GRAFOS − FAQ − v3.0

Sep 2016

1.- ¿Por qué surgen los grafos?

Porque se hacía necesaria una estructura que nos permitiese representar relaciones binarias entre elementos de un conjunto. Los grafos permiten estudiar las interrelaciones entre unidades que interactúan unas con otras.

2.- Explica las razones por las que no es necesario marcar los nodos visitados al realizar el recorrido de un grafo. -> *Examen 2014*

Si es necesario marcar los citados nodos. Existen dos razones. La primera es para no procesar un nodo más de una vez en grafos cíclicos (cuando llegue a un nodo visitado no lo vuelvo a procesar). La segunda razón es para poder procesarlos todos si existe más de una componente conexa en el grafo.

3.- ¿Puede recuperarse un grafo no dirigido a partir de sus recorridos en anchura y profundidad?

No, ya que ambos recorridos (anchura y profundidad) nos devuelven una lista con los nodos que contiene el grafo, pero no nos proporciona información sobre las aristas (existentes o no) entre los diversos nodos.

4.- ¿Se podría mejorar el algoritmo de Dijkstra utilizando un APO? **Revisar.**

Sí, es posible mejorarlo siempre y cuando los nodos estén formados por una estructura que contenga el coste del arco y el vértice que une. Además, la relación de orden del APO debe ser el coste del arco determinada por la operación '<'. Esta modificación sustituye el bucle que usamos para localizar el camino mínimo entre los vértices aún no cerrados por una extracción de la cima del APO. De esta forma conseguimos mejorar su orden original O(n^2), a O(n*log(n)).

5.- El algoritmo de Dijkstra, ¿funciona correctamente con valores negativos?

No, porque no garantizaría los mínimos locales. 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 de costo negativo.

6.- ¿Por qué no se permiten los costes negativos en el algoritmo de Dijkstra?

No, porque no garantizaría los mínimos locales. 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 de costo negativo.

7.- ¿Por qué se coloca infinito en la diagonal principal de la matriz de costes en el algoritmo de Floyd?

En el algoritmo de Floyd la diagonal principal no se establece a infinito, sino a cero, representando de esta manera que el coste de ir de un vértice a sí mismo es nulo. Si suponemos que entre un vértice 'A' y sí mismo existe un camino cíclico y hemos colocado la diagonal principal a infinito (coste A->A = Infinito), en este caso nos daría como resultado que tiene menor coste realizar dicho camino que no movernos de A.

8.- ¿Por qué no se permiten los costes negativos en el algoritmo de Floyd?

Para que haya coherencia numérica. 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 de costo negativo.

9.- ¿Por qué en Floyd se inicializa a cero la diagonal de la matriz resultado?

En el algoritmo de Floyd la diagonal principal no se establece a infinito, sino a cero, representando de esta manera que el coste de ir de un vértice a sí mismo es nulo. Si suponemos que entre un vértice 'A' y sí mismo existe un camino cíclico y hemos colocado la diagonal principal a infinito (coste A->A = Infinito), en este caso nos daría como resultado que tiene menor coste realizar dicho camino que no movernos de A.

10.- ¿Por qué el algoritmo de Prim asegura que no se producen ciclos? **Editada. Revisar.**

Prim asegura que en su algoritmo no se producen ciclos utilizando la siguiente técnica: partiendo de un nodo inicial, va construyendo el árbol generador de coste mínimo marcando los nodos que va visitando. De esta forma, nunca seleccionará una arista que conecte un nodo visitado con un nodo que también esté visitado, evitando así que se produzcan ciclos.

11.- ¿Qué condición debe de cumplir un grafo no dirigido para poder utilizar el algoritmo de Prim?

El algoritmo de Prim encuentra un subconjunto de aristas que forman un árbol con **todos los vértices** del grafo, en donde el peso total de todas las aristas en el árbol es el mínimo posible. Por tanto, la condición es que el grafo debe ser **ponderado y conexo**, ya que si no fuera conexo, sería imposible añadir los nodos inconexos al árbol generador y, si no fuera ponderado, no podríamos obtener un árbol de coste mínimo.

12.- Comente la siguiente afirmación: "Prim y Kruskal resuelven el mismo problema y dan la misma solución". -> ***Examen 2014***

Ambos resuelven el mismo problema que sería el encontrar un árbol generador de coste mínimo, pero no tienen por qué devolver el mismo resultado.

Este resultado no depende únicamente de que los algoritmos sean diferentes, sino además de que el árbol generador de coste mínimo no tiene por qué ser único; depende de la existencia de distintas aristas con el mismo peso. Esto puede implicar que existan varias soluciones factibles que dan lugar a árboles generadores de coste mínimo distintos, si las condiciones iniciales en un algoritmo (el vértice inicial) cambian.

El anterior hecho provoca que el algoritmo pueda tomar diferentes caminos para llegar a soluciones distintas. Por supuesto, si no existen aristas con el mismo peso, la solución será la misma para ambos algoritmos. Indicar además que en todos los casos la suma total de los pesos coincidirá siempre.

13.- ¿Por qué el algoritmo de Kruskal asegura que no se producen ciclos? **Editada.**

Porque el algoritmo trabaja con una partición que contiene todos los nodos, usando esta para comprobar constantemente que componentes del grafo se encuentran conexas (es decir, en el mismo subconjunto), de forma que nunca se va a coger una arista que tenga sus dos nodos dentro de un mismo subconjunto. Si esto llegara a suceder, se produciría un ciclo, puesto que se está insertando una arista a una componente ya conexa, cerrando de esta forma un ciclo.

14.- ¿Es necesario ordenar las aristas en el algoritmo de Kruskal?

El algoritmo de Kruskal es del orden de O(log *n*) en donde *n* es el número de vértices. Para poder conseguir esta complejidad es necesario que las aristas sean ordenadas por su peso usando una ordenación por comparación del orden de O(*m* log *m*), en donde *m* es el número de aristas, lo que permitiría eliminar la arista de peso mínimo en tiempo constante. En el algoritmo de Kruskal que se nos ha propuesto se puede conseguir mediante la utilización de un APO.

15.- ¿Cuál es la condición que debe cumplir un grafo no dirigido para que Kruskal obtenga un resultado?

El algoritmo de Kruskall encuentra un subconjunto de aristas que forman un árbol con **todos los vértices** del grafo, en donde el peso total de todas las aristas en el árbol es el mínimo posible. Por tanto, la condición es que el grafo debe ser **ponderado y conexo**, ya que si no fuera conexo, sería imposible añadir los nodos inconexos al árbol generador y, si no fuera ponderado, no podríamos obtener un árbol de coste mínimo.

16.- ¿Por qué Kruskal no devuelve un grafo? **Editada.**

En nuestra representacion, Kruskal devuelve un grafo, en el que su interior se encuentra el arbol generador,

Se puede optar por cualquier estructura de datos que nos permita representar el árbol generador que Kruskal desarrolla, como pueden ser una matriz, un grafo, un árbol general, etc... Indicar que en la implementación del algoritmo que se nos ha facilitado se devuelve un grafo.

17.- ¿Qué pasaría si Prim y Kruskal operaran sobre un grafo dirigido?

Que no funcionarían correctamente y retornaría un resultado erróneo. El motivo es que ambos están diseñados para trabajar con aristas de grafos no dirigidos, no cumplimos dicha precondición. En el caso del algoritmo de Kruskal que se nos ha facilitado en la asignatura, para copiar las aristas del grafo en el APO, sólo se recorre la diagonal superior de la matriz, ya que se supone simétrica, luego ya estamos presuponiendo que el peso entre dos vértices es el mismo en ambas direcciones. En el caso del algoritmo de Prim que se nos ha facilitado en la asignatura, el coste de la arista se incorpora de forma simétrica en el árbol, ocurriendo lo mismo que en Kruskal.

18.- Dado el algoritmo de Kruskal implementado mediante el TAD Partición, ¿son los mismos árboles los de la partición y los del algoritmo? **Revisar.**

No son los mismos (o es muy improbable) ya que cuando acaba Kruskall, la estructura del árbol se almacena en el objeto partición, depende del orden de inserción de las aristas(por peso) y la raíz de ese árbol no tiene porque coincidir con la del árbol devuelto por el algoritmo.

19.- ¿Qué se consigue con la técnica de unión por tamaño? ¿Y con la unión por altura?

En la unión por tamaño, el árbol con menos nodos se convierte en subárbol del que tiene mayor número de nodos. En la unión por altura, el árbol menos alto se convierte en subárbol del otro.

En ambas técnicas se controla la altura del árbol resultante, consiguiendo que la operación unir() sea del orden de O(1) y que la operación encontrar sea del orden de O(log n).

20.- ¿En la representación mediante bosques de árboles con unión por altura, por qué las raíces de los árboles se representan con números negativos?

Para distinguir si el dato almacenado en el vector (cuyo índice se corresponde con el nodo) se refiere a la altura (número negativo y por lo tanto se trata de un nodo raíz) o a su padre (número no negativo y por tanto no se trata de un nodo raíz). Para saber la altura, al dato almacenado como número negativo se le debe sumar 1 y posteriormente obtener su valor absoluto.

21.- Dado el TAD Partición, decir qué combinación entre estructura de datos y estrategia es necesaria para que tanto la búsqueda como la unión sean de O(1). **Editada. Revisar.**

Es imposible conseguir que ambas sean a la vez operaciones de O(1). Utilizando control de altura podremos conseguir para la búsqueda un orden de O(log n) y para la unión un orden constante. Si además unimos la estrategia de comprensión de caminos, podemos lograr acercarnos a un orden constante en la búsqueda pero nunca llegar a O(1).

En el caso contrario, para obtener la búsqueda de O(1), lo mejor que podemos conseguir para la unión es O(n), siendo la más recomendable la representación con control de la longitud.

22.- En el TAD Partición, ¿es posible emplear la unión en altura y la compresión de caminos a la vez?

Si. La unión por altura garantiza el orden logarítmico en la búsqueda. La compresión de caminos permite acercarnos a un coste constante.

23- ¿Qué beneficio aporta la compresión de caminos en la implementación del TAD partición? -> *Examen 2014*

En la implementación del TAD partición mediante bosques de árboles y **mediante unión por tamaño o por altura**, la compresión de caminos se utiliza para poder acercarnos a órdenes constantes en la función encontrar. Esto lo conseguimos haciendo que los nodos por los que pasamos sean hijos del nodo raíz. Indicar que esta técnica permite acercarnos a coste 1, aunque no se garantiza.

Nota: La unión por tamaño o por altura garantiza el orden logarítmico en la búsqueda. La compresión de caminos permite acercarnos a un coste constante.

24.- Diferencias y similitudes entre Prim y Kruskal. **Editada.**

Ambos algoritmos resuelven el mismo problema, que sería encontrar en un grafo conexo, no dirigido y ponderado; un árbol generador de coste mínimo.

Kruskal y Prim se diferencian en la metodología usada para la obtención del resultado y evitar la generación de ciclos. Esta diferenciación radica en que Prim parte de un nodo cualquiera y construye el árbol a partir de él marcando los nodos que va visitando, mientras que Kruskal trabaja con particiones para asegurar que no se producen ciclos.

También difieren en su orden de complejidad computacional temporal. Mientras que el algoritmo de Prim es de O (n2); el de Kruskal es más eficiente, siendo su orden de O (a log n).

Nota: Se podría explayar mucho más, pero creo que con lo básico es suficiente.

25.- ¿Qué ventajas e inconvenientes plantea el uso de matrices de adyacencia y costes para la representación de grafos?

Como ventaja, son muy eficientes para comprobar si existe una arista entre un vértice y otro. Sin embargo, tienen el inconveniente de que pueden desaprovechar gran cantidad de memoria si el grafo no es completo. Además, si la matriz no es dinámica, no se pueden añadir o eliminar vértices del grafo.

26.- ¿Qué ventajas e inconvenientes plantea el uso de listas de adyacencia para la representación de grafos?

Esta representación aprovecha el espacio de memoria, pues solo se representan los arcos existentes en el grafo. Sin embargo, son pocos eficientes para determinar si existe una arista entre dos vértices del grafo, pues esta operación implica recorrer una lista. Además, si no se utilizan vectores dinámicos, no se pueden añadir o eliminar vértices de los grafos.

27.- ¿Existe alguna estructura para la representación de grafos que permita añadir y suprimir vértices?

Si. Estas operaciones son factibles si se usan como estructuras matrices y vectores dinámicos, o una lista de listas de adyacencia.