

MÉTODO DE LA INGENIERÍA

Fase 1: Identificación del problema

El Masivo Integrado de Occidente(MIO) inaugurado el 15 de noviembre de 2008 en fase de prueba, el cual a partir del 1 de marzo de 2009 empezó su funcionamiento en firme; es el sistema de transporte masivo(SITM) de la ciudad colombiana de Santiago de Cali. El sistema es operado por buses articulados, padrones y complementarios, los cuales se desplazan por medio de corredores troncales, pre troncales y complementarios cubriendo rutas troncales, pre troncales y alimentadoras.

Ante la reciente avalancha de usuarios que están utilizando el sistema se ha presentado una dificultad para conocer que ruta es más corta acorde al lugar de destino.

Muchos de los usuarios siempre han requerido que el SITM presente que ruta para ellos es la más corta con referencia a la distancia del lugar en donde se encuentran hacia el lugar de destino.

- **Definición del problema:**

Se ha requerido el desarrollo de un programa que agrupe la información de mayor relevancia de las rutas del MIO, para así mismo calcular las distancias que recorren ciertas rutas por ciertas vías y de esta manera a través de Grafos representar el SITM y dar al usuario el MIO con la ruta más corta para llegar al lugar de destino.

REQUERIMIENTOS FUNCIONALES

La solución del problema:

R1: Debe registrar las rutas por donde se transportan los masivos, el cual representa cada uno de los caminos posibles que tiene un MIO.

R2: Debe dar la distancia que existe entre una ruta y otra las cuales son recorridas por los masivos, estas distancias ya están predefinidas en el grafo.

R3: Debe dar la ruta más cercana para el usuario acorde a su lugar de destino tomando por referencia el lugar donde se encuentra actualmente.

R4: Debe registrar los masivos que pasan por las rutas ya definidas en el grafo para así conocer cuál es de utilidad para el usuario.

R5: Debe registrar los puntos o paradas donde se encuentran los usuarios para a partir de este conocer la distancia a otro punto y la ruta más cercana.

FASE 2: Recopilación de la información

Definiciones

GRAFOS

En matemáticas y en ciencias de la computación, la teoría de grafos (también llamada teoría de las gráficas) estudia las propiedades de los grafos (también llamadas gráficas). Un grafo es un conjunto, no vacío, de objetos llamados vértices (o nodos) y una selección de pares de vértices, llamados aristas (edges en inglés) que pueden ser orientados o no. Típicamente, un grafo se representa mediante una serie de puntos (los vértices) conectados por líneas (las aristas).

Existen diferentes formas de almacenar grafos en una computadora. La estructura de datos usada depende de las características del grafo y el algoritmo usado para manipularlo. Entre las estructuras más sencillas y usadas se encuentran las listas y las matrices, aunque frecuentemente se usa una combinación de ambas. Las listas son preferidas en grafos dispersos porque tienen un eficiente uso de la memoria. Por otro lado, las matrices proveen acceso rápido, pero pueden consumir grandes cantidades de memoria.

Grafos no dirigidos

- Grado de un vértice:

Número de aristas que lo contienen.

Grafos dirigidos

- Grado de salida de un vértice v :

Número de arcos cuyo vértice inicial es v .

- Grado de entrada de un vértice v :

Número de arcos cuyo vértice final es v.

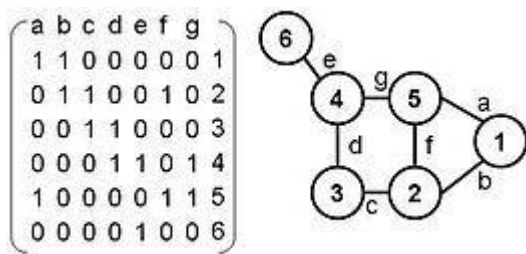
Estructura de lista

- Lista de incidencia - Las aristas son representadas con un vector de pares (ordenados, si el grafo es dirigido), donde cada par representa una de las aristas.
- Lista de adyacencia - Cada vértice tiene una lista de vértices los cuales son adyacentes a él. Esto causa redundancia en un grafo no dirigido (ya que A existe en la lista de adyacencia de B y viceversa), pero las búsquedas son más rápidas, al costo de almacenamiento extra.

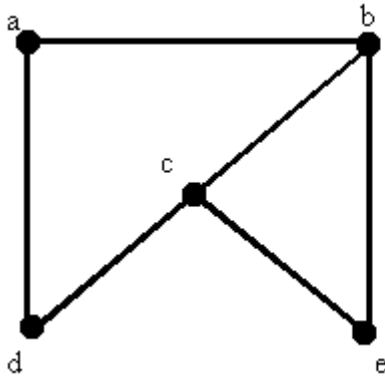
En esta estructura de datos la idea es asociar a cada vértice i del grafo una lista que contenga todos aquellos vértices j que sean adyacentes a él. De esta forma sólo reservará memoria para los arcos adyacentes a i y no para todos los posibles arcos que pudieran tener como origen i. El grafo, por tanto, se representa por medio de un vector de n componentes (si $|V|=n$) donde cada componente va a ser una lista de adyacencia correspondiente a cada uno de los vértices del grafo. Cada elemento de la lista consta de un campo indicando el vértice adyacente. En caso de que el grafo sea etiquetado, habrá que añadir un segundo campo para mostrar el valor de la etiqueta.

Estructuras matriciales

- Matriz de incidencia - El grafo está representado por una matriz de A (aristas) por V (vértices), donde [arista, vértice] contiene la información de la arista (1 - conectado, 0 - no conectado)



- Matriz de adyacencia - El grafo está representado por una matriz cuadrada M de tamaño, donde es el número de vértices. Si hay una arista entre un vértice x y un vértice y, entonces el elemento es 1, de lo contrario, es 0.



$$A = \begin{matrix} & \begin{matrix} a & b & c & d & e \end{matrix} \\ \begin{matrix} a \\ b \\ c \\ d \\ e \end{matrix} & \begin{vmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{vmatrix} \end{matrix}$$

Vértice

Los vértices constituyen uno de los dos elementos que forman un grafo. Como ocurre con el resto de las ramas de las matemáticas, a la Teoría de Grafos no le interesa saber qué son los vértices.

Diferentes situaciones en las que pueden identificarse objetos y relaciones que satisfagan la definición de grafo pueden verse como grafos y así aplicar la Teoría de Grafos en ellos.

Subgrafo

Un subgrafo de un grafo G es un grafo cuyos conjuntos de vértices y aristas son subconjuntos de los de G . Se dice que un grafo G contiene a otro grafo H si algún subgrafo de G es H o es isomorfo a H (dependiendo de las necesidades de la situación).

El subgrafo inducido de G es un subgrafo G' de G tal que contiene todas las aristas adyacentes al subconjunto de vértices de G .

Definición:

Sea $G=(V, A)$. $G'=(V',A')$ se dice subgrafo de G si:

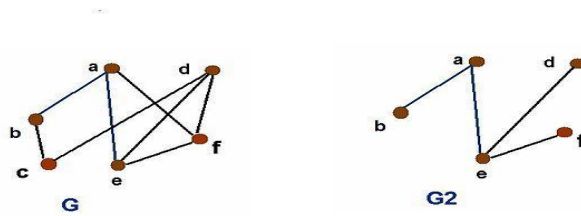
1- $V' \subseteq V$

2- $A' \subseteq A$

3- (V',A') es un grafo

• Si $G'=(V',A')$ es subgrafo de G , para todo $v \in V$ se cumple $\text{gr}(G',v) \leq \text{gr}(G,v)$

G_2 es un subgrafo de G .



Caracterización de grafos

- Grafos simples

Un grafo es simple si a lo más existe una arista uniendo dos vértices cualesquiera. Esto es equivalente a decir que una arista cualquiera es la única que une dos vértices específicos.

Un grafo que no es simple se denomina multigrafo.

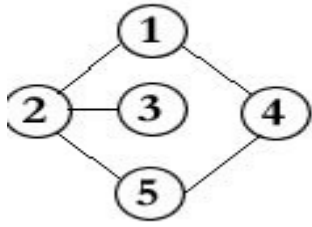
- Grafos conexos

Un grafo es conexo si cada par de vértices está conectado por un camino; es decir, si para cualquier par de vértices (a, b), existe al menos un camino posible desde a hacia b.

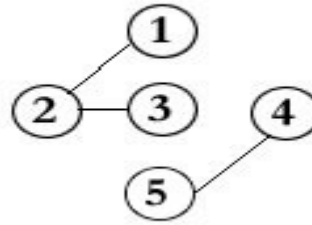
Un grafo es doblemente conexo si cada par de vértices está conectado por al menos dos caminos disjuntos; es decir, es conexo y no existe un vértice tal que al sacarlo el grafo resultante sea disco nexa. Es posible determinar si un grafo es conexo usando un algoritmo Búsqueda en anchura (BFS) o Búsqueda en profundidad (DFS).

En términos matemáticos la propiedad de un grafo de ser (fuertemente) conexo permite establecer con base en él una relación de equivalencia para sus vértices, la cual lleva a una partición de éstos en "componentes (fuertemente) conexas", es decir, porciones del grafo, que son (fuertemente) conexas cuando se consideran como grafos aislados. Esta propiedad es importante para muchas demostraciones en teoría de grafos.

Grafo conexo



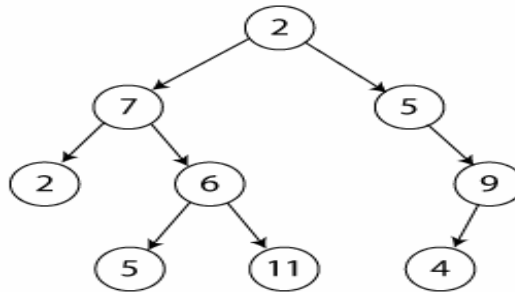
Grafo no conexo



- Árboles

Un grafo que no tiene ciclos y que conecta a todos los puntos, se llama un árbol. En un grafo con n vértices, los árboles tienen exactamente $n - 1$ aristas, y hay n^{n-2} árboles posibles. Su importancia radica en que los árboles son grafos que conectan todos los vértices utilizando el menor número posible de aristas.

Un importante campo de aplicación de su estudio se encuentra en el análisis filogenético, el de la filiación de entidades que derivan unas de otras en un proceso evolutivo, que se aplica sobre todo a la averiguación del parentesco entre especies; aunque se ha usado también, por ejemplo, en el estudio del parentesco entre lenguas.

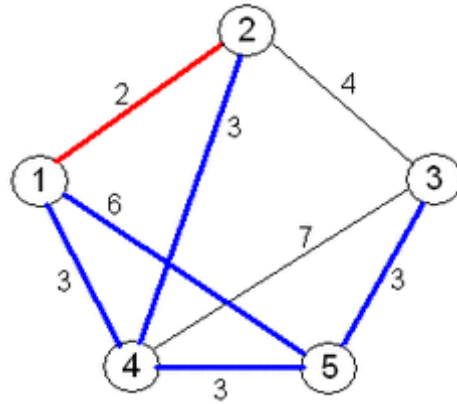


- Grafos ponderados o etiquetados

En muchos casos, es preciso atribuir a cada arista un número específico, llamado valuación, ponderación o coste según el contexto, y se obtiene así un grafo valuado.

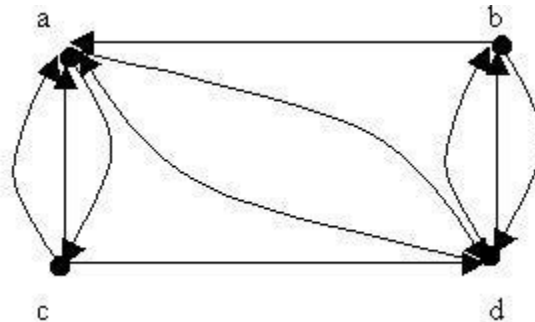
Formalmente, es un grafo con una función $v: A \rightarrow \mathbb{R}^+$

Por ejemplo, un representante comercial tiene que visitar n ciudades conectadas entre sí por carreteras; su interés previsible será minimizar la distancia recorrida (o el tiempo, si se pueden prever atascos). El grafo correspondiente tendrá como vértices las ciudades, como aristas las carreteras y la valuación será la distancia entre ellas. Y, de momento, no se conocen métodos generales para hallar un ciclo de valuación mínima, pero sí para los caminos desde a hasta b , sin más condición.



- Multígrafo:

Grafo en el que se permite que entre dos vértices exista más de una arista o arco.



ALGORITMOS A UTILIZAR PARA EL DESARROLLO DEL PROBLEMA

Algoritmo de Dijkstra

Algoritmo de Dijkstra. También llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto dado un vértice origen al resto de vértices en un grafo con pesos en cada arista. Su nombre se refiere a Edsger Dijkstra, quien lo describió por primera vez en 1959.

- Aplicaciones

En múltiples aplicaciones donde se aplican los grafos, es necesario conocer el camino de menor costo entre dos vértices dados:

Distribución de productos a una red de establecimientos comerciales.

Distribución de correos postales.

Sea $G = (V, A)$ un grafo dirigido ponderado.

El problema del camino más corto de un vértice a otro consiste en determinar el camino de menor costo, desde un vértice u a otro vértice v . El costo de un camino es la suma de los costos (pesos) de los arcos que lo conforman.

- Características del algoritmo

Es un algoritmo greedy.

Trabaja por etapas, y toma en cada etapa la mejor solución sin considerar consecuencias futuras.

El óptimo encontrado en una etapa puede modificarse posteriormente si surge una solución mejor.

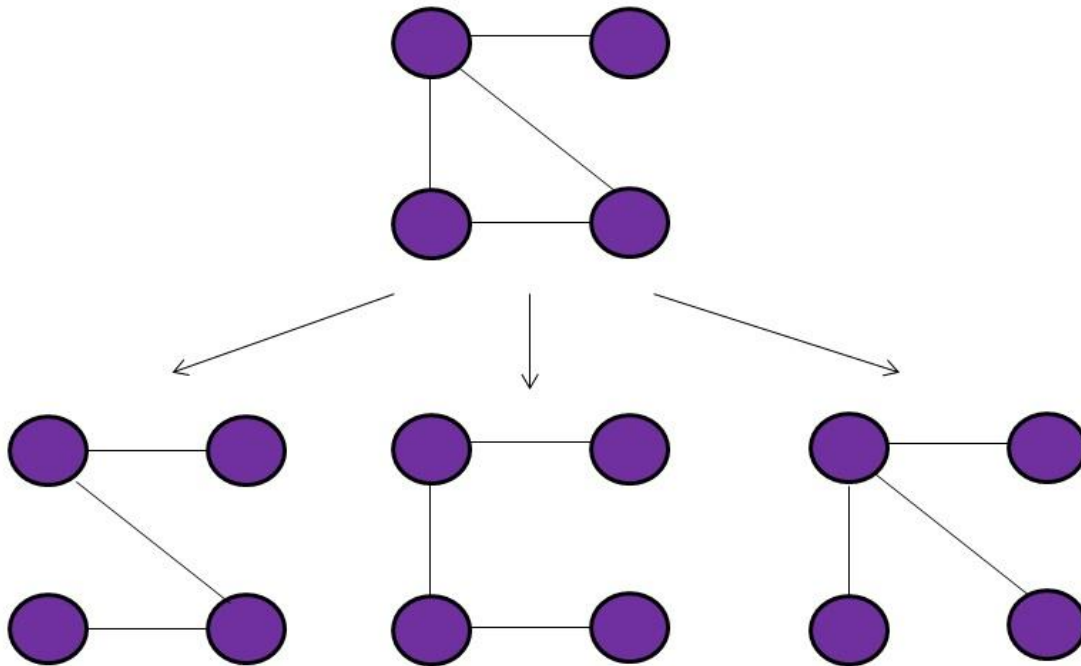
- Pseudocódigo

```
DIJKSTRA (Grafo G, nodo_fuente s)
para  $u \in V[G]$  hacer
    distancia[u] = INFINITO
    padre[u] = NULL
distancia[s] = 0
adicionar (cola, (s, distancia[s]))
mientras que cola no es vacía hacer
     $u = \text{extraer\_minimo}(\text{cola})$ 
    para todos  $v \in \text{adyacencia}[u]$  hacer
        si  $\text{distancia}[v] > \text{distancia}[u] + \text{peso}(u, v)$  hacer
             $\text{distancia}[v] = \text{distancia}[u] + \text{peso}(u, v)$ 
             $\text{padre}[v] = u$ 
            adicionar( $\text{cola}, (v, \text{distancia}[v])$ )
```


ÁRBOL DE EXPANSIÓN MÍNIMA: ALGORITMO DE KRUSKAL

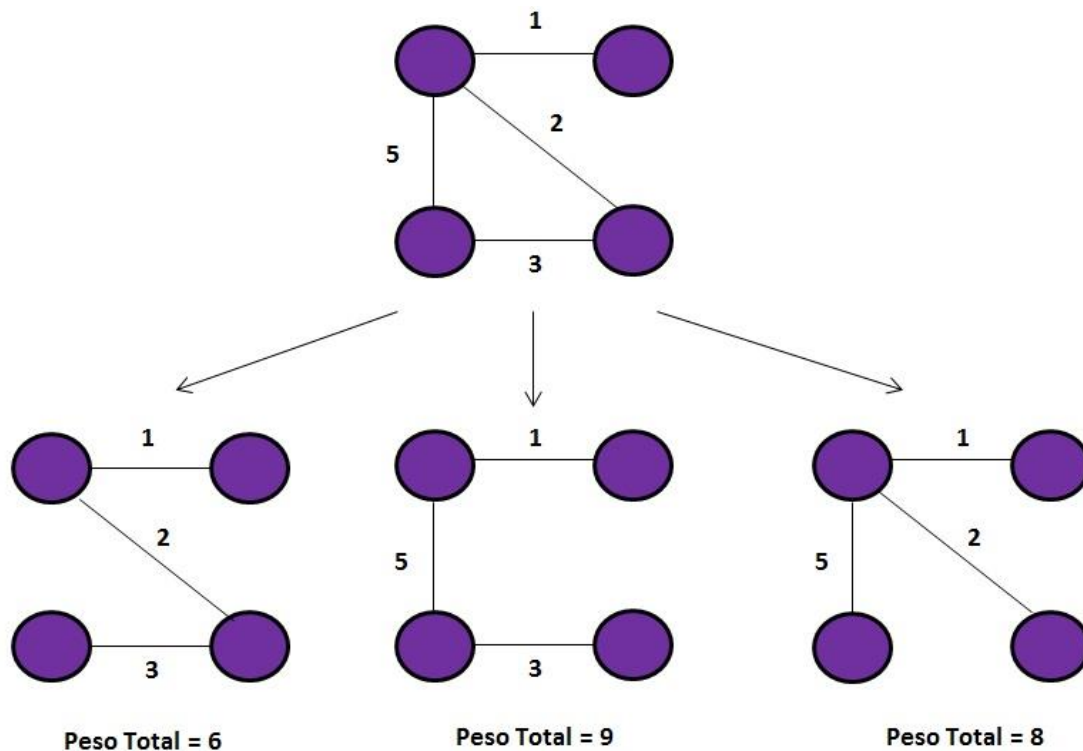
Dado un grafo conexo, no dirigido G . Un árbol de expansión es un árbol compuesto por todos los vértices y algunas (posiblemente todas) de las aristas de G . Al ser creado un árbol no existirán ciclos, además debe existir una ruta entre cada par de vértices.

Un grafo puede tener muchos árboles de expansión, veamos un ejemplo con el siguiente grafo:



Árbol de Expansión Mínima

Dado un grafo conexo, no dirigido y con pesos en las aristas, un árbol de expansión mínima es un árbol compuesto por todos los vértices y cuya suma de sus aristas es la de menor peso. Al ejemplo anterior le agregamos pesos a sus aristas y obtenemos los arboles de expansiones siguientes:



Algoritmos de Kruskal

Primeramente ordenaremos las aristas del grafo por su peso de menor a mayor. Mediante la técnica greedy Kruskal intentará unir cada arista siempre y cuando no se forme un ciclo, ello se realizará mediante Union-Find. Como hemos ordenado las aristas por peso comenzaremos con la arista de menor peso, si los vértices que contienen dicha arista no están en la misma componente conexa entonces los unimos para formar una sola componente mediante $\text{Union}(x, y)$, para revisar si están o no en la misma componente conexa usamos la función $\text{SameComponent}(x, y)$ al hacer esto estamos evitando que se creen ciclos y que la arista que une dos vértices siempre sea la mínima posible.

- Pseudocódigo

```
1 método Kruskal(Grafo):
2   inicializamos MST como vacío
3   inicializamos estructura unión-find
4   ordenamos las aristas del grafo por peso de menor a mayor.
5   para cada arista e que une los vértices u y v
6     si u y v no están en la misma componente
7       agregamos la arista e al MST
8     realizamos la unión de las componentes de u y v
```

CAMINO MAS CORTO: ALGORITMO DE BELLMAN-FORD

Descripción

El algoritmo de Bellman-Ford determina la ruta más corta desde un nodo origen hacia los demás nodos para ello es requerido como entrada un grafo cuyas aristas posean pesos. La diferencia de este algoritmo con los demás es que los pesos pueden tener valores negativos ya que Bellman-Ford me permite detectar la existencia de un ciclo negativo.

Como trabaja

El algoritmo parte de un vértice origen que será ingresado, a diferencia de Dijkstra que utiliza una técnica voraz para seleccionar vértices de menor peso y actualizar sus distancias mediante el paso de relajación, Bellman-Ford simplemente relaja todas las aristas y lo hace $|V| - 1$ veces, siendo $|V|$ el número de vértices del grafo.

Para la detección de ciclos negativos realizamos el paso de relajación una vez más y si se obtuvieron mejores resultados es porque existe un ciclo negativo, para verificar porque tenemos un ciclo podemos seguir relajando las veces que queramos y seguiremos obteniendo mejores resultados.

Pseudocódigo

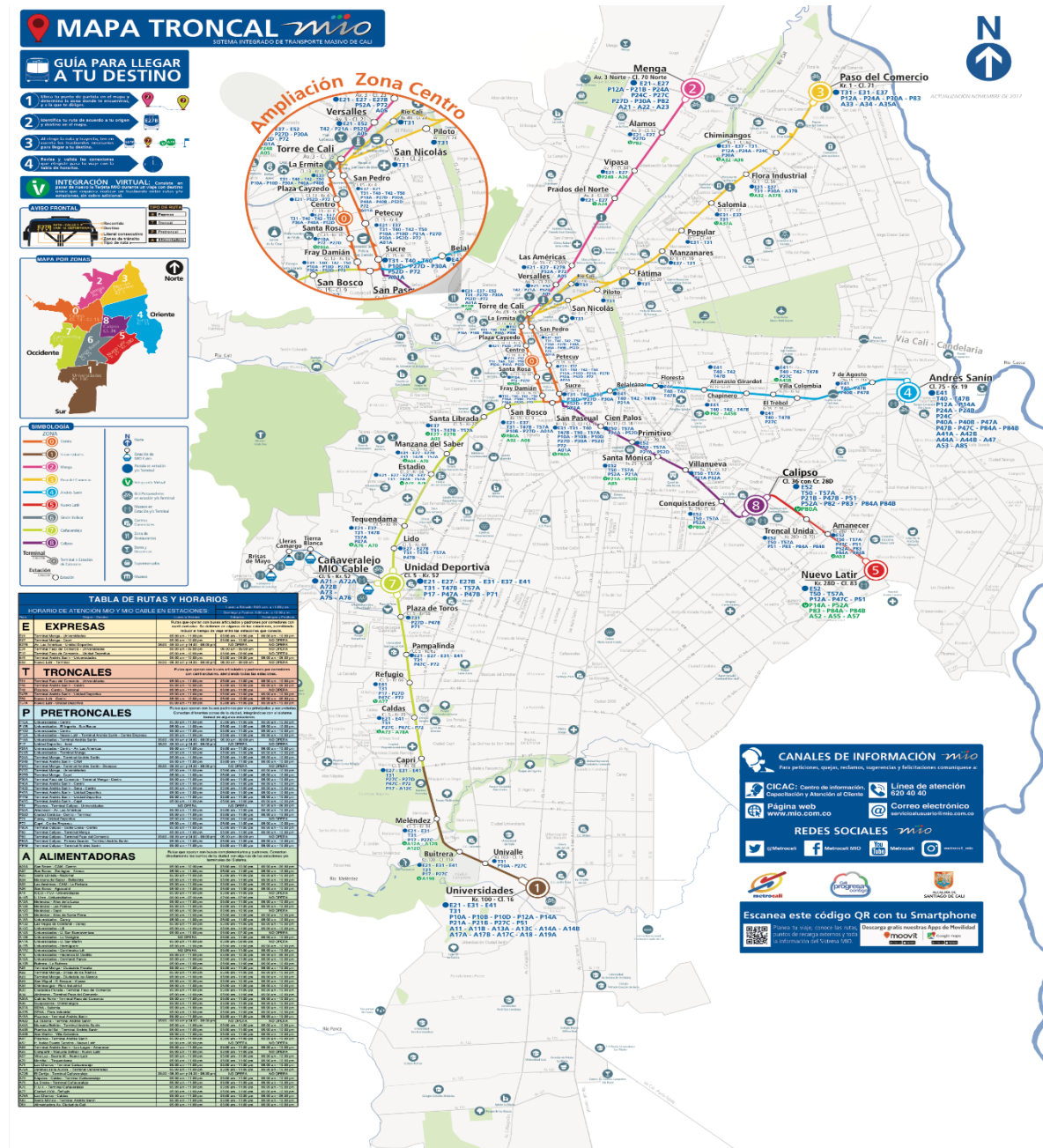
```
1 método BellmanFord(Grafo, origen):
2   inicializamos las distancias con un valor grande
3   distancia[ origen ] = 0
4   para i = 1 hasta  $|V| - 1$ :
5     para cada arista E del Grafo:
6       sea ( u , v ) vértices que unen la arista E
7       sea w el peso entre vértices ( u , v )
8       Relajacion( u , v , w )
```

```
9  para cada arista E del Grafo:
10     sea ( u , v ) vértices que unen la arista E
11     sea w el peso entre vértices ( u , v )
12     si Relajacion( u , v , w )
13         Imprimir "Existe ciclo negativo"
15     Terminar Algoritmo

1  Relajacion( actual , adyacente , peso ):
2     si distancia[ actual ] + peso < distancia[ adyacente ]
3     distancia[ adyacente ] = distancia[ actual ] + peso
```

Información Importante:

Mapa de las rutas del mio, con localidades:



Fase 3: Búsqueda de soluciones creativas

La generación de las ideas se desarrollaron de forma conjunta planeando cuales serían las posibles soluciones al problema, donde aceptamos todo tipo de idea, pero con el fin de que su uso fuese lógico en la solución, sin importar si era el más óptimo, el más eficiente o algún otro criterio.

1. Mediante el uso de algoritmo de recorrido DFS analizar el recorrido mínimo que existe entre una ruta y la otra.
2. Mediante el uso de BFS recorrer todo el grafo.
3. Utilizar las estructuras de datos pila y cola para almacenar cada una de las rutas por donde se transportan los masivos e ingresar acorde a la distancia entre uno y otro.
4. Utilizar tablas hash para guardar cada una de las rutas con sus respectivas claves.
5. Utilizar el algoritmo de Dijkstra.
6. Utilizar el MST.
7. Utilizar el algoritmo de Bellman Ford
8. Utilizar el algoritmo de Floyd-Warshall's

Fase 4: Transición de las ideas a los Diseños Preliminares

Las siguientes ideas las descartaremos tomando como base de criterio el desarrollo por completo y de forma efectiva lo que se nos está pidiendo en este caso con el desarrollo de la factorización de polinomios para hallar sus raíces, no obstante, en cada una de las ideas descartadas que daremos a continuación argumentaremos por qué se tomó la decisión.

Alternativa 1: Uso de algoritmo DFS

- No requerimos el uso de este algoritmo para recorrer el grafo ya que nuestro objetivo es analizar los caminos más cortos, la menor distancia. Pero si utilizaremos BFS para una de nuestras funciones.

Alternativa 3: Utilizar estructuras de Pilas y Colas

- Aunque el uso de estas estructuras de datos esta incorporado para el desarrollo des problema, lo incorporamos en esta lista porque no usamos Pila y Cola como medio fundamental para el desarrollo del programa, pero si como una herramienta de apoyo.

Alternativa 4: Utilizar Tablas Hash

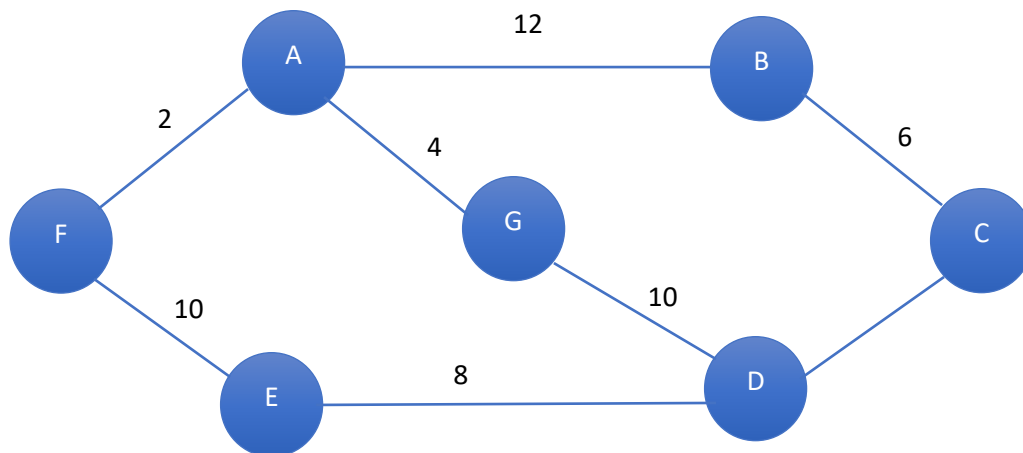
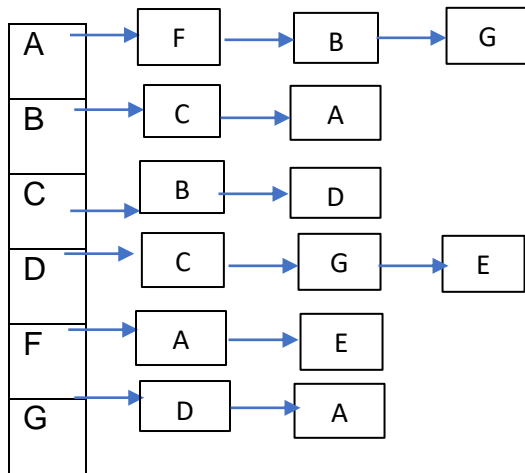
-Aunque sería muy interesante utilizar Tablas Hash en el almacenamiento de cada parada con su respectiva clave, no nos es útil es este caso.

Diseños preliminares:

TAD Grafo (Lista de adyacencia).

TAD: Grafo.

Representación:



Invariantes:

- Un grafo se define como $G = (V, E)$ donde V es el conjunto de vértices y E el conjunto de las aristas
- Cada arista conecta dos vértices.
- Dos vértices son adyacentes si tienen una arista que los conecta.
- Una arista es incidente a un vértice si esta lo une a otro.
- Ponderación es una función que a cada arista se le asocia un valor.

Operaciones

- <ContruirGrafo>:
Grafo



- <AgregarVertice>: Grafo, Valor →
Grafo
- <AgregarArista>: Grafo, Valor1, Valor2, Etiqueta →
Grafo
- <EliminarVertice>: Grafo, Valor →
Grafo
- <RecorrerGrafo>: Grafo, Valor →
Grafo
- <CaminoMasCorto>: Grafo, Valor1 →
Entero
- <EliminarArista>: Grafo, Valor1, Valor2 →
Grafo

ConstruirGrafo()

“Crea un nuevo grafo sin ningún vértice”

{Pre: True}

{Post: Grafo = {Grafo}}

AgregarVertice()

“Agrega un nuevo vértice al grafo, sin ninguna conexión ya que no vértices adyacentes”

{Pre: Grafo = {Grafo}, Valor; Valor = dato}

{Post: Grafo = Grafo con un vértice nuevo agregado}

AgregarArista()

“Agrega una nueva arista al grafo, esta conecta dos vértices especificados, esto se agrega en la lista de adyacencia que representa la conexión entre los vértices”

{Pre: Grafo = {Grafo}, Valor1 = dato del vértice; Valor2 = dato del vértice 2;
Etiqueta = Entero que representa ponderación en la arista}
{Post: Grafo con una nueva arista}

EliminarVertice()

“Elimina el vértice que se requiere de acuerdo con el valor especificado, además si este tenía una arista esta también se eliminara”

{Pre: Grafo = {Grafo}, Valor = dato del vértice}

{Post: Grafo sin el nodo con la llave}

RecorrerGrafo()

“Recorre todo el grafo sin repetir los vértices”

{Pre: Grafo = {Grafo}, Valor = dato del vértice desde donde inicia el recorrido}

{Post: Grafo}

CaminoMasCorto()

“Retorna el valor de la distancia más corta entre dos vértices, esto de acuerdo con la etiqueta de las aristas”

{Pre: Grafo= {Grafo}, Valor1 = dato del vértice; Valor2 = dato del vértice al cual se quiere llegar, valor}

{Post: Entero que representa la distancia más corta}

EliminarArista()

“Elimina la conexión entre dos vértices”

{Pre: Grafo = {Grafo}; Valor1 = Dato del vértice al cual se le va a eliminar la conexión, Valor2 = Dato del vértice al cual se le va a eliminar la conexión}

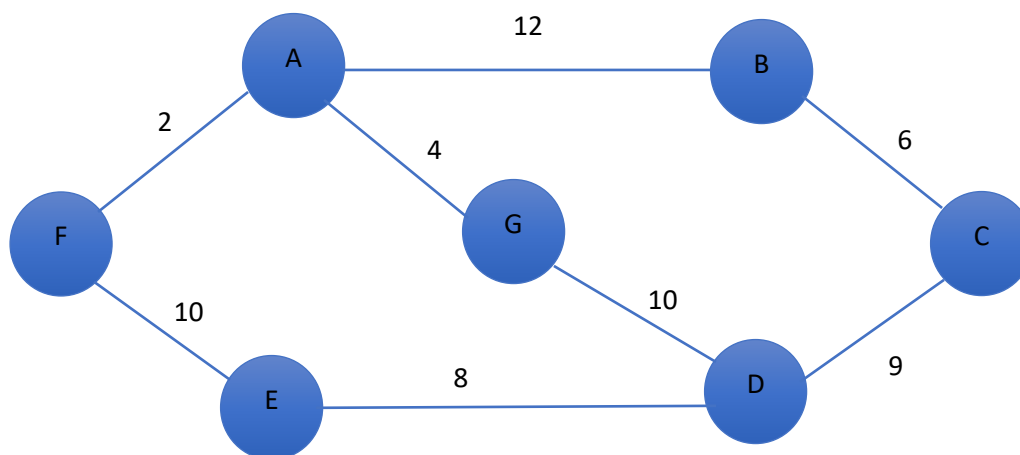
{Post: Grafo sin la arista}

TAD Grafo (Matriz de adyacencia).

TAD: Grafo.

Representacion

| | A | B | C | D | E | F | G |
|---|----|----|---|----|----|----|----|
| A | 0 | 12 | 0 | 0 | 0 | 2 | 4 |
| B | 12 | 0 | 6 | 0 | 0 | 0 | 0 |
| C | 0 | 6 | 0 | 9 | 0 | 0 | 0 |
| D | 0 | 0 | 9 | 0 | 8 | 0 | 10 |
| E | 0 | 0 | 0 | 8 | 0 | 10 | 0 |
| F | 2 | 0 | 0 | 0 | 10 | 0 | 0 |
| G | 4 | 0 | 0 | 10 | 0 | 0 | 0 |






Invariantes:

- Un grafo se define como $G = (V, E)$ donde V es el conjunto de vértices y E el conjunto de las aristas
- Cada arista conecta dos vértices.
- Dos vértices son adyacentes si tienen una arista que los conecta.
- Una arista es incidente a un vértice si esta lo une a otro.
- Ponderación es una función que a cada arista se le asocia un valor.

Operaciones

- <ContruirGrafo>: Grafo \longrightarrow
- <AgregarVertice>: Grafo, Valor \longrightarrow
- <AgregarArista>: Grafo, Valor1, Valor2, Etiqueta \longrightarrow
- <EliminarVertice>: Grafo, Valor \longrightarrow

- <RecorrerGrafo>: Grafo, Valor 
- <CaminoMasCorto>: Grafo, Valor1 
Entero
- <EliminarArista>: Grafo, Valor1, Valor2 
Grafo

ConstruirGrafo()

“Crea un nuevo grafo sin ningún vértice”

{Pre: True}

{Post: Grafo = {Grafo}}

AgregarVertice()

“Agrega un nuevo vértice al grafo, sin ninguna conexión ya que no vértices adyacentes”

{Pre: Grafo = {Grafo}, Valor; Valor = dato}

{Post: Grafo = Grafo con un vértice nuevo agregado}

AgregarArista()

“Agrega una nueva arista al grafo, esta conecta dos vértices especificados, esto se agrega en la matriz de adyacencia que representa la conexión entre los vértices”

{Pre: Grafo = {Grafo}, Valor1 = dato del vértice; Valor2 = dato del vértice 2; Etiqueta = Entero que representa ponderación en la arista}

{Post: Grafo con una nueva arista}

EliminarVertice()

“Elimina el vértice que se requiere de acuerdo con el valor especificado, además si este tenía una arista esta también se eliminara”

{Pre: Grafo = {Grafo}, Valor = dato del vértice}

{Post: Grafo sin el nodo con la llave}

RecorrerGrafo()

“Recorre todo el grafo sin repetir los vértices”

{Pre: Grafo = {Grafo}, Valor = dato del vértice desde donde inicia el recorrido}

{Post: Grafo}

CaminoMasCorto()

“Retorna el valor de la distancia más corta entre dos vértices, esto de acuerdo con la etiqueta de las aristas”

{Pre: Grafo= {Grafo}, Valor1 = dato del vértice; Valor2 = dato del vértice al cual se quiere llegar, valor}

{Post: Entero que representa la distancia más corta}

Fase 5: Evaluación y Selección de la Mejor Solución

Criterios

Estos son los principios por los cuales serán evaluadas las ideas y donde escogeremos las apropiadas para el desarrollo de la solución del problema de forma definitiva.

Criterio A: Complejidad

- [8] Complejidad constante
- [7] Complejidad logarítmica
- [6] Complejidad raíz
- [5] Complejidad lineal
- [4] Complejidad $n \log n$
- [3] Complejidad polinómica
- [2] Complejidad exponencial
- [1] Complejidad factorial

Criterio B: Eficiencia

- [3] Muy eficiente
- [2] Eficiente
- [1] Nada eficiente

Criterio C: Facilidad implementación

- [2] Fácil
- [1] Difícil

Evaluación

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

| | Criterio A | Criterio B | Criterio C | Total |
|----------------------------|----------------|----------------------|------------|-------|
| Alternativas 2: BFS | Constante - 8 | Muy eficiente - 3 | Fácil - 2 | 13 |
| Alternativa 6: MST | $n \log n$ - 4 | Muy eficiente - 3 | Fácil - 2 | 9 |
| Alternativa 5: Dijkstra | $n \log N$ - 4 | Muy eficiente - 3 | Fácil - 2 | 9 |

| | | | | |
|---------------------------|--------------------|---------------|-------------|----|
| Alternativa 7: BellMan | Constante - 8 | Eficiente - 2 | Difícil - 1 | 11 |
| Algoritmos 8: Floyd | Exponencial - 2 | Eficiente - 2 | Difícil - 1 | 5 |

Selección

Teniendo en cuenta la tabla anterior, las alternativas 2,5,6 y 7 deben ser escogidas ya que tuvieron mayor puntuación dados los criterios de evaluación y por motivos de conveniencia en el desarrollo.

Bibliografía

<https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/>

[https://www.ecured.cu/Algoritmo de Dijkstra](https://www.ecured.cu/Algoritmo_de_Dijkstra)

<https://jariasf.wordpress.com/2013/01/01/camino-mas-corto-algoritmo-de-bellman-ford/>

<https://jariasf.wordpress.com/2012/04/19/arbol-de-expansion-minima-algoritmo-de-kruskal/>

<https://www.hackerearth.com/practice/algorithms/graphs/graph-representation/tutorial/>

<http://www.mio.com.co/images/stories/rutas%20pdf/rutas2016/pdf/mapageneral.pdf>