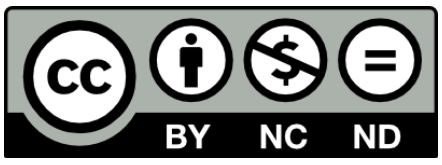
Grafos

Los Del DGIIM, `losdeldgiim.github.io`

Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada



Grafos

Los Del DGIIM, losdeldgiim.github.io

José Juan Urrutia Milán

Granada, 2023-2024

Índice general

Ejemplo. Dados varios ficheros de libros (algunos parecidos pero con leves diferencias), también tenemos libros incluidos dentro de otros. El problema es quedarnos con un conjunto de libros suficientemente pequeño que no contemple libros repetidos (o muy similares).

Podemos llegar a una solución a este problema mediante un grafo, de forma que ciertos libros estén conectados si tienen una estructura similar (mayor que un cierto umbral). Después de conectar todos los nodos similares, sustituiremos la estructura de nodos conectados por un sólo nodo, que cumpla una cierta propiedad (menor tamaño, mayor similitud con el resto, ...).

Definición 0.1 (Grafo)**.** Un grafo es un conjunto de vértices (nodos) y de aristas (que unen los vértices).

- En el caso de que las aristas tengan un inicio y una final determinado (es decir, un sentido), decimos que tenemos grafos dirigidos. En contraparte, grafos no dirigidos.
- Si las aristas presentan un peso (un valor), hablamos de grafos ponderados.

Conceptualmente, todo lo siguiente está permitido (aunque su permisibilidad depende del prblema):

- Unir dos vértices con más de una arista.
- Unir un vértice consigo mismo.

Definición 0.2 (Grafo completo)**.** Decimos que un grafo es completo si dados dos vértices cualesquiera, existe una arista que los une. Es decir, si desde cualquier vértice podemos llegar a cualquier vértice mediante una única arista.

En los grafos dirigidos, sólo podremos avanzar en el sentido de las aristas y no al revés.

Definición 0.3 (Grafo denso)**.** Un grafo es denso si el número de aristas que posee es cercano al número máximo de aristas (es decir, al que tendría si fuese completo).

Definición 0.4 (Grafo plano)**.** Un grafo se llama plano si puede dibujarse en un plano o esfera (espacios homeomorfos) sin que se crucen sus aristas.

Definición 0.5 (Grado de un vértice)**.** El grado de un vértice es el número de aristas que pasan por dicho vértice. En caso de tener una arista que salga de un nodo y llegue a él, añade un grao de 2.

Definición 0.6 (Camino)**.** Es una sucesión (finita) de aristas adyacentes, donde se supone que no hay aristas repetidas en el camino (es decir, sin pasar dos veces por la misma arista).

En grafos dirigidos, la orientación de las aristas es relevante a la hora de considerar caminos.

Definición 0.7 (Ciclo)**.** Un ciclo es un camino que comienza y termina en el mismo vértice.

Definición 0.8 (Grafo conexo)**.** Un grafo se dice conexo si para cualquier par de vértices existe un camino que los una. Es decir,

Teorema 0.1 (Teorema de Euler)**.**

- *Si todos los vértices de un grafo son de grado impar, no existen circuitos eulerianos.*

-

Definición 0.9. Un camino Hamiltoniano es un camino que pasa a través de ada vértice una y sólo una vez, y termina en el vértice que comienza.

Definición 0.10. Un árbol es un grafo conexo que no tiene ciclos.

En caso de ser un grafo dirigido, decimos que tiene una relación padre, descendiente (o hijo).

Definición 0.11. Un poliárbol es un grafo dirigido que si lo consideramos no dirigido (puede ser dirigido), resulta conexo. No puede tener ciclos.

0.1. Representaciones

0.1.1. Matriz de adyacencia

Dado un grafo de n vértices, lo representamos con una matriz $n \times n$, de forma que `a[i][j]` es 1 si los vértices `i` y `j` están conectados mediante una aritsta.

Pros

Simplicidad.

Contras

Una contra de esta representación es el alto coste a la hora de almacenarlo, ya que es de eficiencia $O(n^2)$ en memoria.

0.1.2. Vector de listas de adyacencia

Para cada vértice, tenemos una lista con el conjunto de aristas que salen de él (de forma que en la lista almacenamos el vértice al que llega cada arista).

Podemos representarlo como un vector de listas o como un vector (o conjunto) de pares (la primera coordenada es el vértice del que sale una arista y el segundo al que llega, por ejemplo).

0.1.3. Matriz de incidencia

Para cada vértice, determina qué arcos comienzan por dicho vértice. Dado un grafo, numeramos los vértices y a las aristas. De forma que `a[i][j]` será 1 si del vértice `i` comienza la arista `j` hasta otro vértice.

Presenta una eficiencia de $O(n \cdot p)$, siendo n el número de nodos y p el número de aristas.

En caso de ser dirigido, contamos con 1 o -1 en los valores de la matriz (donde 1 es que parte del vértice y -1 que llega a él).

0.2. Recorridos

0.2.1. Recorrido en anchura (BFS)

Parto de un nodo y visito a todos los adyacentes a ese nodo. Una vez visitados todos, cogemos el primero visitado y visitamos a todos sus adyacentes. Implica que tenemos que usar una cola para mantener los nodos no visitados.

0.2.2. Recorrido en profundidad (DFS)

Cuando llega a uno nodo, se va a el primer adyacente (hasta abajo).
Los nodos que quedan por visitar los tenemos en una pila.

0.3. Arbol generador minimal

Dado un grafo, buscamos un subgrafo conexo tal que la suma de las aristas sea mínima.

Podemos razonar que la solución al problema es un árbol.

0.3.1. Algoritmo de Kruskal

Conocidas sólo las aristas: cogemos las de menor peso. Comprobamos si la elección es factible (proponemos que no genere ciclos). Cuando tengamos $n - 1$ arcos, sabremos que el árbol es conexo, luego el problema estará terminado.

Demostración. Demostremos que esta solución Greedy alcanza la óptima:

1. Existe un árbol minimal de costo mínimo que contiene a la primera arista que toma el algoritmo Greedy.

   Supongamos que existe una solución optimal que no tiene dicha arista:

   - Si al introducir la arista de menor coste se genera un ciclo (también es un ciclo dos aristas que unen los mismo vértices), quitamos cualquier arista (que no sea esa) y que siga dando un árbol conexo.

   Hemos llegado a una solución mejor que la optimal, contradicción.

2. Existen subestructuras optimales. Quitamos la primera arista que escoge el algoritmo Greedy. A continuación, tratamos de demostrar que quitada dicha arista, se nos quedan dos soluciones (cada una de las componentes conexas) para ahora los conjuntos de puntos que no están conectados por quitar la arista anterior. Si una de las subsoluciones no fuese solución, tendríamos que hay una óptima del subproblema que, si le añadimos el arco que antes quitamos, tenemos una óptima menor que la anterior, luego no era la óptima. Contradicción.

□

Tiene una eficiencia de $O(A \cdot V)$.

Función de factibilidad

Debe determinar si una elección es factible. Para ello, comprobamos si se forma un ciclo al coger una solución.

Usamos las compoentes conexas: si nuestra elección une dos componentes conexas, es factible.

Proponemos la siguiente estructura de datos: una lista (cada elemento apunta al siguiente) donde todos los nodos apuntan también a la raíz, de forma que la raíz sepa quienes son el primer y último elemento de la lista. Esta estructura recibe el nombre `union find`.

Algoritmo de Kruskal

El algoritmo de Kruskal sigue la heurística Greedy planteada pero de forma eficiente. Hace uso de la estructura `union find`. Puede garantizarse que la unión se realiza en orden $O(\log V)$. De esta forma, Kruskal tiene un orden $O(A \log V)$.

0.3.2. Algoritmo de Prim

Elegimos un vértice del grafo.

En cada paso, elegimos la arista de menor costo que une un nodo del conjunto de seleccionados con uno del conjunto de no seleccionados. Este nuevo nodo lo metemos al conjunto.

Funcion de factibilidad: no es necesaria, siempre elegimos nodos seleccionados con no seleccionados.

Mejora

Buscamos reducir el orden de eficiencia de selección de aristas.

Almacenamos en cada nodo cual es la arista de menor costo que lo une con el conjunto de nodos seleccionados, notando el costo a dicho conjunto y el nodo de los seleccionados con el que conectamos.

Tras cada elección, deberemos actualizar dicha lista. La actualizamos de forma que sólo actualizamos los valores de los nodos conectados con el nuevo nodo.

0.4. Problema de Caminos Mínimos

Dado un grafo y dos nodos del grafo, buscar el camino mínimo que los una.

El algoritmo Greedy es capaz de dar solución óptima al problema siempre y cuando no tengamos caminos con pesos negativos. Sí que la alcanzaríamos con costos nulos.

0.4.1. Algoritmo de Dijkstra

Dado un grafo y un nodo del mismo, es capaz de determinar el camino de costo mínimo desde dicho nodo hasta cualquier nodo del grafo.

Es aplicable tanto a grafos dirigidos como no dirigidos.

Supuesto que no hay caminos negativos: Propiedades

1. Es fácil probar que existen subestructuras optimales. Es decir, una solución óptima al problema está compuesta de subsoluciones que son óptimas para problemas que resuelven.

2. $M(u, v) <= M(u, s) + M(s, v)$ para cualquier nodo s. Siendo M la longitud de camino minima.

Del conjunto de candidatos no conozco el camino de costo minimo al origen. Sin embargo, en cada paso voy a calcular el camino de costo minimo pasando sólo por el conjunto de nodos seleccionados. Si no existe, no ponemos costo (o infinito).

Camino especial: Un camino que existe desde el origen hasta cualquier nodo que pasa solamente por el conjunto de vértices seleccionados, salvo el último.

En cada paso, escoger el nodo de los no seleccionados con una menor distancia al origen. Cada vez que incluimos un nodo tenemos que actualizar la información. Sólo actualizamos la información de los nodos adyacentes al obtenido, que son los susceptibles de tener alterado su camino mínimo al conjunto de nodos especiales. Si el camino anterior es mayor, actualizamos el camino. Si no, no.

La implementación más sencilla es $O(V^2)$. Con cola de prioridad es $O(A \cdot \log V)$. La sencilla es deseable para grafos densos y la de prioridad para grafos dispersos.

Demostración. Para cada v del conjunto de seleccionados, coincide con el camino minimo desde el origen hasta dicho vértice. Por inducción sobre el tamano del conjunto de vertices (caso $n = 0$, trivial).

En el paso de inducción, supongamos que el algoritmo de Dijkstra añade el vértice v. Dicho vértice representa la longitud de un camino de S a v.

Si al añadirlo no es el camino mínimo desde S a v, entonces tiene que haber otro camino de costo menor entre s y v. Dicho camino no puede pasar sólo por vértices de S (porque si no, el que hemos cogido que era el padre de v no era el padre de v).

Sea x el último nodo de los seleccionados que se encuentra en dicho camino y y el último vértice de dicho camino fuera de los seleccionados.

$$\begin{aligned} d(v) > M(s, v) &= M(s, x) + c(x, y) + M(y, v) \\ &\geqslant M(s, x) + c(x, y) = d(x) + c(x, y) \geqslant d(y) \end{aligned}$$

Por tanto, en el paso de elección de v no lo habríamos escogido, sino que habríamos escogido y. Contradicción. □

Observación. No podemos usar `priority_queue`, al tener que actualizar los adyacentes.

Tener algo que permita obtener el menor y tener algo que permita actualizar el menor. Le vale un vector.

0.4.2. Problema del Viajante de Comercio

Nunca podremos alcanzar la óptima. En este problema priorizamos el mejor resultado a la mejor eficiencia.

Hay distintas heurísticas Greedy que pueden dar una solución.

0.4.3. Problema de la Mochila

Si ahora los objetos no pueden fraccionarse (el objeto se selecciona o no), el algoritmo Greedy no es capaz de encontrar la solución óptima. Podemos dar heurísticas Greedy que nos lleven a una solución.