**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, se debe poder dar un paquete en el que se ofrezca un viaje a cada ciudad disponible en la aerolínea y regresando al sitio de partida. Este último paquete, de igual modo, se buscará con las rutas más cortas entre las ciudades.

**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.

**BFS:**

El algoritmo BFS (Breadth-First Search) es una forma de encontrar todos los vértices alcanzables de un grafo partiendo de un vértice origen dado. Como en el algoritmo de búsqueda en profundidad, BFS recorre una componente conexa de un grafo y define un árbol de expansión.

Intuitivamente, la idea básica de la búsqueda en anchura es la siguiente: lanzar una ola desde el origen s. La ola golpea a todos los vértices situados a una distancia de una arista de s. Desde allí, la ola golpea a todos los vértices situados a una distancia de dos aristas de s, y así sucesivamente.

Este algoritmo de grafos es muy útil en diversos problemas de programación. Por ejemplo, halla la ruta más corta entre dos vértices cuando el peso entre todos los nodos es 1, cuando se requiere llegar con un movimiento de caballo de un punto a otro con el menor número de pasos, o para salir de un laberinto con el menor número de pasos, etc.

**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.

**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.

**3ra idea:** Para hallar el ciclo euleriano se utilizar bfs para poder identificar si se podría tener un viaje con todas las cuidades disponibles en la aerolínea comparando sus tiempos para obtener un viaje entre todas las ciudades recorriendo la menor distancia.

**4ta idea:** Utilizar dfs para hallar el ciclo euleriano para identificar si la aerolínea puede tener un paquete que incluya un viaje en todas las ciudades que esta presenta y regresando al mismo punto de partida. Además, consideraría si es la mejor ruta que puede tomar el viaje o si existe una más corta.