5 Cammini minimi

In questa sezione si considera il problema di trovare i cammini minimi/calcolare le distanze tra nodi di un grafo G con archi etichettati da numeri reali (che posso essere interpretati come lunghezze o costi). Per il corretto funzionamento di alcuni algoritmi (p.e. quello di Dijkstra) e' essenziale che i numeri reali siano $> geq0$ (e le dimostrazioni si semplificano ulteriormente se sono > 0). Inoltre gli algoritmi possono essere generalizzati al caso in cui le etichette, invece che numeri reali, sono elementi di *semianelli chiusi*. Tali generalizzazione saranno discusse in seguito. Diamo le definizioni necessarie per definire formalmente il problema:

- la **lunghezza** di un cammino $p = (u_i|0 \leq i \leq n)$ di G e' data da $l(p) \stackrel{\Delta}{=} \sum_{0\leq i<n} l(u_i, u_{i+1})$, cioe' la somma delle lunghezze degli archi che compongono il cammino p. In particolare, $l(p) = 0$ quando il numero di archi $n = 0$.
- la **distanza** di v da u in G e' data da $d(u,v) \stackrel{\Delta}{=} \inf\{l(p)|p{:}\, u \rightsquigarrow v\}$, cioe' l'estremo inferiore delle lunghezze dei cammini da u a v. In particolare, $d(u,v) = +\infty$ quando non esistono cammini $p{:}\, u \rightsquigarrow v$. In presenza di archi di lunghezza negativa e' possibile avere anche $d(u,v) = -\infty$. p.e. se $u = v$ e si ha un cappio con $l(u,u) = -1$ (infatti piu' giri si fa' intorno al cappio, e piu' si riduce la lunghezza del cammino).
- dato $U \subseteq V$ diciamo che il cammino $p = (u_i|0 \leq i \leq n)$ ha nodi intermedi in $U \stackrel{\Delta}{\iff} u_i \in U$ quando $0 < i < n$. Inoltre definiamo la distanza $d_U(u,v) \stackrel{\Delta}{=} \inf\{l(p)|p{:}\, u \rightsquigarrow v \wedge p$ ha nodi intermedi in $U\}$. Proprieta', che discendono immediatamente dalla definizione, sono: $d_U(u,v)$ decresce al crescere di U, e $d_V(u,v) = d(u,v)$.

Per rappresentare un grafo G con archi etichettati con numeri reali (ed eventualmente $-\infty$) si puo' usare una matrice M di dimensioni $V \times V$ a valori nei reali estesi con $+\infty$ (e $-\infty$), piu' precisamente M rappresenta $G \stackrel{\Delta}{\iff}$

- $M[u,v] = +\infty$ sse $(u,v) \notin E$
- $M[u,v] = l < +\infty$ sse $(u,v) \in E$ e $l(u,v) = l$

5.1 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 con gli archi etichettati da numeri reali (che posso essere interpretati come lunghezze o costi). Per il corretto funzionamento dell'algoritmo e' essenziale che i numeri reali siano ≥ 0 (e le dimostrazioni si semplificano ulteriormente se i pesi sono > 0). 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.

```
input  M: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] <- M[r,v]
*invariante
*r in U
*d[v]=d(r,v) se v in U, d_U(r,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]+M[v,x] < d[x] then d[x] <- d[v]+M[v,x]
```

Le seguenti proprieta' dei grafi con pesi > 0 sono essenziali per dimostrare la correttezza dell'algoritmo:

- $d(r,v) = 0$ sse $v = r$
- se $p{:}\, r \rightsquigarrow u$ e $q{:}\, u \rightsquigarrow q$ sono cammini t.c. $l(p) + l(q) = d(r,v)$, allora $l(p) = d(r,u) \leq d(r,v)$
- se $U \subseteq V$ e' t.c. $r \in U$ e $d(r,u) \leq d(r,v)$ per ogni $u \in U$ e $v \in V - U$, mentre $w \in V - U$ e' t.c. $\forall v \in V - U.d_U(r,w) \leq d_U(r,v)$, allora

1. $\forall v \in V - U.d(r,w) \leq d(r,v)$ e $d(r,w) = d_U(r,w)$
   si consideri il primo nodo $\notin U$ del cammino $p{:}\, r \rightsquigarrow v$ di lunghezza minima
2. $\forall v \in V - (U+w).d_{U+w}(r,v) = \min\{d_U(r,v), d_U(r,w) + l(w,v)\}$
   si consideri il penultimo nodo del cammino $p{:}\, r \rightsquigarrow v$ di lunghezza minima con nodi intermedi in $U + w$.

Per calcolarsi i cammini minimi basta mantenere un array p t.c. $p[v]$ contenga il padre di v nel cammino minimo da r a v con nodi intermedi in U.

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

5.2 Algoritmo di Floyd

L'algoritmo di Floyd serve a trovare i cammini minimi/calcolare le distanze tra tutte le possibili coppie di nodi 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*. 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 D_k di dimensione $V \times V$ t.c. $D_k[u,v]$ 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 $D_U[u,v] = d_U(u,v)$ con $U \subseteq V$. L'idea fondamentale e' come calcolare D_{U+w}, con $w \in V - U$, a partire da D_U. Un cammino da u a v con nodi intermedi un $U + w$ deve necessariamente essere di una delle seguenti forme:

- un cammino $p{:}\, u \rightsquigarrow v$ con nodi intermedi in U, e scriveremo $p{:}\, u \rightsquigarrow_U v$
- un cammino $p{:}\, u \rightsquigarrow v$ che ha w come nodo intermedio esattamente $k > 0$ volte, e quindi avremo che p e' ottenuto concatenando i cammini propri (cioe' con almeno un arco)
  - $p_0{:}\, u \rightsquigarrow_U w$
  - $p_i{:}\, w \rightsquigarrow_U w$ con $0 < i < k$
  - $p_k{:}\, w \rightsquigarrow_U v$

A questo punto si tratta di esaminare come puo' essere fatto il cammino $p{:}\, u \rightsquigarrow_{U+w} v$ di lunghezza minima:

- se $p{:}\, u \rightsquigarrow_U v$ allora la sua lunghezza sara' $D_U[u,v]$
- se $p{:}\, u \rightsquigarrow_U w \rightsquigarrow_U v$ allora la sua lunghezza sara' $D_U[u,w] + D_U[w,v]$
- se invece esiste un ciclo $q{:}\, w \rightsquigarrow_U w$ di lunghezza < 0, e quindi $D_U[w,w] < 0$, allora vi sono cammini $p{:}\, u \rightsquigarrow_{U+w} v$ di lunghezza negativa arbitrariamente grande (ripetendo il ciclo q). Indicheremo questa terza eventualita' ponendo $D_{U+w}[u,v] = -\infty$.

Occorre fare attenzione nel manipolare $+\infty$ e $-\infty$. 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 $(+\infty) + (-\infty) = +\infty$.

```
input  M:array [V,V] of pesi {+infinito indica assenza di arco}
output D:array [V,V] of pesi {+infinito indica assenza di cammini, -infinito
         indica cammini di lunghezza negativi arbitrariamente grande}
var    D:array [0..|V|,V,V] of pesi
for x,y in V do if x=y then D[0,x,y] <- min(0,M[x,y]) else D[0,x,y] <- M[x,y]

for k <- 1 to |V| do
  for x,y in V do
    D[k,x,y] <- min(D[k-1,x,y] , D[k-1,x,v_k]+D[k-1,v_k,v_k]^*+D[k-1,v_k,y])

for x,y in V do D[x,y] <- D[|V|,x,y]
```

dove $c^* = 0$ se $c \geq 0$, e $c^* = -\infty$ se $c < 0$. Per risparmiare memoria, si puo' evitare di mantenere un array di matrici e lavorare direttamente sulla matrice di output D, in tal caso occorre fare attenzione a sovrascrivere un valore solo quando non serve piu' per i calcoli intermedi.

```
input  M:array [V,V] of pesi
output D:array [V,V] of pesi
for x,y in V do
  if x=y and 0<=D[x,y] then D[x,y] <- 0 else D[x,y] <- M[x,y]

for k <- 1 to |V| do
  D[v_k,v_k] <- D[v_k,v_k]^* {ovvero 0 o -infinito}
  for x,y in V-v_k do
    D[x,y] <- min(D[x,y] , D[x,v_k]+D[v_k,v_k]+D[v_k,y])
  for x in V-v_k do
    D[x,v_k] <- D[x,v_k]+D[v_k,v_k]
  for y in V-v_k do
    D[v_k,y] <- D[v_k,v_k]+D[v_k,y]
```

Con questa ottimizzazione la complessita' spazio passa da $O(|V|^3)$ a $O(|V|^2)$, mentre la complessita' tempo rimane in $O(|V|^3)$. Dal punto di vista della complessita' l'algoritmo di Floyd e' equivalente ad eseguire l'algoritmo di Dijkstra $|V|$ volte, una per ogni $r \in V$. Tuttavia, l'algoritmo di Dijkstra e' corretto solo nell'ipotesi di pesi non negativi, mentre Floyd non richiede ipotesi aggiuntive. Infine consideriamo due varianti dell'algoritmo di Floyd, che oltre alla distanza $d(x, y)$ permettono di calcolare anche il cammino minimo da x a y, se esiste (cioe' se $-\infty < D[x, y] < +\infty$). In entrambe le varianti abbiamo in output matrice `D` delle distanze e una matrice `P:array [V,V] of V+{nil}` t.c. $P[x, y]$ contiene un nodo del cammino minimo da x a y. Nella prima variante (similmente a quanto fatto per Dijkstra) $P[x, y]$ e' il penultimo nodo del cammino minimo da x a y

```
input  M:array [V,V] of pesi
output D:array [V,V] of pesi
       P:array [V,V] of V+{nil}
for x,y in V do
  if x=y and 0<=M[x,y]
  then D[x,y] <- 0; P[x,y] <- nil
  else D[x,y] <- M[x,y]; P[x,y] <- x

for k <- 1 to |V| do
  D[v_k,v_k] <- D[v_k,v_k]^*; P[v_k,v_k] <- nil
  for x,y in V-v_k do
    if D[x,v_k]+D[v_k,v_k]+D[v_k,y] < D[x,y]
    then D[x,y] <- D[x,v_k]+D[v_k,v_k]+D[v_k,y]; P[x,y] <- P[v_k,y]
  for x in V-v_k do
    D[x,v_k] <- D[x,v_k]+D[v_k,v_k]
  for y in V-v_k do
    D[v_k,y] <- D[v_k,v_k]+D[v_k,y]
```

Nella seconda variante $P[x, y] = v_k$ sse v_k e' il nodo intermedio sul cammino minimo da x a y con il valore k piu' grande. La seconda variante e' piu' in sintonia con l'algoritmo di Floyd, tuttavia la ricostruzione del cammino minimo da x a y richiede una procedura ricorsiva, invece che iterativa.

```
input  M:array [V,V] of pesi
output D:array [V,V] of pesi
       P:array [V,V] of V+{nil}
for x,y in V do
  if x=y and 0<=D[x,y] then D[x,y] <- 0 else D[x,y] <- M[x,y]
  P[x,y] <- nil

for k <- 1 to |V| do
  D[v_k,v_k] <- D[v_k,v_k]^*; P[v_k,v_k] <- nil
  for x,y in V-v_k do
    if D[x,v_k]+D[v_k,v_k]+D[v_k,y] < D[x,y]
    then D[x,y] <- D[x,v_k]+D[v_k,v_k]+D[v_k,y]; P[x,y] <- v_k
  for x in V-v_k do
    D[x,v_k] <- D[x,v_k]+D[v_k,v_k]
  for y in V-v_k do
    D[v_k,y] <- D[v_k,v_k]+D[v_k,y]
```