Algoritmi e Strutture Dati

Capitolo 13
Cammini minimi:
Algoritmo di Dijkstra (*)

(ACM in grafi diretti e non diretti senza archi di peso negativo)

Punto della situazione

- Algoritmo basato sull'ordinamento topologico: albero dei cammini minimi in grafi diretti aciclici. Complessità $\Theta(n+m)$ (grafo rappresentato con liste di adiacenza).
- Algoritmo di Bellman&Ford: albero dei cammini minimi in grafi diretti che non contengono cicli negativi, ovvero grafi non diretti che non contengono archi di peso negativo. Complessità $\Theta(n \cdot m)$ (grafo rappresentato con liste di adiacenza).

Copyright © 2004 - The McGraw - Hill Companies, srl

Algoritmo di Dijkstra (1959)

(albero dei cammini minimi in grafi (sia diretti che non diretti) con pesi non negativi)

Copyright © 2004 - The McGraw - Hill Companies, srl

Richiamo sulla notazione

$w(u,v)$: peso dell'arco (u,v) del grafo pesato (diretto o non diretto) $G=(V,E,w)$

$w(\pi)$: lunghezza di un cammino π nel grafo, ovvero somma dei pesi di tutti gli archi del cammino

π_{uv}: cammino minimo nel grafo tra i nodi u e v

d_{uv}: distanza nel grafo (ovvero lunghezza del cammino minimo) tra i nodi u e v

D_{uv}: sovrastima della distanza nel grafo tra i nodi u e v (cioè $D_{uv} \geq d_{uv}$)

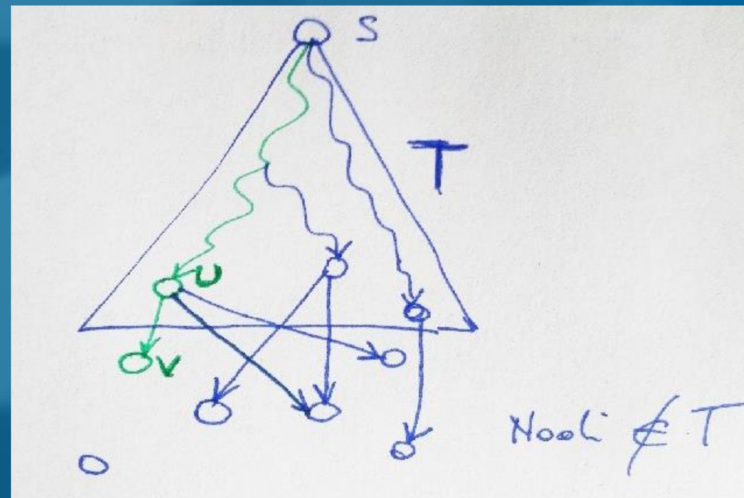
Copyright © 2004 - The McGraw - Hill Companies, srl

Estendere l'albero dei cammini minimi

Lemma di Dijkstra (1959): Sia G=(V,E,w) (diretto o non diretto) con pesi non negativi, e sia T un sottoalbero dell'albero dei cammini minimi radicato in s che include s ma non include tutti i vertici raggiungibili da s. Sia:

$$(u,v) = \arg\min\{d_{st} + w(t,z) \mid (t,z) \in E,\ t \in T \text{ e } z \notin T\}$$

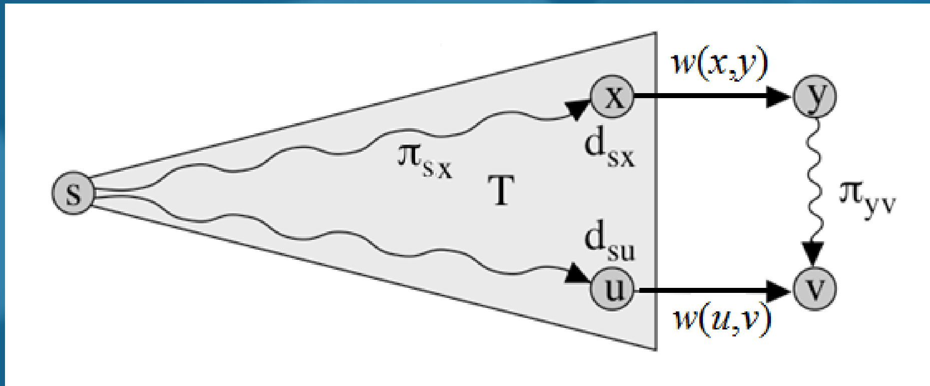
(in altre parole, v è il nodo non appartenente a T più vicino ad s).
Allora, (u,v) appartiene a un cammino minimo da s a v.



Copyright © 2004 - The McGraw - Hill Companies, srl

Prova del lemma di Dijkstra (1/2)

Dim.: Supponiamo per assurdo che (u,v) non appartenga ad un cammino minimo da s a v, e quindi che $d_{sv} < d_{su} + w(u,v)$. Allora, d_{sv} è la lunghezza di un cammino minimo da s a v che non passa per (u,v). Tale cammino, per uscire da T, passerà allora (almeno una prima volta) per un qualche arco $(x,y) \neq (u,v)$, con $x \in T$ e $y \notin T$. Sia quindi $\pi_{sv} = \langle s,\ldots,x,y,\ldots,v \rangle$.



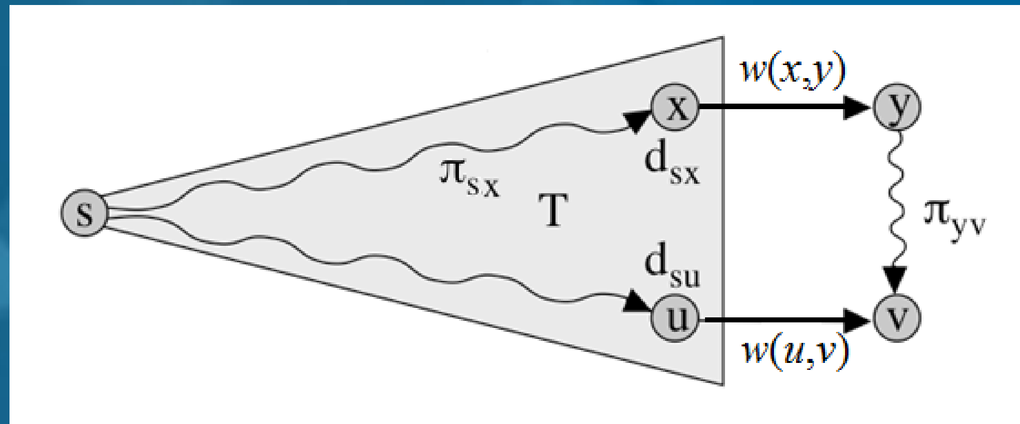
Si noti che il cammino π_{yv} potrebbe rientrare in T in quanto abbiamo ammesso la possibilità che gli archi abbiano peso pari a 0

Per la minimalità dei sottocammini di un cammino minimo, il cammino da s a x deve essere minimo e avere quindi lunghezza d_{sx}

$$w(\pi_{sv}) = w(\pi_{sx}) + w(x,y) + w(\pi_{yv}) = d_{sx} + w(x,y) + w(\pi_{yv}).$$

Copyright © 2004 - The McGraw - Hill Companies, srl

Prova del lemma di Dijkstra (2/2)



Ma poiché (u,v) minimizza d_{st}+ w(t,z) per ogni t∈T e z∉T, allora:

$$d_{sx} + w(x,y) \geq d_{su} + w(u,v)$$

e quindi:

$$w(\pi_{sv}) = d_{sx} + w(x,y) + w(\pi_{yv}) \geq d_{su} + w(u,v) + w(\pi_{yv})$$

e poiché $w(\pi_{yv}) \geq 0$ (si noti che in questo punto si sfrutta il fatto che i pesi di G sono non negativi, e quindi i cammini hanno sempre lunghezza ≥ 0), ne segue $d_{sv} \equiv w(\pi_{sv}) \geq d_{su} + w(u,v)$, assurdo (avevamo supposto $d_{sv} < d_{su} + w(u,v)$). ■

Copyright © 2004 - The McGraw - Hill Companies, srl

Approccio di Dijkstra

Sia T il sottoalbero dei cammini minimi costruito sinora (all'inizio T contiene solo il nodo sorgente s). Supponiamo quindi che l'algoritmo abbia già trovato d_{su} per ogni $u \in T$, e abbia invece una sovrastima D_{sv} per ogni $v \notin T$. Al passo successivo, seleziona il nodo $v \notin T$ che minimizza la quantità $D_{sv} := d_{su} + w(u,v)$ (si noti che D_{sv} coincide con d_{sv} per il lemma di Dijkstra), con $u \in T$; quindi, aggiungi v a T ed effettua il passo di rilassamento su ogni nodo $y \notin T$ adiacente a v (per il quale cioè $(v,y) \in E$).

I nodi da aggiungere progressivamente a T sono mantenuti in una coda di priorità, associati ad un unico arco che li connette a T. Se y è in coda con arco (x,y) associato, e se dopo aver aggiunto v a T troviamo un arco (v,y) tale che $D_{sv} + w(v,y) < D_{sx} + w(x,y)$, allora rimpiazziamo (x,y) con (v,y), ed aggiorniamo D_{sy}.

Copyright © 2004 - The McGraw - Hill Companies, srl

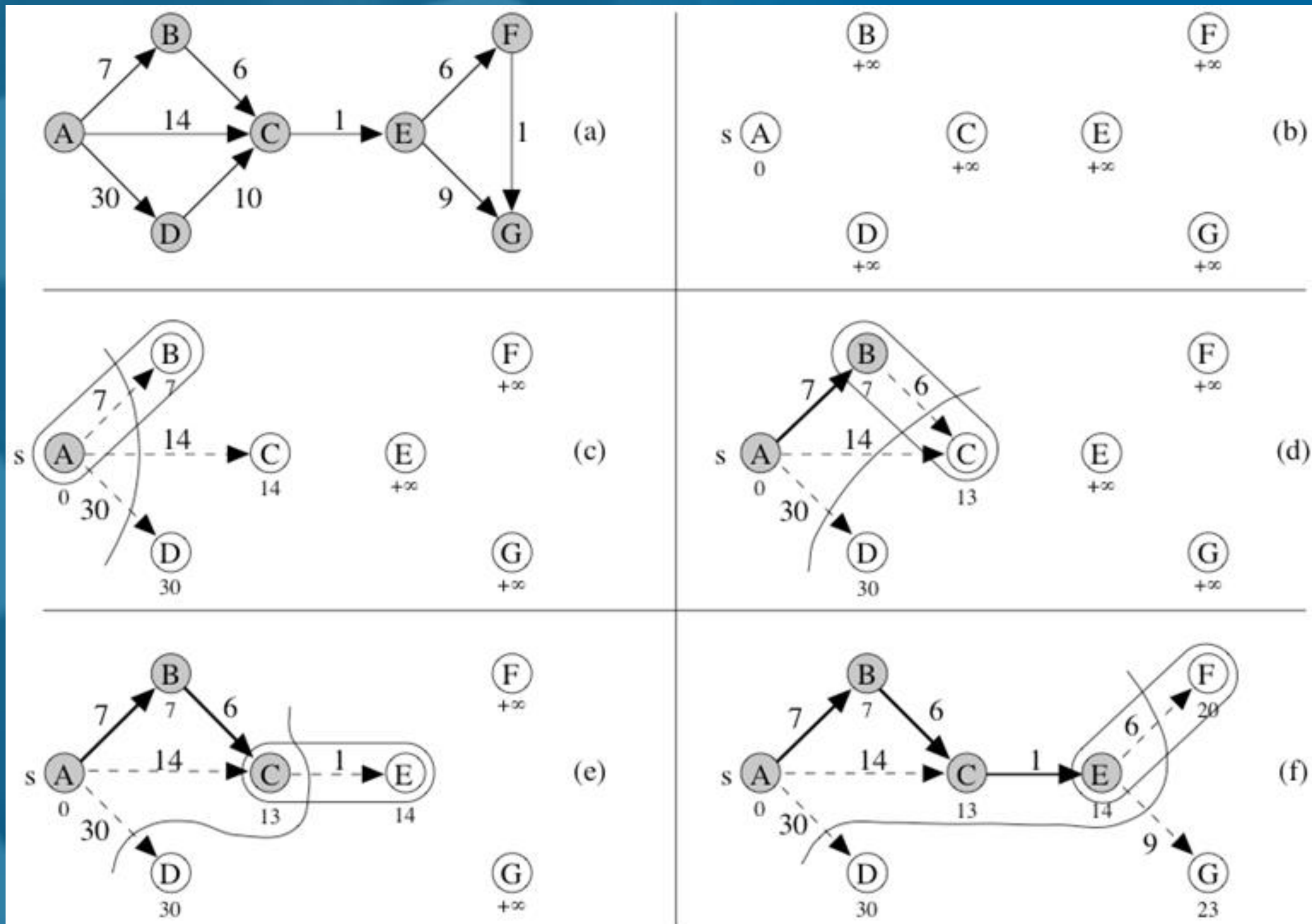
Pseudocodice

algoritmo `Dijkstra`$(grafo\ G, vertice\ s) \rightarrow albero$
  for each (vertice u in G) **do** $D_{su} \leftarrow +\infty$
  $\widehat{T} \leftarrow$ albero formato dal solo nodo s
  `CodaPriorita S`
  $D_{ss} \leftarrow 0$
  `S.insert`$(s, 0)$
  while (**not** `S.isEmpty`()) **do**
    $v \leftarrow$`S.deleteMin`()
    for each (arco (v, y) in G) **do**
      if $(D_{sy} = +\infty)$ **then**
        `S.insert`$(y, D_{sv} + w(v, y))$
        $D_{sy} \leftarrow D_{sv} + w(v, y)$
        rendi v padre di y in $\widehat{T}$
      else if $(D_{sv} + w(v, y) < D_{sy})$ **then**
        `S.decreaseKey`$(y, D_{sy} - D_{sv} - w(v, y))$
        $D_{sy} \leftarrow D_{sv} + w(v, y)$
        rendi v nuovo padre di y in $\widehat{T}$
  return $\widehat{T}$

Nota: $\widehat{T}$ è un albero che contiene tutti i nodi già aggiunti alla soluzione (ovvero quelli in T secondo la notazione del lemma di Dijkstra, per i quali è già stato trovato il cammino minimo da s), più i nodi correntemente contenuti nella coda di priorità, cioè “appesi” a T, ciascuno connesso al rispettivo nodo padre. Si noti che a differenza dei due algoritmi già visti, in questo caso viene restituito l’ACM (e non le distanze da s)

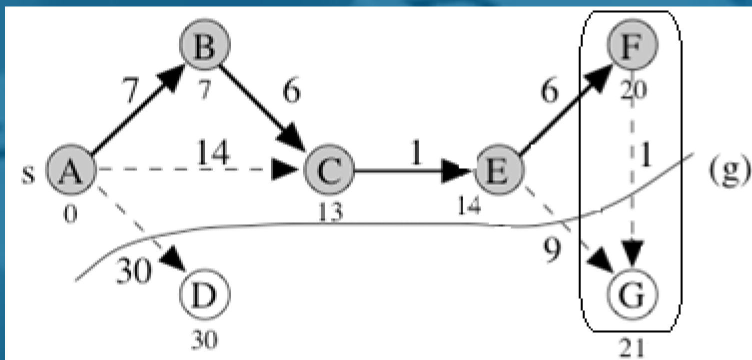
Copyright © 2004 - The McGraw - Hill Companies, srl

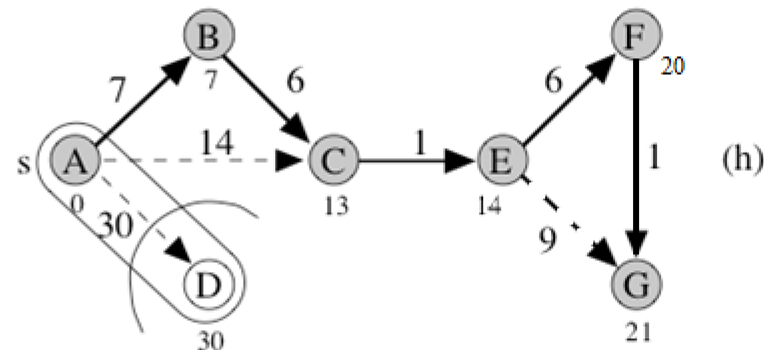
Esempio (1/2)

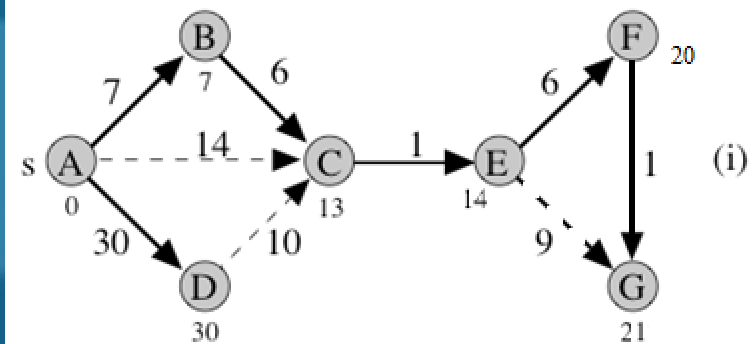


Copyright © 2004 - The McGraw - Hill Companies, srl

Esempio (2/2)







Copyright © 2004 - The McGraw - Hill Companies, srl

Tempo di esecuzione: implementazioni elementari

Supponendo che il grafo $G=(V,E,w)$ sia rappresentato tramite liste di adiacenza e che $|V|=n$ ed $|E|=m$, e supponendo che G sia fortemente connesso rispetto ad s (e quindi $m \geq n-1$), avremo n `insert`, n `deleteMin` e al più m `decreaseKey` nella coda di priorità S, al costo di:

	Insert	DelMin	DecKey
Array non ord.	$O(1)$	$\Theta(\|S\|)=O(n)$	$O(1)$
Array ordinato	$O(\|S\|)=O(n)$	$O(1)$	$O(\|S\|)=O(n)$
Lista non ord.	$O(1)$	$\Theta(\|S\|)=O(n)$	$O(1)$
Lista ordinata	$O(\|S\|)=O(n)$	$O(1)$	$O(\|S\|)=O(n)$

- $n \cdot O(1) + n \cdot O(n) + O(m) \cdot O(1) = O(n^2)$ con array non ordinati
- $n \cdot O(n) + n \cdot O(1) + O(m) \cdot O(n) = O(m \cdot n)$ con array ordinati
- $n \cdot O(1) + n \cdot O(n) + O(m) \cdot O(1) = O(n^2)$ con liste non ordinate
- $n \cdot O(n) + n \cdot O(1) + O(m) \cdot O(n) = O(m \cdot n)$ con liste ordinate

Copyright © 2004 - The McGraw - Hill Companies, srl

Osservazione sulla `decreaseKey`

- Ricordiamo che le complessità computazionali esposte per la `decreaseKey` sono valide supponendo di avere un puntatore diretto all'elemento su cui eseguire l'operazione. Come possiamo garantire tale condizione?
- Semplicemente inizializzando un puntatore tra il nodo v nell'array dei nodi della lista di adiacenza del grafo e l'elemento nella coda di priorità associato al nodo v; tale puntatore viene inizializzato nella fase di inserimento di quest'ultimo all'interno della coda.

Copyright © 2004 - The McGraw - Hill Companies, srl

Tempo di esecuzione: implementazioni efficienti

Supponendo che il grafo G=(V,E,w) sia rappresentato tramite liste di adiacenza e che |V|=n ed |E|=m, e supponendo che G sia fortemente connesso rispetto ad s (e quindi m≥n-1), avremo n `insert`, n `deleteMin` e al più m `decreaseKey` nella coda di priorità, al costo di:

	Insert	DelMin	DecKey
Heap binario	O(log n)	O(log n)	O(log n)
Heap Binom.	O(log n)	O(log n)	O(log n)
Heap Fibon.	O(1)	O(log n)* (ammortizzata)	O(1)* (ammortizzata)

- $n \cdot O(\log n) + n \cdot O(\log n) + O(m) \cdot O(\log n) = O(m \cdot \log n)$ utilizzando heap binari o binomiali
- $n \cdot O(1) + n \cdot O(\log n)^* + O(m) \cdot O(1)^* = O(m + n \cdot \log n)$ utilizzando heap di Fibonacci (Esercizio di approfondimento: si dimostri che questa è l'implementazione più efficiente tra quelle descritte)

Copyright © 2004 - The McGraw - Hill Companies, srl

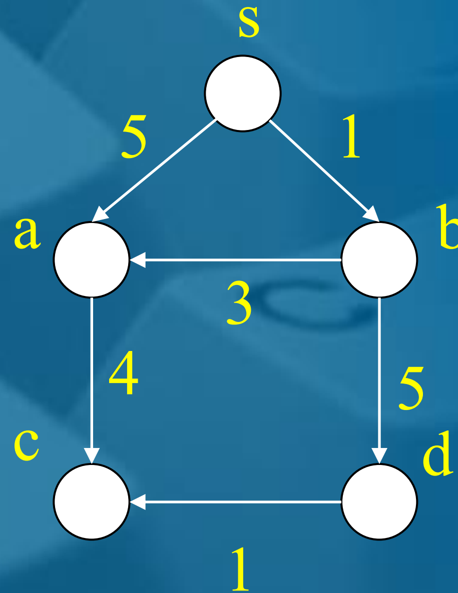
Confronto tra Bellman&Ford e Dijkstra

- Innanzitutto, si noti che B&F si applica ad una classe molto più vasta di grafi, ovvero su tutti i grafi orientati/non orientati che non hanno cicli/archi di costo negativo, che sono poi tutti e solo quelli per i quali i cammini minimi possono effettivamente essere calcolati
- Si noti però che l'implementazione di Dijkstra con heap di Fibonacci è sempre più efficiente di B&F; infatti, $O(m+n \log n)=o(n\,m)$, poiché $m=o(n\,m)$ e $n \log n = o(n\,m)$
- Nonostante l'algoritmo di Dijkstra abbia oltre mezzo secolo di vita, è al momento l'algoritmo più efficiente per la classe di grafi a cui si applica; ricordando però che l'unico lower bound noto per il problema del cammino minimo tra due nodi (e quindi anche per l'ACM) è $\Omega(m+n)$, ne consegue che non lo si può definire ottimo, in quanto per $m=o(n \log n)$, permane un gap di $\log n$ rispetto al termine additivo in n; invece, per $m=\Omega(n \log n)$, si ha che $O(m + n \log n) = O(m) = O(m+n)$, e quindi diventa ottimo
- Per istanze pratiche, l'algoritmo di Dijkstra è molto efficiente; infatti, ad esempio, su grafi planari (con pesi non negativi, come le reti stradali), usando la formula di Eulero si può dimostrare che $m \leq 3n-6$, cioè $m=\Theta(n)$, ovvero $O(m + n \log n) = O(n \log n)$, e quindi l'algoritmo è quasi lineare in n; nell'esempio del navigatore satellitare, questa complessità si traduce in un tempo di risposta vicino al secondo

Copyright © 2004 - The McGraw - Hill Companies, srl

Approfondimento

Applicare l'algoritmo di Dijkstra con sorgente s sul seguente grafo:



Copyright © 2004 - The McGraw - Hill Companies, srl