**Método de ingeniería TI3.**

Juan Esteban Duque Taborda – A00376778.

Sebastián Escobar Marín - A00374994.

**FASE 1: IDENTIFICACIÓN DEL PROBLEMA.**

El problema que debemos solucionar con base en el problema es que debemos realizar un sistema que pueda buscar el viaje más corto entre dos ciudades disponibles para poder generar un viaje entre estas. Este viaje puede ser con o sin escalas. Cuando el viaje se muestra con escalas debe realizar este mismo proceso, buscar el viaje más corto entre la primera ciudad con la segunda y luego de la segunda a la tercera. Además de esto, la aerolínea desea conocer si con las ciudades registradas se puede hacer un viaje a cualquier par de ciudades disponibles. En el momento en el que se encuentre una ciudad desde la cual no se pueda hacer un trayecto a todas las otras ciudades se avisará al usuario de que no es posible ir de todas las ciudades a todas, igual en el caso en el que una ciudad si pueda ir a todas las otras cuidades.

**Requerimientos:**

**R1.** El sistema debe permitir la creación de nuevas ciudades destino en el sistema, de un destino se almacena únicamente el nombre y las rutas que salen de él.

**R2.** El sistema debe permitir eliminar cualquier destino del sistema, eliminando con él cualquier ruta que se dirija o despegue de ese destino.

**R3.** El sistema debe permitir crear rutas entre cualquier par de destinos, de una ruta se almacena la ciudad de salida, ciudad de llegada y el tiempo promedio de vuelo.

**R4.** El sistema debe permitir eliminar una ruta, al eliminar una ruta únicamente se elimina la ruta, no las ciudades asociadas.

**R5.** El sistema debe calcular las rutas mínimas en tiempos entre cada par de ciudades cada que se modifican las rutas y ciudades almacenadas.

**R6.** El sistema debe verificar si es posible, con las rutas registradas, hacer un trayecto entre cualquier par de ciudades.

**R7.** El sistema debe permitir cambiar la representación del grafo en cualquier momento, este cambio se hará entre un grafo representado como matriz de adyacencia y un grafo representado como lista de vecinos.

**FASE 2: RECOPILACIÓN DE INFORMACIÓN NECESARIA.**

**Floyd-Warshall:**

El algoritmo de Floyd-Warshall compara todos los posibles caminos a través del [grafo](https://es.wikipedia.org/wiki/Grafo) entre cada par de vértices. El algoritmo es capaz de hacer esto con sólo {\displaystyle V^{3}} comparaciones (esto es notable considerando que puede haber hasta {\displaystyle V^{2}} aristas en el grafo, y que cada combinación de aristas se prueba). Lo hace mejorando paulatinamente una estimación del camino más corto entre dos vértices, hasta que se sabe que la estimación es óptima. Sea un grafo {\displaystyle V^{2}}{\displaystyle G} con conjunto de vértices {\displaystyle V^{2}}{\displaystyle G} {\displaystyle V}, numerados de 1 a N. Sea además una función {\displaystyle {\textrm {caminoMinimo}}(i,j,k)}que devuelve el camino mínimo de {\displaystyle i}***i*** a {\displaystyle j}***jjjj*** usando únicamente los vértices de 1 a ***k***{\displaystyle k} como puntos intermedios en el camino. Ahora, dada esta función, nuestro objetivo es encontrar el camino mínimo desde cada ***i***{\displaystyle i} a cada ***j***{\displaystyle j}***jj*** usando únicamente los vértices de 1 hasta ***k+1***{\displaystyle k+1}. Hay dos candidatos para este camino: un camino mínimo, que utiliza únicamente los vértices del conjunto ***(1…k)***{\displaystyle (1...k)}; o bien existe un camino que va desde ***i***{\displaystyle i} hasta ***k+1***{\displaystyle k+1}, y de ***k+1*** {\displaystyle k+1} hasta ***j***{\displaystyle j}, que es mejor. Sabemos que el camino óptimo de ***i***{\displaystyle i}***i*** a ***j***{\displaystyle j} que únicamente utiliza los vértices de 1 hasta ***k***{\displaystyle k} está definido por {\displaystyle {\textrm {caminoMinimo}}(i,j,k)}, y está claro que si hubiera un camino mejor de ***i***{\displaystyle i} a ***k+1***{\displaystyle k+1} a ***j***{\displaystyle j} , la longitud de este camino sería la concatenación del camino mínimo de ***i***{\displaystyle i} a ***k+1***{\displaystyle k+1} (utilizando vértices de ***(1…k)***{\displaystyle (1...k)}) y el camino mínimo de ***k+1***{\displaystyle k+1} a NKj{\displaystyle j} (que también utiliza los vértices en ***(1…k)***{\displaystyle (1...k)}).

**Dijkstra:**

La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen, al resto de vértices que componen el grafo, el algoritmo se detiene. El algoritmo es una especialización de la búsqueda de costo uniforme, y como tal, no funciona en grafos con aristas de coste negativo (al elegir siempre el nodo con distancia menor, pueden quedar excluidos de la búsqueda nodos que en próximas iteraciones bajarían el costo general del camino al pasar por una arista con costo negativo). Teniendo un grafo dirigido ponderado de N nodos no aislados, sea x el nodo inicial, un vector D de tamaño N guardará al final del algoritmo las distancias desde x al resto de los nodos.

1. Inicializar todas las distancias en D con un valor infinito relativo ya que son desconocidas al principio, exceptuando la de x que se debe colocar en 0 debido a que la distancia de x a x sería 0.
2. Sea a = x (tomamos a como nodo actual).
3. Recorremos todos los nodos adyacentes de *a*, excepto los nodos marcados, llamaremos a estos nodos no marcados vi.
4. Para el nodo actual, calculamos la distancia tentativa desde dicho nodo a sus vecinos con la siguiente fórmula: dt(vi) = Da + d(a,vi). Es decir, la distancia tentativa del nodo ‘vi’ es la distancia que actualmente tiene el nodo en el vector D más la distancia desde dicho el nodo ‘a’ (el actual) al nodo vi. Si la distancia tentativa es menor que la distancia almacenada en el vector, actualizamos el vector con esta distancia tentativa. Es decir: Si dt(vi) < Dvi → Dvi = dt(vi)
5. Marcamos como completo el nodo a.
6. Tomamos como próximo nodo actual el de menor valor en D (puede hacerse almacenando los valores en una cola de prioridad) y volvemos al paso 3 mientras existan nodos no marcados.

Una vez terminado al algoritmo, D estará completamente lleno.

**DFS:**

El algoritmo DFS (Depth-First Search) es una forma sistemática de encontrar todos los vértices alcanzables de un grafo desde un vértice de origen. Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto. Cuando ya no quedan más nodos que visitar en dicho camino, regresa, de modo que repite el mismo proceso con cada uno de los hermanos del nodo ya procesado.

Si G es un grafo un grafo conexo, el algoritmo de búsqueda en profundidad obtiene un árbol recubridor de G. Se trata de un grafo en el que aparecen todos los vértices de G, pero no todas sus aristas. El árbol recubridor no es único, depende del vértice de partida.

**FASE 3: BÚSQUEDA DE SOLUCIONES CREATIVAS.**

**1ra idea:** Utilizar **Floyd-Warshall** haciendo una matriz de adyacencia de todos los nodos para poder conocer los caminos incluyendo escalas de todas las ciudades de la aerolínea. Esto para conocer el camino más corto de un viaje específico que se postularía en la aerolínea. Este método es viable para el grafo que se realice con matriz de adyacencia.

**2da idea:** Utilizar **Dijsktra** aplicándolo a cada nodo para conocer las distancias menores de una ciudad a otra incluyendo escalas. Este consideraría el camino más corto sin revisar los otros posibles caminos por si llega a haber menor tráfico aéreo para la disponibilidad de la aerolínea. Esta opción es más viable para el grafo que se representa como lista de vecinos.

**3ra idea:** Para hallar si él grafo es fuertemente conexo se utilizará **BFS** para poder identificar si partiendo de cualquier nodo son visitados el resto de nodos.

**4ta idea:** Utilizar **DFS** para identificar si el grado es fuertemente conexo para descubrir si la aerolínea tiene conexiones de todas las ciudades disponibles entre ellas mismas.

**FASE 4: TRANSICIÓN DE LA FORMULACIÓN DE IDEAS A LOS DISEÑOS PRELIMINARES.**

Las ideas que son factibles con respecto a las anteriormente mencionadas son la **1ra idea**, **2da idea** y **4ta idea**. Esto porque las primeras dos ideas a pesar de solucionar el mismo problema, no se descarta ninguna de las dos, puesto que se necesita mostrar la matriz de adyacencia del grafo como su lista de vecinos, y cada método usa una de estas dos opciones para solucionar el problema. Y la última idea, porque se necesita conocer si el grafo es fuertemente conexo para identificar si se pueden hacer viajes, además, es más factible hacer un método recursivo que sería el **DFS** con respecto a tener un método iterativo que es el **BFS** que necesita otras estructuras como la cola de prioridad que es útil para la lista de vecinos, pero no para la matriz de adyacencia.

**FASE 5: EVALUACIÓN Y SELECCIÓN DE LA MEJOR SOLUCIÓN.**

Las soluciones seleccionadas son el método **Floyd-Warshall** y **Dijkstra** porque cada uno de estos funciona son más sencillos de implementar para una representación de grafo en específico, **Floyd-Warshall** funcionando con matrices de adyacencia, mientras que **Dijkstra** funciona con lista de vecinos. Para el otro caso, se eligió **DFS** puesto que era más sencillo de implementar para las dos representaciones de grafos.