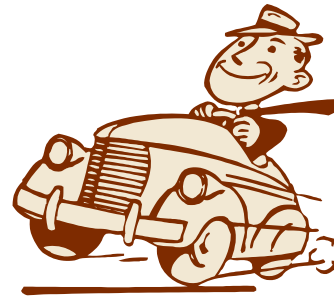
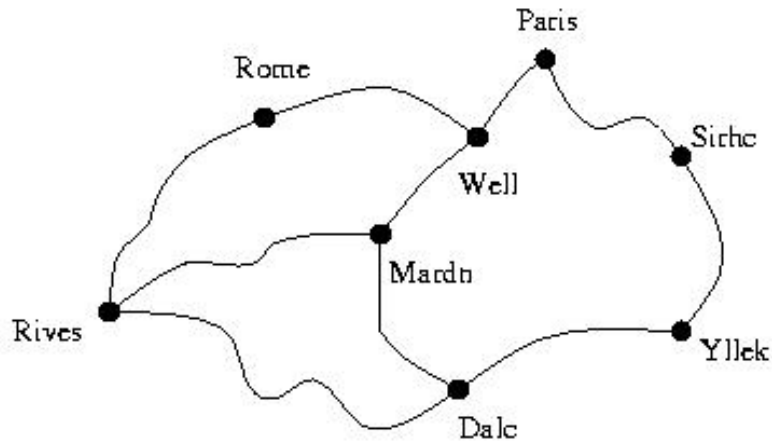

Cammini minimi in un grafo orientato pesato

Algoritmi di Dijkstra e Bellman-Ford
per il problema del cammino minimo
da sorgente singola

Un problema di “percorso”



Dati una mappa stradale (con distanze ad es. in chilometri) ed un punto di partenza s trovare i percorsi più brevi da s a ciascuna delle altre località

Problemi di ottimizzazione

Problema: data una misura (costo o obiettivo) individuare una soluzione ammissibile la cui misura sia minima o massima

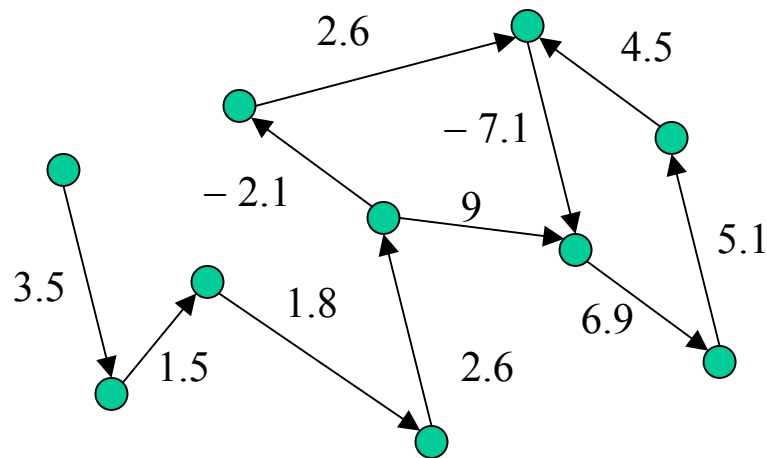


Senza offesa, questi due
vorrei vederli il meno
possibile



Grafi pesati

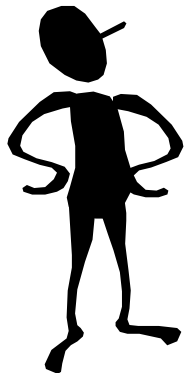
$$p : A \rightarrow \mathbb{R}$$



$$G = (N, A, p)$$

Funzione “peso”

I pesi possono essere anche negativi



Anche se non nel mio caso



Peso di un cammino e distanze

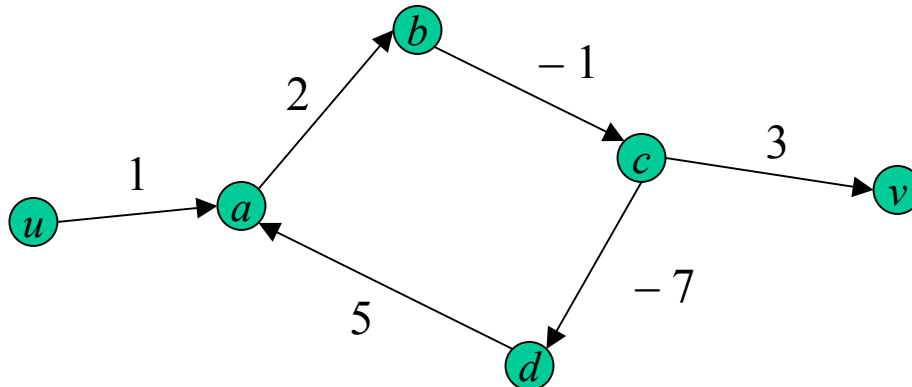
Il *peso di un cammino* è la somma dei pesi dei suoi archi:

$$p(v_0, v_1, \dots, v_k) = p(v_0, v_1) + p(v_1, v_2) + \dots + p(v_{k-1}, v_k)$$

La *distanza* di un vertice dall'altro è il minimo dei pesi tra tutti i cammini che congiungono il primo al secondo (se esistono):

$$\delta(u, v) = \begin{cases} \min \{p(c) : c \text{ cammino da } u \text{ a } v\} & \text{se ne esistono} \\ \infty & \text{altrimenti} \end{cases}$$

Cicli a somma negativa



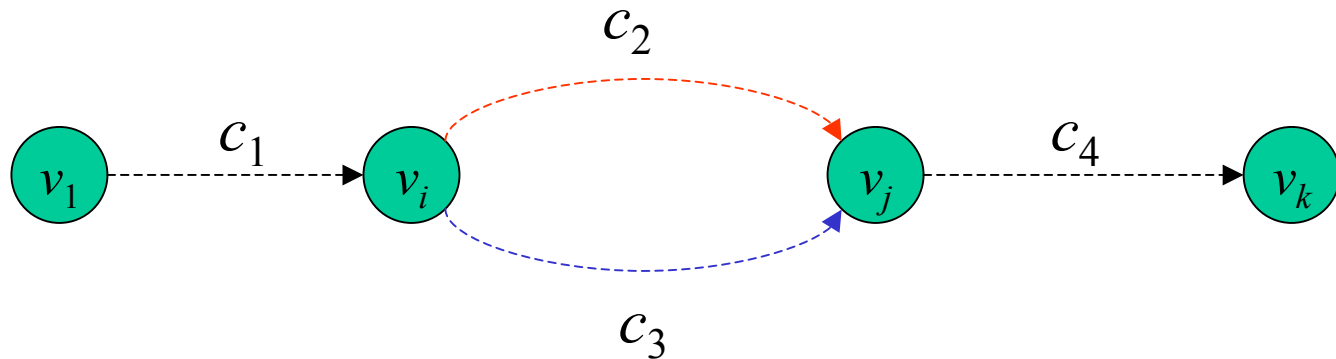
Ma quanto vale
 $\delta(u,v)$?

$$p(a,b,c,d) = 2 - 1 - 7 + 5 = -1$$

Perché $\delta(u,v)$ sia sempre
definito assumeremo che non ci
siano cicli a somma negativa.



Sottocammini di quello ottimo



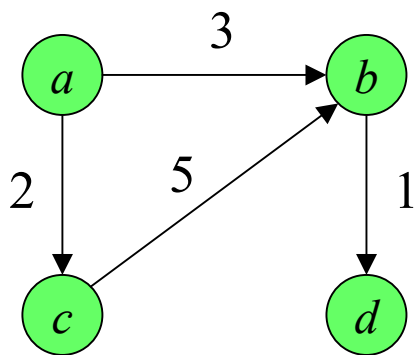
se $p(c_3) < p(c_2)$ allora

$$p(c_1 c_3 c_4) = p(c_1) + p(c_3) + p(c_4) < p(c_1) + p(c_2) + p(c_4) = p(c_1 c_2 c_4)$$

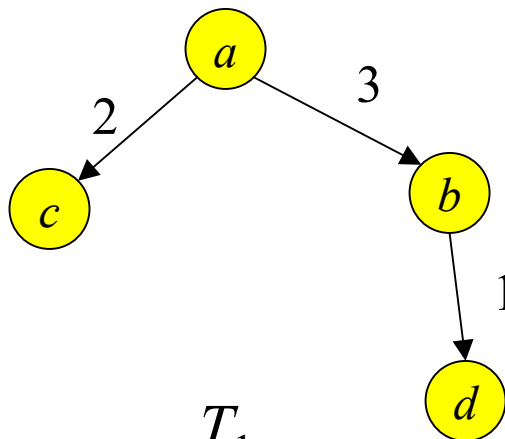
Se un cammino è ottimo, tali sono tutti i
suoi sottocammini.

Coperture, ovvero soluzioni

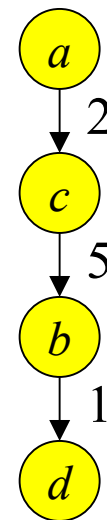
Un albero T è una *copertura* di G se è un albero sorgente ed un sottografo di G che comprende tutti i suoi vertici.



G



T_1



T_2

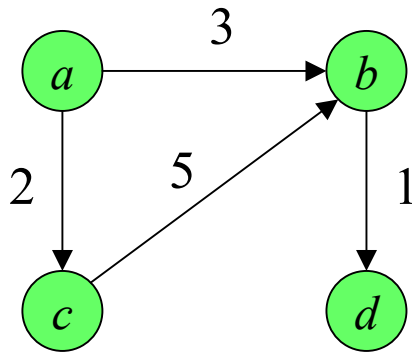
Ci sono due
coperture-soluzioni

Coperture, ovvero soluzioni

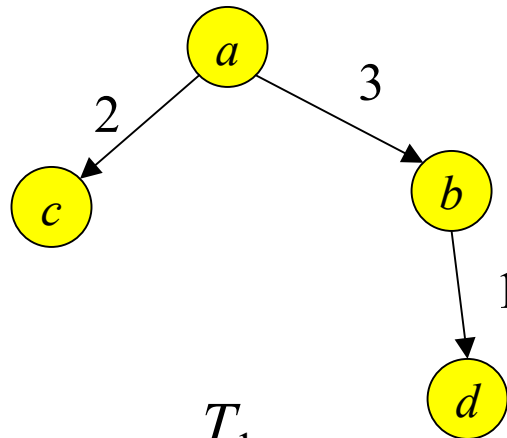
Se s è la radice di T , v in T e c il cammino da s a v allora:

$$d[v] = p(c) = \delta_T(s, v)$$

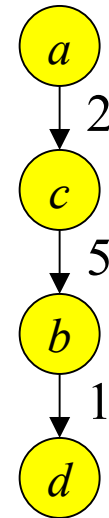
La distanza s, v
in T



G



T_1



T_2

$$d_1[c] = 2, d_1[b] = 3, d_1[d] = 4$$

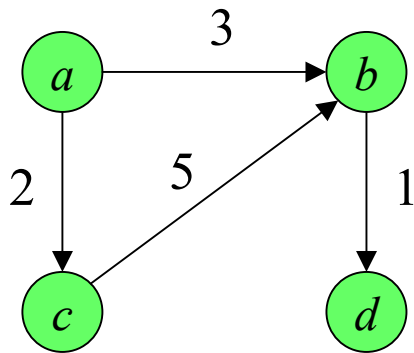
$$d_2[c] = 2, d_2[b] = 7, d_2[d] = 8$$

Soluzioni ottime

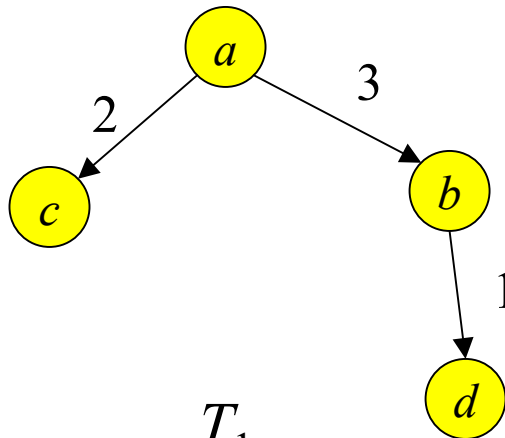
T è ottimo per G da s se è una soluzione e per ogni v :

$$d[v] = \delta_G(s, v)$$

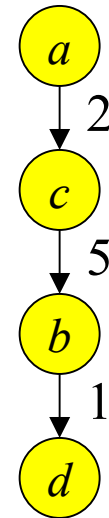
La distanza s, v
in G



G

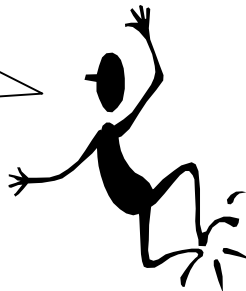


T_1



T_2

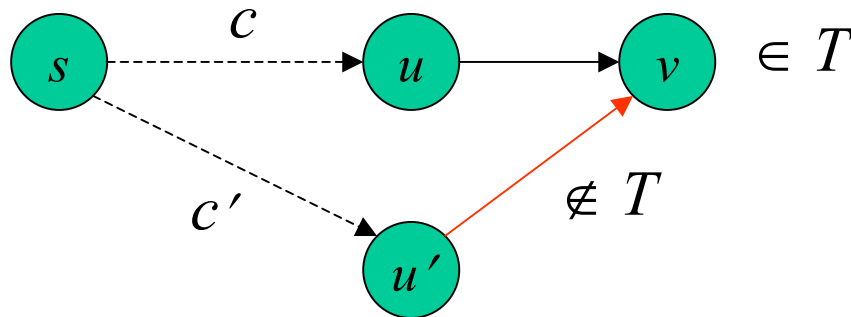
T_1 è ottimo per
 G da a



Archivi di una soluzione ottima

Supponiamo che T sia ottimo per G da s :

se $(u, v) \in T$ allora $d[v] = d[u] + p(u, v)$



se $(u', v) \notin T$ allora $d[v] \leq d[u'] + p(u', v)$

Il teorema di Bellman

Teorema. Se per ogni arco (u,v) , $d[v] \leq d[u] + p(u, v)$,
con $=$ quando $(u,v) \in T$, allora T è ottimo

Per assurdo sia ottimo T' ma non T : allora esiste w t.c.

$$d'[w] < d[w]$$

Nel cammino che in T' conduce a w , consideriamo il primo arco (u, v) per cui $d'[u] = d[u]$ ma $d'[v] < d[v]$:

$$\begin{aligned} d'[v] &= d'[u] + p(u, v) && \text{perché } T' \text{ è ottimo} \\ &= d[u] + p(u, v) && d'[u] = d[u] \\ &\geq d[v] && \text{per ipotesi} \end{aligned}$$



Schema di risoluzione

CamminiMinimi ($G = (N, A, p), s$)

foreach $v \in N$ **do** $d[v] \leftarrow \infty$

$d[s] \leftarrow 0, T \leftarrow \emptyset, S \leftarrow \{s\}$

while $S \neq \emptyset$ **do**

 sia $u \in S, S \leftarrow S - \{u\}$

foreach $(u, v) \in A$ **do**

if $d[v] > d[u] + p(u, v)$ **then**

 // c'è un cammino migliore da s a v

$d[v] \leftarrow d[u] + p(u, v)$

 rimpiazza in T l'arco incidente in v con (u, v)

$S \leftarrow S \cup \{u\}$

return T

Se nessun arco della forma (w, v) era in T , allora “rimpiazza” = aggiungi

rilassamento

Cammini minimi e BFS


Somiglianze

Da un nodo consideriamo tutti i suoi adiacenti, inoltre:

1. S è la *Frontiera*
2. $Esterni = \{v: d[v] = \infty\}$
3. $Interni = \{v: d[v] < \infty\} - S$

Differenze

Se v è raggiungibile da più di un cammino allora sarà in S più di una volta



Non si tiene conto
del fatto che un
vertice possa essere
stato visitato

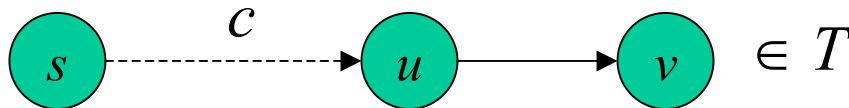
Algoritmo di Dijkstra

Supponiamo che i pesi siano positivi:

- S è una coda di priorità dove

$$\text{Priorità}(v, S) = d[v]$$

- Se $d[v] < \infty$ e $v \notin S$, allora $d[v] = \delta(s, v)$:



$$d[u] = p(c) = \delta(s, u) \quad \text{ipotesi induttiva}$$

$$p(u, v) = \min \{ p(u, w) : (u, w) \in A \}$$

$\delta(s, v)$ non dipende dai nodi ancora da scoprire

Qui usiamo
l'ipotesi che i pesi
siano positivi

Algoritmo di Dijkstra

Dijkstra-Johnson ($G = (N, A, p), s$) // Pre: p ha valori positivi

foreach $v \in N$ **do** $d[v] \leftarrow \infty$;

$d[s] \leftarrow 0$; $Q \leftarrow \text{EmptyPriorityQueue}()$

foreach $v \in N$ **do** $\text{Enqueue}(v, d[v], Q)$ // la priorità di ogni v in Q è $d[v]$

while not $\text{IsEmptyQueue}(Q)$ **do**

$u \leftarrow \text{ExtractMin}(Q)$;

foreach $(u, v) \in A$ **do**

if $d[v] > d[u] + p(u, v)$ **then**

$d[v] \leftarrow d[u] + p(u, v)$

 rimpiazza in T l'arco incidente in v con (u, v)

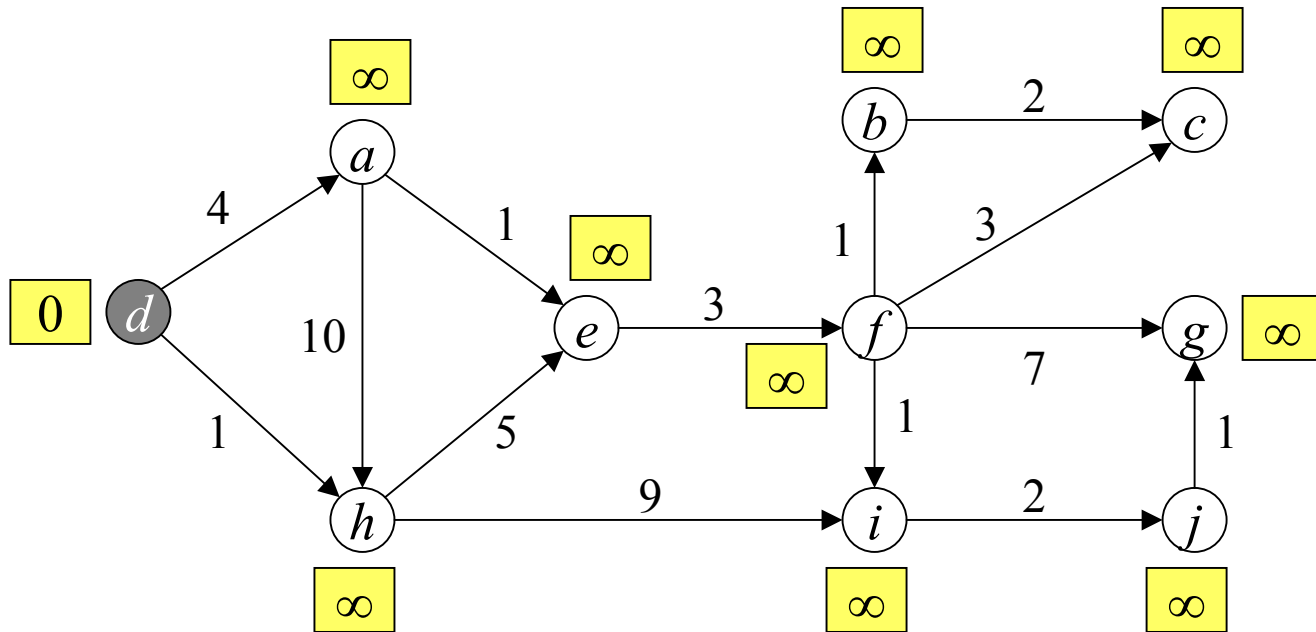
$\text{RedefinePrior}(v, d[v], Q)$

return T

Nessun
reinserimento

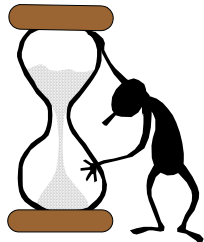
Essenziale se Q è
uno heap: la
posizione di v
cambia

Esecuzione dell'algoritmo di Dijkstra

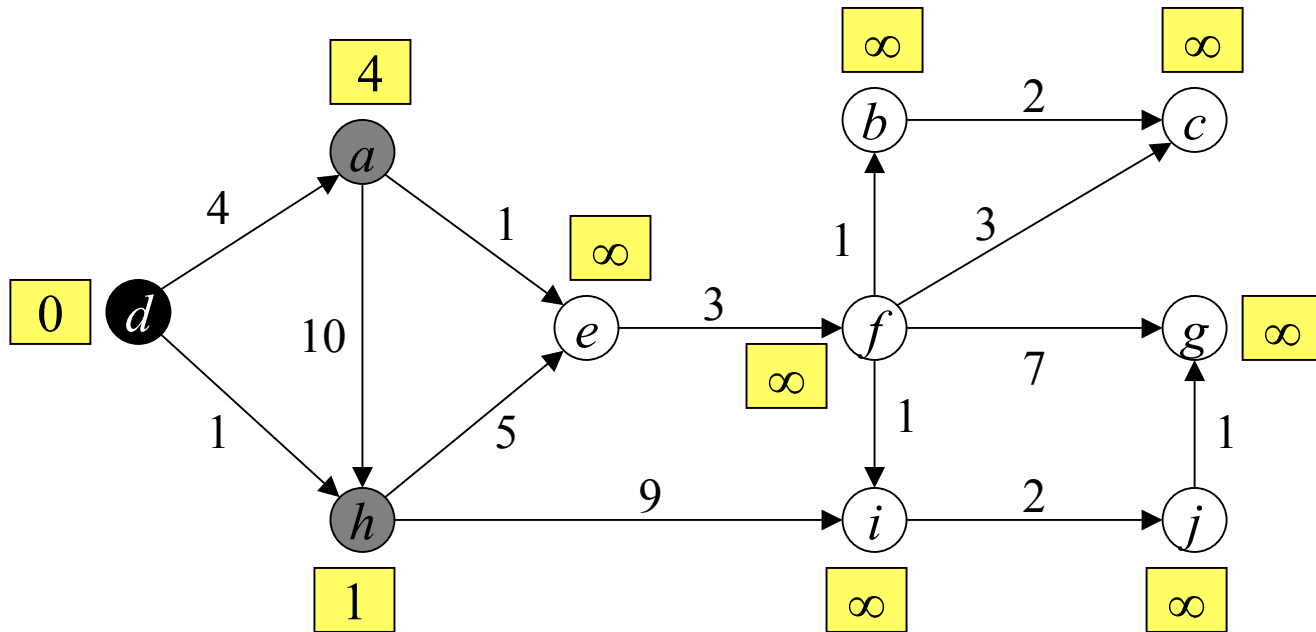


Iterazione N°

0

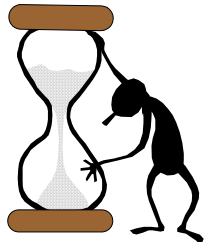


Esecuzione dell'algoritmo di Dijkstra

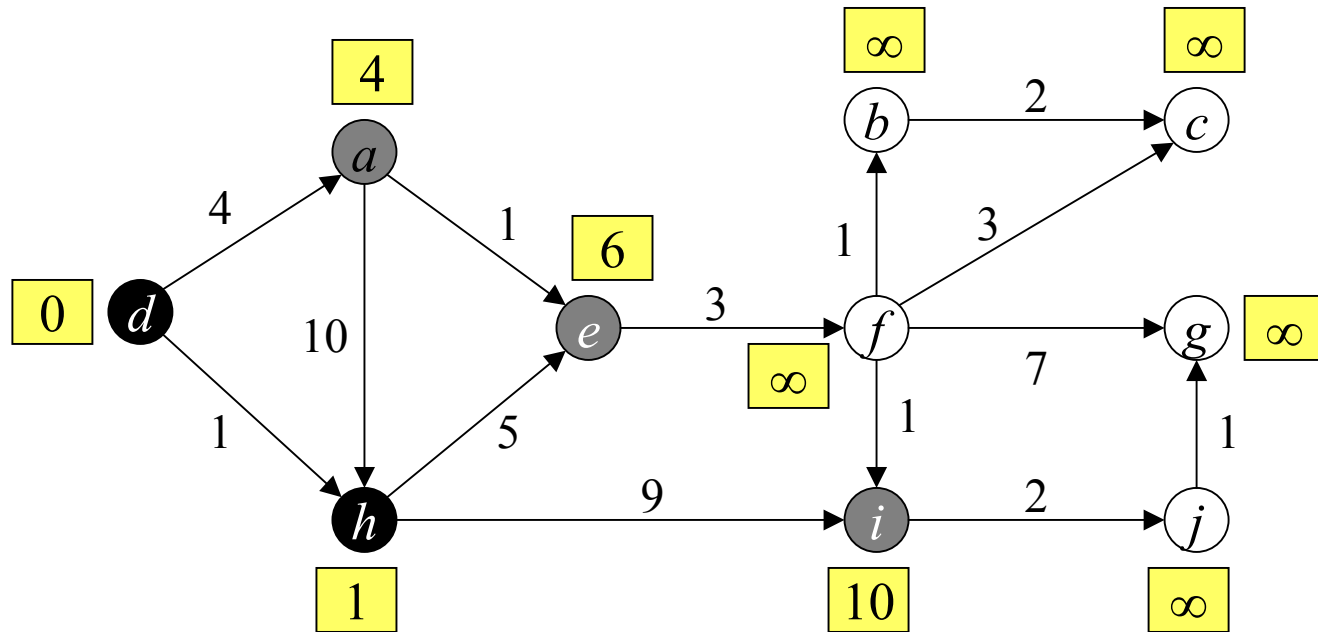


Iterazione N°

1

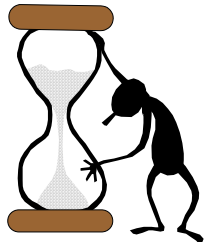


Esecuzione dell'algoritmo di Dijkstra

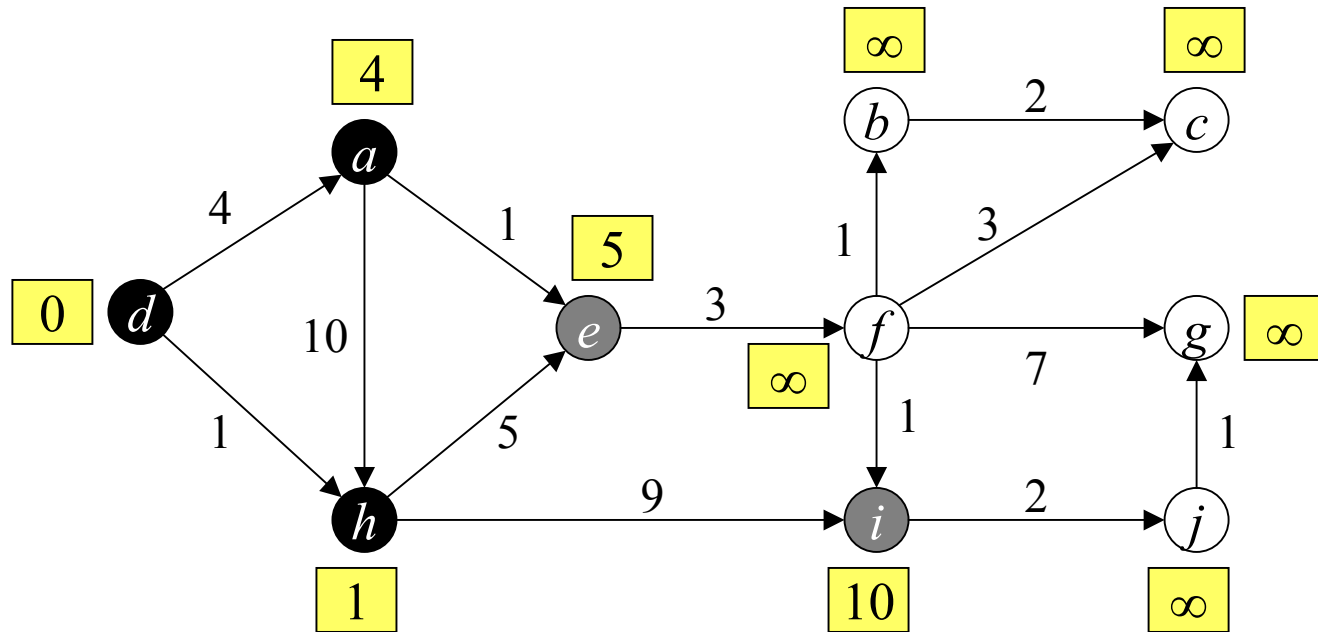


Iterazione N°

2

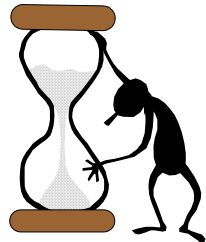


Esecuzione dell'algoritmo di Dijkstra

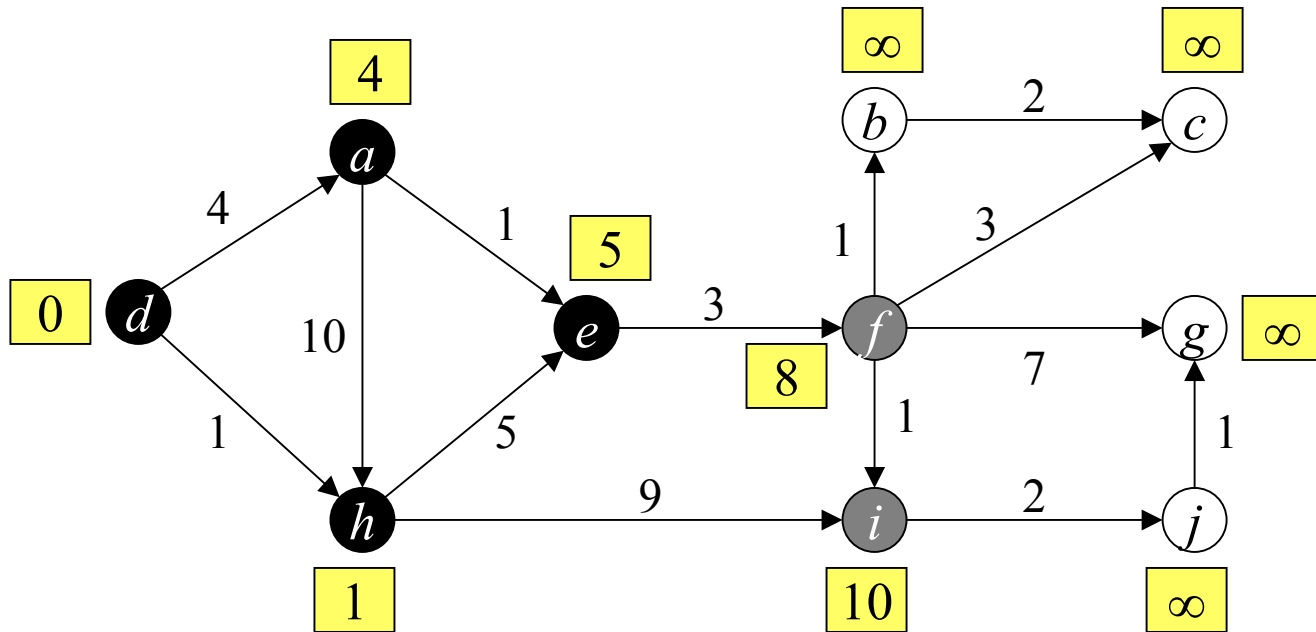


Iterazione N°

3

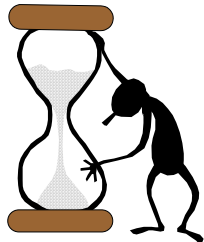


Esecuzione dell'algoritmo di Dijkstra

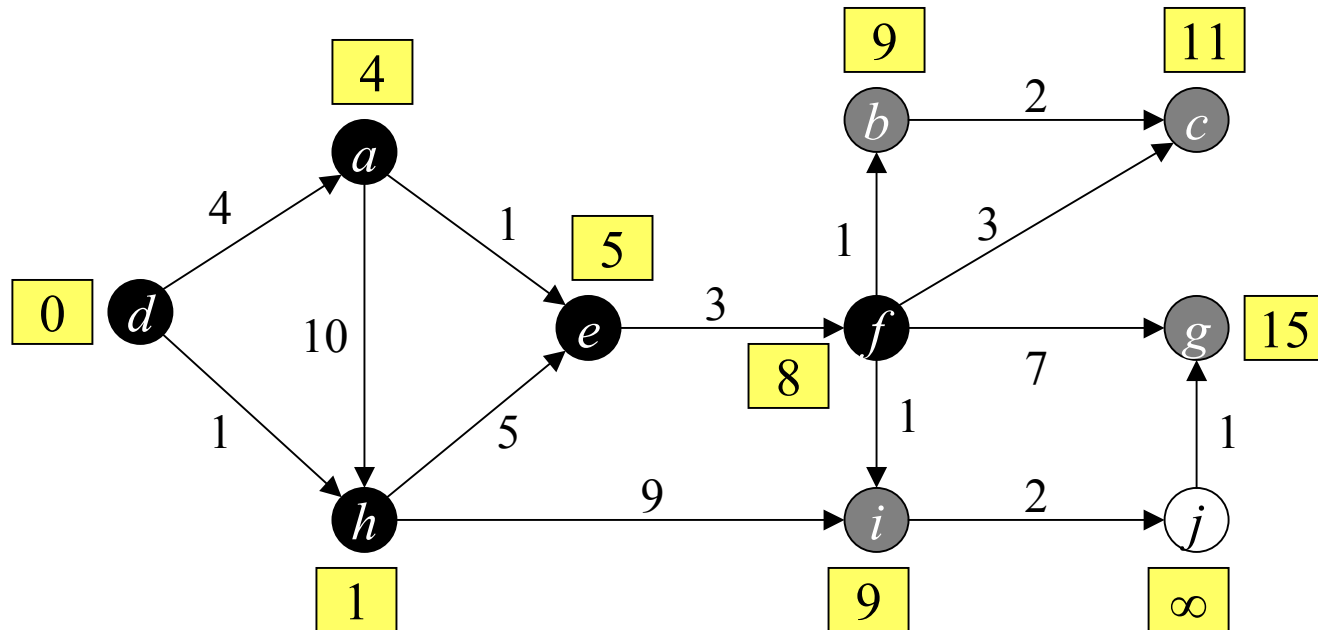


Iterazione N°

4

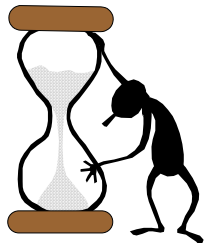


Esecuzione dell'algoritmo di Dijkstra

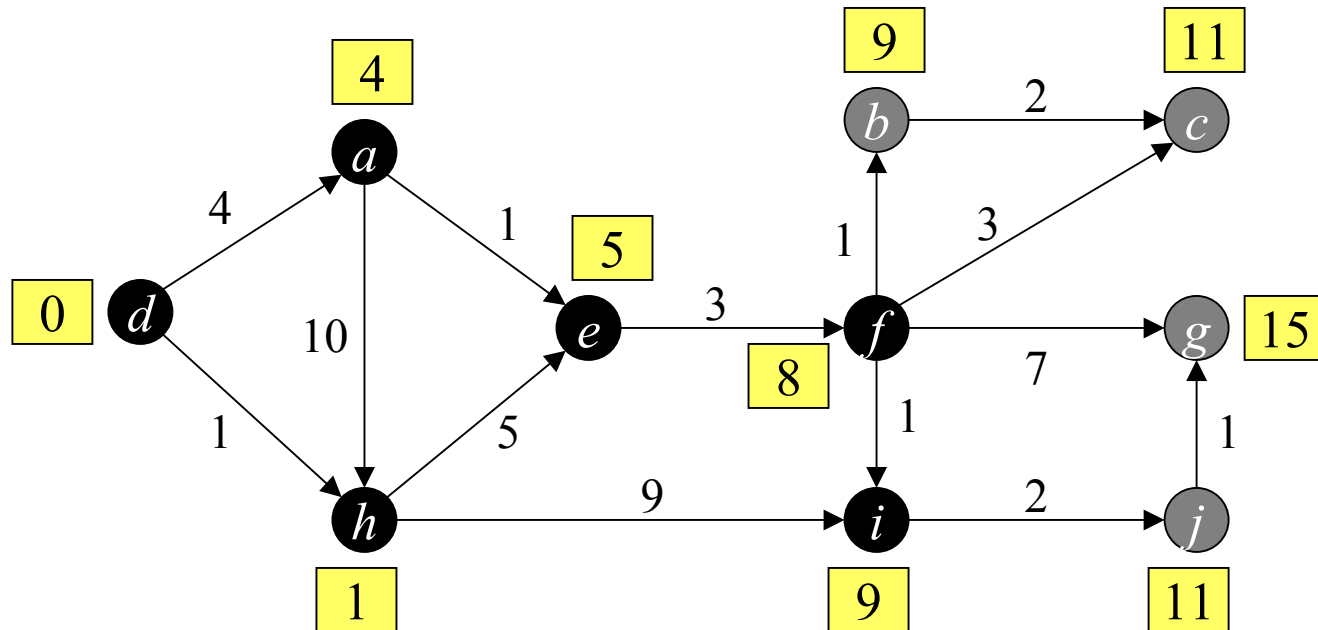


Iterazione N°

5

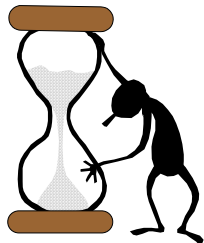


Esecuzione dell'algoritmo di Dijkstra

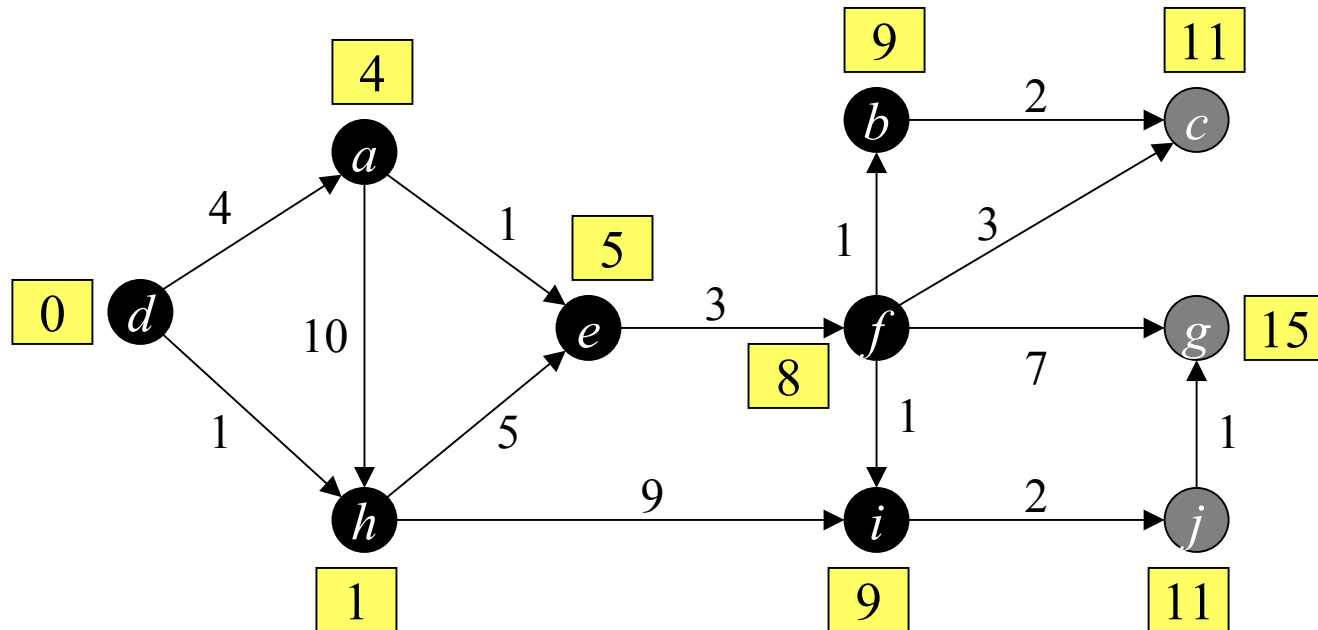


Iterazione N°

6

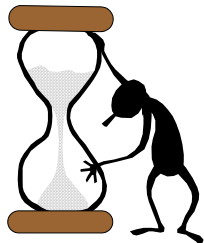


Esecuzione dell'algoritmo di Dijkstra

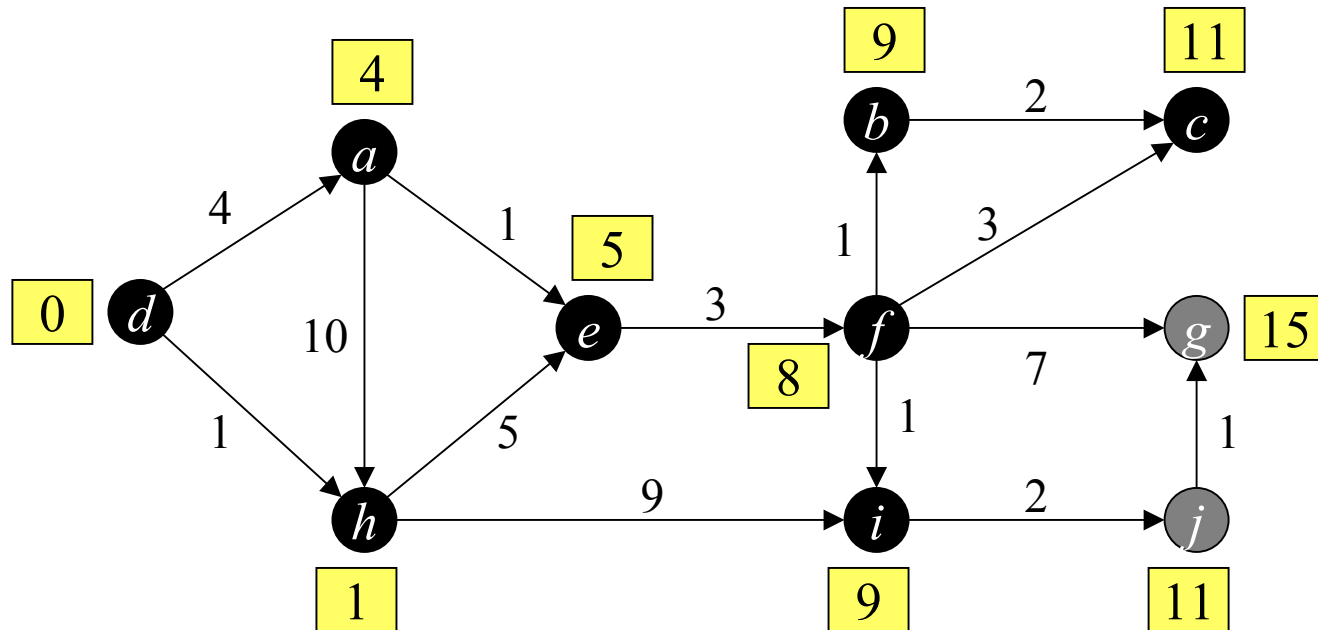


Iterazione N°

7

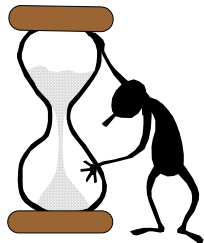


Esecuzione dell'algoritmo di Dijkstra

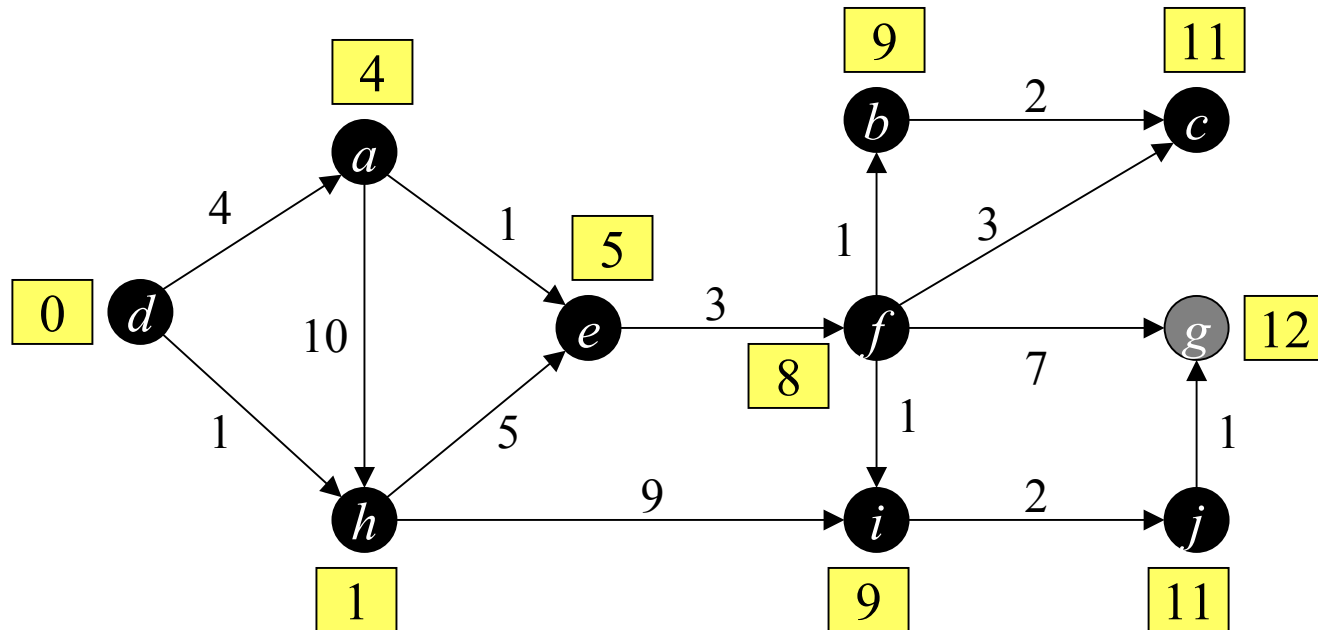


Iterazione N°

8

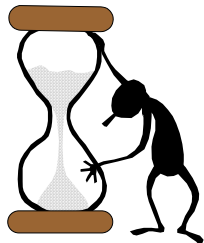


Esecuzione dell'algoritmo di Dijkstra

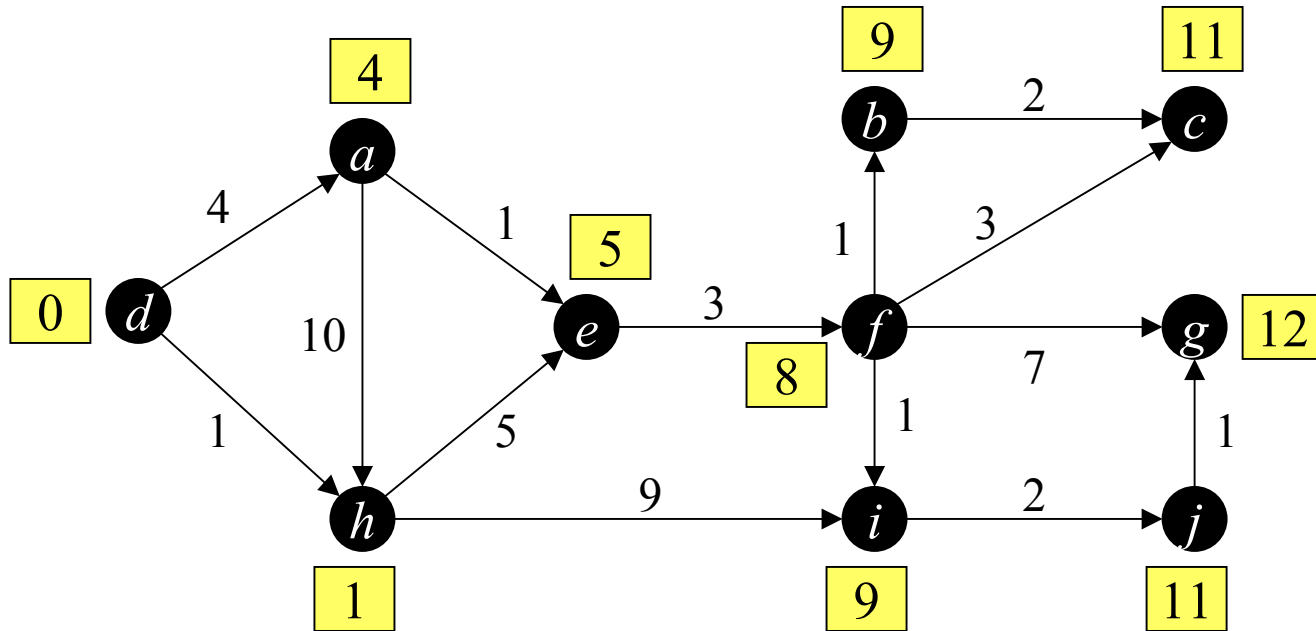


Iterazione N°

9

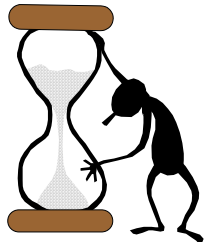


Esecuzione dell'algoritmo di Dijkstra

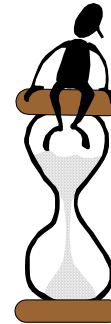
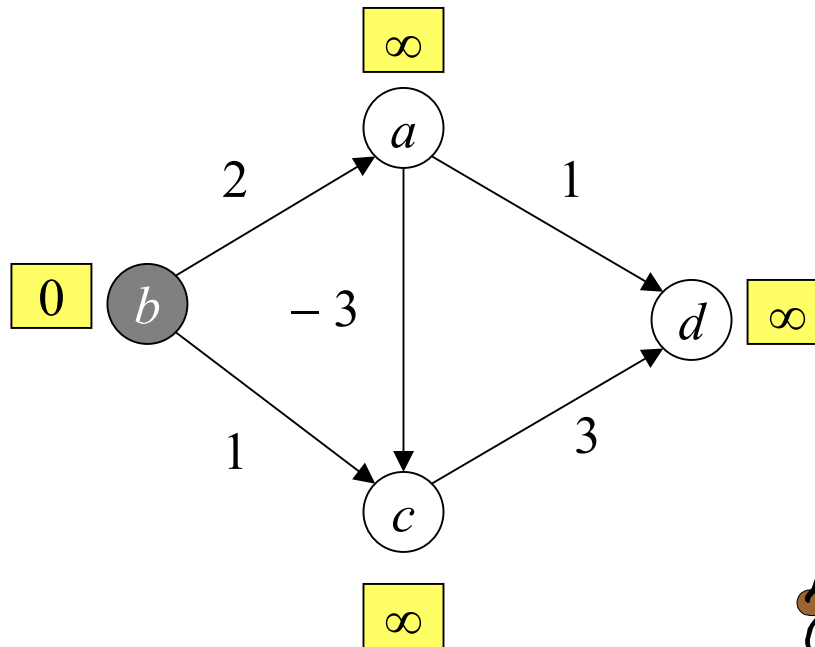


Iterazione N°

10



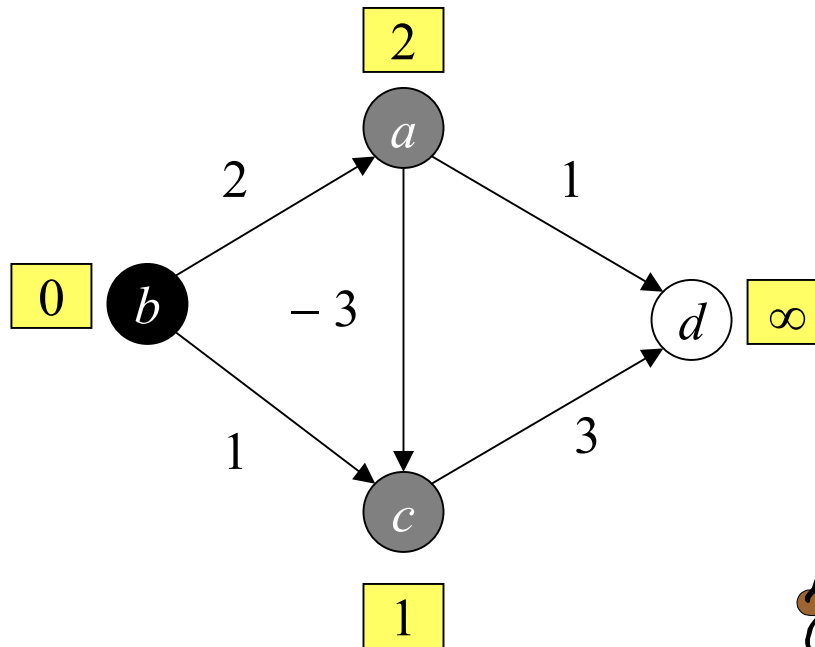
E se ci sono pesi negativi?



Iterazione N°

0

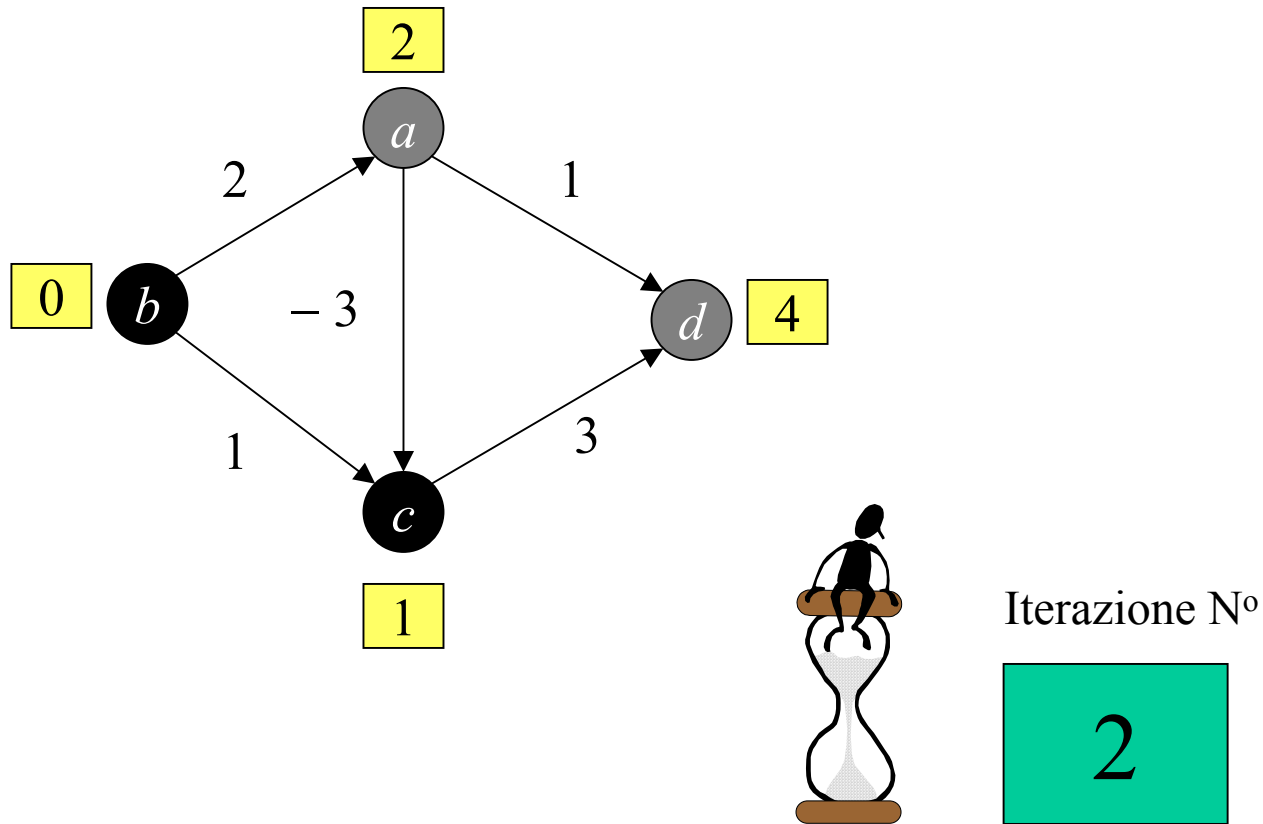
E se ci sono pesi negativi?



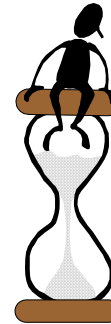
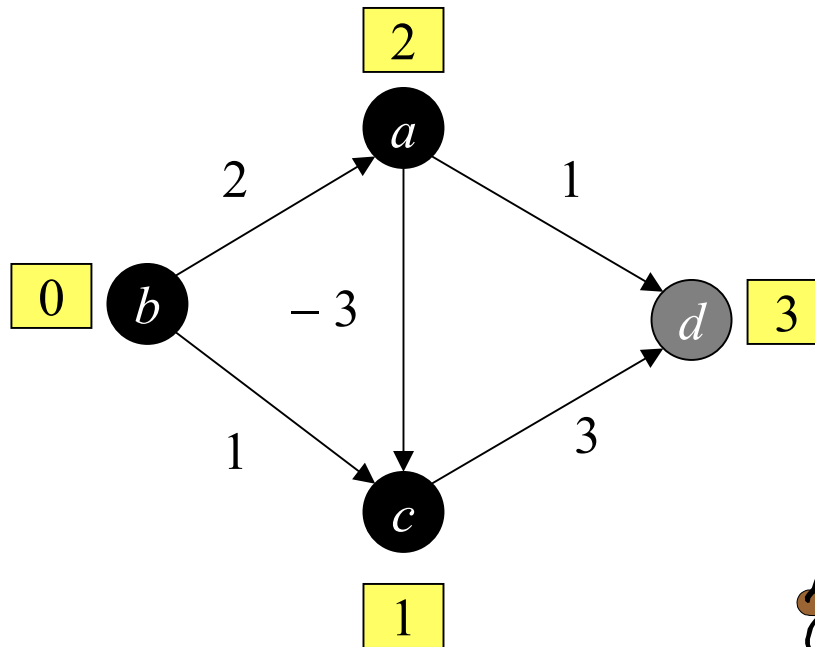
Iterazione N°

1

E se ci sono pesi negativi?



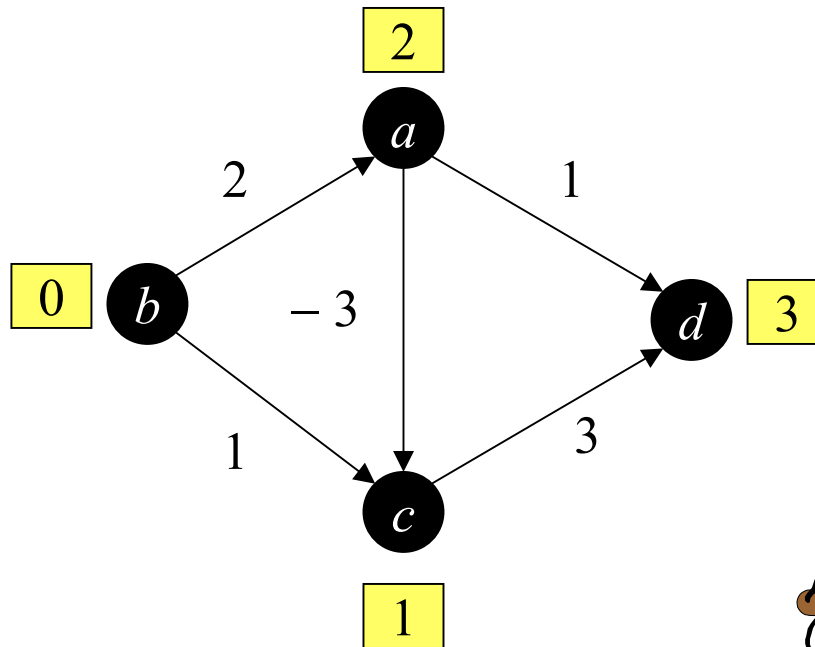
E se ci sono pesi negativi?



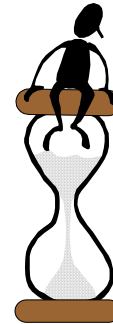
Iterazione N°

3

E se ci sono pesi negativi?



$$\delta(b, d) = p(b, a, c, d) = 2 - 3 + 3 = 2$$



Iterazione N°

4

Un problema con pesi negativi

“Un proprietario di TIR è libero di spostare containers da una città all'altra. Un viaggio dalla città i alla città j porta un profitto di p_{ij} euro se c'è un container da trasportare, ma provoca una perdita di p_{ij} euro se il TIR viaggia scarico. Assumendo profitti $p_{ij} > 0$ e perdite $p_{ij} < 0$, il percorso più vantaggioso tra due città corrisponde ad un cammino minimo, in cui le lunghezze (i pesi) degli archi sono $-p_{ij}$ ”.

Bertossi, esempio 10.1



Per aggirare l'ostacolo
proviamo a “rilassare” tutti
gli archi a ripetizione

Algoritmo di Bellman-Ford

BF ($G = (N, A, p), s$)

foreach $v \in N$ **do** $d[v] \leftarrow \infty$

$d[s] \leftarrow 0, T \leftarrow \emptyset, relax \leftarrow true$

while $relax$ **do**

$relax \leftarrow false$

foreach $(u, v) \in A$ **do**

if $d[v] > d[u] + p(u, v)$ **then**

$relax \leftarrow true$

$d[v] \leftarrow d[u] + p(u, v)$

rimpiazza in T l'arco incidente in v con (u, v)

return T

Questo algoritmo
termina? Se si, entro
quante iterazioni?

Algoritmo di Bellman-Ford

Teorema. Se G non ha cicli a somma negativa ed esiste una copertura con radice in s , allora BF termina entro $n - 1$ iterazioni, dove $n = |N|$.

Sia $c = v_0, \dots, v_k$ un cammino ottimo in G , con $s = v_0$: per induzione su k dimostriamo che:

i. $d[v_i] \geq \delta(s, v_i)$, per $i = 0, \dots, k$ è un invariante di ciclo

ii. $d[v_k] = \delta(s, v_k)$ dopo al più k iterazioni

E in un cammino
ottimo ci sono $\leq n - 1$
archi

Base: $k = 0$, allora $\delta(s, v_0) = \delta(s, s) = 0 = d[v_0]$ per
inizializzazione

Algoritmo di Bellman-Ford

Teorema. Se G non ha cicli a somma negativa ed esiste una copertura con radice in s , allora BF termina entro $n - 1$ iterazioni, dove $n = |N|$.

Passo: $k > 0$ e $c = v_0, \dots, v_{k-1}, v_k$:

$$\begin{aligned}\delta(s, v_k) &= p(c) = p(v_0, \dots, v_{k-1}) + p(v_{k-1}, v_k) \\ &= \delta(v_0, v_{k-1}) + p(v_{k-1}, v_k) && \text{sott. ottimo} \\ &= d[v_{k-1}] + p(v_{k-1}, v_k) && \text{ip. ind.}\end{aligned}$$

Per ip. ind. $d[v_k] \geq \delta(s, v_k)$, e se è $>$ allora per rilassamento

$d[v_k] = d[v_{k-1}] + p(v_{k-1}, v_k) = \delta(s, v_k)$ dopo la k -esima iterazione. 😊

Algoritmo di Bellman-Ford

Bellman-Ford ($G = (N, A, p), s$)

foreach $v \in N$ **do** $d[v] \leftarrow \infty$

$d[s] \leftarrow 0, T \leftarrow \emptyset$

for $i \leftarrow 1$ **to** $|N| - 1$ **do**

foreach $(u, v) \in A$ **do**

if $d[v] > d[u] + p(u, v)$ **then**

$d[v] \leftarrow d[u] + p(u, v)$

 rimpiazza in T l'arco incidente in v con (u, v)

return T

Analisi della complessità

Dijkstra-Johnson: **ExtractMin** si esegue $n = |N|$ volte, mentre **RedefinePrior** si esegue al più $m = |A|$ volte.

- Se Q è un vettore, **ExtractMin** è $O(n)$, ma **RedefinePrior** è $O(1)$, dunque

$$O(n^2 + m) = O(n^2)$$

- Se Q è uno heap, sia **ExtractMin** che **RedefinePrior** sono $O(\log n)$ mentre l'inizializzazione è $O(n)$, dunque, se una copertura esiste,

$$O(n + n \log n + m \log n) = O(m \log n)$$

Lo heap conviene se m non è $\Omega(n^2)$ (grafo sparso)

Analisi della complessità

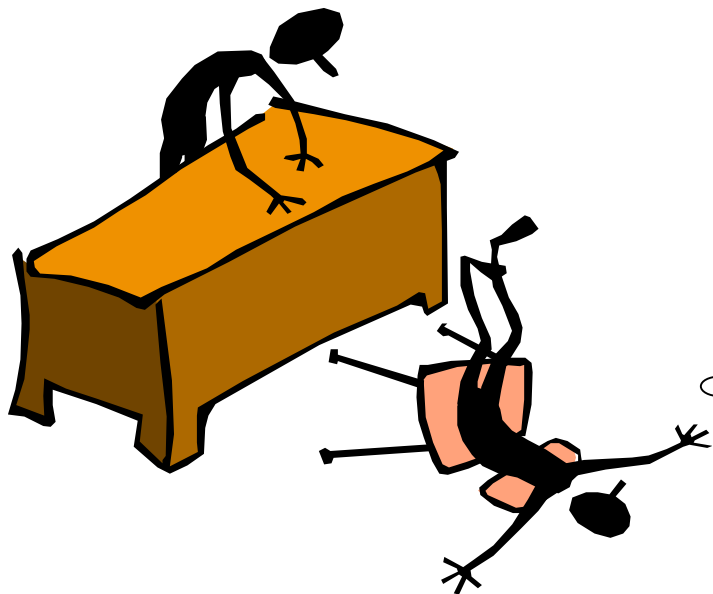
Bellman-Ford: in ognuna delle $n - 1$ iterazioni si esplorano al più m archi:

$$O(nm)$$

Poiché m è $O(n^2)$, abbiamo

$$O(n^3).$$

Il corso di
“Algoritmi” è finito:
non sei contento?



Sicuro se
sopravvivo anche
questa volta ...