Algoritmos y Estructura de Datos

2020-2 Tarea Integradora 3

Nombre: **Sebastian Villa Ávila** **Jhon sebastian Ijai Ortiz**	**Código:** A00361589 A00362423
Profesor: **Andres Aristizabal**	

Enunciado:

Problema de la Aerolínea

Una reconocida Aerolínea los ha contratado para implementar un programa que calcule cual es la ruta más corta entre su lugar de partida y su destino. Es decir, el programa debe permitir al usuario ingresar su punto de partida y su punto de llegada de tal forma que este pueda calcular las distintas rutas que existen para el mismo trayecto y mostrarle al usuario la ruta más corta según sus destinos y a la vez le muestre el valor del vuelo.

Método de la Ingeniería

1. Identificación del Problema

- **Definición:**
  Calcular la ruta más corta o la ruta más barata según sea seleccionado por el cliente

- **Contexto:**
  El problema consiste en implementar un programa que haga uso de los conocimientos adquiridos en la última unidad del curso (Grafos), para diseñar un software que calcule el camino más corto entre dos destinos y a la vez cual es la y a la vez el valor de cada vuelo de tal forma que le muestre al usuario la ruta mas rapida para su viaje o la rutas más barata según sea la necesidad del cliente.

Síntomas y Necesidades:

- Los usuarios requieren un Software capaz de calcular a la ruta más corta
- Los usuarios requieren un software capaz de calcular el valor de los vuelos
- La solución al problema debe hacer uso de dos implementaciones de grafos
- La solución al problema debe ser eficiente para que el servicio pueda ser entregado a la mayor cantidad de usuarios con el mínimo consumo de recursos.

Requerimientos Funcionales:

El sistema está en capacidad de:

- Permitir ingresar el lugar de partida y el destino
- Calcular la ruta más corta para el viaje
- Calcular la ruta más barata
- Proporcionar el valor del viaje

Requerimientos no Funcionales:

- Implementar dos versiones de grafos
- Implementar interfaz gráfica
- Aplicar BFS, Dijkstra, Prim

2. Recopilacion de Informacion

Conceptos:

- **Aerolíneas:**
  una aerolínea, línea aérea o compañía aérea es una empresa que se dedica al transporte de pasajeros o carga —y, en algunos casos, animales— por avión

- **Rutas vuelo:**
  En aviación, una aerovía, o ruta aérea, es una ruta designada en el espacio aéreo. Las aerovías son establecidas entre varios elementos de ayuda a la navegación como los radiofaros omnidireccionales VHF, radiofaros no direccionales en intersecciones. Son componentes básicos de los planes de vuelo de los aviones.

- **Tiquete Aéreo:**
  Un boleto de avión es un documento o registro electrónico, emitido por una aerolínea o una agencia de viajes, que confirma que una persona tiene derecho a un asiento en un vuelo en un avión. El boleto de avión puede ser de dos tipos: un boleto de papel, que comprende cupones o vales; y un boleto electrónico

Contexto general del problema:

Cualquier ruta de un aeropuerto X a otro Y hace uso de esta clase de rastreo aéreo. Una aeronave no vuela en línea recta; se mueve de un lugar a otro. En distancias más grandes, dicha ruta en forma poligonal casi se ajusta a la línea directa. La razón es simple y lógica: cuanto más corta es la distancia, menor es la cantidad de combustible consumido.

Muchos creen que un avión vuela en la línea curva por alguna razón. En Flightradar la ruta se representa, al menos, como una curva, tal como se aprecia en las pantallas a bordo (algo que no guardar secreto alguno). Sin embargo, la tierra es una esfera, mientras que los mapas y los monitores la representan como plana. Cuanto más cerca de la ruta está un polo, más distorsionada se representa.

Por ejemplo, una ruta de Moscú a Los Ángeles se asemeja a una parábola en un mapa. Pero si agarras un globo terráqueo y estiramos una cuerda entre estas dos ciudades, verás como la ruta de del avión es próxima a dicha línea que es, obviamente, la distancia más corta.

Adjunto link:
Video que muestra cómo se realizan las rutas aéreas
https://youtu.be/s2b06qtqpp4

3. Búsqueda de soluciones

Lluvia de ideas:

1. Realizar una aplicación que solicite al usuario registrarse para realizar el seguimiento de las rutas que va a tomar, que implemente una base de datos y sea persistente.

2. Realizar una aplicación que maneje un mapa, que automáticamente halle la ubicación en la que se encuentra y permita seleccionar en el mapa la ubicación a la que quiere llegar.

3. Realizar un software en el cual se ingrese el punto de partida y llegada, de tal forma que se muestre todas las rutas y el usuario pueda elegir su preferida.

4. Realizar un software que permita ingresar el destino de partida y el destino de llega para calcular los precios y las diferentes rutas, de tal forma que se le muestre al usuario la ruta más corta o la ruta más económica según sea el caso elegido por el usuario haciendo uso de grafos y de sus diferentes operaciones para realizar una solución óptima con una interfaz gráfica amigable y fácil de usar.

5. Diseñar una aplicación que ingresados los destinos de partida y llegada, pueda registrarse el usuario y mediante una archivo de texto creado se le entregue la ruta más corta y el valor de esta, todo mediante una interfaz de usuario y una implementación de árboles.

6. Diseñar una aplicación que tenga una interfaz de usuario amigable, permita al usuario ingresar la ruta que quiere tomar, informarle si esa ruta es óptima y sugerirle otras rutas similares o mejores.

4. Diseños preliminares

Descarte de Alternativas:

En primer lugar tenemos que descartar las alternativas 2, 5 y 6 dado que estas son poco eficientes y no cumplen con las funcionalidades necesarias para el funcionamiento completo del programa. Por tanto, se hace imposible el uso de alguna de estas alternativas.

Revisión de Alternativas:

- **Alternativa 1:**
  - ❖ Esta opción no cumple con los requerimientos solicitados
  - ❖ No es necesario hacer el registro de usuarios por tanto hace la solución poco eficiente
  - ❖ No es necesaria la persistencia

- **Alternativa 3:**
  - ❖ Cumple con los requerimientos pedidos por el usuario, ya que calcula el tiempo de vuelo de cada ruta
  - ❖ No muestra únicamente la ruta mas rapida
  - ❖ No calcula el valor de los de los vuelos

- **Alternativa 4:**
  - ❖ Cumple con todos los requerimientos solicitados por el usuario
  - ❖ Calcula la ruta más corta y rápida y el valor de la rutas para mostrar las más económicas
  - ❖ Utiliza una estructura de datos eficiente y adecuada para la solución

5. Evaluación y Selección de Soluciones

Criterio A. Precisión de la solución. La alternativa entrega una solución:
- [2] Exacta (se prefiere una solución exacta)
- [1] Aproximada

- Criterio B. Eficiencia. Se prefiere una solución con mejor eficiencia que las otras consideradas. La eficiencia puede ser:
- [4] Constante
- [3] Mayor a constante
- [2] Logarítmica
- [1] Lineal

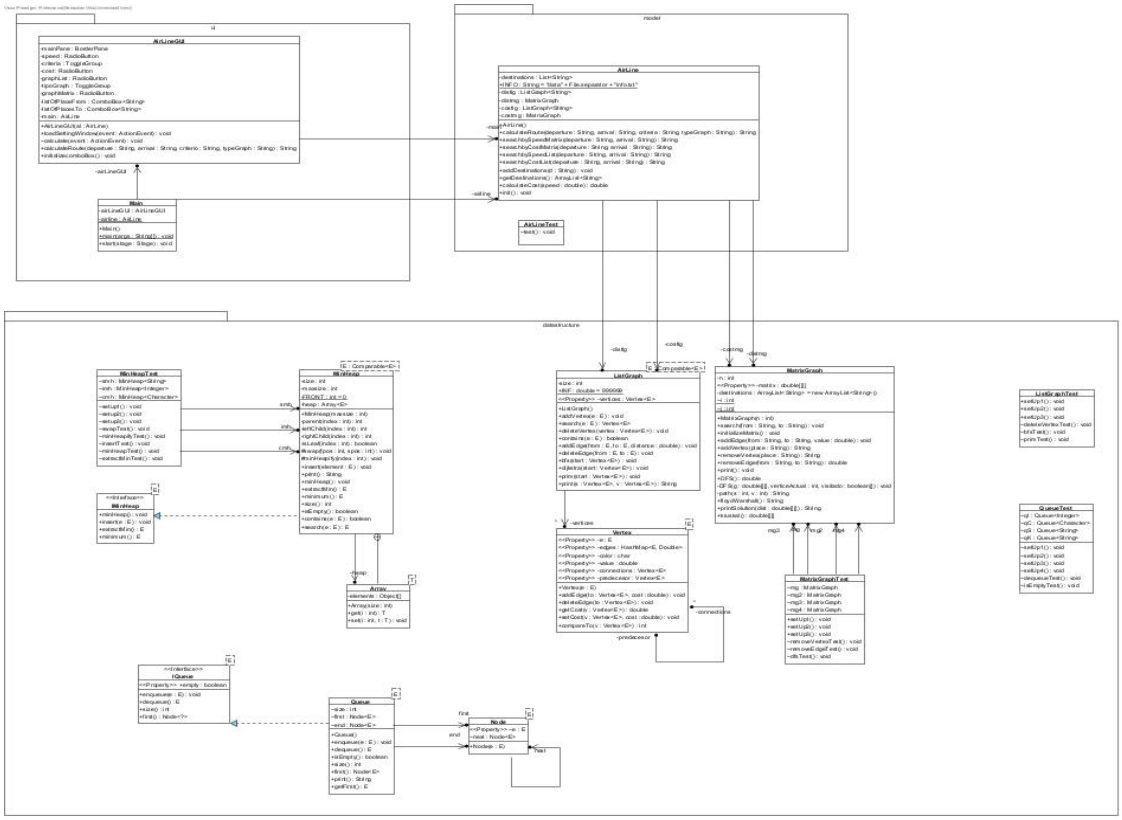
- Criterio C. Completitud. Se prefiere una solución que encuentre todas las soluciones. Cuántas soluciones entrega:
- [3] Todas
- [2] Más de una si las hay, aunque no todas
- [1] Sólo una o ninguna

- Criterio D. Facilidad en implementación algorítmica:
- [2] Cumple con todas las soluciones y a la vez se facilita la completa implementación del software
- [1] No cumple con todas las soluciones y no se facilita la completa implementación del software

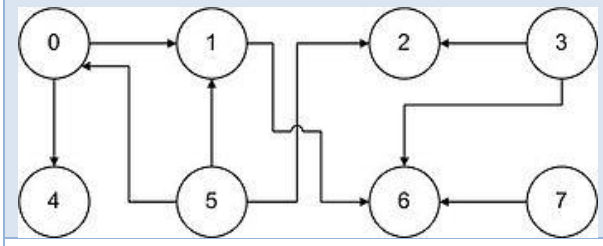
Opción	Criterio A	Criterio B	Criterio C	Criterio D	Total
1	1	2	1	1	5
3	1	1	2	2	6
4	2	2	3	2	9

En conclusión, se define la opción 5 como la mejor opción dado que esta cumple con los requerimientos solicitados, es más efectiva y utiliza una estructura de datos adecuada para la solución.

6. Preparación de informes



TAD adjacency matrix graph



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

- Para un grafo no dirigido la matriz de adyacencia es simétrica.
- El número de caminos $C_{i,j}(k)$, atravesando k aristas desde el nodo i al nodo j, viene dado por un elemento de la potencia k-ésima de la matriz de adyacencia:
  {inv: Ci,j(k)= $[A^k]_{ij}$ }

Operations:

- **Search** **from: String to: String** **-> void**
- **initializeMatrix** **->void**
- **addEdge** **from: String to: String Value: double** **->void**
- **addVertex** **place: String** **->void**
- **removeVertex** **place:String** **->void**
- **removeEgde** **from:String to:String** **->void**
- **print** **->void**
- **getMatrix** **[][] double**
- **dfs** **g:double[][] verAct: int visitado: Boolean[]** **->void**
- **floydWarshall** **String**
- **printSolution** **dis: [][]double** **String**
- **kruskal** **[][] double**

Search
"Once the data has been entered, to create the edge, verify that the vertices are created and if not, create them and add them to the matrix"

{pre: the graph must be created}
{post: a new place added to the list or to the matrix as the case may be}

Modifier

initializeMatrix
"is responsible for assigning the value 0 to all positions of the matrix"

{pre: the matrix must be created with n vertices}
{post: a matrix of n vertices with value 0}

Modifier

AddEdge
“adds an edge to the matrix according to its vertices assigns a value to the position”

{pre: an initialized matrix must exist}
{post: a new value in a position of the matrix}

Modifier

AddVertex
“takes care of adding a new vertex to the matrix”

{pre: an initialized matrix must exist}
{post: a new vertex in the matrix}

Modifier

removeVertex
“is responsible for removing a vertex from the matrix”

{pre: an initialized matrix must exist}
{post: one vertex less in the matrix}

Modifier

removeEdge
“removes an edge from the matrix”

{pre: an initialized matrix must exist}
{post: one less edge in the matrix}

Modifier

print
“takes care of printing the adjacency matrix”

{pre: an initialized matrix must exist}
{post: }

Analyzer

getMatrix
“returns the matrix”

{pre: an initialized matrix must exist}
{post: }

Analyzer

dfs
“traverse the graph advancing as far as possible, until at the moment when it is not possible to advance further, it goes back some step until there is again a new branch of the graph on which to advance”

{pre: an array created, initialized and with values corresponding to its positions}
{post: }

Analyzer

floydWarshall
“find the minimum path in weighted directed graphs. The algorithm finds the path between all pairs of vertices in a single execution”

{pre: an matrix created, initialized and with values corresponding to its positions }
{post: the minimum path between a pair of vertices }

Analyzer

printSoltion
“is responsible for printing the solution of the algorithm floyd warshall”

{pre: running the floyd warshall algorithm }
{post: the minimum path between a pair of vertices }

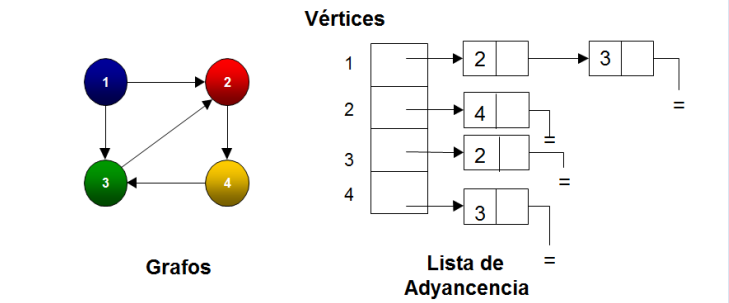
Analyzer

Kruskal
"find a minimal covering tree in a connected and weighted graph. That is, it looks for a subset of edges that, forming a tree, include all the vertices and where the value of the sum of all the edges of the tree is the minimum"

{pre: an matrix created, initialized and with values corresponding to its positions }
{post: the value of the sum of all edges of the tree }

Analyzer

TAD adjacency list graph



Inv: $\forall v \epsilon G, G: \{v_1, .., v_n\}$

addVertex	Vertex<E>	void
search	Vertex<E>	Vertex<E>
deleteVertex	E	void
addEdge	Vertex<E>, Vertex<E>, double	void
deleteEdge	Vertex<E>, Vertex<E>,	void
contains	E	boolean
bfs	Vertex<E>	void
dijkstra	Vertex<E>	void
prim	Vertex<E>	void

addVertex(v)
"Add a new vertex to the adjacent list graph."
pre: A vertex created
post: The vertex has been added
Modifier

searchVertex(e)
"Search the vertex in the adjacent list graph."
pre: The vertex exist
post: The vertex has been found and returned
Analyzer

deleteVertex(v)
"Delete a vertex in the adjacent list graph."
pre: The vertex must exist
post: The vertex has been deleted.

Modifier

addEdge(v,v,f)
“Add a new vertex to the adjacent list to the graph.”
pre: TRUE
post: The edge has been created.
Modifier

deleteEdge(v,v)
“Add a new vertex to the adjacent list to the graph.”
pre: The edge must exist.
post: The edge has been deleted.
Modifier

contains(e)
“Search if the adjacent list graph contains the vertex or not.”
pre: A graph with vertices.
post: If the graph contains the vertex.
Analyzer

bfs(v)
“Search the way with the minimum cost in the adjacent list graph.”
pre: A graph with vertices and edges and the start vertex.
post: The simple path with minimum cost has been found.

dijkstra(v)
“Search the path with the minimum cost in the adjacent list graph.”
pre: G: weighted connected simple graph, with
all weights positive.
post: A shortest path has been found.

prim(v)
“Search the path with the minimum cost in the adjacent list to the graph.”
pre: G: weighted connected undirected graph with n vertices.
post: The path with minimum cost has been found.