Algoritmi e Strutture Dati

Capitolo 13
Cammini minimi:
Algoritmo di Floyd e Warshall (1962)

Punto della situazione

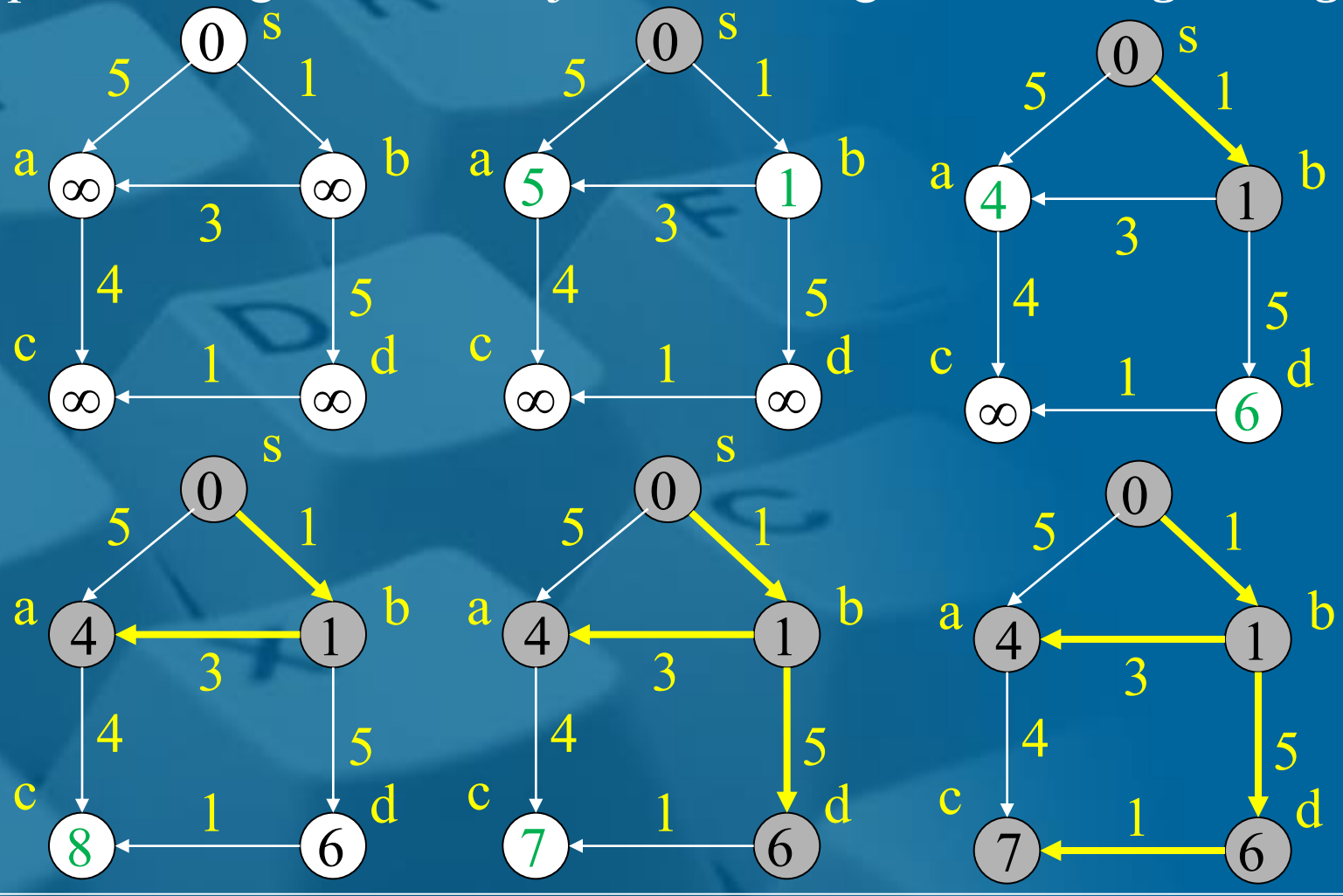
- Algoritmo basato sull'ordinamento topologico: albero dei cammini minimi in grafi diretti aciclici. Complessità $\Theta(n+m)$ (grafo rappresentato con liste di adiacenza).
- Algoritmo di Bellman&Ford: albero dei cammini minimi in grafi diretti che non contengono cicli negativi, o in grafi non diretti che non contengono archi di peso negativo. Complessità $\Theta(n \cdot m)$ (grafo rappresentato con liste di adiacenza).
- Algoritmo di Dijkstra: albero dei cammini minimi in grafi (diretti e non diretti) che non contengono archi di peso negativo. Complessità $O(m + n \log n)$ (grafo rappresentato con liste di adiacenza e uso degli Heap di Fibonacci). Come si confronta tale implementazione con quelle alternative basate su vettori/liste/heap binari/heap binomiali viste a lezione?

Soluzione esercizio di approfondimento: confronto tra le varie implementazioni di Dijkstra

- L'implementazione di Dijkstra con heap di Fibonacci è la più efficiente delle implementazioni presentate, poiché:
  - $m + n \log n = O(n^2)$ (liste/array non ord.), in quanto $m = O(n^2)$ e $n \log n = o(n^2)$ (si noti che i due approcci sono equivalenti solo nel caso in cui $m=\Theta(n^2)$)
  - $m + n \log n = o(n\, m)$ (liste/array ord.), in quanto $m = o(n\, m)$ e $n \log n = o(n\, m)$
  - $m + n \log n = O(m \log n)$ (heap binari/binomiali), in quanto $m = o(m \log n)$ e $n \log n = O(m \log n)$ (si noti che i due approcci sono equivalenti solo nel caso in cui $m=\Theta(n)$)

Soluzione esercizio di approfondimento: esecuzione di Dijkstra

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



Domanda

- Quanto costano i vari algoritmi se il grafo è rappresentato mediante una matrice di adiacenza (pensateci, lo chiederò all'orale)?
- Algoritmo basato sull'ordinamento topologico: $\Theta(n^2)$
- Algoritmo di Bellman&Ford: $\Theta(n^3)$
- Algoritmo di Dijkstra:
  - liste/array non ordinate: $O(n^2)$
  - liste/array ordinate: $O(n\ m)$
  - heap binari/binomiali: $O(n^2 + m \log n)$
  - heap di Fibonacci: $O(n^2)$

Domanda: Qual è la miglior soluzione da adottare per Dijkstra se G fosse rappresentato mediante matrice di adiacenza?

Algoritmo di Floyd e Warshall

(cammini minimi tra tutte le coppie di nodi in grafi diretti che non contengono cicli negativi, o in grafi non diretti che non contengono archi di peso negativo)

Premessa

- Una **tecnica algoritmica** può essere vista come un meta-algoritmo che permette di risolvere un'ampia classe di problemi, i quali, per loro natura, si prestano ad essere risolti appunto in modo analogo
- In particolare, abbiamo già visto dettagliatamente la tecnica *divide et impera*
- Oggi vedremo la tecnica della Programmazione Dinamica, che è esattamente quella adottata dall'algoritmo di Floyd&Warshall

Richiamo: la tecnica *divide et impera*

Tecnica top-down:

1. Dividi l'istanza del problema in due o più sottoistanze (sottoproblemi)
2. Risolvi ricorsivamente il problema sulle sottoistanze
3. Ricombina opportunamente le soluzioni dei sottoproblemi allo scopo di ottenere la soluzione globale del problema

Esempi: mergesort, quicksort

Programmazione dinamica

Tecnica bottom-up:

1. Identifica dei sottoproblemi del problema originario, procedendo logicamente dai problemi più piccoli verso quelli più grandi
2. Utilizza una tabella per memorizzare le soluzioni dei sottoproblemi incontrati: quando si incontra lo stesso sottoproblema, sarà sufficiente esaminare la tabella
3. Si usa quando i sottoproblemi non sono indipendenti, ovvero quando vale il principo di sottostruttura ottima della soluzione: una sottosoluzione può essere usata direttamente per costruire la soluzione del problema!

Esempi?

Un esempio banale: `Fibonacci3`

algoritmo `fibonacci3`*(intero n)* → *intero*

sia Fib un array di n interi

Fib[1] ← *Fib[2]* ← *1*

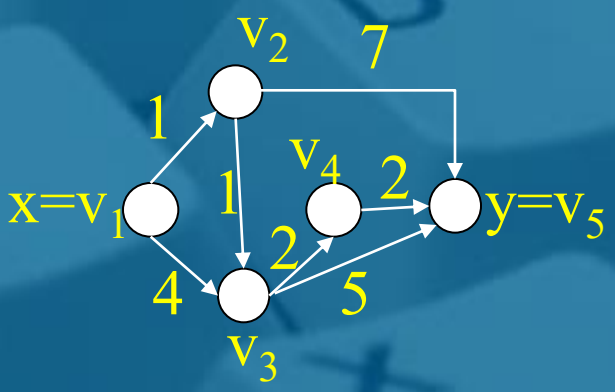
for i = 3 **to** n **do**

Fib[i] ← *Fib[i-1]* + *Fib[i-2]*

return *Fib[n]*

Approccio di Floyd e Warshall

- Elegante applicazione della tecnica della programmazione dinamica
- Supponiamo di enumerare i vertici di $G=(V,A,w)$ da 1 a n, cioè $V=\{v_1, v_2,\ldots, v_n\}$. Un cammino minimo k-vincolato da x a y è un cammino di costo minimo tra tutti i cammini da x a y che possono usare come vertici intermedi solo un sottoinsieme (anche vuoto) dei vertici $I_k=\{v_1, v_2,\ldots, v_k\}$ (in particolare, un cammino minimo 0-vincolato tra due vertici x e y non può usare vertici intermedi, e quindi esiste se e solo se esiste l'arco (x,y) in G)



Tra x e y, il cammino minimo:

- 0-vincolato è lungo $+\infty$
- 1-vincolato è lungo $+\infty$
- 2-vincolato è lungo 8: $\langle x,v_2,y\rangle$;
- 3-vincolato è lungo 7: $\langle x,v_2,v_3,y\rangle$;
- 4-vincolato è lungo 6: $\langle x,v_2,v_3,v_4,y\rangle$;
- 5-vincolato (ovvero senza vincoli, e quindi il cammino minimo) è lungo 6: $\langle x,v_2,v_3,v_4,y\rangle$.

- Idea di Floyd e Warshall: calcolare cammini minimi k-vincolati per k=0,1,…, n

Relazioni tra distanze vincolate

- Sia d_{xy}^{k} il costo di un cammino minimo k-vincolato da x a y. Chiaramente, valgono le seguenti proprietà:
  - $d_{xy}^{0} = w(x,y)$ se $(x,y) \in A$, $+\infty$ altrimenti
  - $d_{xv_k}^{k-1} = d_{xv_k}^{k}$ e $d_{v_kx}^{k-1} = d_{v_kx}^{k}$
  - $d_{xy}^{n} = d_{xy}$
- Per le proprietà di cui sopra e per la proprietà di minimalità dei sottocammini di cammini minimi, si ha:

$$d_{xy}^{k} = \min\{ d_{xy}^{k-1}, d_{xv_k}^{k} + d_{v_ky}^{k} \} = \min\{ d_{xy}^{k-1}, d_{xv_k}^{k-1} + d_{v_ky}^{k-1} \}$$

$\Rightarrow$L'algoritmo calcola d_{xy} dal basso verso l'alto, incrementando k da 0 a n

Pseudocodice (su grafi diretti)

algoritmo `FloydWarshall`$(grafo\ diretto\ G) \rightarrow distanze$
  inizializza D tale che $D_{xy} = w(x,y)$ se $(x,y) \in A$, e $D_{xy} = +\infty$ altrimenti
  for each $v \in V$ **do**
    for each $((x,y) \in V \times V)$ **do**
      if $(D_{xv} + D_{vy} < D_{xy})$ **then** $D_{xy} \leftarrow D_{xv} + D_{vy}$
  return D

Tempo di esecuzione: $\Theta(n^3)$

(sia con liste di adiacenza che con matrice di adiacenza)

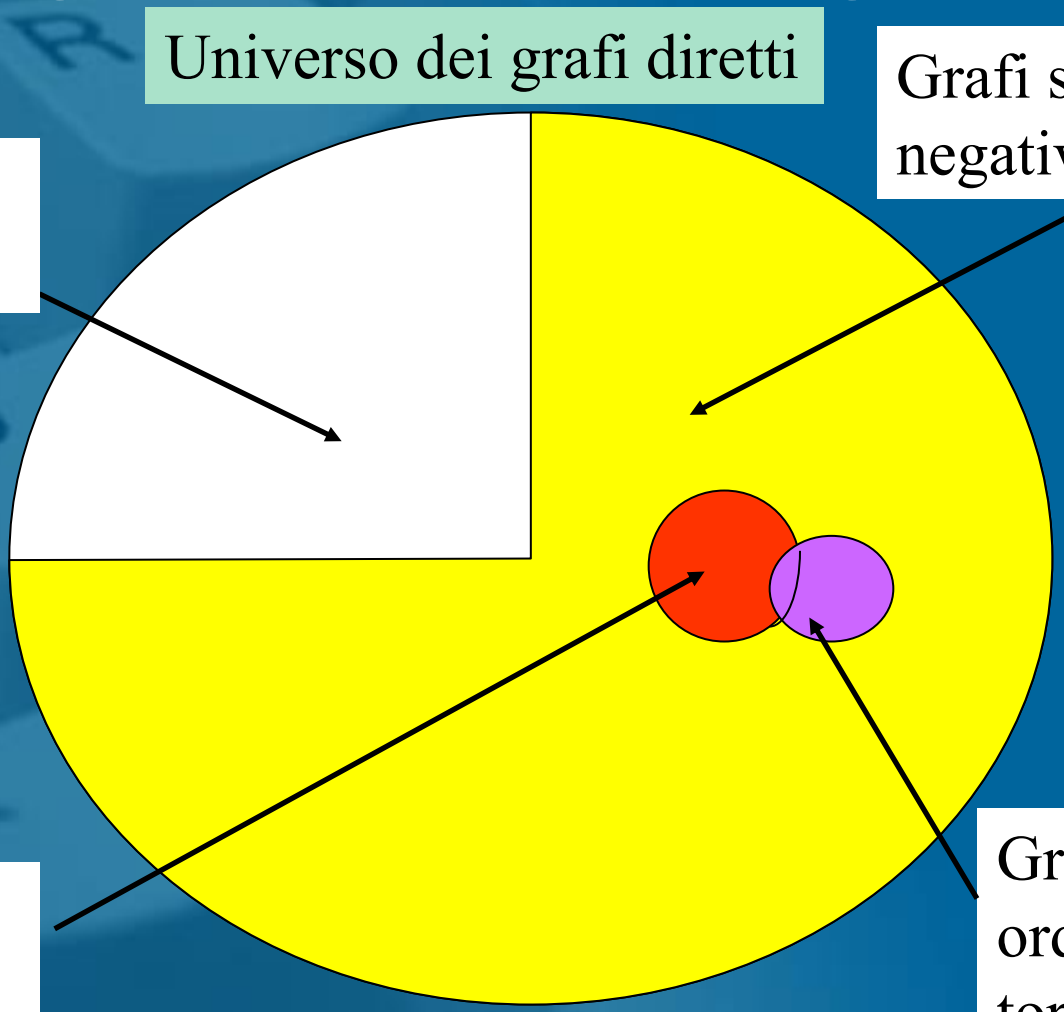
D: Come si confronta con l'applicazione ripetuta di Dijkstra?

R: Utilizzando gli Heap di Fibonacci, n applicazioni dell'algoritmo di Dijkstra richiederanno tempo $O(n\ (m+n \log n)) = O(n\ m+n^2 \log n) = O(n^3)$. Quindi, Dijkstra è più efficiente. Tuttavia, si applica solo su un sottoinsieme delle istanze ammissibili per F&W.

D: Come si confronta con l'applicazione ripetuta di Bellman e Ford?

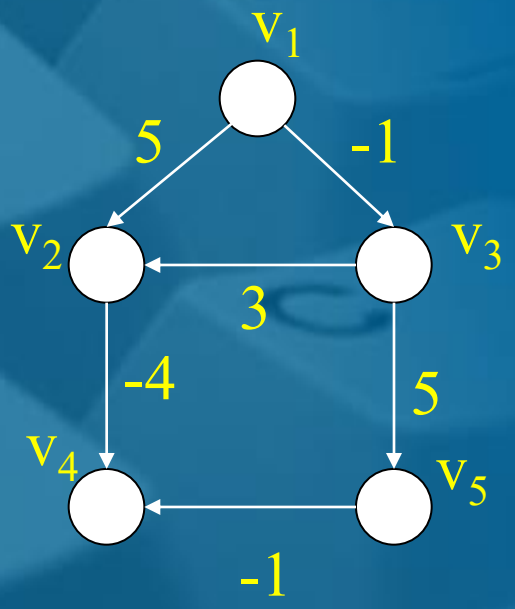
R: Si osservi innanzitutto che i due algoritmi si applicano alla stessa classe di grafi . Le n applicazioni dell'algoritmo di B&F richiederanno tempo $\Theta(n\ (m\ n)) = \Theta(m\ n^2) = \Omega(n^3)$ (sotto l'ipotesi realistica che $m=\Omega(n)$). Quindi, F&W è sempre più efficiente, escluso il caso in cui $m = \Theta(n)$, in cui i due approcci si equivalgono. Tuttavia, B&F è in grado di rilevare cicli di costo negativo.

Riepilogo grafico di applicabilità dei vari algoritmi studiati (per grafi diretti)



Approfondimento

Applicare l'algoritmo di Floyd e Warshall al seguente grafo:

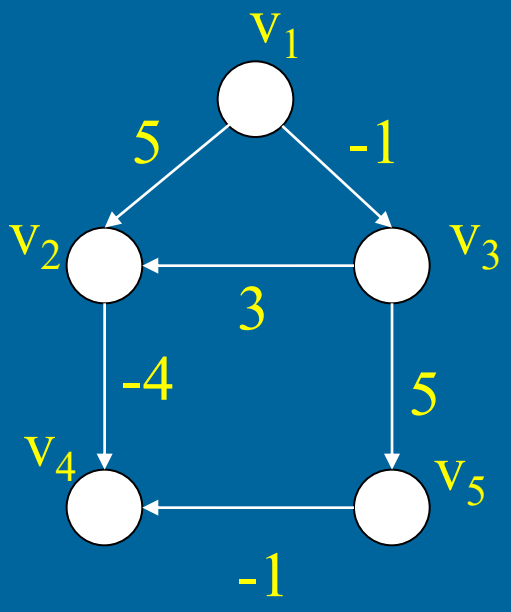


Soluzione

Posso applicare F&W?

Sì, non ci sono cicli (negativi)!

Inizializziamo la matrice delle distanze:



$$D_0 = \begin{bmatrix} 0 & 5 & -1 & +\infty & +\infty \\ +\infty & 0 & +\infty & -4 & +\infty \\ +\infty & 3 & 0 & +\infty & 5 \\ +\infty & +\infty & +\infty & 0 & +\infty \\ +\infty & +\infty & +\infty & -1 & 0 \end{bmatrix} \quad D_1 = \begin{bmatrix} 0 & 5 & -1 & +\infty & +\infty \\ +\infty & 0 & +\infty & -4 & +\infty \\ +\infty & 3 & 0 & +\infty & 5 \\ +\infty & +\infty & +\infty & 0 & +\infty \\ +\infty & +\infty & +\infty & -1 & 0 \end{bmatrix}$$

$$D_2 = \begin{bmatrix} 0 & 5 & -1 & \mathbf{1} & +\infty \\ +\infty & 0 & +\infty & -4 & +\infty \\ +\infty & 3 & 0 & \mathbf{-1} & 5 \\ +\infty & +\infty & +\infty & 0 & +\infty \\ +\infty & +\infty & +\infty & -1 & 0 \end{bmatrix} \quad D_3 = \begin{bmatrix} 0 & \mathbf{2} & -1 & \mathbf{-2} & \mathbf{4} \\ +\infty & 0 & +\infty & -4 & +\infty \\ +\infty & 3 & 0 & -1 & 5 \\ +\infty & +\infty & +\infty & 0 & +\infty \\ +\infty & +\infty & +\infty & -1 & 0 \end{bmatrix}$$

$$D_4 = \begin{bmatrix} 0 & 2 & -1 & -2 & 4 \\ +\infty & 0 & +\infty & -4 & +\infty \\ +\infty & 3 & 0 & -1 & 5 \\ +\infty & +\infty & +\infty & 0 & +\infty \\ +\infty & +\infty & +\infty & -1 & 0 \end{bmatrix} \quad D_5 = \begin{bmatrix} 0 & 2 & -1 & -2 & 4 \\ +\infty & 0 & +\infty & -4 & +\infty \\ +\infty & 3 & 0 & -1 & 5 \\ +\infty & +\infty & +\infty & 0 & +\infty \\ +\infty & +\infty & +\infty & -1 & 0 \end{bmatrix}$$