Algoritmi e Strutture Dati (II anno) - a.a. 1996-97

Ultima modifica: 14 Aprile 1997. Commenti a Eugenio Moggi.

Visite di un grafo orientato

Fissato un nodo r di un grafo orientato G=(V,E) diciamo che una sequenza S = (e_1,...,e_k) di archi e' una **visita** a partire da r sse

- l' i-esimo arco e_i=(u_i,v_i) della sequenza e' t.c. u_i appartiene a V_{i-1} mentre v_i non vi appartiene, dove
  - V_0 = {r}
  - V_i = V_{i-1} + {v_i}
- non esiste un arco (u,v) t.c. u appartiene a V_k ma v non vi appartiene.

Ogni visita soddisfa le seguenti proprieta':

- V_k e' l'insieme dei nodi raggiungibili da r. Si proceda per assurdo, se v e' raggiungibile da r ma non e' in V_k, si consideri il primo nodo in un cammino da r ad v che non sta in V_k, e quindi si ottenga una contraddizione con la seconda condizione nella definizione di visita.
- ogni nodo di G_i=(V_i,E_i) e' raggiungibile da r tramite un unico cammino, quindi per l'esercizio 5.5-3 G_i e' un albero radicato in r (con gli archi che vanno dal padre ai figli).

Il seguente algoritmo fornisce uno schema generale per produrre una visita di G=(V,E) a partire da r.

```
input:  A: array [V] of sottoinsiemi di V {liste di adiacenza per G}
        r: V {nodo di partenza}
output: S: sequenza di archi {usati nella visita}

var U: sottoinsieme di V {dei nodi visitati}
    F: sottoinsieme di V {dei nodi visitati frontiera}

U <- F <- {r}
S <- sequenza vuota
while F non vuoto do
  u <- "un elemento di F"
  if A[u] incluso in U
  then "rimuovi u da F"
  else v <- "un elemento di A[u]-U"
       "aggiungi v ad U ed F"
       S <- S*(u,v)
```

Le visite si possono sfruttare per risolvere una varieta' di problemi sui grafi p.e.: test di connettivita' (forte) e calcolo delle componenti (fortemente) connesse (vedi note di Costa).

Se S e' una visita, introduciamo la seguente notazione

- v_i e' l'i-esimo nodo visitato. Quindi v_0 e' r e l'i-esimo arco di S e' della forma (u_i,v_i).
- p_i e' la posizione nella lista dei nodi visitati di u_i, cioe' il padre di v_i nell'albero associato ad S, ovvero u_i = v_{p_i}. Quindi p_0 e' indefinito, e p_i < i se i > 0.

- V_i e' l'insieme {v_0,...,v_i}.

Algoritmi di visita BFS e DFS

Nella visita BFS si implementa F come una coda

```
var U: sottoinsieme di V {dei nodi visitati}
    F: coda di V {dei nodi visitati frontiera}
U <- {r}
F <- {r}
S <- sequenza vuota
while F non vuoto do
  get(F,u)
  for v in A[u] do
    if "v not in U"
    then "aggiungi v ad U"
         put(F,v)
         S <- S*(u,v)
```

Nella visita DFS si implementa F come uno stack, o piu' semplicemente si usa una procedura ricorsiva

```
var U: sottoinsieme di V {dei nodi visitati}
U <- insieme vuoto
S <- sequenza vuota
DFS(r)

DFS(u)
  "aggiungi u a U"
  for v in A[u] do
    if "v not in U"
    then S <- S*(u,v)
         DFS(v)
```

Come esempio, vediamo come si puo' adattare l'algoritmo DFS per determinare le componenti connesse in un grafo non orientato.

```
input:  A: array [V] of sottoinsiemi di V
output: c: array [V] of 1..|V| {c[v] numero della componente connessa}
        k: 1..|V| {numero componenti connesse}

var U: sottoinsieme di V {dei nodi visitati}
U <- insieme vuoto
k <- 0
for v in V do c[v] <- NIL

while "V-U non vuoto" do
  k <- k+1
  r <- "un elemento di V-U"
* invariante:
* c[v] <= k se v in U, c[v] = NIL altrimenti
* {v|c[v]=i} componente connessa se i < k
* c[v]=k sse v in U e v nella stessa componente connessa di r
  DFS'(r)

DFS'(u)
  "aggiungi u a U"
  c[u] <- k
  for v in A[u] do
    if "v not in U"
    then DFS'(v)
```

Caratterizzazione della BFS

Astrattamente una visita S e' una BFS sse per ogni $0 < i <= |S|$

- $A[v_j]$ e' incluso in V_{i-1} se $0 <= j < p_i$

Una facile consequenza della definizione di visita BFS e': $p_i <= p_j$ per ogni $0 < i < j <= |S|$.

Le proprieta' piu' interessanti delle visite BFS si possono dare in termini di distanza. Definiamo la distanza d(u,v) da u a v come la lunghezza minima dei cammini da u a v, e se non esistono cammini da u a v conveniamo che d(u,v) sia infinito. La distanza in un grafo orientato gode di proprieta' simili a quelle della distanza in uno spazio metrico:

- $d(u,v)=0$ sse $u=v$
- $d(u,w) <= d(u,v) + d(v,w)$
- mentre $d(u,v) = d(v,u)$ in generale non vale (ma vale se il grafo e' non orientato).

Teorema: il cammino da r a v_i nell'albero indotto da una visita BFS S, ha lunghezza minima tra i cammini da r a v_i nel grafo.
Tale proprieta' si dimostra facilmente una volta che si e' stabilito il seguente lemma tecnico (che si dimostra per induzione su i): per ogni $0 < i <= |S|$

- $d(r,v_j) <= d(r,v_k)$ se $0 <= j < k <= i$, cioe' la funzione d(r,_) sull'intervallo 0..i e' non decrescente
- $d(r,v_i) = d(r,v_{p_i})+1$
- u e' in V_i se $d(r,u) < d(r,v_i)$, cioe' se u e' piu' vicino ad r di v_i allora e' visitato prima di v_i.

Il Lemma ci dice che nelle visite BFS prima si visitano i nodi vicini ad r e poi quelli piu' distanti, quindi graficamente una visita BFS procede a cerchi concentrici a partire da r.

Il Teorema ci fornisce un modo semplice di calcolare le distanze ed i cammini minimi da r di tutti i nodi del grafo, adattando l'algoritmo di visita BFS.

```
input:  come in algoritmo di visita
output: d: array [V] of distanze {d[v] = distanza di v da r}
        p: array [V] of V u {NIL} {p[v] = padre di v nell'albero BFS}

var U: sottoinsieme di V
    F: coda di V
U <- {r}
F <- {r}
for v in V do
  if v=r then d[v] <- 0 else d[v] <- infinito
for v in V do p[v] <- NIL

while F non vuoto do
* invariante:
* d[v] = d(r,v) se v e' in U, infinito altrimenti
* p[v] = padre di v se v e' in U, NIL altrimenti
  get(F,u)
  for v in A[u] do
    if "v not in U"
    then "aggiungi v ad U"
         put(F,v)
```

```
d[v] <- d[u]+1
p[v] <- u
```

Per dimostrare che l’invariante rimane vero, occorre sfruttare in modo essenziale il Teorema.

Caratterizzazione della DFS

Le visite DFS si possono definire astrattamente come il "duale" delle BFS, cioe’ una visita S e’ una DFS sse per ogni $0 < i <= |S|$

- $A[v_j]$ e’ incluso in V_{i-1} se $p_i < j < i$

Teorema: Se S e’ una visita DFS per ogni $0 <= i <= |S|$ esiste un f_i t.c. i discendenti di v_i nell’albero indotto da S, sono esattamente i nodi $\{v_j | i <= j <= f_i\}$.