Estructuras de Datos y Algoritmos – IIC2133

Preparación para el C5 (miércoles 22/5)

16 de mayo, 2019

1. El algoritmo de Dijkstra puede resumirse así: Primero, colocamos el vértice de partida s en la solución —recordemos que la solución es un árbol de rutas mínimas, T, inicialmente vacío. Luego, construimos T de a una arista a la vez, agregando siempre a continuación la arista que da una ruta más corta **desde *s*** a un vértice que no está en T; es decir, agregamos vértices a T en el orden de sus distancias a s (distancias a través de las aristas que están en T). Esta es la versión abstracta del algoritmo.

a) Una versión concreta particular del algoritmo es la que estudiamos en clase, y que toma tiempo $O(E \log V)$. Este algoritmo es conveniente cuando el número de aristas de G es $O(V)$, o en general, mucho menor que V^2. Pero si G es muy denso, es decir, si el número de aristas es $O(V^2)$, entonces sería preferible una versión del algoritmo que tome tiempo $O(V^2)$, y que por lo tanto sea lineal en el número de aristas de G. Da una versión del algoritmo con esta propiedad y justifica que es así.

b) Muestra que en general el algoritmo de Dijkstra efectivamente no encuentra (todas) las rutas más cortas a partir de s si G tiene algunas aristas con costos negativos.

2. El algoritmo de Kruskal se puede resumir así: Primero, ordenamos las aristas, según sus costos, de menor a mayor; el árbol T que queremos encontrar está inicialmente vacío. Luego, vamos considerando una a una las aristas en el orden dado, y unimos una arista a T a menos que cierre un ciclo. El algoritmo termina cuando hay $|V|-1$ aristas en T.

a) ¿Cuál es la complejidad de este algoritmo si simplemente usamos DFS para decidir si una arista cierra un ciclo? Deduce paso a paso tu respuesta.

b) Si el algoritmo es implementado usando conjuntos disjuntos, ¿cuál es la relación entre los conjuntos disjuntos y los (sub)árboles que constituyen las componentes conectadas de T, en cada paso del algoritmo?

c) [Aparte de a) y b)] Si C es cualquier ciclo de G, entonces demuestra que la arista más costosa de C no puede pertenecer a un MST de G.

3. Sea $G = (V, E)$ un grafo no direccional con costos; y supongamos que todos los costos son distintos. Se define un **corte** de G como una partición de V en dos conjuntos disjuntos no vacíos (como lo vimos en clase); y una **arista que cruza el corte** como una arista que conecta un vértice en un conjunto con un vértice en el otro (como también lo vimos en clase).

El algoritmo de Kruskal, para determinar el árbol de cobertura de costo mínimo de G, está basado en la siguiente proposición: Dado cualquier corte de G, **la arista de menor costo que cruza el corte pertenece al MST de *G***. Demuestra esta proposición (p.ej., por contradicción).

4. Conjuntos disjuntos

a) Para cada una de las siguientes implementaciones de conjuntos disjuntos, ¿cuál es la complejidad —peor caso— de ejecutar una única operación *union*? ¿y una única operación *find*? Justifica, preferentemente ayudado por un dibujo.

implementación	*union*	*find*
arreglo simple: $x[i] = j \Leftrightarrow$ el nodo i pertenece al conjunto j		
listas ligadas lineales: cada elemento de la lista tiene un puntero al elemento que le sigue y otro al primer elemento de lista (el representante)		
árboles con raíz (representados mediante arreglos, y en que los hijos apuntan a los padres)		
árboles con raíz (como arriba), en que, además, en la *union* el árbol más pequeño apunta al más grande		

b) Considera el problema de crear un laberinto a partir de un arreglo rectangular de $n \times m$ celdas, en el que la entrada está en la celda de la esquina de arriba a la izquierda y la salida está en la celda de la esquina de abajo a la derecha; las celdas están separadas de sus celdas vecinas por murallas. Inicialmente, están todas las murallas, excepto a la entrada y a la salida. La idea es repetidamente elegir una muralla de manera aleatoria y botarla si las celdas que la muralla separa no están conectadas entre ellas, y así las conectamos (por supuesto, nunca botamos las murallas del perímetro del arreglo). Terminamos cuando las celdas de la entrada y la salida queden conectadas entre ellas (aunque para obtener un "mejor" laberinto conviene seguir botando murallas hasta que cualquier celda sea alcanzable desde cualquier otra celda). Explica cómo puedes resolver este problema usando una estructura de datos para conjuntos disjuntos.

5. Considera el siguiente grafo no direccional con costos, representado matricialmente:

	a	***b***	***c***	***d***	***e***	***f***	***g***
a		2	4	1			
b	2			3	10		
c	4			2		5	
d	1	3	2		7	8	4
e		10		7			6
f			5	8			1
g				4	6	1	

	¿sol?	*dist*	*padre*
a	*F*	0	–
b	*F*	∞	–
c	*F*	∞	–
d	*F*	∞	–
e	*F*	∞	–
f	*F*	∞	–
g	*F*	∞	–

Ejecuta paso a paso el algoritmo de Prim* para determinar un árbol de cobertura de costo mínimo, tomando ***a*** como vértice de partida. En cada paso muestra la versión actualizada de la tabla a la derecha, en que *¿sol?* indica si el vértice ya está en la solución: *V* o *F*.

*Ya sé que el algoritmo de Prim no es tema para el control; pero entender bien su funcionamiento puede ayudar a entender el algoritmo de Dijkstra.