Cantidad de integrantes: 1.

Nombre: Genaro Facundo.

Email: FacundoGenaro@hotmail.com

Ayudante asignado:

Correcciones de la primera parte:

Las correcciones solicitadas en el código fueron hechas. Las del informe las enviaré por mail el sábado o la entregaré el miércoles.

Segunda parte:

Introducción al problema:

Para esta segunda entrega se solicitó resolver el problema del viajante utilizando Backtracking y un algoritmo Greedy.

El problema del viajante (Travelling salesman problema (TSP)) es un problema que plantea encontrar el camino más corto en un grafo para recorrerlo desde un vértice de origen, pasando por todos los demás vértices hasta llegar otra vez hasta el primero, sin pasar dos veces por el mismo vértice y con el menor costo posible.

Análisis del problema:

El problema requiere un grafo ponderado y no dirigido donde pueda encontrarse un ciclo hamiltoniano (un camino que recorra todo el grafo sin repetir vértices).

Backtracking: encuentra este camino en caso de que exista con pura fuerza bruta, recorriendo cada una de las posibilidades y volviendo atrás en caso de ser necesario para solucionar el problema planteado.

Características:

- Siempre va a encontrar una o más soluciones (en caso de que exista).
- Las funciones de poda achican el espacio de búsqueda y por lo tanto el tiempo de ejecución. Pero en el peor de los casos tendrá costo exponencial.
- Su implementación es relativamente sencilla.
- Es adaptable según el problema a resolver.
- Consume mucha memoria debido a tener que almacenar los ciclos de búsqueda.

- Al parecer su única limitación es la gran cantidad de memoria que consume, además del caso en que exista una solución pero sea infinita. Esto llevaría a que la solución nunca se encuentre.

Para implementar el backtracking decidí tomar como base mi implementación del servicio 2 de la primera parte de este trabajo practico y adaptarlo al problema solicitado.

(voy a dar en el folio un dibujo de mi grafo con un seguimiento de ambos algoritmos)

Para lograr el backtracking decidí adaptar un DFS que consta de una lista de Aeropuertos visitados, una lista de Rutas que será mi solución y una condición de poda.

La condición de poda fue implementada usando una interfaz por si en algún momento se decide utilizar otra condición de poda

Mi condición de poda es la siguiente: si la suma actual de las distancias (ósea la sumatoria de las distancias que recorrí hasta el momento) es menor que mi mejor distancia se permite avanzar, caso contrario se termina la iteración

Pequeña aclaración: Debido a que tuve muchos problemas trabajando con valores Float decidí pasar las distancias a BigDecimal.

Busco entre todos los aeropuertos al de origen y empiezo el recorrido recursivo donde a medida que obtengo rutas que satisfacen mi condición de búsqueda, voy podando las posibilidades para llegar a obtener el mejor camino que resuelva el problema del viajante.

A medida que encuentro posibles rutas, si se cumple la condición de poda la agrego a mi lista de rutas a retornar (cada vez que encuentro una ruta nueva que cumple con mi condición de poda, sé que esa ruta será mejor que la que ya tengo y por ende mi lista de rutas a devolver es limpiada antes de agregar una nueva ruta para preservar solo la mejor)

El backtracking sin poda encontró 18 posibles soluciones con el origen “Ministro Pistarini” y con poda encontró solo 4 (las cuales son eliminadas para conservar solo con la mejor).

Analizando los datos obtenidos por pantalla puedo concluir en que la cantidad de iteraciones recursivas en el método varían según que tan “lejos” tenga la posibilidad de encontrar una ruta cuya sumatoria de distancias sea menor que mi “mejor distancia”.

La condición de poda se ve funcional porque por lo general es distinta a la cantidad de entradas recursivas del método.

Greedy: intenta encontrar un camino recorriendo solo los vértices cuyas aristas tengan el costo más bajo. Este algoritmo intenta resolver el problema tomando decisiones en función a la información disponible en el momento, una vez que elige el camino a recorrer no hay vuelta atrás y no garantiza encontrar una solución (ni una óptima tampoco)

Características:

- Puede o no encontrar una solución. Esta solución puede o no ser la solución óptima.
- Con el criterio greedy va seleccionando el mejor candidato y guardándolo en el conjunto solución.

- Suelen ser rápidos y fáciles de implementar si tenemos en claro cual va a ser nuestro criterio greedy.
- Una vez que se toma una decisión, no se puede deshacer.
- Consumo de memoria muy bajo.

Para implementar el algoritmo greedy decidí seguir el pseudo código provisto por la catedra e intentar encontrar el camino más corto a partir de un origen yendo por la ruta de menor peso que esté en la lista de rutas del vértice en el que estoy parado.

El metodo greedy(origen) es el que tiene el trabajo de conseguir los vertices a seguir según el peso de sus rutas. Para esto utiliza una lista de grafos sin visitar y utiliza otros métodos para verificar si mi ArrayList Solución es una solución factible para el problema del viajante, en caso de no serlo retorna la lista de aeropuertos hasta donde los pudo obtener.

Haciendo el seguimiento de lo que se muestra en pantalla puedo deducir que el sistema de búsqueda de la ruta mas corta funciona bien, siempre selecciona de entre las rutas disponibles la de menor distancia y que siga estando en la queue.

El metodo isFactible() cumple bien su función también, la cual es verificar si el aeropuerto encontrado posee dentro de sus rutas un vértice sin visitar. Es un método boolean y según su retorno, se agrega dicho aeropuerto a lo lista solución y también se remueve de la queue o en el caso contrario solo agrega el aeropuerto y termina la función.

El metodo getRutasGreedy() obtiene los aeropuertos provistos por greedy() y en base a eso obtiene las rutas a seguir para retornar una lista de rutas con el camino mas corto que se pudo encontrar.

Conclusiones:

~~Hacer esto sin internet fue una pesadilla.~~

El algoritmo de backtracking que implementé se siente horriblemente “lento y pesado” pero creo que cumple su función de encontrar la ruta más corta en base a fuerza bruta.

Su implementación no causó mayores problemas, salvo por el hecho de que cuando quería implementar la poda, mis valores Float de las rutas eran muy imprecisos, lo que conllevó a cambiarlas por BigDecimal. Otro problema que tuve fue a la hora de hacer la poda, debido a que era un algoritmo recursivo no podía guardar apropiadamente los valores de mis variables para hacer las comparaciones de las distancias (sigo sin saber si esto era un error mío o no se podía hacer), lo cual solucioné utilizando variables en la clase.

El algoritmo greedy por otra parte, lo implementé en base al pseudo código del material de la catedra porque creo que es lo más correcto… me parece bastante funcional pero no pude llegar nunca a una solución utilizándolo. Mas allá de esos detalles su implementación no fue complicada, cumplió lo que prometió “Los algoritmos greedy suelen ser rápidos y fáciles de implementar”