EDA – GRUPO 2

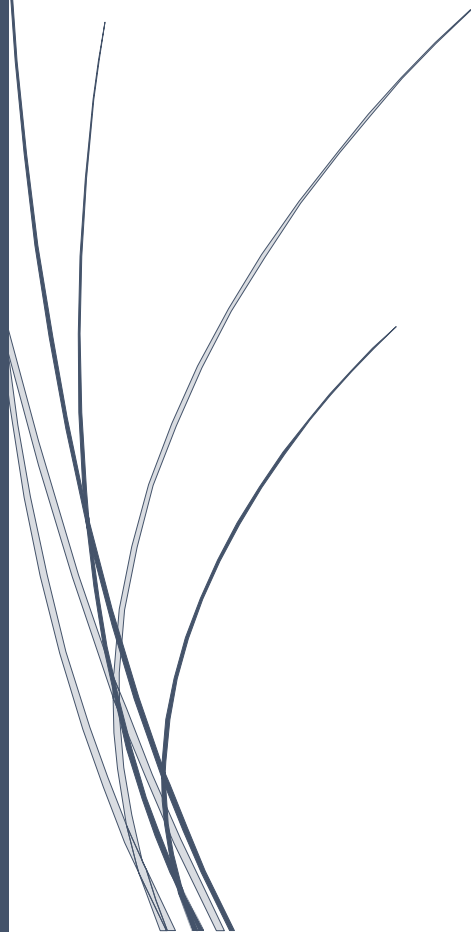
INFORME

TRABAJO N°2

Segundo Cuatrimestre 2017

Integrantes:

Della Sala, Rocío
Giorgi, María Florencia
Heimann, Matías
Rodríguez, Ariel Andrés
Santoflaminio, Alejandro



ITBA
Instituto Tecnológico de Buenos Aires

Find Route

En esta parte del informe se hablará sobre el método Find Route, se mencionarán las decisiones de eficiencia y de programación que se han tomado.

En el método, luego de verificar que los aeropuertos existan, se lleva a cabo el algoritmo de Djikstra. El mismo tiene como objetivo encontrar el camino, si es que existe, de menor peso bajo algún criterio pedido previamente.

El algoritmo de Djikstra utiliza una cola de prioridad. Nuestro equipo eligió que la cola contenga objetos de la clase PQNode (Priority Queue Node). Esta clase fue creada por nosotros como una clase auxiliar.

Los atributos de la clase PQNode son el peso que se utilizará como comparación, el precio acumulado, las horas de vuelo, las horas totales (de vuelo y de escala) hasta ese entonces, el día en el cual partió el vuelo, el vuelo, el aeropuerto del que partió el vuelo y una lista de PQNode previos. Esta decisión se tomó debido a que se retornan todos los PQNode del camino de menor peso encontrado. Luego se explicará por qué se decide retornar los PQNode.

Cuando se usa como criterio que el precio sea mínimo o que el tiempo de horas de vuelo sea mínimo sólo se utiliza una vez el aeropuerto origen y se pasa una vez por cada nodo. Esto es porque en ningún momento depende del día de salida o de algún factor que no sea las aristas. Se puede afirmar esto debido a que, sin importar el día de salida, el tiempo de vuelo y el precio del pasaje será el mismo.

No se puede decir lo mismo del tiempo total, debido a que el día de partida puede afectar seriamente el tiempo de la escala, así afectando el tiempo total. Por lo tanto, la decisión que tomó el equipo fue analizar todas las posibilidades del nodo origen, es decir, se toma en cuenta cada día de cada vuelo. Un ejemplo que podría respaldar nuestra decisión es que no es lo mismo tener el primer vuelo un Lunes y el segundo un Miércoles que tener el primer vuelo un Martes y el segundo vuelo el ya mencionado del Miércoles. El tiempo de escala en el segundo caso sería notablemente menor.

Como se mencionó previamente se utiliza una cola de prioridades a la cual se le ingresan PQNode. Se genera un PQNode por día en el que el vuelo puede salir, es decir, se crea uno para cada día en particular.

El tiempo total se calcula mediante la función getTimeDifference, que tiene como parámetro dos vuelos y el día en el que estos salen, y luego calcula la cantidad de minutos que pasan desde que arribo el primer vuelo al aeropuerto hasta que el siguiente llega al otro aeropuerto.

Como se dijo previamente, la función retorna una lista con todos los PQNode del camino. Esto se hace porque se pide que para la salida de la función se muestre cada vuelo, cada día en el que parte el vuelo, el precio total, el tiempo de vuelo y el tiempo total. Por lo tanto, al equipo le pareció mucho más simple guardarlo en la

estructura y retornar la lista de nodos. Esta decisión también facilitó la impresión en pantalla con el formato pedido y el armado del KML.

Por último, la ruta recibida será impresa por salida estándar, guardada en un archivo de texto, o guardada en un archivo KML (que también puede ser impreso en pantalla pero no cumple su principal función de poder verse la ruta óptima en el mapa).

Aclaraciones sobre Find Route:

En caso de elegir los criterios de precio o de tiempo de vuelo no se asegura que la ruta propuesta sea la de menor tiempo total, es decir, puede que exista un vuelo que tenga menor tiempo total debido a los días seleccionados. Esto ocurre debido a que el comparador sólo tiene en cuenta el criterio pedido y al encolarse en la lista de prioridades el primero en desencolarse será el día utilizado para proponer el vuelo.

World Trip

En este apartado se redactarán decisiones de eficiencia y programación que se tomaron en cuenta para la realización de “World Trip”.

Primero se tuvieron en cuenta ciertas restricciones sobre el grafo, para saber si al menos existía un ciclo hamiltoniano. De esta manera determinábamos muy rápidamente, si valía la pena o no buscar dicho ciclo.

Dichas restricciones sobre el grafo son:

1) El grafo debe ser fuertemente conexo (fuertemente, para grafos dirigidos, implica que para cada par de vértices existe un camino viendo el grafo con sus aristas dirigidas)
2) El grafo (no dirigido) no puede contener vértices de corte. Si el grafo dirigido tiene vértices de corte no es prueba suficiente de que no existe un ciclo hamiltoniano, más adelante mostraremos un ejemplo de este caso.

Para ver si el grafo era fuertemente conexo, se tomó un vértice inicial y se vio si con este vértice podíamos llegar a todos a través de un recorrido DFS. Luego tomamos los demás vértices y determinamos si podían llegar al vértice inicial anterior nombrado. De esta manera lográbamos corroborar que para cada par de vértices existe un camino. Más adelante mostraremos una demostración de porque el grafo debe ser fuertemente conexo para permitir la existencia de un ciclo hamiltoniano.

Para ver si el grafo no dirigido contenía vértices de corte, construimos un grafo idéntico al inicial, pero generando una arista dirigida hacia el lado del vértice que no existe, de esta manera las 2 aristas dirigidas funcionan como una arista no dirigida. Luego, por cada vértice, se realizó un recorrido DFS en el que dicho vértice ya estaba visitado para determinar si podíamos encontrar los vértices restantes y así determinar si dicho vértice es de corte o no. Más adelante mostraremos una demostración de porque el grafo no dirigido no debe contener vértices de corte para permitir la existencia de un ciclo hamiltoniano.

Mejoras en cuanto a la eficiencia tomadas en cuenta en el back-tracking del “World Trip”:

1) Si 2 pares de vértices están conectados por más de una arista, se elige la mejor de todas en función de la prioridad buscada por el usuario, para así eliminar muchos recorridos innecesarios. Una cuestión a tomar en cuenta, es que esto empeora la eficiencia espacial, ya que necesitamos por cada iteración una lista que marque que nodos fueron visitados por el nodo en cuestión, para descartar las subsiguiente multiaristas. Pero a lo sumo tendremos n listas con n nodos en el peor de los casos (en realidad, un poco

menos de lo dicho anteriormente), y tomando en cuenta que esta decisión mejora mucho la cantidad de tiempo empleado en el algoritmo, decidimos que era mejor tomar esta decisión.

2) Como en el algoritmo tenemos 2 listas, una “efficient” y la otra “current”, si durante el transcurso del algoritmo, “current” supera el valor de prioridad de “efficient”, entonces el algoritmo retorna puesto que se da cuenta que por ese camino no obtendrá un ciclo mejor, de esta manera también eliminamos recorridos innecesarios.

Nota: Siempre que hablamos de “n” hablamos de la cantidad de nodos.

Para dar algunos ejemplos que probamos, utilizando un backtracking que no tome en cuenta las eficiencias anteriores nombradas se obtuvieron los siguientes resultados:

Vértices	Aristas	Backtracking Completo	Backtracking eficientes
5	50	1532 iteraciones	78 iteraciones en general, dependiendo la prioridad
8	85	123990 iteraciones	1506 iteraciones en general, dependiendo la prioridad
10	95	563786 iteraciones	Entre 10000 y 21182 iteraciones en general, dependiendo la prioridad

Demostraciones:

Demostraremos que si el grafo no es fuertemente conexo, entonces no existe un ciclo hamiltoniano. Para ello haremos la contrarrecíproca.

Existe un ciclo hamiltoniano ⟶ El grafo es fuertemente conexo

Existe un ciclo hamiltoniano

⟶ Existe un ciclo V -> … -> … -> T -> V / V llega a todos y T llega a V

⟶ Para cualquier par de vértices A,B existe un camino que debe ser de la forma

1) A -> B si están conectados directamente
2) A -> … -> B si no están conectados directamente, pero el ciclo mostrado anteriormente los muestra conectados apareciendo A primero en el ciclo y luego B
3) A -> … -> T -> V -> … -> B si en el ciclo mostrado anteriormente aparece primero B y luego A.

→ El grafo G está conectado fuertemente

Demostraremos que si el grafo dirigido tiene vértices de corte, entonces no existe un ciclo hamiltoniano. Para ello haremos el absurdo.

Existen vértices de corte → Existe un ciclo hamiltoniano

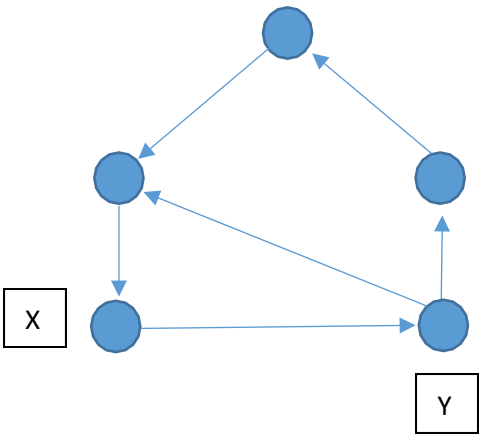
1) Empezamos el ciclo en el vértice de corte, recorremos la primera estructura conexa, pero no podemos recorrer la segunda estructura conexa, que está conectada por el vértice de corte ya que no sería hamiltoniano el ciclo. ABSURDO

2) Empezamos el ciclo en cualquiera de las estructuras conexas. Recorremos la estructura conexa, pasamos por el vértice de corte, recorremos la otra estructura conexa, pero para que el ciclo sea hamiltoniano, debemos volver al vértice inicial que está en la primera estructura conexa elegida, por lo que tenemos que pasar por el vértice de corte. ABSURDO

Queda demostrado que:

Existen vértices de corte → No existe ciclo hamiltoniano

Acá mostraremos un ejemplo de un grafo dirigido con vértices de corte que tiene ciclo hamiltoniano, para demostrar que el grafo no dirigo no debe tener vértices de corte para que exista un ciclo hamiltoniano



Como vemos existe un claro ciclo hamiltoniano, pero el vértice X, es vértice de corte puesto que al sacarlo, no existe camino entre Z, Y siendo Z cualquier vértice menos X.

Aclaración sobre Testing:

En el programa se pueden encontrar dos clases de tests mediante JUnit. Sin embargo, estos tests se limitan a comprobar el tiempo que demoran los dos algoritmos más importantes del sistema.

Manualmente se testearon los distintos métodos, ya que no era posible hacerlo mediante JUnit dado que la mayoría eran privados o si eran públicos eran void y no retornaban nada sino que imprimían en consola.