Grafos

Amalia Duch Brown

Octubre de 2007

Índice

1. Definiciones Básicas

Intuitivamente un grafo es un conjunto de vértices unidos por un conjunto de líneas o flechas dependiendo de si el grafo es dirigido o no dirigido.

Gráficamente los vértices se representan por círculos, las líneas (o aristas) pertenecen a los grafos no dirigidos y las flechas (o arcos) a los grafos dirigidos.

Formalmente, un **grafo no dirigido** (o simplemente grafo) consta de:

1. un conjunto finito de vértices V
2. un cojunto de aristas E en el que cada arista es un conjunto de exactamente dos vértices.

Un **grafo dirigido** (o digrafo) consta de:

1. un conjunto finito de vértices V
2. un cojunto de arcos $E \subset V \times V$ (obsérvese que cada arco es un par ordenado vértices)

Tanto en los grafos dirigidos como en los no dirigidos las secuencias de vértices pueden formar caminos y ciclos. Definimos un **camino de longitud** ℓ como una secuencia de vértices $u_0, u_1, \ldots, u_\ell$ tales que, para todo i tal que $1 \leq i \leq \ell$, $(u_{i-1}, u_i) \in E$ (si se trata de un digrafo) o $\{u_{i-1}, u_i\} \in E$ (si se trata de un grafo). Un camino es **simple** si todos los vértices del camino, excepto quizás el primero y el último, son diferentes. Un **ciclo** es un camino simple que comienza y acaba en el mismo vértice.

Decimos que un grafo $G = (V, E)$ es **conexo** si para todo par de vértices $u, v \in V$ existe un camino en el grafo G que comienza en u y acaba en v.

Un tipo especial de grafo conexo es el **árbol** (árbol libre) que es un grafo no dirigido, conexo y acíclico. Un árbol también puede definirse como un grafo no dirigido en el que hay exactamente un camino entre todo par de vértices. Los árboles tienen algunas sencillas propiedades que pueden resultar muy útiles, como por ejemplo:

1. un árbol con n vértices contiene exactamente $n-1$ aristas
2. si se añade una única arista a un árbol, el grafo resultante contiene un único ciclo
3. si se elimina una única arista de un árbol, entonces el grafo resultante deja de ser conexo

Decimos que un digrafo es **fuertemente conexo** si para cualquier par de vértices existe un camino que los une y decimos que es **débilmente conexo** si el grafo resultante de convertir los arcos en aristas es conexo.

En un grafo (o digrafo), decimos que un vértice $v \in V$ es **adyacente** a un vértice $u \in V$ si y solo si $\{u, v\} \in E$ en el caso de los grafos o $(u, v) \in E$ en el caso de los digrafos. En un digrafo, si un vértice v es adyacente a un vértice u, decimos que el vértice u es **incidente** al vértice v.

En un grafo, el **grado de un vértice** es él número de vértices adyacentes a él y el **grado del grafo** el máximo de los grados de sus vértices.

Teorema 1.1 *En un grafo no dirigido $G = (V, E)$ se tiene que $\sum_{v \in V} grado(u) = 2|E|$.*

La demostración hecha en clase es por inducción sobre el número de vértices del grafo.

En un digrafo el **grado de entrada de un vértice** es el número de sus vértices incidentes y el **grado de salida de un vértice** es el número de sus vértices adyacentes.

Decimos que un grafo (digrafo) es **completo** si contiene el máximo número de aristas (arcos) posible. ¿Cuántas aristas (arcos) son?

Ejemplos de todo lo anterior: Dados en clase.

2. Representación

Existen varias estructuras de datos que pueden utilizarse para representar grafos y digrafos. La elección de la estructura de datos adecuada depende del tipo de operaciones que se quieran aplicar al conjunto de vértices y aristas (arcos) del grafo (digrafo) en cuestión. Las representaciones más comunes son las matrices de adyacencia y las listas de adyacencia.

2.1. Matrices de Adyacencia

Dado un grafo (digrafo) $G = (V, E)$ con $V = \{1, 2, \ldots, n\}$, la matriz de adyacencias de G es una matriz A de boleanos de tamaño $n \times n$ en la que $A[i][j]$ es cierto si y sólo si la arista (arco) que une al vértice i con el vértice j está en E ($\{i, j\}$ en el caso de grafos y (i, j) en el caso de digrafos).

Es fácil observar que la matriz de adyacencias de un grafo no dirigido es una matriz simétrica y que podemos ahorrar espacio (la mitad) guardando solo su parte inferior.

En un grafo representado por matrices de adyacencia el tiempo que se requiere para accesar un elemento es independiente de las tallas de V y de E, por tanto, ésta puede ser una representación adecuada en las aplicaciones en las que es necesario saber con mucha frecuencia si una determinada arista (arco) está presente en el grafo (digrafo).

La desventaja principal de utilizar una matriz de adyacencias para representar un grafo (digrafo) es que la matriz requiere un espacio $\Omega(n^2)$ incluso si el grafo (digrafo) es **esparso**, es decir, si tiene bastante menos de n^2 aristas (arcos). Sólo leer o examinar la matriz requerirá tiempo $\mathcal{O}(n^2)$, en perjuicio de posibles algoritmos de tiempo $\mathcal{O}(n)$ para manipular grafos (digrafos) con $\mathcal{O}(n)$ aristas (arcos). Una alternativa para evitar esta desventaja es utilizar listas para representar un grafo.

Ejemplos: Dados en clase.

2.2. Listas de Adyacencia

Dado un grafo $G = (V, E)$ la lista de adyacencias de un vértice i de G, es una lista, en un orden cualquiera, de todos los vértices adyacentes a i. Se puede representar G como un vector L en el que $L[i]$ es un puntero a la lista de adyacencias del vértice i.

La cantidad de memoria que requiere esta representación es proporcional a la suma del número de vértices más el número de punteros (que corresponde al número de aristas o arcos según sea el caso). Es decir, el coste en memoria es $\Theta(n + m)$ con $n = |V|$ y $m = |E|$. Si el grafo es esparso este coste es mucho menor que el requerido por la representación matricial, en cambio, si el grafo es **denso** (lo que sucede cuando el número de aristas o arcos es $\Theta(n^2)$) la diferencia en requerimientos de memoria entre una representación u otra no es tan significativa.

La desventaja de esta representación es que el determinar si una arista (arco) está o no en el grafo puede tomar tiempo $\mathcal{O}(n)$ ya que el número máximo de vértices que pueden haber en la lista de adyacencias de un vértice dado es n.

Ejemplos: Dados en clase.

3. TAD Grafo

Es posible definir formalmente los TADs correspondientes a los grafos y a los grafos dirigidos y estudiar las implementaciones de sus operaciones. No entraremos en los detalles porque la mayoría de ellos se han estudiado antes en la asignatura. Las operaciones más comunes en grafos y digrafos incluyen las operaciones de leer la etiqueta de un vértice o de una arista (arco), insertar o

borrar vértices y aristas (o arcos), navegar por el grafo (digrafo) siguiendo sus aristas (arcos). Esta última operación requiere la definición de un tipo *índice* que nos permita recorrer todos los vértices adyacentes a uno dado. Para ello, definimos las macros:

1. `forall_ver(u,G)` que recorre todos los vértices de G
2. `forall_adj(uv,G[u])` que hace que el iterador `uv` recorra toda la lista de adyacencias `G[u]`. En este caso es posible imaginar que `u` es el vértice de salida, que `uv` es la arista (o arco) y que `*uv` es el vértice de destino.

La implementación correspondiente se encuentra en el fichero `graph.hh` incluido en el código de la asignatura (`http://www.lsi.upc.edu/~ada`).

4. Recorridos

Un **recorrido** de un grafo es una manera sistemática de explorar sus vértices siguiendo la estructura del grafo.

Los recorridos dan lugar a esquemas para el tratamiento de grafos. Varias preguntas sobre grafos (como por ejemplo saber si un grafo es conexo o no) pueden resolverse mediante recorridos.

En un recorrido puede aplicarse una determinada operación en cada visita. Nosotros nos centraremos en generar una secuencia de los vértices en el orden en el que se visitan (igual que se había hecho para árboles).

4.1. Recorrido en Profundidad (DFS)

En cada paso se escoge seguir por una de las aristas que salen del vértice visitado más recientemente.

Implementación y Ejemplos: Dados en clase.

Dado un grafo $G = (V, E)$ el coste de recorrerlo utilizando el algoritmo DFS es $\Theta(|V| + |E|)$, ya que los bucles del algoritmo DFS toman un tiempo $\Theta(|V|)$ que es independiente del tiempo que toma hacer las llamadas a las visitas en profundidad de cada vértice. Para cada visita a un vértice v el bucle correspondiente se ejecuta tantas veces como el número de vértices adyacentes a v ($|Adj[v]|$) y como $\sum_{v \in V} |Adj[v]| = \Theta(|E|)$, el coste total de ejecutar el bucle es $\Theta(|E|)$. La combinación de los dos costes descritos resulta en el coste del algoritmo.

4.2. Recorrido en Amplitud (BFS)

En cada paso se visitan todos los vértices (aún no visitados) adyacentes al vértice visitado más recientemente.

Implementación y Ejemplos: Dados en clase.

Dado un grafo $G = (V, E)$ el coste en tiempo de recorrerlo en amplitud es $\mathcal{O}(|V|+|E|)$ ya que cada vértice del grafo se pone en la cola y se saca de ella una única vez. Las operaciones de agregar un elemento a la cola y de sacarlo toman tiempo $\Theta(1)$, por tanto el tiempo total de agregar y sacar elementos de la cola es $\Theta(|V|)$. Por otra parte, la lista de adyacencias de cada vértice se recorre sólo cuando el vertice se saca de la cola, es decir, se recorre una única vez y como la suma de las longitudes de todas las listas de adyacencias es $\Theta(|E|)$, el tiempo total de recorrer las listas es $\mathcal{O}(|E|)$. De las dos consideraciones anteriores se desprende el coste del algoritmo BFS.

5. Ordenación Topológica

Dado un grafo dirigido y acíclico una ordenación topológica de sus vértices es una ordenación de sus vértices en la que un vértice v no aparece antes que un vértice u si en el grafo hay un camino de u a v.

Partiendo de las versiones iterativa y recursiva del DFS, pueden determinarse dos algoritmos (iterativo y recursivo, respectivamente) eficientes para obtener una posible ordenación topológica de un grafo dirigido y acíclico dado.

Implementación y Ejemplos: Dados en clase.