Una guida alla visita di grafi

Mauro Torelli – a. a. 2004-2005

Note al corso di *Algoritmi e strutture di dati*

Lo scopo di questa nota è presentare molto sinteticamente delle *linee guida* per comprendere svariati algoritmi su grafi. Il contenuto della nota, l'esame attento delle *figure del testo*, che illustrano il comportamento dei diversi algoritmi su esempi specifici (come mostrato a lezione), ed eventualmente il materiale accessibile in rete dalla pagina www.algoteam.dsi.unimi.it sono sufficienti per l'esame, mentre per una comprensione approfondita e l'eventuale realizzazione pratica degli algoritmi è necessario lo studio dettagliato del testo.

Per visitare sistematicamente un grafo occorre in qualche modo *marcare* i vertici già visitati: più precisamente, avremo a che fare con vertici *inesplorati* (che supporremo colorati in *bianco* (*white*: denoteremo con *W* l'insieme dei vertici bianchi), vertici *in corso di visita* (*grigi*, insieme *C*: non abbiamo usato la lettera *G* per non fare confusione con il simbolo che denota l'intero grafo) e vertici *completamente esplorati* (*neri* – *black*, insieme *B*). I vertici assumono via via prima il colore bianco, poi diventano grigi e infine neri.

Nel corso della visita si può costruire un *albero di visita A*, in cui *ogni nodo ha come padre il nodo a partire dal quale è stato visitato la prima volta*: l'albero A ha come elementi *lati* del grafo, mentre gli altri insiemi hanno come elementi *vertici* del grafo. In realtà, se il grafo *non è connesso*, l'insieme A è una *foresta ricoprente* tutti i vertici del grafo assegnato.

L'*algoritmo di visita* può essere espresso genericamente come segue, con una notazione insiemistica che ignora i dettagli realizzativi ma aiuta a capire l'essenziale. Il grafo in input ha *vertici* V e *lati* E. Denoteremo con Adj[*u*] l'insieme dei *vicini* del vertice *u* (contenuti nella *lista di adiacenza*): Adj[*u*]:= {*v* ∈ V: *uv* ∈ E}. Ometteremo come di consueto le parentesi graffe: per esempio, indicheremo con *uv* il lato {*u*, *v*}. Il simbolo ▷ introduce un *commento*.

ALGORITMO VISITA(V, E)

0. W ← V; A ← B ← C ← Ø
1. *finché* C **»** W ≠ Ø
2. *se* C = Ø ***scegli1*** *u* ∈ W *altrimenti* ***scegli2*** *u* ∈ C
3. ∀*v* ∈ Adj[*u*] ∩ W: A ← A **»** *uv* ▷ *u* è il *padre* di *v* nell'albero A
4. B ← B **»** *u*; C ← (C **»** Adj[*u*]) – B; W ← W – B – C
5. *restituisci* A.

Alla riga 2 dell'algoritmo, la prima procedura ***scegli1*** sceglie, in genere *casualmente*, un vertice *u* da W: ciò è necessario soltanto se il grafo *non è connesso*, e in tal caso l'algoritmo può essere usato per individuare le diverse *componenti connesse* del grafo, in tempo $\Theta(V + E)$ (infatti si devono colorare di bianco i vertici di V e poi esaminare – in tempo costante per ciascuno – i lati di E).

La seconda procedura ***scegli2***, che sceglie un vertice *u* in C, è invece cruciale per determinare le caratteristiche della visita: se interessa solo visitare sistematicamente il grafo, la scelta può essere ancora *casuale*, altrimenti occorre seguire un ordine stabilito da un'opportuna *coda con priorità*.

1. Come possibile applicazione molto semplice di una visita in ordine *casuale* possiamo citare lo stabilire se il grafo sia *bipartito* ovvero 2-*colorabile* (si veda il problema 5-1 del testo). In questo caso non interessano gli insiemi A, B e C: si colora il nodo di partenza con un colore, tutti i suoi vicini con un altro e poi si alternano i colori. Se non occorre mai cambiare il colore di un nodo già colorato, il grafo è bipartito, altrimenti non lo è.

2. Se ***scegli2*** estrae il vertice *u* da una *coda* (FIFO), allora abbiamo la *visita in ampiezza* (*breadth-first search*, *BFS*): i vertici a distanza 1 da quello di partenza sono visitati prima di quelli a distanza 2, e così via: *estendendo* la struttura dati del grafo con un campo *distanza* è possibile tener traccia delle distanze e quindi fornire la distanza minima di ciascun nodo da quello di partenza (si veda la figura 23.3 del testo).

3. Se ***scegli2*** estrae il vertice *u* da una *pila* (LIFO), allora abbiamo la *visita in profondità* (*depth-first search*, *DFS*), in cui i nodi a distanza 1 da quello di partenza sono in fondo alla pila e la visita tende invece ad allontanarsi il più possibile dal nodo di partenza, prima di tornare ad avvicinarsi. In realtà qui abbiamo semplificato un po' troppo: per alcune applicazioni è essenziale la versione *ricorsiva* dell'algoritmo, in cui solo un primo vicino viene immesso nella pila (non *tutti* i vicini). La pila è gestita automaticamente dalla ricorsione. Il libro mostra come, estendendo la struttura dati per registrare gli istanti di inizio visita e di fine visita (si veda la figura 23.4) sia poi possibile, per esempio, realizzare il cosiddetto *ordinamento topologico* (§23.4 del testo), sempre in tempo $\Theta(V + E)$.

4. Se ***scegli2*** estrae il vertice *u* da una *coda con priorità* che fornisce il lato *uv* di peso *minimo* tra tutti quelli che collegano un vertice *v* nero a un vertice grigio (come nella figura 24.5 del testo), allora si costruisce "ingordamente" un albero di lati minimi e si ottiene alla fine un *albero ricoprente minimo* (se il grafo G è *connesso*) con l'*algoritmo di Prim*. Se la coda con priorità è realizzata con un *heap* binario (come nell'algoritmo di Huffman) il tempo di esecuzione risulta $O(E \lg E) = O(E \lg V)$.

5. In alternativa all'algoritmo di Prim si possono scegliere ingordamente i lati minimi badando solo a non chiudere *cicli*, quindi costruendo una *foresta*, che alla fine però risulterà un *albero ricoprente minimo*, se il grafo è connesso (si veda la figura 24.4: *algoritmo di Kruskal*). Per stabilire se un lato chiude o no un ciclo si possono usare le procedure per *insiemi disgiunti* (UNION e FIND-SET) e il tempo di esecuzione è dominato dal tempo $O(E \lg E)$ per ordinare i lati.

6. Un'ulteriore alternativa (meno nota, ma di interesse non solamente storico) è data dall'*algoritmo di Borůvka*, che risale al 1926 ed è quindi anteriore alla data di nascita "ufficiale" della teoria dei grafi moderna, che è il 1936, data di pubblicazione del libro di Dénes König, *Theorie der endlichen und unendlichen Graphen*. L'articolo originale di Borůvka, tradotto dal ceco in inglese e accompagnato da utili commenti si può leggere alla pagina http://citeseer.nj.nec.com/nesetril00otakar.html.

7. In un grafo *orientato* con *lati pesati* in cui i pesi rappresentano le *distanze* (non negative), se ***scegli2*** estrae il vertice *u* da una *coda con priorità* che fornisce il vertice *più vicino all'albero* A costruito sino a quel momento, allora l'algoritmo fornisce l'albero che collega tutti i vertici al nodo di partenza con le *distanze minime*, in tempo O(E lg V) (*algoritmo di Dijkstra*, si veda la figura 25.5 del testo).