EDA II

PRÁCTICA 2

Algoritmos Greedy

EL PROBLEMA DEL VIAJANTE

Implementación de algoritmos de Prim y Kruskal

Francisco Ramírez Vaquero

Introducción

El problema por resolver es obtener el árbol de recubrimiento mínimo, es decir, el árbol de recubrimiento cuyo peso total (suma de todas las aristas) sea el menor. Un árbol en este contexto es un grafo conexo acíclico. Para este problema los algoritmos greedy suelen funcionar bien, y de hecho ya existen algoritmos que resuelven este problema, que son los que se van a implementar.

Elementos de los algoritmos greedy:

Conjunto de candidatos (C) En este problema, el grafo del que queremos obtener el árbol de recubrimiento mínimo.

Solución parcial (S) Este algoritmo va construyendo la solución añadiendo elementos del conjunto de candidatos. En este caso, es una parte del árbol de recubrimiento mínimo.

Función de selección selecciona(C) Cómo elegir el siguiente elemento (arista, en este caso) que se va a añadir a la solución parcial.

Función de factibilidad Se comprueba si se puede añadir a la solución. En este caso, sería, por ejemplo, comprobar que al añadir la arista no se obtenga un ciclo.

Función de solución Comprueba si se ha llegado a la solución.

Función objetivo Es más bien teórica, se refiere a lo que se quiere maximizar/minimizar. En este caso, el objetivo es minimizar el coste o peso asociado a las aristas.

Estudio de la implementación

Para almacenar tanto el grafo como los algoritmos se ha creado la clase Graph. En ella se almacenan los métodos que aplican los algoritmos de Prim (con la variante usando una cola de prioridad) y Kruskal.

Al principio del algoritmo se comprueba que tenga ciclos, o “aristas de más”. Formalmente, que el número de aristas menos 1 sea mayor (estricto) que el número de vértices.

Algoritmo de Prim (sin cola de prioridad)

Se comprueba que haya una fuente, y si no hay una, se obtiene. La fuente que se elija no es relevante, por eso se escoge una sin preguntar al usuario.

Lo siguiente que se hace en la implementación es inicializar las variables y estructuras que se van a necesitar. Creamos un conjunto (Hash Set) en el que añadimos todos los vértices menos el vértice fuente. Este conjunto almacenará los vértices del grafo original que quedan por explorar.

Posteriormente, creamos dos mapas, uno en el que almacenaremos los pares de vértices, y otro en el que almacenaremos el peso de las aristas entre cada par de vértices. Se inicializan los valores de estos mapas con los vértices del conjunto de no visitados como clave en ambos, y el valor dependerá de si el vértice es adyacente al origen o no. Si es adyacente, el valor del mapa de vértices será el vértice de origen, y el del mapa de pesos será la distancia entre ambos vértices. Si no es adyacente, el valor del mapa de vértices es null, y el de pesos un valor muy grande. Para ello recorro el conjunto de no visitados y obtengo el peso entre el origen y cada uno de los vértices. Será adyacente si el peso no es null. Finalmente, inserto la clave fuente, asociando los valores fuente y 0 a los mapas de vértices y pesos, respectivamente.

Lo último que necesitamos antes de comenzar el algoritmo como tal, es instanciar la estructura en la que almacenaremos el resultado, que será un mapa de adyacencias, y una variable auxiliar que almacenará el vértice que estamos examinando.

Ahora es cuando realmente comienza el algoritmo de Prim, que se ejecutará mientras haya elementos en el conjunto de no visitados.

Primero se obtiene el menor peso asociado a los vértices que hay en no visitados. Una vez se ha encontrado (llamado leastWeight), se elimina y obtenemos el vértice que hay asociado en el mapa de vértices (llamado vertAssoc). Habiendo obtenido ambas aristas y el peso, se añaden a la estructura final.

Ahora se actualizan los mapas de pesos y vértices recorriendo los vértices que continúan sin visitarse, comprobando si leastWeight hace menor algún peso.

Se han elegido las estructuras Hash Set y Hash Map porque su complejidad al insertar y buscar es casi constante, y el orden no es relevante.

Prim con PriorityQueue

La inicialización es parecida, las diferencias empiezan con el algoritmo en sí. Utiliza la estructura final, un conjunto de aristas, y una auxiliar, una cola de prioridad de aristas.

El algoritmo se ejecutará mientras queden aristas por visitar. Igual que cuando no había cola de prioridad, se recorren los vértices adyacentes al vértice de inicio, y sólo se añaden a la cola de prioridad los que no estén visitados. Una vez terminado este recorrido, se saca el menor elemento de la cola, y esto se ejecutará hasta que se obtenga un elemento que esté en la lista de no visitados, para evitar ciclos (se guardaría más de una vez el mismo elemento). Dicho elemento se guardará en la estructura final y se actualiza el vértice de origen al que en esta ocasión ha sido el de destino.

Kruskal

El algoritmo de Kruskal se basa en tratar las aristas como subgrafos, e ir juntando esos subgrafos hasta conectar todos los vértices.

Se inicializa también una estructura con todos los vértices (que serán los no visitados), pero en este caso también le añadimos el coste (será un Hash Map). Entonces se inicializa otra estructura en la que se almacenarán pares de vértices. Esta estructura nos servirá de apoyo, y será en la que nos apoyaremos para almacenar los datos que finalmente volcaremos en la estructura final.

Para rellenar esa estructura, primero obtenemos el menor peso (o coste) y el vértice asociado del conjunto de no visitados y se elimina de la estructura. Entonces se recorren los vértices conectados al seleccionado, y en caso de obtener un peso menor, se guarda en la estructura. Finalmente, se vuelcan los vértices en la estructura final.

Estudio experimental

El generador de grafos aleatorio permite crear grafos conexos dirigidos y no dirigidos, especificando un número de aristas concreto. Los ficheros utilizados para las pruebas de rendimiento se han creado utilizando este método. Se ha llegado a aceptar un máximo de 1.000 aristas por vértice, ya

que por motivos de rendimiento no era factible aceptar más aristas y no es habitual que se de una situación así en una red de carreteras o instalación eléctrica.

Para conseguir que el grafo siempre sea conexo, se comprueba que el número de aristas siempre sea mayor o igual que el número de aristas – 1. En caso de que sea menor, se modifica el número de aristas para garantizar que sea conexo.

Una vez obtenidos los límites tanto inferior como superior, se comienza desde el vértice 0 creando una arista con el vértice 1, y generando un peso aleatorio. Para facilitar esta tarea, los vértices se han nombrado utilizando números naturales. Como estos grafos sólo sirven para probar tanto el rendimiento como el correcto funcionamiento de los distintos algoritmos, el nombre no es relevante.

Anexo

Fuentes

https://es.wikipedia.org/wiki/Algoritmo_de_Prim

https://es.wikipedia.org/wiki/Algoritmo_de_Kruskal

Apuntes de clase

https://visualgo.net/en/mst

https://examples.javacodegeeks.com/core-java/io/bufferedoutputstream/java-bufferedoutputstream-example/

https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

https://www.geeksforgeeks.org/kruskals-minimum-spanning-tree-algorithm-greedy-algo-2/