Caroselli Stefano

ALGORITMI & STRUTTURE DATI

Tesina nº 1

Studio delle componenti fortemente connesse di un grafo orientato

1. ESPOSIZIONE DEL PROBLEMA

Una *componente fortemente connessa* (cfc) di un grafo orientato G(V,E) è un insieme massimale di vertici $U \subseteq V$ tale che per ogni coppia di vertici $A,B \in U$ esiste sia il cammino da A a B che il cammino da B ad A [ossia i vertici A e B sono raggiungibili l'uno dall'altro].

La scomposizione di un grafo nelle sue cfc si può effettuare per mezzo di un'unica VISITA IN PROFONDITÀ (VIP) : quest'ultima ci fornisce infatti notevoli informazioni sul grafo, in particolare distingue tra i possibili archi del grafo quelli di un suo *albero ricoprente DFS* (eventualmente una foresta, se esistono componenti non raggiungibili con un'unica VIP).

In una VIP su G si possono formare quattro tipi di archi :

1. Gli archi facenti parte dell'albero ricoprente ;
2. Gli archi *all'indietro* : un arco (x,y) è all'indietro se y è un antenato di x nell'albero ricoprente (es. : un cappio genera sempre un arco all'indietro).
3. Gli archi *in avanti* : sono quegli archi (x,y) che connettono un vertice x ad un suo successore y nell'albero DFS.
4. Gli archi *di attraversamento* : sono archi (x,y) da un vertice x ad un altro y visitato in precedenza, ma che non è antenato di x nell'albero ricoprente. Sono archi di attraversamento anche archi che connettono vertici in alberi DFS distinti.

Prima di descrivere l'algoritmo riprendiamo alcuni principali risultati sui grafi orientati.

Lemma 1

Un grafo orientato G è aciclico se e solo se una visita in profondità di G non produce archi all'indietro.

Dimostrazione

($\Rightarrow$) Per assurdo: supponiamo che esista un arco all'indietro (B,A). Allora A è un antenato di B nella foresta DFS; quindi vi è un cammino da A ad B in G e l'arco (B,A) completa il ciclo: si arriva ad un assurdo perché il grafo per ipotesi è aciclico.

($\Leftarrow$) Per assurdo: supponiamo che il grafo G contenga un ciclo C ; sia A il primo vertice di C scoperto e sia (B,A) l'arco che A precede in C. Ad un determinato istante della visita vi è un cammino di vertici di colore = 0 (non scoperti) da A ad B. Ne segue, per l'ipotesi, che B sarà un discendente di A nella foresta DFS e quindi (B,A) è un arco all'indietro: si ha nuovamente un assurdo. ////

Da questo risultato segue che se durante la visita non si trova un arco all'indietro il grafo non contiene cicli e di conseguenza la scomposizione di un grafo nelle sue componenti fortemente connesse è banale: ogni nodo costituisce una componente connessa a sé. Ovviamente questa situazione è di poco interesse per il problema di partenza.

Lemma 2

Se due vertici sono nella stessa componente fortemente connessa allora nessun cammino tra di essi può abbandonare la componente fortemente connessa.

Dimostrazione

Siano A e B due vertici della stessa componente fortemente connessa : esiste quindi un AB-cammino ed un BA-cammino. Sia P un vertice appartenente ad un cammino cha va da A a B : ne segue che P è raggiungibile da A. Inoltre, poiché esiste un cammino da B ad A, si ha che A è aggiungibile da P con il cammino che va da P ad B ed ancora a A. Quindi A e B sono nella stessa componente fortemente connessa. Per l'arbitrarietà di P il lemma è dimostrato. ////

Teorema

In una qualunque visita in profondità tutti i vertici della stessa componente fortemente connessa sono posti nello stesso albero DFS.

Dimostrazione

Sia A il primo dei vertici di una componente fortemente connessa che viene scoperto: poiché A è il primo, al momento della sua scoperta tutti gli altri vertici nella componente fortemente connessa hanno ancore colore = 0. Vi sono cammini da A ad ogni altro vertice della componente fortemente connessa e poiché, per il lemma precedente, questi cammini non lasciano mai la componente fortemente connessa, tutti i vertici su di essi hanno colore = 0.

Ricordando che un vertice Y è un discendente di un vertice X se e solo se al tempo in cui le la visita scopre X il vertice Y è raggiungibile da X con un cammino contenente esclusivamente vertici di colore = 0, ne segue che ogni vertice della componente fortemente connessa diventa un discendente di A nell'albero DFS. ////

---

2. DESCRIZIONE DELL'ALGORITMO

Riportiamo ora di seguito i passi principali dell'algoritmo che scompone un grafo orientato nelle sue c.f.c. e che sfrutta i risultati precedentemente visti.

Questo algoritmo utilizza più strutture ausiliari, per diminuirne il costo:

- Una lista di adiacenza "v" che esamineremo durante la VIP;
- Un vettore che memorizza il predecessore di ogni nodo (padre) "π" ;
- Un vettore che memorizza l'ordine di visita "dfn" e il vettore inverso "udfn";
- Un vettore che memorizza il dfn del più alto vertice raggiungibile con archi all'indietro "low" ;

- Un vettore che memorizza lo stato di visita di ogni nodo “color” : vale 0 se il nodo non è stato visitato, 1 non appena è stato visitato, 2 se esso e tutta la sua lista di adiacenza sono stati visitati ;
- Due vettori di puntatori “comp” e “top”, per gestire un array di code [∀ nodo P, la coda di P contiene i vertici fortemente connessi ad P] ;
- Un vettore di booleani “ctr” che controlla le code da studiare ;

Quando un vertice Y viene scoperto durante la scansione della lista di adiacenza di un vertice X già scoperto [color(X) = 1], la visita in profondità memorizza questo evento assegnando X = π[Y]. La VIP produce in generale più alberi DFS, a seconda che la visita sia ripetuta da più sorgenti.

Il programma, dopo aver generato un grafo orientato con la procedura “creagrafo” (che qui è omessa) effettua una VIP, per mezzo della procedura “DFS” (deep-first-search) descritta di seguito :

```
Procedure DFS (u : integer);
var p : link; w : integer;
begin
    color[u]:=1;      x:=x+1;      dfn[u]:=x;      udfn[x]:=u;
low[u]:=x;      p:=v[u];
    while (p<>nil) do
      begin w:=p^.inf;
      if  (color[w]=0)
         then  begin       padre[w]:=u;    DFS(w);
                     low[u]:= min(low[u],low[w]);
               end
         else
      if  (color[w]=1)
         then  begin       low[u]:=min(low[u],low[w]);
                     low[padre[u]]:=min(low[u],low[padre[u]]);
               end
         else
      if (color[udfn[low[w]]]=1)
         then  low[u]:=min(low[u],low[w]);
      p:=p^.pr;
      end;
    if  (low[u]<dfn[u])       then accoda(u,udfn[low[u]]);
    color[u]:=2;
end;
```

Il cuore della procedura DFS è la procedura “accoda” che opera sulle code del vettore comp, (e top) generando le componenti :

```
Procedure accoda (z,y : integer);        {attacca la lista di z in
quella di y}
var p:link;
  begin
      top[y]^.pr:=comp[z];
      top[y]:=top[z];

      ctr[z]:=false;
      ctr[y]:=true;
  end;
```

Come si può osservare, durante la DFS di un nodo A si calcola il dfn(A) e il low(A), studiando gli archi di ritorno (caratterizzati dal fatto che i nodi di arrivo hanno colore = 1) : infatti tutti i nodi compresi da un arco di ritorno fanno parte di una stessa componente connessa in quanto formano un ciclo; si studiano anche i successori di A che possono avere archi di ritorno a vertici antenati di A e quindi formare un ciclo; infine anche archi di attraversamento possono generare cicli, ma solo se il low del vertice di arrivo ha ancora colore = 1 (ossia se si può risalire ad un antenato comune).

Tra tutti i possibili nodi raggiungibili da A tramite più archi di ritorno, si cerca quello più in alto possibile nell'albero DFS, e si pone tale nodo = low(A); a questo punto, dopo aver effettuato ricorsivamente la DFS in tutto il sottoalbero di A si invoca la procedura accoda al nodo B indicato dal low di A : la coda di A assume ctr = **false** e si accoda a quella di B che assume ctr = **true**; risalendo nell'albero, chiudendo le varie DFS, tutte i nodi di una c.f.c. aggregano la propria coda ad un unico rappresentante, dato dal più alto nodo raggiungibile da ogni nodo della c.f.c. (passando più volte per archi dell'albero, archi di attraversamento o archi di ritorno); tutte le code dei nodi della c.f.c. assumono ctr = **false**, eccetto quella del nodo rappresentante, che assume ctr = **true**.

Ovviamente un'unica DFS non è in generale sufficiente a visitare tutto il grafo G, di qui la necessità di un **for** sugli n nodi di G nel programma generale, che richiami la DFS ogni volta che trovi un nodo non visitato (colore = 0) iniziando la nuova visita da tale nodo.

Una volte terminato il ciclo **for**, basterà stampare solo le code che hanno ctr = **true**.

---

3. COSTO DELL'ALGORITMO

L'operazione principale nella procedura DFS è l'unione di due code, che si effettua prima di porre il colore del nodo = 2 ; poiché una volta che un nodo assume colore = 2, esce dalla visita, il numero di volte che si pone colore = 2 è uguale al numero di volte che si scopre un nodo, ossia che si chiama una DFS; tale operazione può avvenire al più n volte, quindi : $c(DFS) \in O(n)$.

Se un grafo è costituito da n nodi isolati, allora nel **for** generale verrà chiamata n volte la procedura DFS, e ogni volta terminerà subito; in questo caso la procedura accoda non verrà mai chiamata.

Al contrario se G ha un'unica c.f.c. il **for** chiamerà una sola volta la DFS ma essa al suo interno chiamerà altre n – 1 volte la DFS, per ricorsione; in questo caso la procedura accoda verrà chiamata per ogni nodo che si visita (eccetto la radice), quindi n – 1 volte.