Método de la Ingeniería
Aplicación para la Solución de un Problema
Carlos Jimmy Pantoja A00366287 - TI3

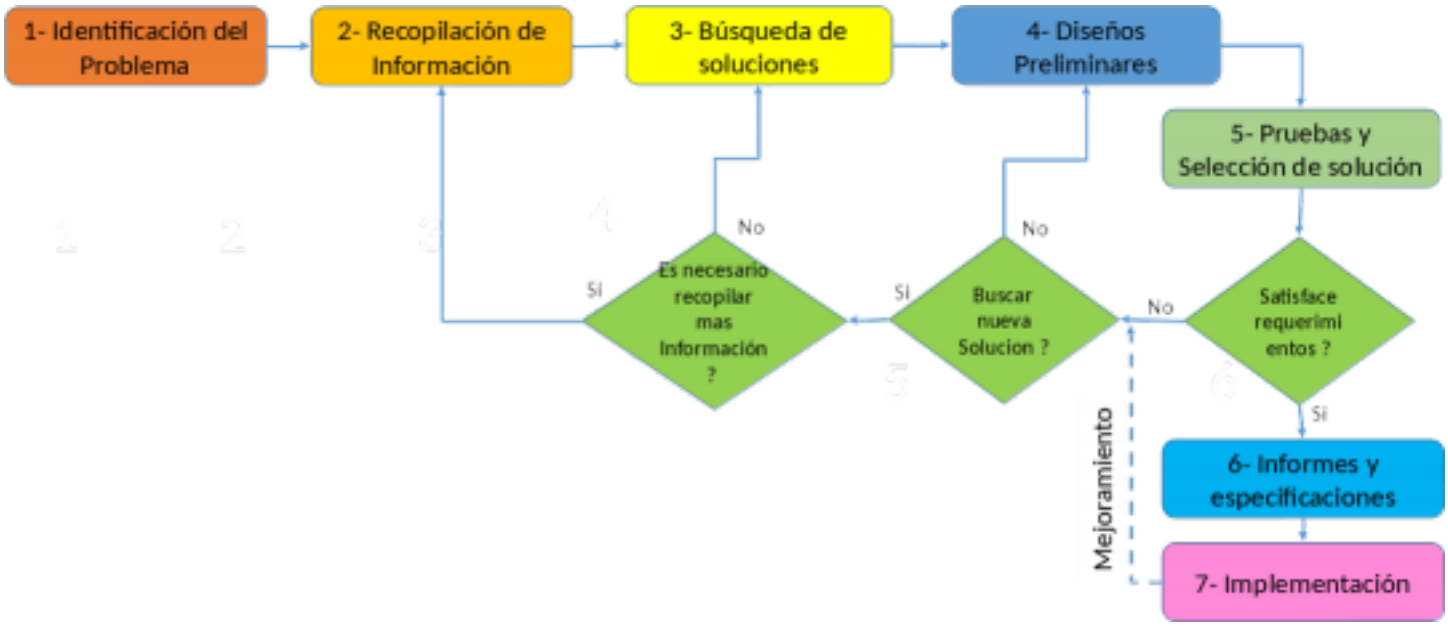
Contexto Problemático

El instituto nacional de vías INVIAS está actualizando sus plataformas web en las que desea ofrecer información útil a la hora de viajar entre ciudades en el territorio colombiano, requiere solucionar el problema más típico cuando se viaja ¿Cuál es el camino más corto?.

Desarrollo de la Solución

Para resolver la situación anterior se eligió el Método de la Ingeniería para desarrollar la solución siguiendo un enfoque sistemático y acorde con la situación problemática planteada.

Con base en la descripción del Método de la Ingeniería del libro “Introduction to Engineering” de Paul Wright, se definió el siguiente diagrama de flujo, cuyos pasos seguiremos en el desarrollo de la solución.



Paso 1. Identificación del Problema

Se reconocen de manera concreta las necesidades propias de la situación problemática así como sus síntomas y condiciones bajo las cuales debe ser resuelta.

Identificación de necesidades y síntomas

Los usuarios de los sitios webs desean conocer la ruta más rápida para viajar entre ciudades.
No existe una base de datos actualizada con todas las vías del país.
El usuario será capaz de añadir rutas a su preferencia sin mucho problema.
Se desea una solución rápida, sin tiempos de espera prolongados.

Definición del Problema

El instituto nacional de vías INVIAS requiere desarrollar un software capaz de calcular la ruta más corta entre dos ciudades.

Paso 2. Recopilación de Información

Con el objetivo de tener total claridad en los conceptos involucrados se hace una búsqueda de las definiciones de los términos más estrechamente relacionados con el problema planteado. Es importante realizar esta búsqueda en fuentes reconocidas y confiables para conocer cuáles elementos hacen parte del problema y cuáles no.

Definiciones

Fuente:

https://es.wikipedia.org

Ruta

Vía que conecta dos puntos (ciudades/pueblos/municipios), normalmente tienen un nombre o placa.

Millas

Indican la extensión de la ruta.

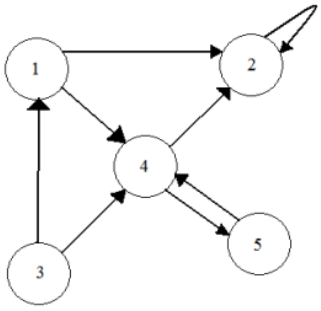
Ciudad

Asentamiento de personas.

Paso 3. Búsqueda de Soluciones Creativas

Para este paso, aunque podemos pensar en soluciones propias, y como resulta que el problema es uno clásico de programación, haremos uso de los algoritmos más conocidos.

Alternativa 1. Algoritmo de Floyd-Warshall



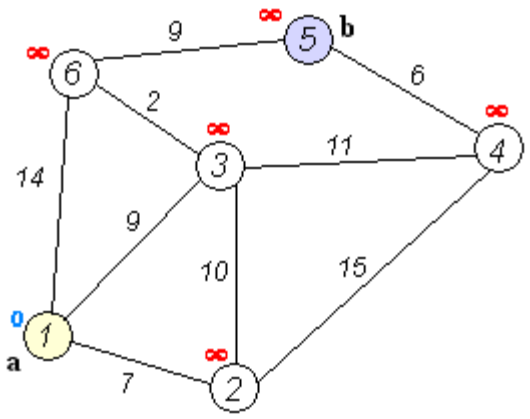
	1	2	3	4	5
1	False	True	False	True	False
2	False	True	False	False	False
3	True	False	False	True	False
4	False	True	False	False	True
5	false	False	False	True	False

Es un método general para encontrar aproximaciones de las raíces de una función real.

En informática, el algoritmo de Floyd-Warshall, descrito en 1959 por Bernard Roy, es un algoritmo de análisis sobre grafos para encontrar el camino mínimo en grafos dirigidos ponderados. El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución. El algoritmo de Floyd-Warshall es un ejemplo de programación dinámica.

Donde f' denota la derivada de f con respecto a x.

Alternativa 2. Algoritmo de Dijkstra



El algoritmo de Dijkstra, también llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto, dado un vértice origen, hacia el resto de los vértices en un grafo que tiene pesos en cada arista. Su nombre alude a Edsger Dijkstra, científico de la computación de los Países Bajos que lo concibió en 1956 y lo publicó por primera vez en 1959.12

Paso 4. Transición de las Ideas a los Diseños Preliminares

Lo primero que hacemos en este paso es descartar las ideas que no son factibles. En este sentido no descartamos ninguna de las dos ideas.

La revisión cuidadosa de las otras alternativas nos conduce a lo siguiente:

Alternativa 1. Algoritmo de Floyd-Warshall.

Es un algoritmo que nos produce los caminos más cortos entre todas las ciudades
Esto implica iterar todo el grafo para hallar un solo camino
No requiere un nodo fuente

Alternativa 3. Fórmula de la Ecuación Cuadrática

Es un algoritmo que dado un vértice nos da el camino hacia todos los demás a partir de este
A comparación de la alternativa 1 nos ahorra iteraciones y tiempo de ejecución
Requiere un nodo fuente

Paso 5. Evaluación y Selección de la Mejor Solución

Criterios

Deben definirse los criterios que permitirán evaluar las alternativas de solución y con base en este resultado elegir la solución que mejor satisface las necesidades del problema planteado. Los criterios que escogimos en este caso son los que enumeramos a continuación. Al lado de cada uno se ha establecido un valor numérico con el objetivo de establecer un peso que indique cuáles de los valores posibles de cada criterio tienen más peso (i.e., son más deseables).

Criterio A. Precisión de la solución. La alternativa entrega una solución:

[2] La más corta posible
[1] Aproximada

Criterio B. Eficiencia. Se prefiere una solución con mejor eficiencia que las otras consideradas. La eficiencia puede ser:

[4] Constante
[3] Logarítmica
[2] Lineal
[1] Exponencial

Criterio C. Completitud. Se prefiere una solución que encuentre todas las soluciones. Cuántas soluciones entrega:

[3] Todas
[2] Mas de una si las hay, aunque no todas
[1] Solo una o ninguna

Criterio D. Facilidad en implementación algorítmica:

[2] Implementación en dos o más tipos de grafos
[1] Implementación en un tipo de grafo

Criterio E. Velocidad según el tamaño de entrada

[2] Mantiene un tiempo aceptable
[1] El tiempo se dispara al aumentar la entrada

Evaluación

Evaluando los criterios anteriores en las alternativas que se mantienen, obtenemos la siguiente tabla:

	Criterio A	**Criterio B**	**Criterio C**	**Criterio D**	**Total**
Alternativa 1. Algoritmo de Floyd-Warshall	La más corta posible 2	Exponencial 1	Todas 3	Implementación en dos o más tipos de grafos 2	8
Alternativa 2. Algorimo de Dijkstra	La más corta posible 2	Exponencial 1	Más de una si las hay, aunque no todas 2	Implementación en dos o más tipos de grafos 2	7

Selección

De acuerdo con la evaluación anterior se debe seleccionar la Alternativa 1, ya que obtuvo la mayor puntuación de acuerdo con los criterios definidos. Se debe tener en cuenta que hay que hacer un manejo adecuado del criterio en el cual la alternativa fue peor evaluada que la otra alternativa.

Paso 6. Preparación de Informes y Especificaciones

Especificación del Problema (en términos de entrada/salida)

Problema: Raíces de una función cuadrática

Entradas: Punto de partida y punto de llegada.

2 2

Salida: Si la hay, la ruta más corta entre los dos puntos con sus nombres y distancias.

Consideraciones

Se deben tener en cuenta:

1. El punto de partida y llegada no puede ser el mismo
2. Debe haber por lo menos una ruta que conecte los dos puntos
3. Si se agregan varias rutas entre los mismo puntos se almacenará la más corta

Pseudocódigo del Algoritmo

```
int camino[][];

procedimiento FloydWarshall ()

para k: = 0 hasta n – 1

camino[i][j] = mín ( camino[i][j], camino[i][k]+camino[k][j])

fin para
```

Paso 7. Implementación del Diseño

Implementación en un Lenguaje de Programación.

Lista de Tareas a implementar:

a. Agregar las rutas
b. Ordenar las rutas en un grafo de tipo matrix y otro de lista de aristas
c. Verificar si hay rutas repetidas y almacenar la más corta
d. Agregar rutas de manera masiva desde un CSV
e. Algoritmo de Floyd-Warshall

Especificación de Subrutinas

Agregar las rutas

Nombre:	addRoute
Descripción:	Agrega una ruta al grafo
Entrada:	from: String, partida to: String, llegada route: String, Nombre de la ruta Miles: int, Distancia
Retorno:	Sin retorno

Ordenar las rutas en un grafo de tipo matrix y otro de lista de aristas

Nombre:	config
Descripción:	Configura la matrix cuando se agregan rutas
Entrada:	city: String, ciudad a agregar
Retorno:	int, el indice de la ciudad agregada

Verificar si hay rutas repetidas y almacenar la más corta

Nombre:	replace
Descripción:	Verificar si hay rutas repetidas y almacenar la más corta
Entrada:	route: String, nombre de la ruta miles: int, distancia de la ruta i, j: int, indices de la ruta
Retorno:	Sin retorno

Agregar rutas de manera masiva desde un CSV

Nombre:	Graph
Descripción:	Crea un grafo desde un CSV
Entrada:	URL: String, ruta del CSV
Retorno:	Graph, retorna un grafo

Algoritmo de Floyd-Warshall

Nombre:	floydWarshall
Descripción:	Encuentra la ruta más corta entre cada uno de los vertices
Entrada:	i, j, indices del vetice a consultar
Retorno:	String, contiene el camino más cortos

Construcción

Escritura del código en un Lenguaje de Programación Java en este caso

Agregar las rutas

```
public void addRoute(String from, String to, String route, int miles) {
        add(from, to, route, miles);
        bows.add(new Road(into(from), into(to), route, miles));
    }
private void add(String from, String to, String route, int miles) {
        int i = config(from);
        int j = config(to);
        replace(route, miles, i, j);
        replace(route, miles, j, i);
    }
```

Ordenar las rutas en un grafo de tipo matrix y otro de lista de aristas

```
private int config(String city) {
        int n = into(city);
        if(n==-1) {
            matrix.add(new ArrayList<Road>());
            nodes.add(city);
            resize();
            n = nodes.size()-1;
        }
        return n;
    }

private int into(String city) {
        for(int i=0; i<nodes.size(); i++) {
            if(city.equals(nodes.get(i)))
                return i;
        }
        return -1;
    }
private void resize() {
        for(int i=0; i<nodes.size(); i++) {
            for(int j=matrix.get(i).size(); j<nodes.size(); j++) {
                if(i==j)

matrix.get(i).add(new Road(i, j, "", 0));
                else
matrix.get(i).add(new Road(i, j, "", 1000000));
            }
        }
    }
```

Verificar si hay rutas repetidas y almacenar la más corta

```
private void replace(String route, int miles, int i, int j) {
		if(matrix.get(i).get(j).getMiles()>miles) {
			matrix.get(i).set(j, new Road(i, j, route, miles));
		}
	}
```

Agregar rutas de manera masiva desde un CSV

```
public Graph(String URL) throws IOException, NumberFormatException, IndexOutOfBoundsException {
		BufferedReader br = new BufferedReader(new FileReader(URL));
		String line = br.readLine();
		while(line != null && !line.equals("")) {
			String[] input = line.split(", ");
			add(input[0], input[1], input[2], Integer.parseInt(input[3]));
			bows.add(new Road(into(input[0]), into(input[1]), input[2], Integer.parseInt(input[3])));
			line = br.readLine();
		}
		br.close();
	}
```

Algoritmo de Floyd-Warshall

```
private String floyd(int from, int to){
		int[][] shortest = listToArray();
		String[][] roads = new String[shortest.length][shortest.length];
		int temp1, temp2;
		for(int k=0; k<nodes.size(); k++){
			for(int i=0; i<nodes.size(); i++){
				for (int j=0; j<nodes.size(); j++){
					temp1 = shortest[i][j];
					temp2 = shortest[i][k] + shortest[k][j];
					if(temp2<temp1) {
shortest[i][j] = temp2;
roads[i][j] = isEmpty(roads[i][j], i, j);
roads[i][k] = isEmpty(roads[i][k], i, k);
roads[k][j] = isEmpty(roads[k][j], k, j);
roads[i][j] = roads[i][k].trim() + "  " + roads[k][j].trim();
					}
				}
			}
		}
		return roads[from][to];
	}
```