Algoritmi greedy
III parte

Progettazione di Algoritmi a.a. 2018-19
Matricole congrue a 1
Docente: Annalisa De Bonis

Cammini minimi

- Si vuole andare da Napoli a Milano in auto percorrendo il minor numero di chilometri
- Si dispone di una mappa stradale su cui sono evidenziate le intersezioni tra le strade ed e` indicata la distanza tra ciascuna coppia di intersezioni adiacenti
- Come si puo` individuare il percorso piu` breve da Napoli a Milano?

Cammini minimi

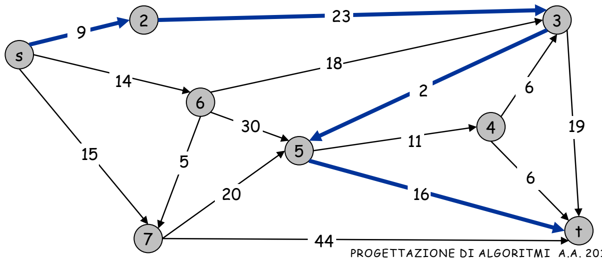
- Esempi di applicazioni dei cammini minimi in una rete
- Trovare il cammino di **tempo minimo** in una rete
- Se i pesi esprimono l'inaffidabilità delle connessioni in una rete, trovare il collegamento che è **più sicuro**

Il problema dei cammini minimi

- Input:
  - Grafo direzionato $G = (V, E)$.
  - Per ogni arco e, ℓ_e = lunghezza del tratto rappresentato da e.
  - s = sorgente
- Def. Per ogni percorso direzionato P, $\ell(P)$ = somma delle lunghezze degli archi in P.

Il problema dei cammini minimi: trova i percorsi direzionati più corti da s verso tutti gli altri nodi.

NB: Se il grafo non è direzionato possiamo sostituire ogni arco (u,v) con i due archi direzionati (u,v) e (v,u)



Costo del percorso da s a t
s-2-3-5-t = 9 + 23 + 2 + 16
= 48.

Varianti del problema dei cammini minimi

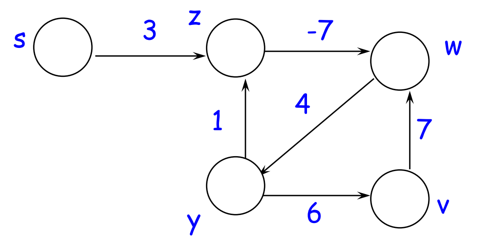
- Single Source Shortest Paths: determinare il cammino minimo da un dato vertice sorgente s ad ogni altro vertice
- Single Destination Shortest Paths: determinare i cammini minimi ad un dato vertice destinazione t da tutti gli altri vertici
  - Si riduce a Single Source Shortest Path invertendo le direzioni degli archi
- Single-Pair Shortest Path: per una data coppia di vertici u e v determinare un cammino minimo da un dato vertice u a v
  - i migliori algoritmi noti per questo problema hanno lo stesso tempo di esecuzione asintotico dei migliori algoritmi per Single Source Shortest Path.
- All Pairs Shortest Paths: per ogni coppia di vertici u e v, determinare un cammino minimo da u a v

Cammini minimi

- Soluzione inefficiente:
  - si considerano tutti i percorsi possibili e se ne calcola la lunghezza
  - l'algoritmo non termina in presenza di cicli
- Si noti che l'algoritmo di visita BFS è un algoritmo per Single Source Shortest Paths nel caso in cui tutti gli archi hanno lo stesso peso

Cicli negativi

- Se esiste un ciclo negativo lungo un percorso da s a v, allora non è possibile definire il cammino minimo da s a v



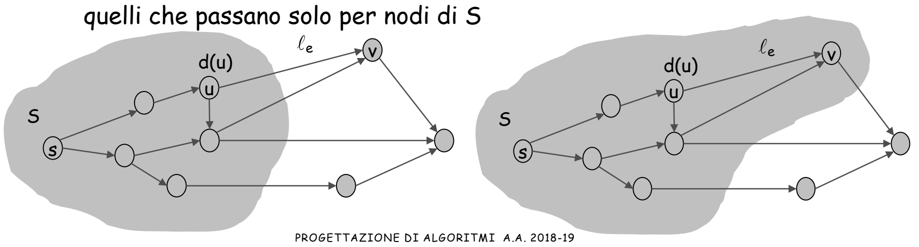
Il ciclo <z,w,y,z> ha peso -2

- Attraversando il ciclo <z,w,y,z> un numero arbitrario di volte possiamo trovare percorsi da s a v di peso arbitariamente piccolo

Algoritmo di Dijkstra

Algoritmo di Dijkstra (1959).

- Ad ogni passo mantiene l'insieme S dei nodi esplorati, cioé di quei nodi u per cui è già stata calcolata la distanza minima d(u) da s.
- Inizializzazione S = { s }, d(s) = 0.
- Ad ogni passo, sceglie tra i nodi non ancora in S ma adiacenti a qualche nodo di S, quello che può essere raggiunto nel modo più economico possibile (scelta greedy)
- In altra parole sceglie v che minimizza $d'(v) = \min_{e=(u,v):\, u \in S} d(u) + \ell_e$, aggiunge v a S e pone d(v) = d'(v).
- d'(v) rappresenta la lunghezza del percorso piu` corto da s a v tra quelli che passano solo per nodi di S



Algoritmo di Dijkstra

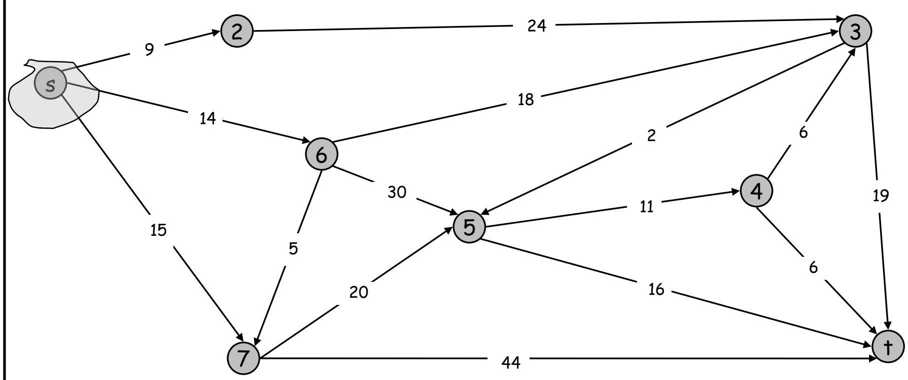
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each $u \in S$, we store a distance $d(u)$
Initially $S = \{s\}$ and $d(s) = 0$
While $S \neq V$
    Select a node $v \notin S$ with at least one edge from S for which
        $d'(v) = \min_{e=(u,v):u \in S} d(u) + \ell_e$ is as small as possible
    Add v to S and define $d(v) = d'(v)$
EndWhile

Per come e` scritta, questa versione dell'algoritmo di Dijkstra assume che tutti i nodi siano raggiungibili da s.

Esercizio: Ad ogni passo l'algoritmo di Dijkstra aggiunge un certo nodo v ad S. Dimostrare per induzione che il percorso piu` corto da s a v computato dall'algoritmo passa solo attraverso nodi gia` inseriti in S.

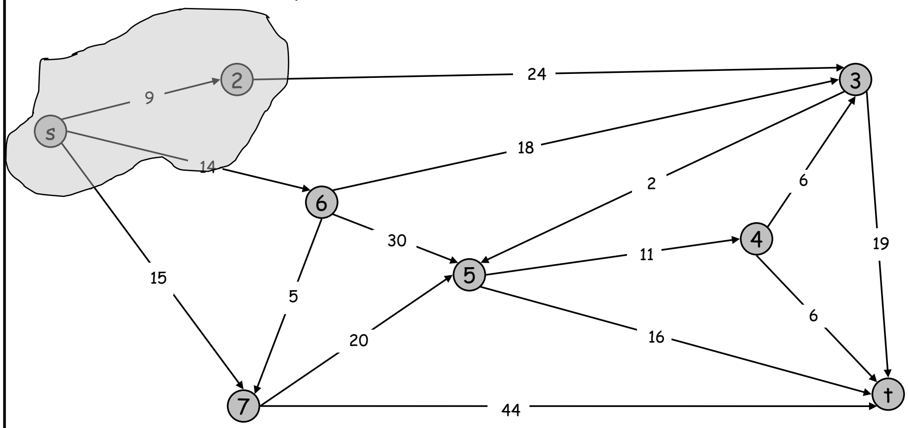
Algoritmo di Dijkstra

S={s} computo d'(2)=9 , d'(6)=14, d'(7)=15 → scelgo 2 e pongo d(2)=9, shortest path = s-2



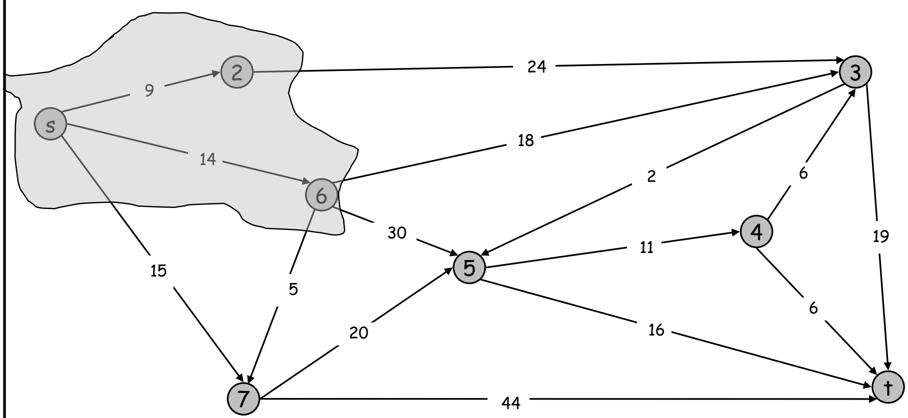
Algoritmo di Dijkstra

S={s,2} computo d'(6)=14, d'(7)=15, d'(3)=9+24=33 → scelgo 6 e pongo d(6)=14 , shortest path = s-6



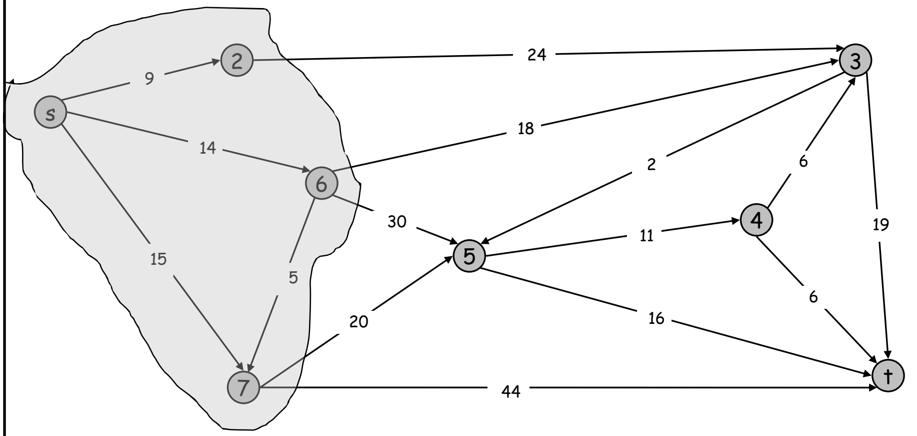
Algoritmo di Dijkstra

S={s,2,6} computo d'(7)=min{15,14+5}=15, d'(5)=14+30=44, d'(3)=min{9+24, 14+18} =32 → scelgo 7 e pongo d(7)=15, shortest path = s-7



Algoritmo di Dijkstra

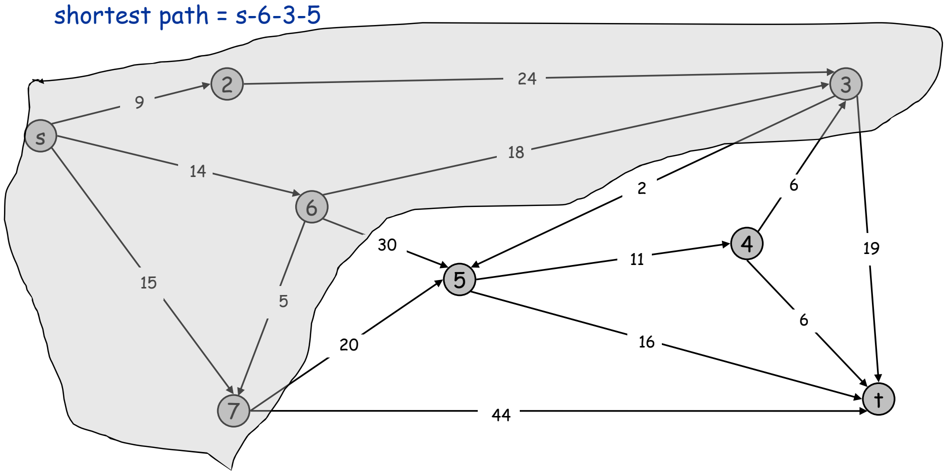
S={s,2,6,7} computo d'(5)=min{14+30,15+20}=35, d'(3)=min{9+24, 14+18} =32, d'(t)=15+44=59 → scelgo 3 e pongo d(3)=32 , shortest path = s-6-3



73

Algoritmo di Dijkstra

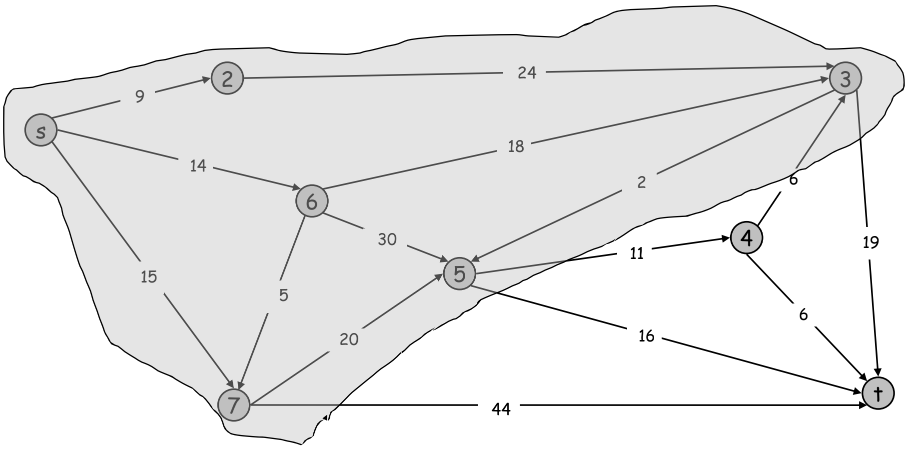
S={s,2,6,7,3} computo d'(5)=min{14+30,15+20,32+2}=34, d'(t)=min{15+44, 32+19}=51 → scelgo 5 e pongo d(5)=34 , shortest path = s-6-3-5



74

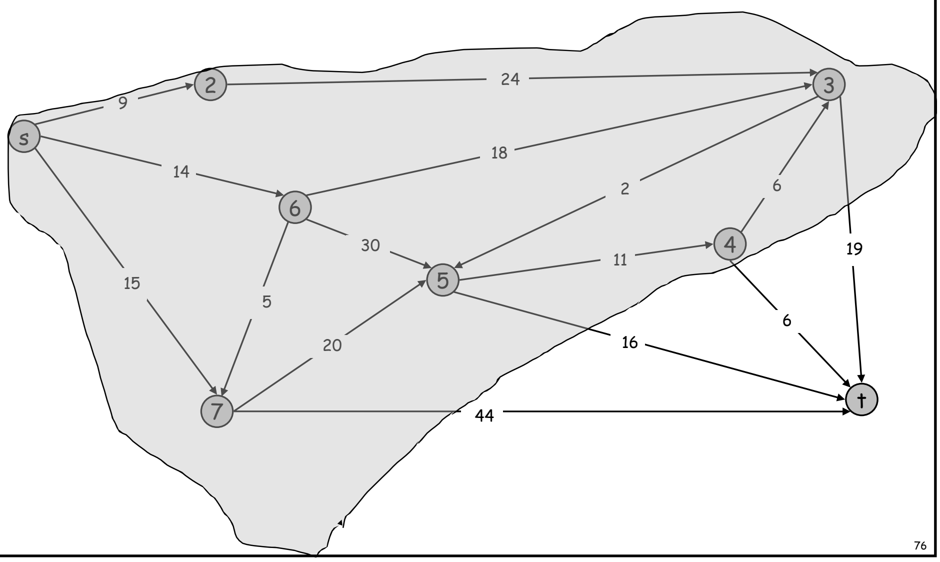
Algoritmo di Dijkstra

S={s,2,6,7,3,5} computo d'(4)=34+11=45, d'(t)=min{15+44, 32+19, 34+16}=50
→ scelgo 4 e pongo d(4)=45 , shortest path = s-6-3-5-4



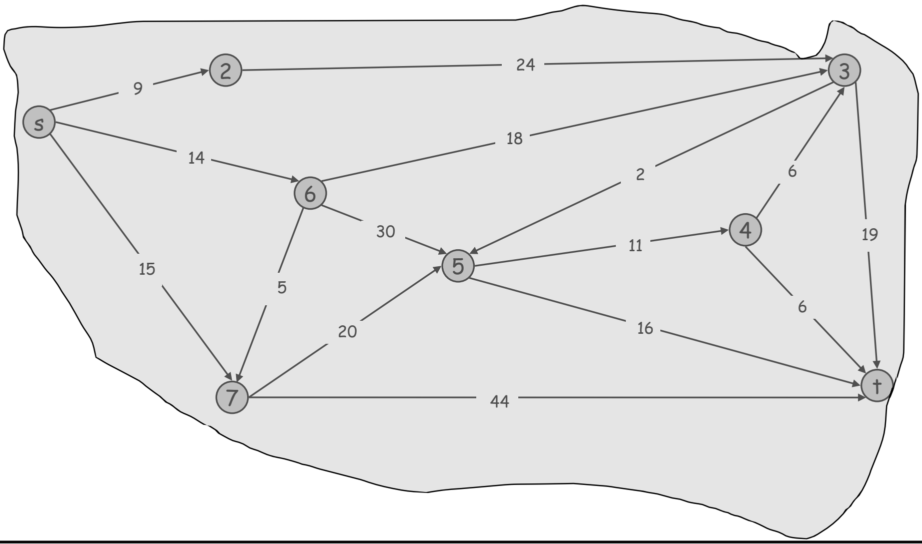
Algoritmo di Dijkstra

S={s,2,6,7,3,5,4} computo d'(t)=min{15+44, 32+19, 34+16,45+6}=50
→ scelgo t e pongo d(t)=50 , shortest path = s-6-3-5-t



Algoritmo di Dijkstra

S={s,2,6,7,3,5,4,t}



Algortimo di Dijksta: Correttezza

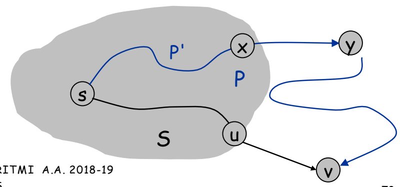
Teorema. Sia G un grafo in cui per ogni arco e è definita una lunghezza $\ell_e \geq 0$ (è fondamentale che ℓ_e **non sia negativa**). Per ogni nodo u ∈ S il valore d(u) calcolato dall'algoritmo di Dijkstra è la lunghezza del percorso più corto da s a u.

Dim. (per induzione sulla cardinalità |S| di S)

Base: |S| = 1. In questo caso S={s} e d(s)=0 per cui la tesi vale banalmente.

Ipotesi induttiva: Assumiamo vera la tesi per |S| = k ≥ 1.

- Sia v il prossimo nodo inserito in S dall'algoritmo e sia (u,v) l'arco attraverso il quale è stato raggiunto v, cioè quello per cui si ottiene
$$d'(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e ,$$
- Consideriamo il percorso di lunghezza d'(v), cioé quello formato dal percorso più corto da s ad u più l'arco (u,v)
- Consideriamo un **qualsiasi** altro percorso P da s a v. Dimostriamo che P non è più corto di d'(v)
- Sia (x,y) il primo arco di P che esce da S.
- Sia P' il sottocammino di P fino a x.
- P' più l'arco (x,y) ha gia` lunghezza non piu` piccola di d'(v) altrimenti l'algoritmo non avrebbe scelto v.



Algoritmo di Dijkstra: analisi tempo di esecuzione

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
      d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```

•While iterato n volte

•Se non usiamo nessuna struttura dati per trovare in modo efficiente il minimo d'(v) il calcolo del minimo richiede di scandire tutti gli archi che congiungono un vertice in S con un vertice non in S → O(m) ad ogni iterazione del while → O(nm) in totale.

Algoritmo di Dijkstra: come renderlo piu` efficiente?

```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
      d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to S and define d(v) = d'(v)
EndWhile
```

Miglioramenti: ad ogni iterazione del while, per ogni z ∉ S

1. ricomputiamo il valore d'(z) solo se e` stato appena aggiunto ad S un nodo u per cui esiste l'arco (u,z)

2. teniamo traccia dell'ultimo valore computato d'(z) in modo tale che per calcolare il nuovo valore di d'(z) occorre solo calcolare min{d'(z), d(u)+ℓ_e }, dove e=(u,z) e` l'arco che congiunge z al nodo u appena introdotto in S.

3. memorizziamo z e d'(z) in una struttura che ci consenta di individuare in modo efficiente il nodo v per cui d'(v)=$\min_{z \notin S}${d'(z)} (coda a priorita`)

Algoritmo di Dijkstra con coda a priorità: analisi del tempo di esecuzione

```
Dijkstra's Algorithm (G,s,ℓ)
Let S be the set of explored nodes
For each u not in S, we store a distance d'(u)
Let Q be a priority queue of pairs (d'(u),u) s.t. u is not in S
For each u ∈ S, we store a distance d(u)
Insert(Q,(0,s))
For each u≠s insert (Q, ∞,u)  in Q EndFor
While  S≠ V
 (d(u),u)<-- ExtractMin(Q)
  Add u to S
  For each edge e=(u,v)
   If  v not in S && d(u)+ ℓe< d'(v)
     ChangeKey(Q,v, d(u)+ ℓe)

EndWhile
```

In una singola iterazione del while, il for è iterato un numero di volte pari al numero di archi uscenti da u.
Se consideriamo tutte le iterazioni del while, il for viene iterato in totale m volte

•Se usiamo una min priority queue che per ogni vertice v **non** in S contiene la coppia (d'(u),u) allora con un'operazione di ExtractMin allora possiamo ottenere il vertice v con il valore d'(v) più piccolo possibile

•Tempo inizializzazione O(n) più tempo per effettuare gli n inserimenti in Q

•Tempo While: O(n) più il tempo per fare le n ExtractMin e le al più m changeKey

81

Algoritmo di Dijkstra con coda a priorità: analisi del tempo di esecuzione

•Se usiamo una min priority queue che per ogni vertice v **non** in S contiene la coppia (d'[u],v) allora con un'operazione di ExtractMin possiamo ottenere il vertice v con il valore d'[v] più piccolo possibile

•Tempo inizializzazione O(n) più tempo per effettuare gli n inserimenti in Q

•Tempo While: O(n) più il tempo per fare le n ExtractMin e le m changeKey

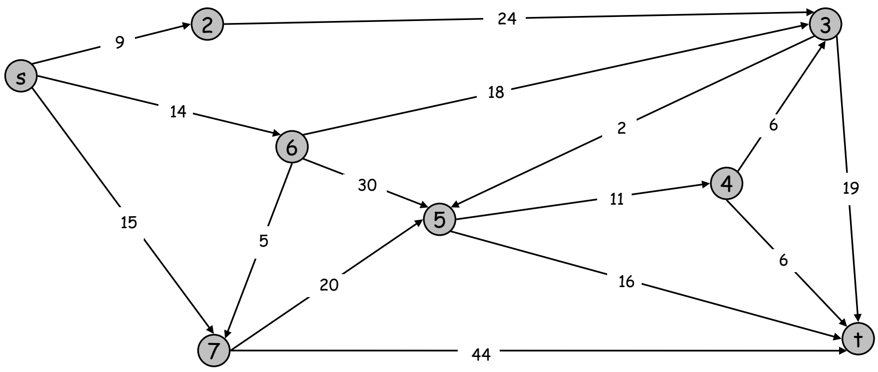
Se la coda è implementata mediante una lista o con un array non ordinato:
Inizializzazione: O(n)
While: $O(n^2)$ per le n ExtractMin; O(m) per le m ChangeKey
Tempo algoritmo: $O(n^2)$

Se la coda è implementata mediante un heap:
Inizializzazione: O(n) con costruzione bottom up oppure O(n logn) con n inserimenti
While: O(nlog n) per le n ExtractMin; O(m log n) per le m ChangeKey
Tempo algoritmo: O(n log n+ m log n)

82

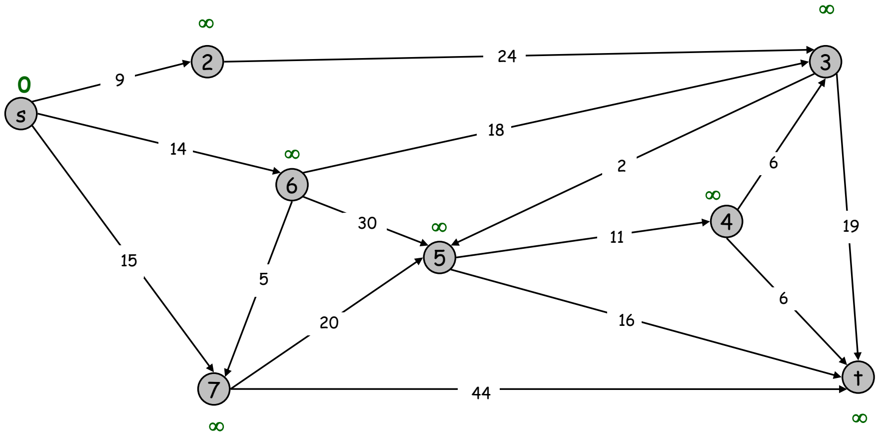
Algoritmo di Dijkstra

Trova i percorsi più corti



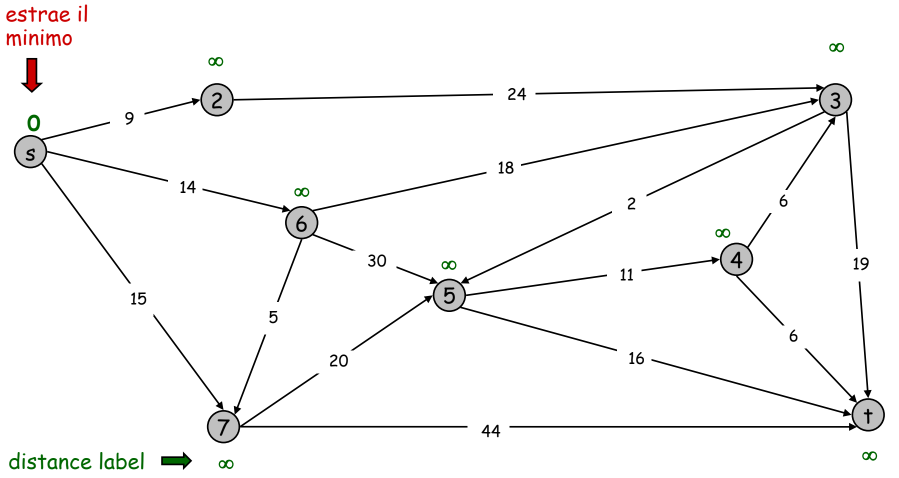
Algoritmo di Dijkstra

S = { }
Q = { s, 2, 3, 4, 5, 6, 7, t }



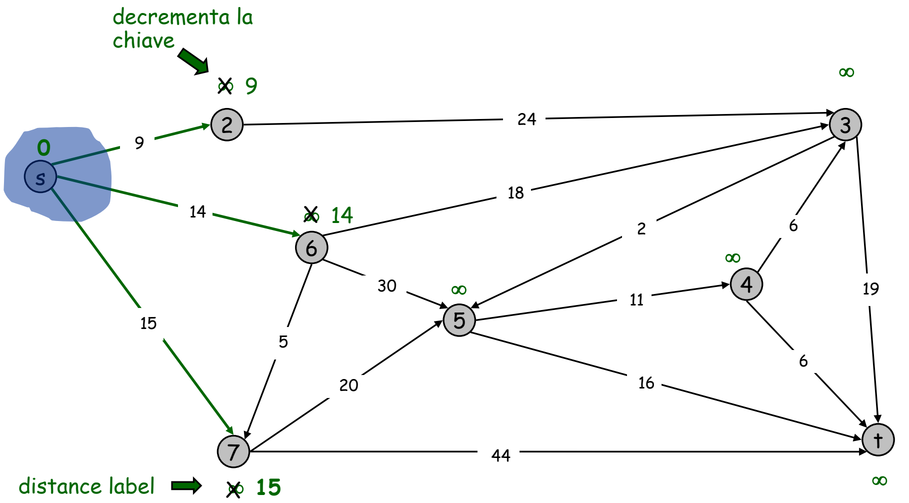
Algoritmo di Dijkstra

S = { }
Q = { s, 2, 3, 4, 5, 6, 7, t }



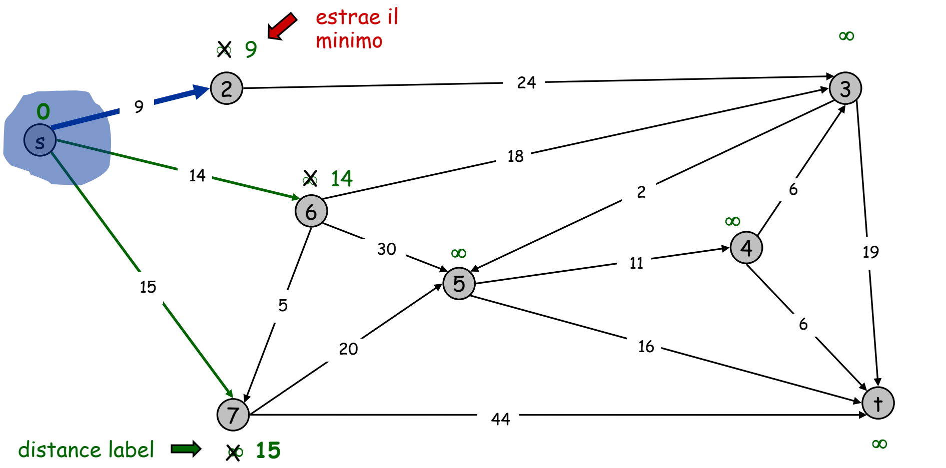
Algoritmo di Dijkstra

S = { s }
Q = { 2, 3, 4, 5, 6, 7, t }



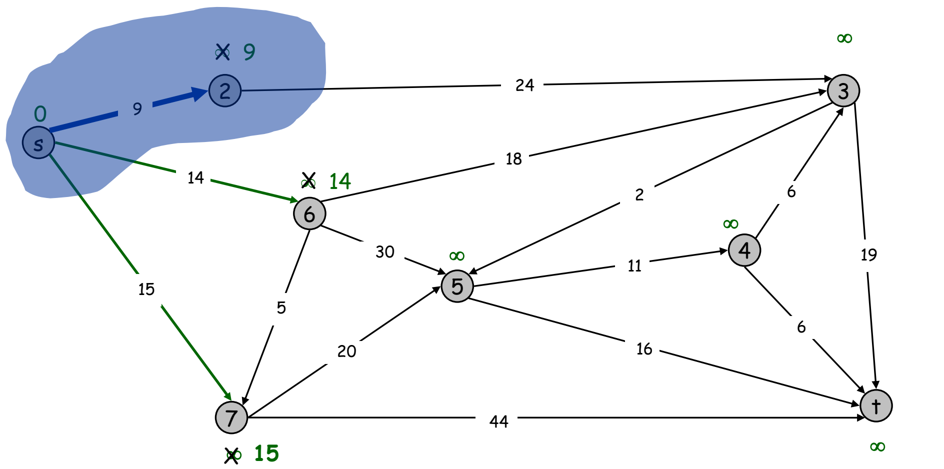
Algoritmo di Dijkstra

S = { s }
Q = { 2, 3, 4, 5, 6, 7, t }



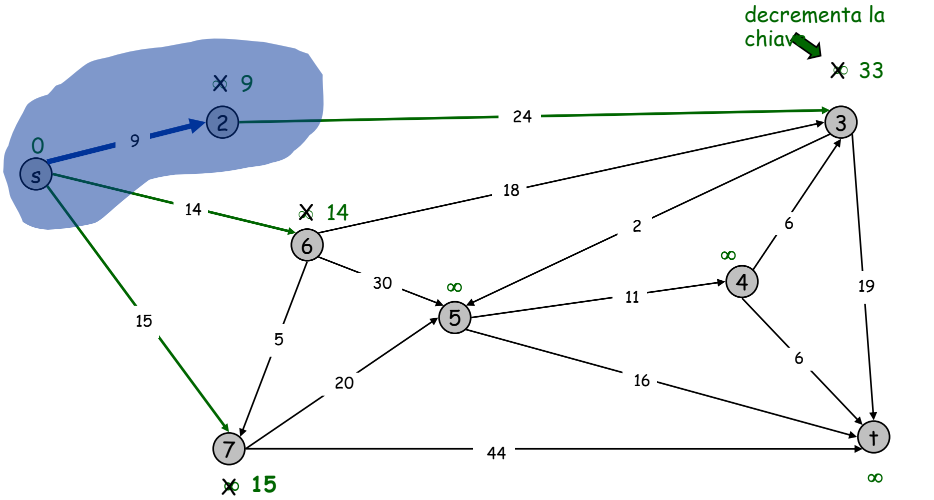
Algoritmo di Dijkstra

S = { s, 2 }
Q = { 3, 4, 5, 6, 7, t }



Algoritmo di Dijkstra

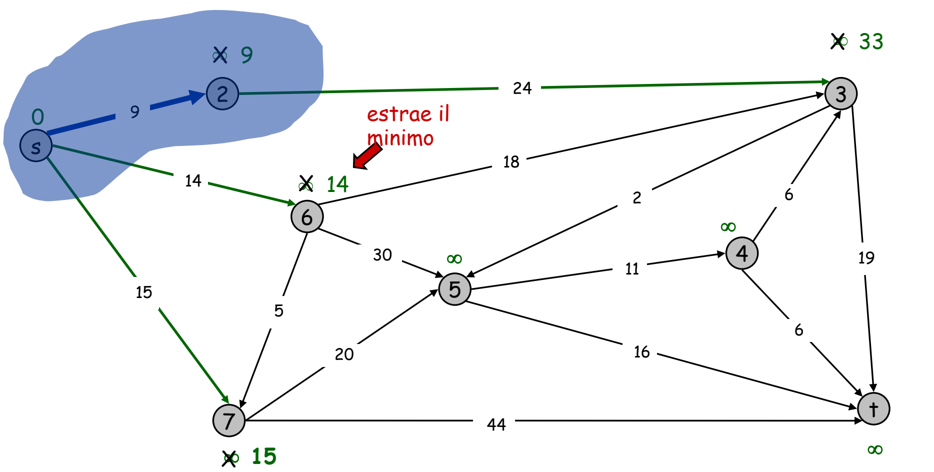
S = { s, 2 }
Q = { 3, 4, 5, 6, 7, t }



89

Algoritmo di Dijkstra

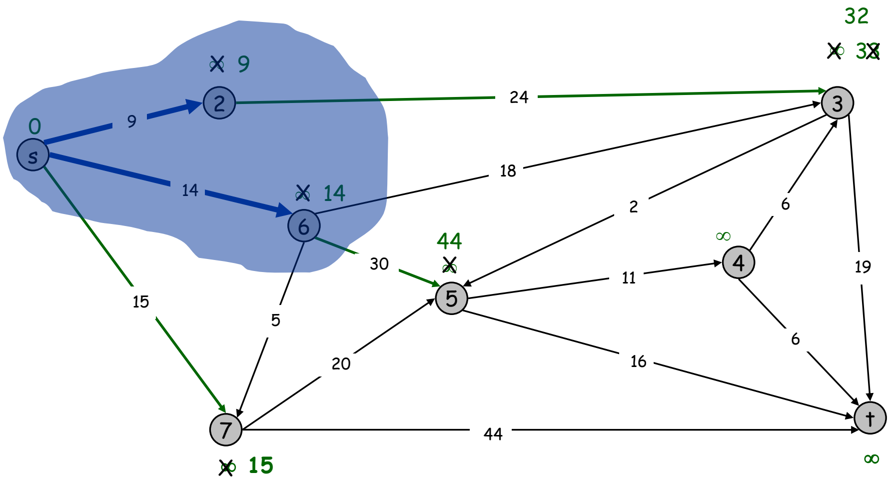
S = { s, 2 }
Q = { 3, 4, 5, 6, 7, t }



90

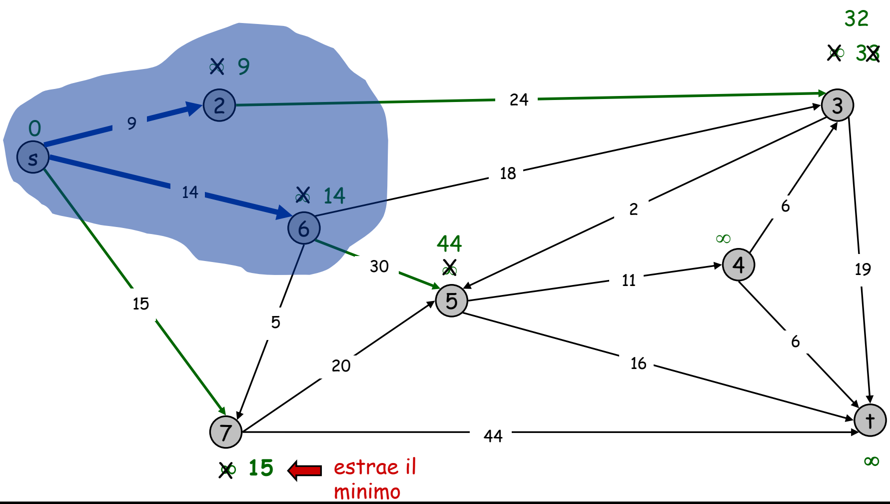
Algoritmo di Dijkstra

S = { s, 2, 6 }
Q = { 3, 4, 5, 7, t }



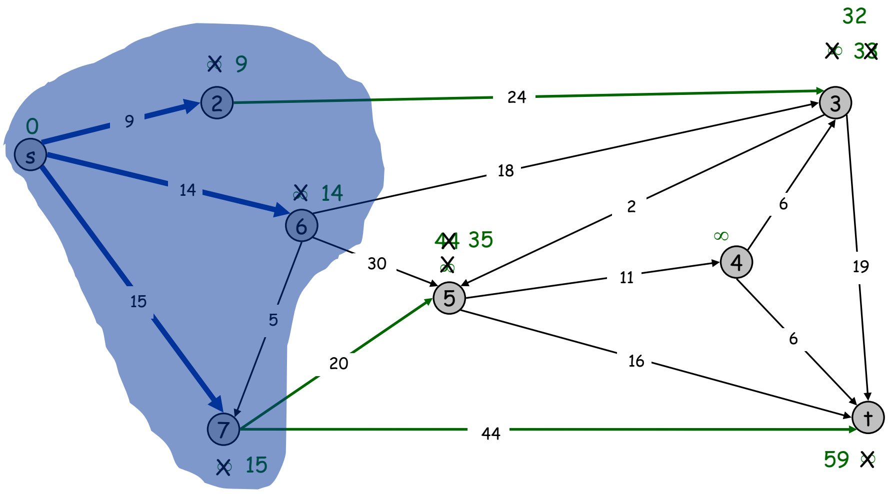
Algoritmo di Dijkstra

S = { s, 2, 6 }
Q = { 3, 4, 5, 7, t }



Algoritmo di Dijkstra

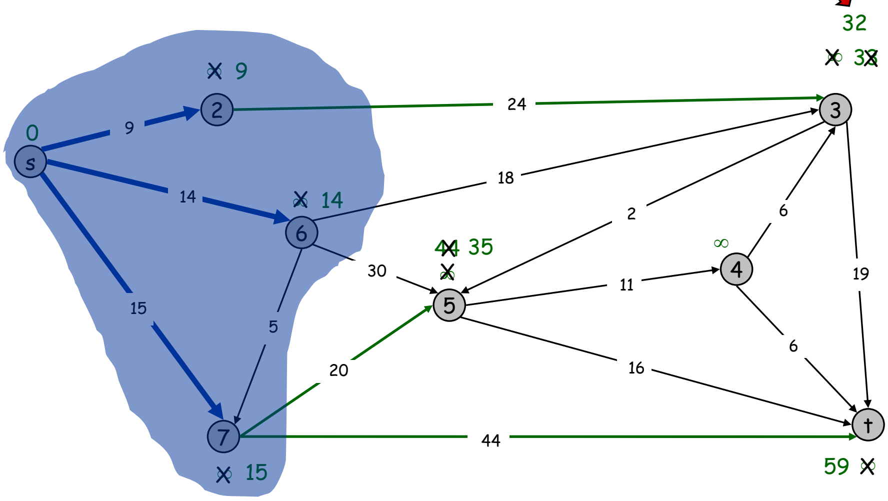
S = { s, 2, 6, 7 }
Q = { 3, 4, 5, t }



Algoritmo di Dijkstra

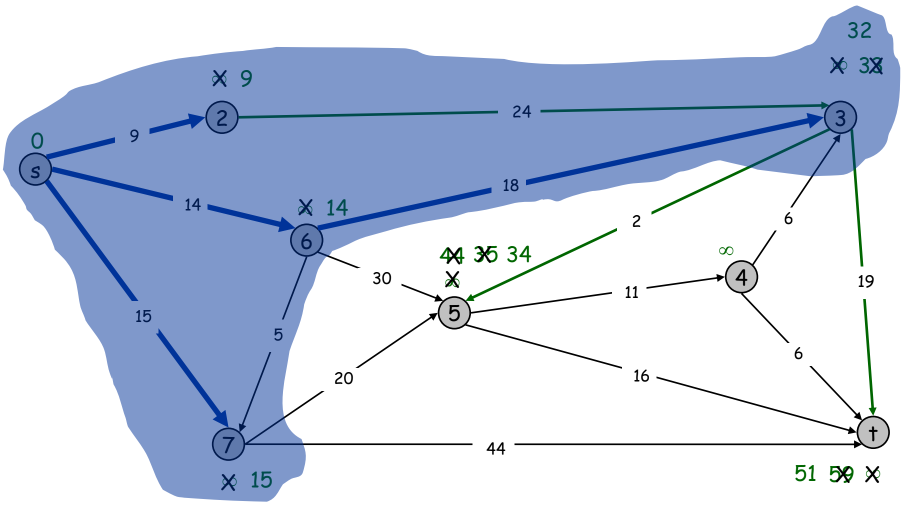
S = { s, 2, 6, 7 }
Q = { 3, 4, 5, t }

estrae il minimo



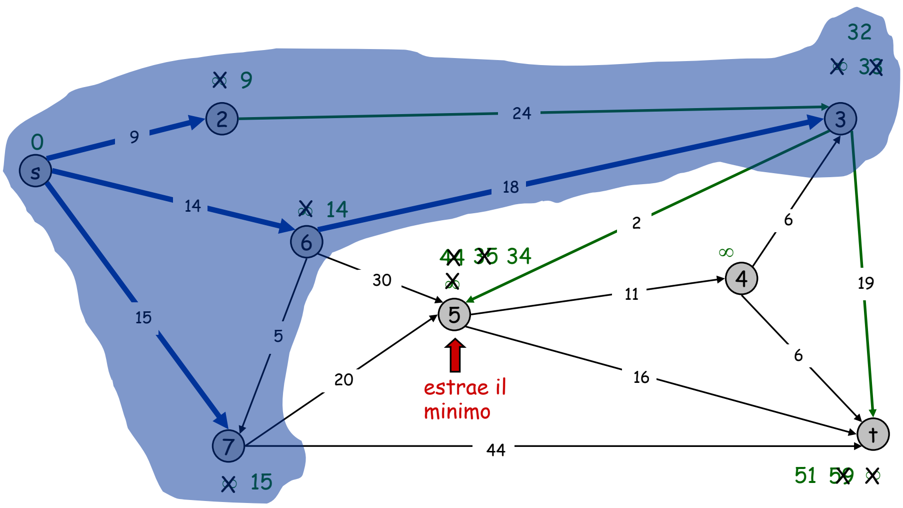
Algoritmo di Dijkstra

S = { s, 2, 3, 6, 7 }
Q = { 4, 5, t }



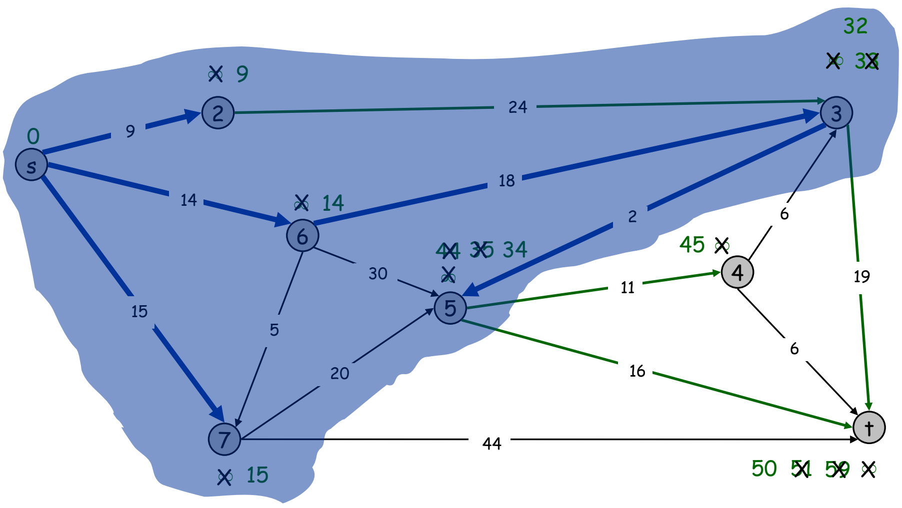
Algoritmo di Dijkstra

S = { s, 2, 3, 6, 7 }
Q = { 4, 5, t }



Algoritmo di Dijkstra

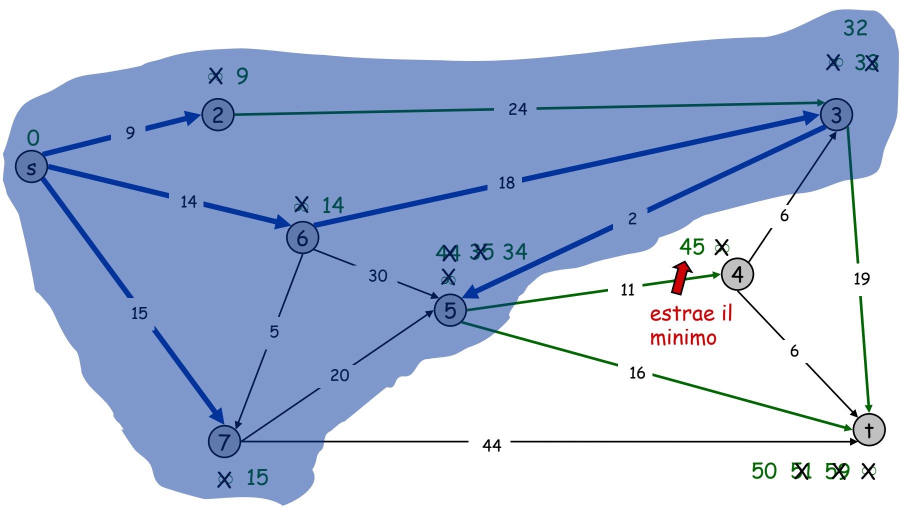
S = { s, 2, 3, 5, 6, 7 }
Q = { 4, t }



97

Algoritmo di Dijkstra

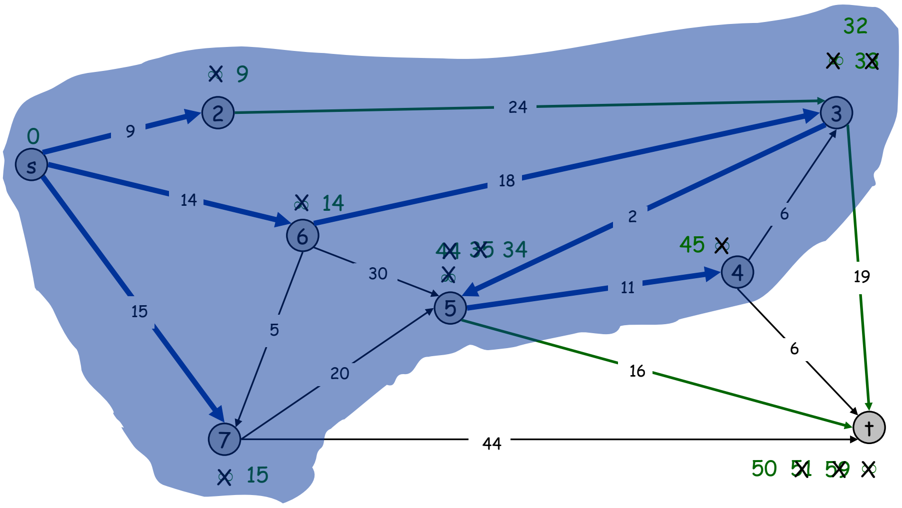
S = { s, 2, 3, 5, 6, 7 }
Q = { 4, t }



98

Algoritmo di Dijkstra

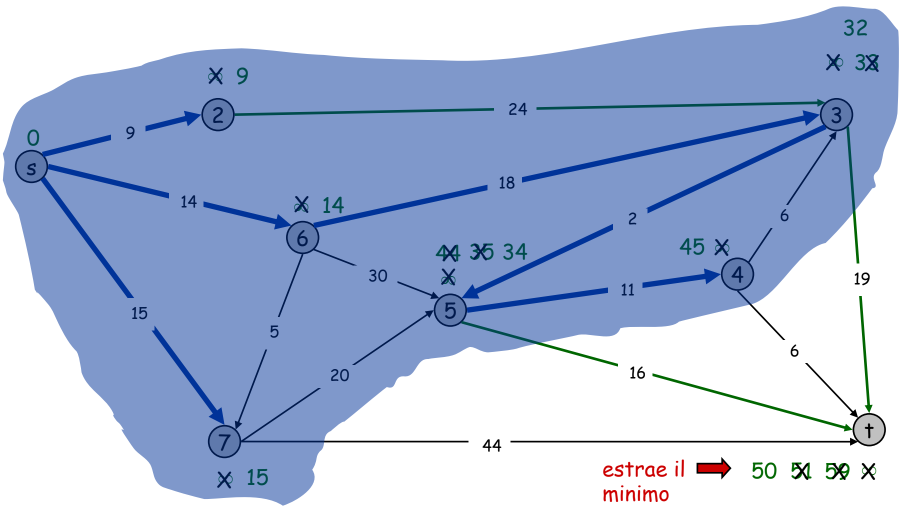
S = { s, 2, 3, 4, 5, 6, 7 }
Q = { t }



99

Algoritmo di Dijkstra

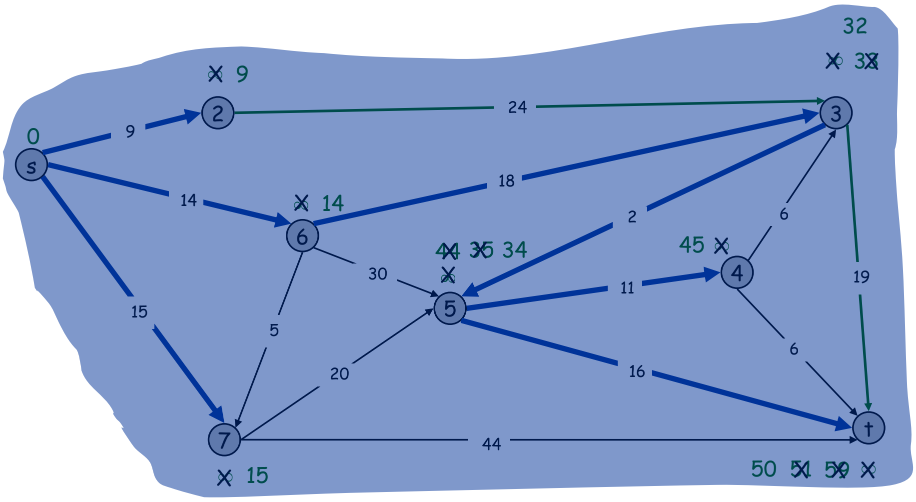
S = { s, 2, 3, 4, 5, 6, 7 }
Q = { t }



100

Algoritmo di Dijkstra

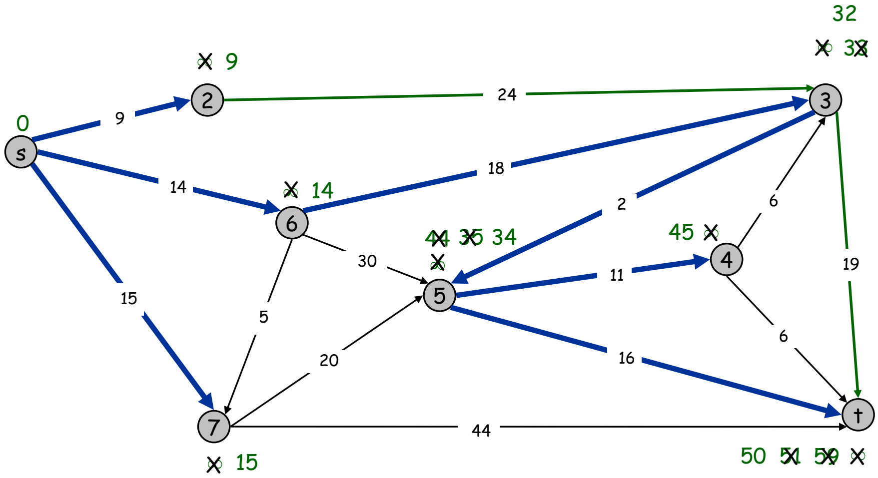
S = { s, 2, 3, 4, 5, 6, 7, t }
Q = { }



101

Algoritmo di Dijkstra

S = { s, 2, 3, 4, 5, 6, 7, t }
Q = { }



102