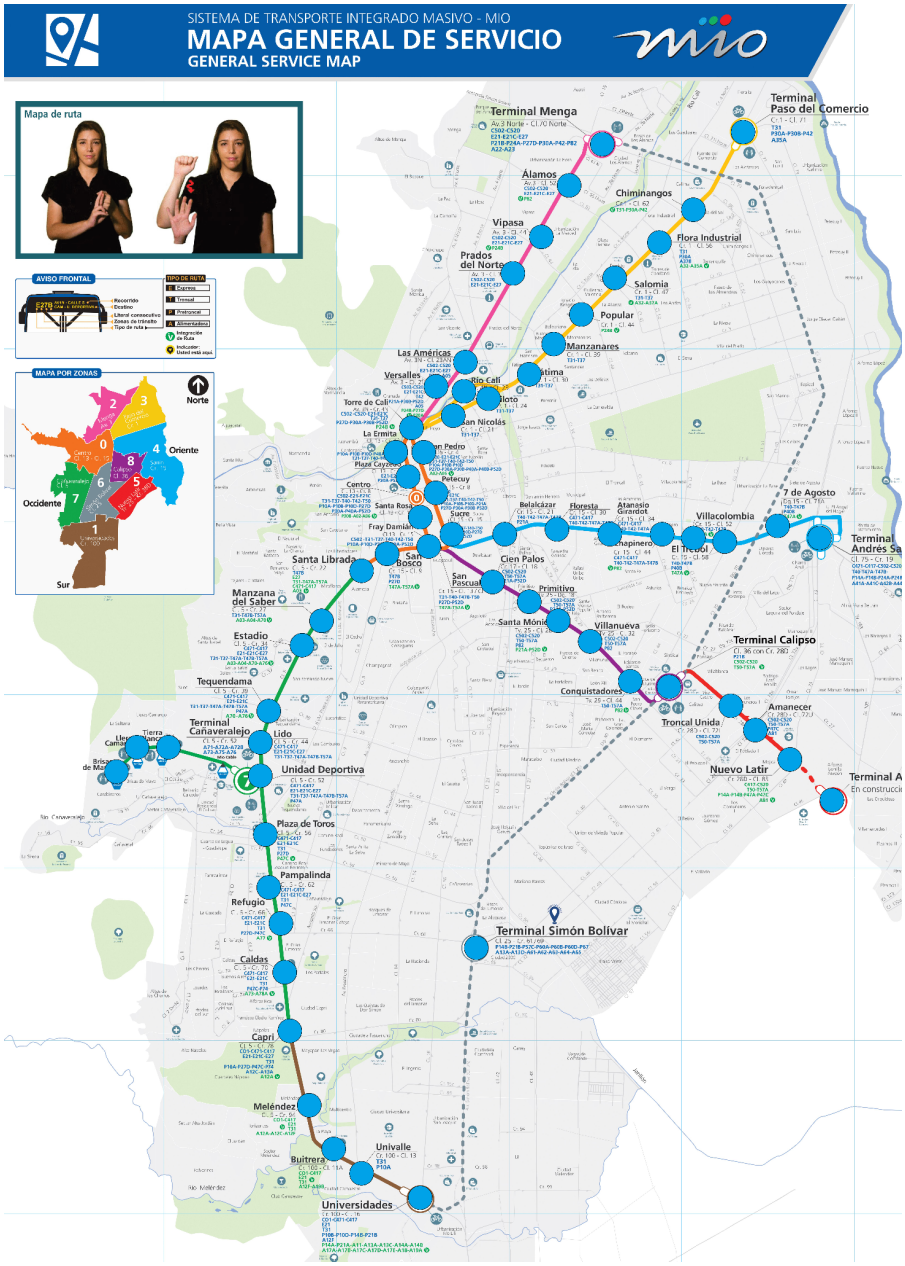
Algunas personas se han dado cuenta que tardan mucho en llegar a su destino usando el sistema de transporte masivo de la ciudad de Cali, esto porque no saben qué ruta deben usar exactamente, por lo que usualmente escogen la ruta equivocada, lo que hace que el tiempo del trayecto aumente.

Se contratan tres ingenieros para crear un sistema que permita al usuario del transporte conocer cuál es la ruta más eficiente para llegar a su destino desde la estación en la que se encuentren.

Propuesta de solución

Los tres ingenieros llegan a un acuerdo para la solución que van a construir, la cuál se va a componer de grafos, con el uso de vértices representando las estaciones, unidos por aristas que representan el tiempo que tarda el transporte entre dos estaciones. Para esto deberán implementar las estructuras del grafo y métodos de camino mínimo.

Ilustración (Un vértice por estación)

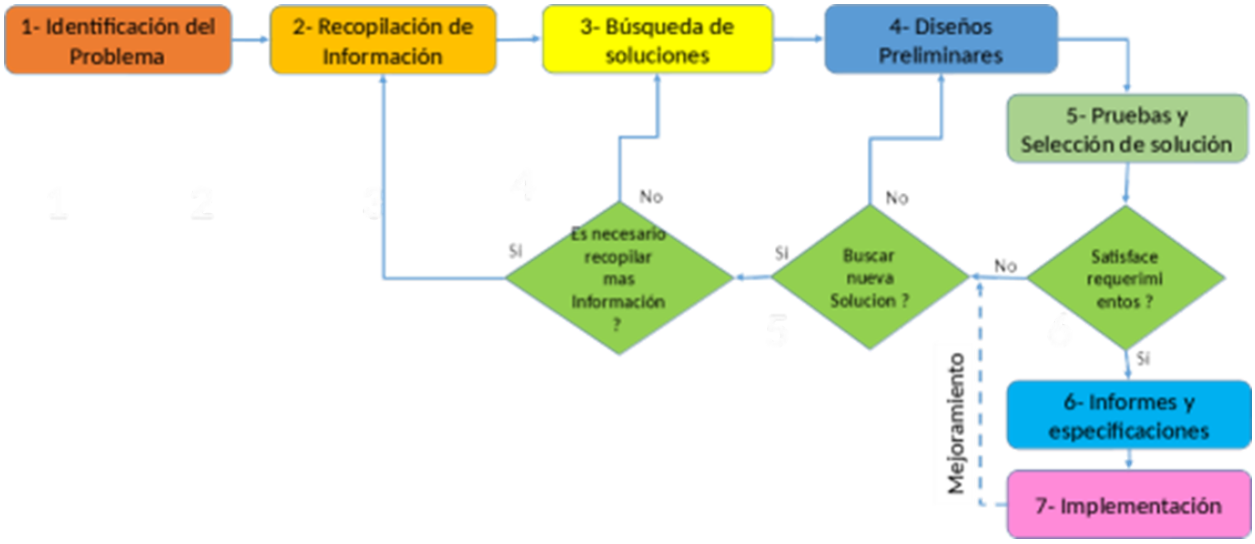


Tomado de [Rutas (mio.com.co)]

Desarrollo de la solución

A lo largo del presente documento se evidenciará el uso del método de ingeniería para resolver de la mejor forma el problema que se busca solucionar.

Con base en la descripción del Método de la Ingeniería del libro “Introduction to Engineering” de Paul Wright, el método se llevará a cabo de la siguiente forma dividido en 7 fases.



1. Identificación del problema

En este paso debemos identificar cuales son los requerimientos del programa, describir cada uno de ellos, analizar cuales son las entradas que necesitan y la salida que otorgan.

Los requerimientos son los siguientes:

Nombre o identificador	R1-Permitir al usuario ver las rutas mínimas que debe utilizar para llegar a su destino.		
Resumen	El sistema permite al usuario ingresar la estación en la que se encuentra, y la estación a la que quiere dirigirse, luego muestra las rutas del MIO que tiene que usar para llegar más rápido a su destino.		
Entradas	Nombre entrada	Tipo de dato	Condición de selección o repetición
	PuntoOrigen	String	Que la estación se encuentre en la base de datos de la aplicación.
	PuntoDestino	String	Que la estación se encuentre en la base de datos de la aplicación.
Actividades generales necesarias para obtener los resultados	Ingresar al apartado de Buscar ruta más eficiente.		
Resultado o postcondición	Se le muestra al usuario las rutas o ruta que debe tomar, y todas las paradas.		
Salidas	Nombre entrada	Tipo de dato	Condición de selección o repetición
	ruta	String	

Nombre o identificador	R2-Permitir al usuario ver las rutas que pasan por una estación.		
Resumen	Permite al usuario buscar una estación del MIO, y revisar que rutas del MIO pasan por esa estación.		
Entradas	Nombre entrada	Tipo de dato	Condición de selección o repetición
	EstacionMIO	String	Que la estación se encuentre en la base de datos de la aplicación.
Actividades generales necesarias para obtener los resultados	Ingresar al apartado de ver rutas en estación MIO.		
Resultado o postcondición	Lista de todas las rutas que pasan por la estación.		
Salidas	Nombre entrada	Tipo de dato	Condición de selección o repetición
	Rutas	String	

Nombre o identificador	R3-Permitir al usuario ver las paradas activas en ese momento de una ruta en especifico.		
Resumen	Permite al usuario buscar una ruta especifica en el sistema para conocer las paradas que esta realiza.		
Entradas	Nombre entrada	Tipo de dato	Condición de selección o repetición
	Ruta	String	La ruta se encuentre en la base de datos.
Actividades generales necesarias para obtener los resultados	Ingresar al apartado de paradas del MIO.		
Resultado o postcondición	Listado de paradas de esa ruta.		
Salidas	Nombre entrada	Tipo de dato	Condición de selección o repetición
	Paradas	String	

2. Recopilación de Información

La información que se recopiló fue con base a la página oficial del MIO, en la cual se encuentran los se encuentran los datos de las estaciones, sus conexiones con otras estaciones y las rutas que paran sobre estas.

3. Búsqueda de soluciones

Para cumplir con los requerimientos se optó por realizar algoritmos de camino mínimo y algoritmos de búsqueda dentro del grafo.

En un principio se planteó usar BFS para esto, sin embargo, no fue muy útil. Entonces lo que se decidió usar finalmente fueron los dos métodos siguientes:

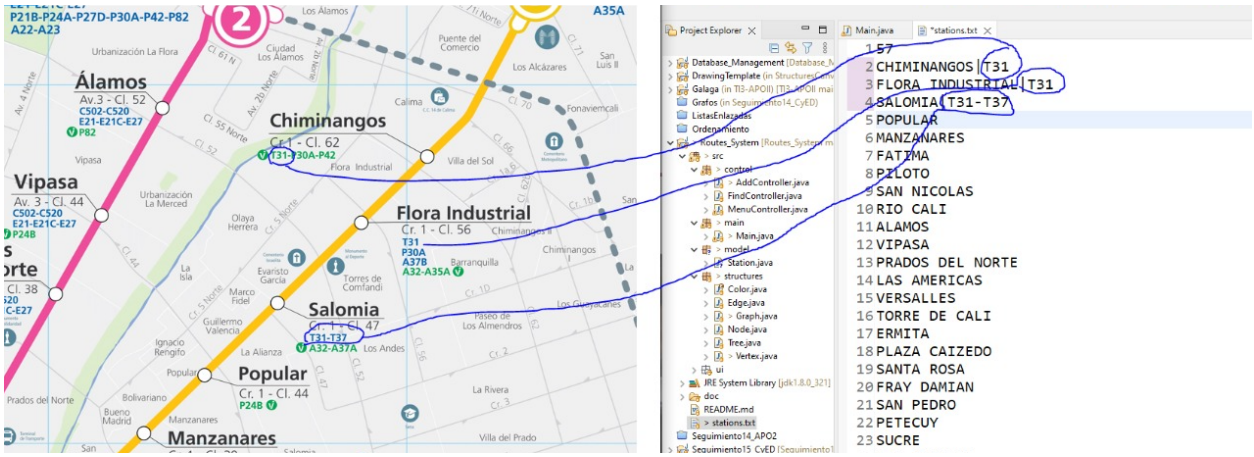
1. Dijkstra: Normalmente el algoritmo de dijkstra funciona para hallar el camino mínimo desde un vértice hacia todos los demás, sin embargo, el método fue modificado teniendo en cuenta las necesidades. En primer lugar, se modificó para que en lugar de calcular el camino mínimo desde un vértice hasta todos los demás, lo hiciera con un solo vértice de destino, esto para permitir hallar el camino mínimo entre estos. Por otro lado, se modificó también para que, al recorrer el recorrer árbol generador mínimo del vértice de origen, en lugar de guardar la distancia mínima al vértice de destino, guarde el los vértices por los que tuvo que pasar a través del camino mínimo para llegar a este.

2. Floyd-Warshall: El algoritmo de floyd warshall se usó debido a que a pesar de que se había guardado los vértices por los que pasamos, no guardamos la distancia recorrida, y como este método realiza una matriz de distancias, fue muy útil para poder mostrar la distancia recorrida al final del programa.

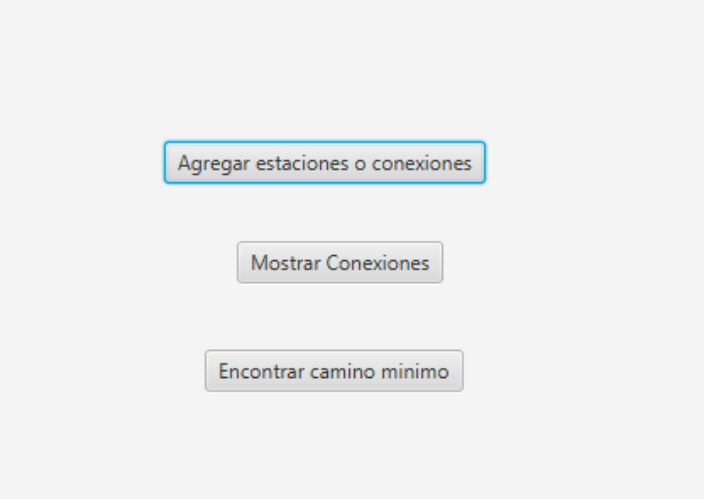
4. Diseños preliminares

En este caso, solo tuvimos dos alternativas en cuanto a los métodos de camino mínimo, del cual el uso del BFS, aunque pudo haber sido útil, fue descartado rápidamente, ya que se hizo más eficiente la alternativa de dijkstra y floyd-warshall.

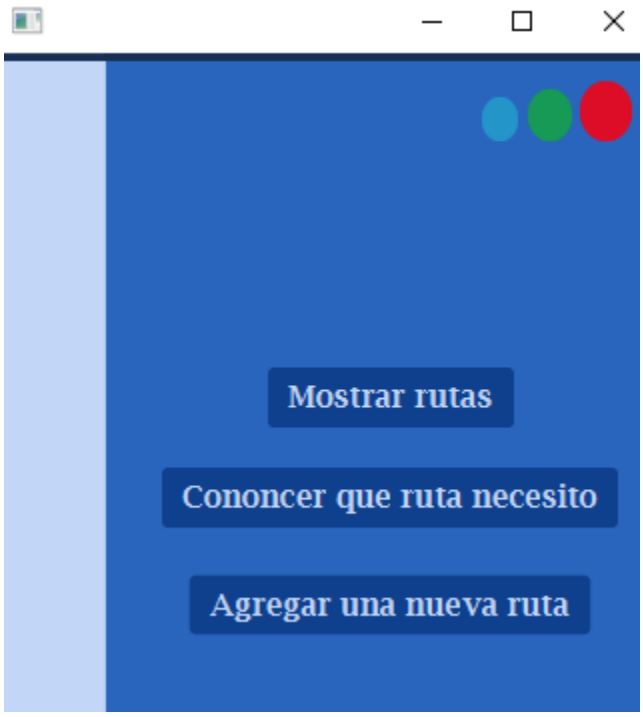
El diseño de la base de datos en la que se encuentran los datos de las rutas se hizo de la siguiente manera:



En cuanto a la interfaz gráfica, siempre hubo un diseño básico, el cual contiene tres opciones:



La interfaz evolucionó en la siguiente:



Además, las estaciones, al momento de conocer la ruta necesaria, ya están previamente incluidas en un choicebox, cuando anteriormente el usuario tenía que escribirlas.

5. Pruebas y selección de Solución

En este caso solo hubo una propuesta de solución, por lo que esta fue la seleccionada.

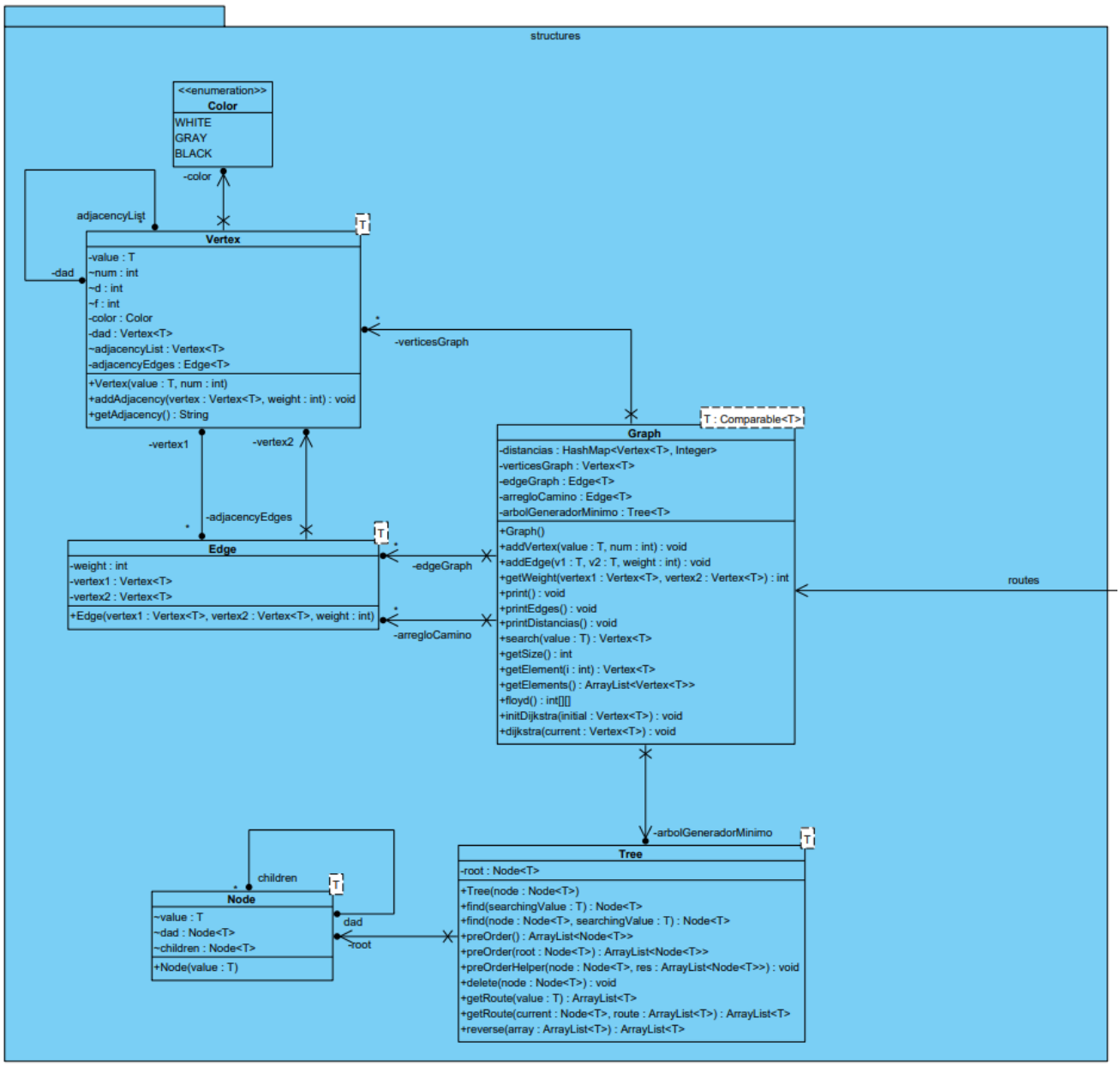
6. Informes y Especificaciones

Grafo

- Dos algoritmos (Floyd y Dijkstra)
- Implementación de grafo por lista de adyacencia

7. Implementación

Estructuras (véase más claro en el diagrama de clases).



Algoritmo dijkstra

```
public void dijkstra(Vertex<T> current) {
    for(int i=0;i<current.getAdjacencyList().size();i++) {
        Vertex<T> aux=current.getAdjacencyList().get(i);


        int distanciaActual=distancias.get(current);

        int weight=getWeight(current, aux);

        if((distanciaActual+weight)<distancias.get(aux)) {

            for(int j=0;j<current.getAdjacencyEdges().size();j++) {
                if(current.getAdjacencyEdges().get(j).getVertex2()==aux) {
                    arregloCamino.add(current.getAdjacencyEdges().get(j));
                    T nodeValue = current.getAdjacencyEdges().get(j).getVertex2().getValue();
                    Node<T> node = new Node<T>(nodeValue);

                    T dadValue = current.getAdjacencyEdges().get(j).getVertex1().getValue();

                    node.setDad(arbolGeneradorMinimo.find(dadValue));

                    arbolGeneradorMinimo.find(dadValue).getChildren().add(node);
                }
            }

            distancias.remove(aux);
            distancias.put(aux, distanciaActual+weight);
        }
    }
    current.setColor(Color.BLACK);

    for(int i=0;i<current.getAdjacencyList().size();i++) {
        if(current.getAdjacencyList().get(i).getColor()==Color.WHITE) {
            dijkstra(current.getAdjacencyList().get(i));
        }
    }

}
```

Algoritmo Floyd-Warshall

```
public int[][] floyd() {

    int dist[][];

    dist = new int[verticesGraph.size()][verticesGraph.size()];

    for(Vertex<T> v : this.getElements()) {
        v.setColor(Color.WHITE);
    }

    for(int i=0;i<verticesGraph.size();i++) {
        for(int j=0;j<verticesGraph.size();j++) {
            if(i!=j) {
                dist[i][j]=1000000000;
            }
        }
    }

    for(int i=0;i<edgeGraph.size();i++) {
        Vertex<T> aux1=edgeGraph.get(i).getVertex1();
        Vertex<T> aux2=edgeGraph.get(i).getVertex2();
        dist[aux1.getNum()][aux2.getNum()]=edgeGraph.get(i).getWeight();
        dist[aux2.getNum()][aux1.getNum()]=edgeGraph.get(i).getWeight();
    }

    for(int k=0;k<verticesGraph.size();k++) {
        for(int i=0;i<verticesGraph.size();i++) {
            for(int j=0;j<verticesGraph.size();j++) {
                if(dist[i][j]>dist[i][k]+dist[k][j]) {
                    dist[i][j]=dist[i][k]+dist[k][j];
                }
            }
        }
    }
    return dist;
}
```