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

Ultima modifica: 24 Maggio 1997. Commenti a Eugenio Moggi.

Algoritmo di Dijkstra

L'algoritmo di Dijkstra serve a trovare i cammini minimi/calcolare le distanze di tutti i nodi da un nodo prefissato r in un grafo pesato. Per il corretto funzionamento dell'algoritmo e' essenziale che i pesi siano non negativi. L'algoritmo generalizza al caso pesato il comportamento delle visite BFS, cioe' visita i nodi del grafo in ordine di distanza crescente da r. Il punto cruciale e' come determinare il prossimo nodo da visitare. Le seguenti proprieta' dei grafi con pesi non negativi sono essenziali per dimostrare la correttezza dell'algoritmo:

- se u e' un nodo in un cammino da r a v allora d(r,u) <= d(r,v) [si usa la non negativita' dei pesi]

Supponiamo di aver visitato un certo insieme U di nodi t.c.

1. r e' in U
2. d(r,u) <= d(r,v), se u e' in U e v e' in V-U
3. esiste un cammino di lunghezza minima da r ad u contenuto in U, se u e' in U

e di conoscere le distanza d(r,u) per ogni u in U. Come possiamo selezionare un nodo v in V-U da aggiungere ad U in modo di mantenere le proprieta' su indicate?

1. la prima proprieta' rimane vera
2. per garantire la seconda occorre scegliere v in modo che d(r,v) <= d(r,w) per ogni w in V-U. Certamente un tale v esiste (pero' non conosciamo ancora le distanze)
3. per garantire la terza proprieta' e' necessario che v sia adiacente ad un nodo di U, in questo modo possiamo dare una stima di d(r,v) per eccesso prendendo d_U(v) = min { d(r,u)+c(u,v) | u in U }

Per semplicita' assumeremo di aggiungere gli archi mancanti con peso infinito, in questo modo d_U(v) e' definito per tutti i nodi in V-U. L'algoritmo di Dijkstra sceglie un v in V-u che minimizza d_U(v). Dimostriamo che U u {v} ha le proprieta' richieste, e che d(r,v) = d_U(v):

- Per assurdo sia y in V-U t.c. d(r,y) < d(r,v), sia x il primo nodo di un cammino minimo da r a y non appartenete ad U, allora d_U(x) = d(r,x) <= d(r,y) < d(r,v) <= d_U(v). Quindi abbiamo una contraddizione con il criterio di scelta di v.
- Sia u in U t.c. d_U(v) = d(r,u) + c(u,v) facciamo vedere che il cammino di lunghezza minima p : r -> u seguito da v ha lunghezza minima tra i cammini da r a v. La dimostrazione e' per assurdo, prendendo y=v nel caso precedente.
- Il precedente risultato ci dice che d(r,v) = d_U(v).

Per ottenere una bassa complessita' dell'algoritmo, occorre avere un modo efficiente di calcolare d_U(v). O piu' precisamente, per calcolare d_{U'}(_) per U' = U u {v} a partire da d_U(_)

- d_{U'}(x) = min { d_U(x) , d_U(v) + c(v,x) } per ogni x in V-U'

- si osservi che d_U(x) = d(r,x) se x e’ in U’.

```
input:  P: array [V,V] of pesi {matrice delle lunghezze degli archi}
        r: V
output: d: array [V] of pesi {vettore delle distanze da r}
var U: sottoinsiemi di V {dei nodi visitati}
*inizializzazione
U <- {r}
for v in V do
  if v=r then d[v] <- 0 else d[v] <- P[r,v]
*invariante
*d[v]=d(r,v) se v in U, d_U(v) altrimenti
*d(r,u) <= d(r,v) se u in U ed v in V-U
while V-U non vuoto do
  v <- un elemento di V-U che minimizza d[_]
  U <- U+{v}
  for x in V-U do
    if d[v]+P[v,x] < d[x]
    then d[x] <- d[v]+P[v,x]
```

Per calcolarsi i cammini minimi basta mantenere un array p t.c. p[v] contenga il padre vi v nel cammino minimo da r a v se v e’ in U (altrimenti indefinito). Alternativamente, p[v] potrebbe contiene il padre di v nel cammino minimo da r a v con nodi intermedi in U (vedi note di Costa).

Esercizio Si identifichi il punto della dimostrazione di correttezza dell’algoritmo di Dijkstra che non funziona in presenza di pesi negativi.

Algoritmo di Floyd

L’algoritmo di Floyd serve a trovare i cammini minimi/calcolare le distanze tra tutte le possibili coppie di nodi r in un grafo pesato. Al contrario dell’algoritmo di Dijkstra non e’ necessario assumere che i pesi siano non negativi. Inoltre, l’algoritmo di Floyd puo’ essere generalizzato al caso di pesi a valori in un *semianello chiuso* A. In questo modo, problemi apparentemente non correlati (sia su grafi che su matrici) possono essere risolti con l’algoritmo di Floyd scegliendo opportunamente il semianello chiuso.

L’idea base dell’algoritmo e’ di calcolarsi le matrici A_k di dimensione VxV t.c. A_k[u,v] e’ la lunghezza del cammino minimo da u a v i cui nodi intermedi (cioe’ senza considerare il primo ed ultimo nodo del cammino) sono contenuti nei primi k nodi del grafo (dove l’ordine lineare dei nodi puo’ essere scelto in modo arbitrario). Poiche’ l’ordine sui nodi e’ del tutto irrilevante, potremo ignorarlo e considerare invece le matrici A_U con U sottoinsieme di V. L’idea fondamentale e’ come calcolare A_{U’}, con U’ = U+{v} e v in V-U, a partire da A_U. Un cammino da x a y con nodi intermedi un U+{v} deve necessariamente essere di una delle seguenti forme:

- un cammino da p:x -> y con nodi intermedi in U, e scriveremo p:x ->_U y
- un cammino da p:x -> y che ha v come nodo intermedio esattamente k>0 volte, e quindi avremo che p e’ ottenuto concatenando i cammini propri (cioe’ con almeno un arco)
  - p_0:x ->_U v
  - p_i:v ->_U v con 0 < i < k
  - p_k:v ->_U y

A questo punto si tratta di esaminare come puo’ essere fatto il cammino p:x ->_{U’} y di lunghezza minima:

- se p:x ->_U y allora la sua lunghezza sara’ A_U[x,y]

- se p:x ->_U v ->_U y allora la sua lunghezza sara' A_U[x,v]+A[v,y]
- se invece esiste un ciclo q: v ->_U v di lunghezza < 0, e quindi A_U[v,v] < 0, allora vi sono cammini p:x ->_{U'} y di lunghezza negativa arbitrariamente grande (ripetendo il ciclo q). Indicheremo questa terza eventualita' ponendo A_{U'}[x,y] = -infinito.

Occorre fare attenzione nel manipolare +infinito e -infinito. Il primo indica l'assenza di cammini, l'altro la presenza di cammini con lunghezza negativa grande a piacere. Con questa interpretazione e' sensato convenire che (+infinito) + (-infinito) = +infinito.

```
input:  P: array [V,V] of pesi {+infinito indica assenza di arco}
output: A: array [V,V] of pesi {+infinito indica assenza di cammini,
        -infinito indica cammini di lunghezza negativi arbitrariamente
        grande}
var     A [0..|V|] of array [V,V] of pesi
*inizializzazione
for x,y in V do
  if x=y
  then A[0,x,y] <- min(0,P[x,y])  else A[0,x,y] <- P[x,y]
for k <- 1 to |V| do
  for x,y in V do
    A[k,x,y] <- min(A[k-1,x,y] , A[k-1,x,v_k]+A[k-1,v_k,v_k]^*+A[k-1,v_k,y])
A <- A[|V|]
```

dove c^* = 0 se c >= 0, e c^* = -infinito se c < 0. Si puo' evitare di mantenere un array di matrici e lavorare direttamente sulla matrice di output A.