Algoritmi greedy
IV parte

Progettazione di Algoritmi a.a. 2019-20
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 è indicata la distanza tra ciascuna coppia di intersezioni adiacenti
- Come si può individuare il percorso più 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**

64

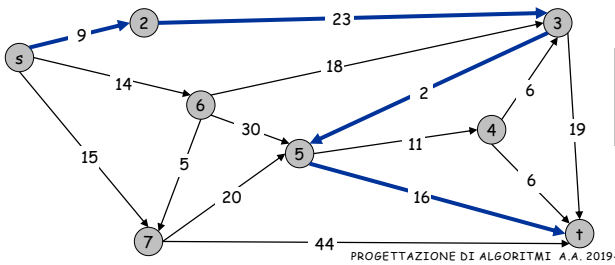
64

Il problema dei cammini minimi

- Input:
  - Grafo direzionato G = (V, E).
  - Per ogni arco e, valore ℓ_e (costo, peso o lunghezza dell'arco 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 = 50

65

65

Varianti del problema dei cammini minimi

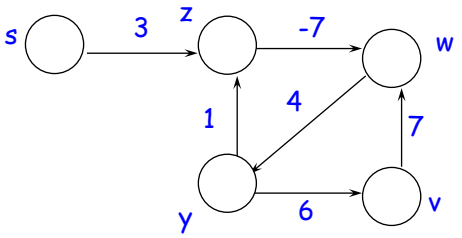
- Single Source Shortest Paths: determinare il cammino minimo da un dato vertice sorgente s ad ogni altro vertice
- Single Destination Shortest Paths: determinare i cammini minimi ad un dato vertice destinazione t da tutti gli altri vertici
  - Si riduce a Single Source Shortest Path invertendo le direzioni 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

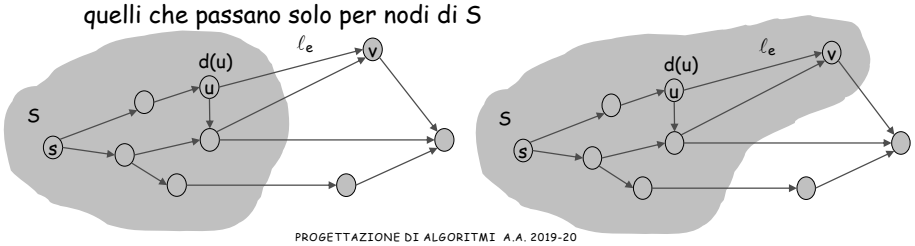
- Attraversando il ciclo <z,w,y,z> un numero arbitrario di volte possiamo trovare percorsi da s a v di peso arbitariamente piccolo

Algoritmo di Dijkstra

in G solo archi con lunghezze maggiori o uguali di zero

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 più corto da s a v tra quelli che passano solo per nodi di S



Algoritmo di Dijkstra

in G solo archi con lunghezze maggiori o uguali di zero

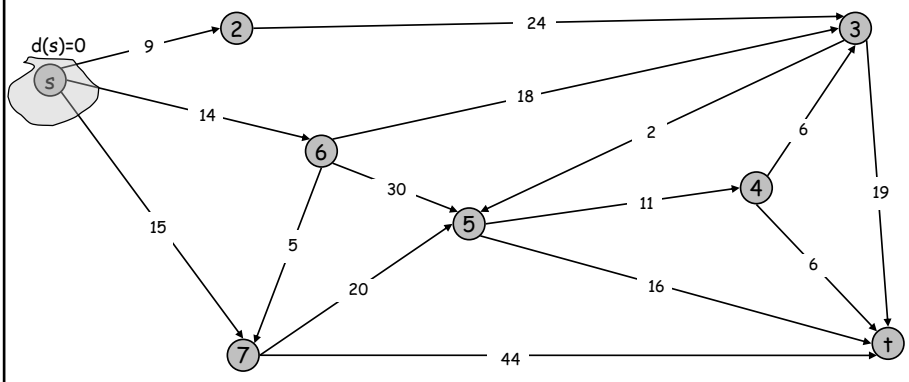
```
Dijkstra's Algorithm (G, ℓ)
Let S be the set of explored nodes
    For each u ∈ S, we store a distance d(u)
Initially S = {s} and d(s) = 0
While S ≠ V
    Select a node v ∉ S with at least one edge from S for which
        d'(v) = min_{e=(u,v):u∈S} d(u) + ℓ_e is as small as possible
    Add v to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

Quests 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 più corto da s a v computato dall'algoritmo passa solo attraverso nodi già inseriti in S.

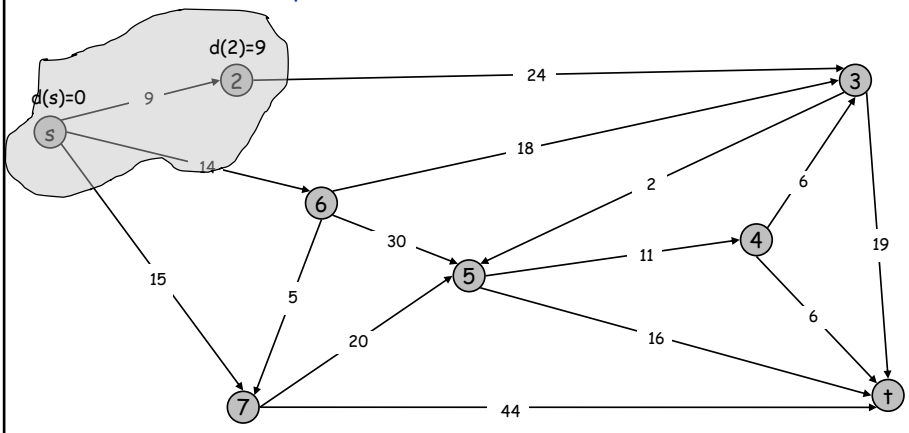
Algoritmo di Dijkstra

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

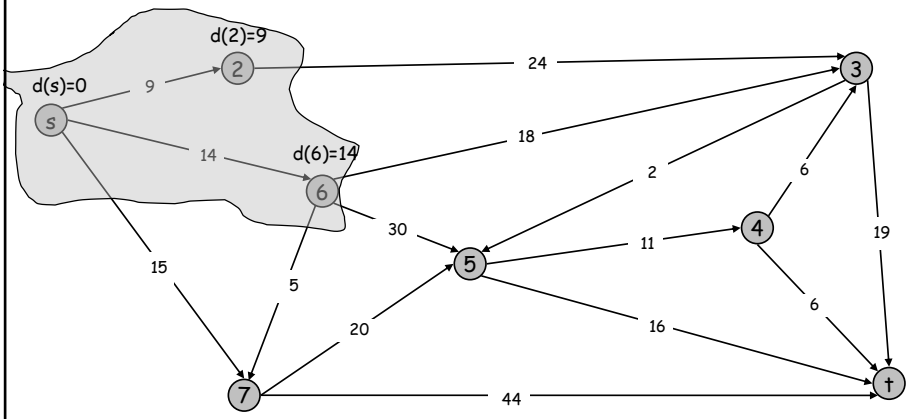


72

72

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

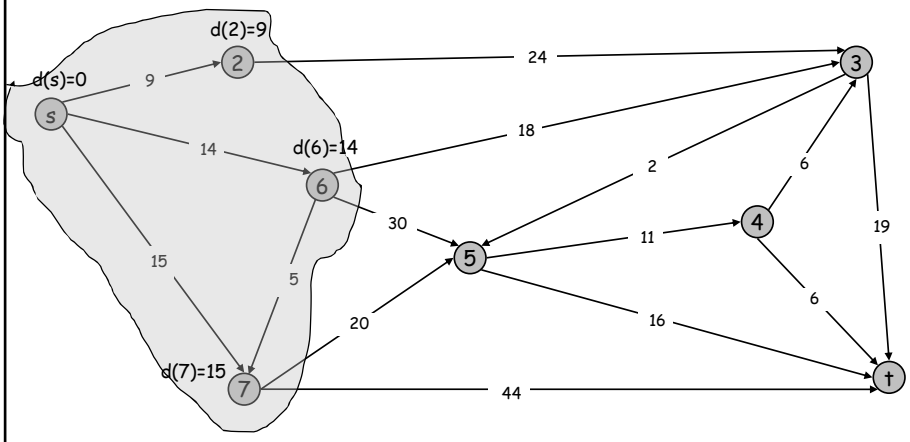


73

73

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

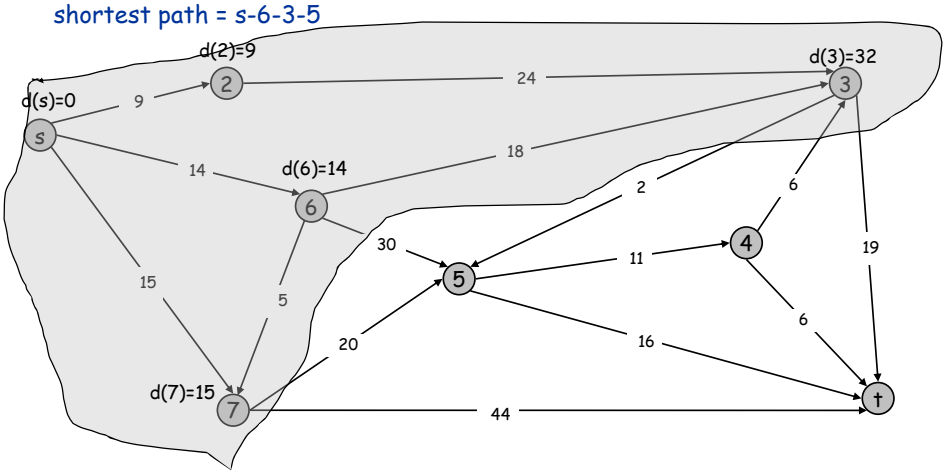


74

74

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

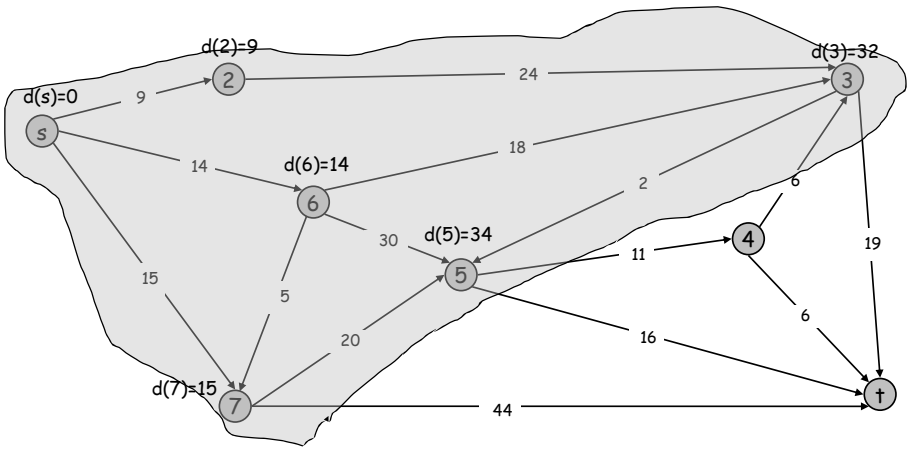


75

75

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

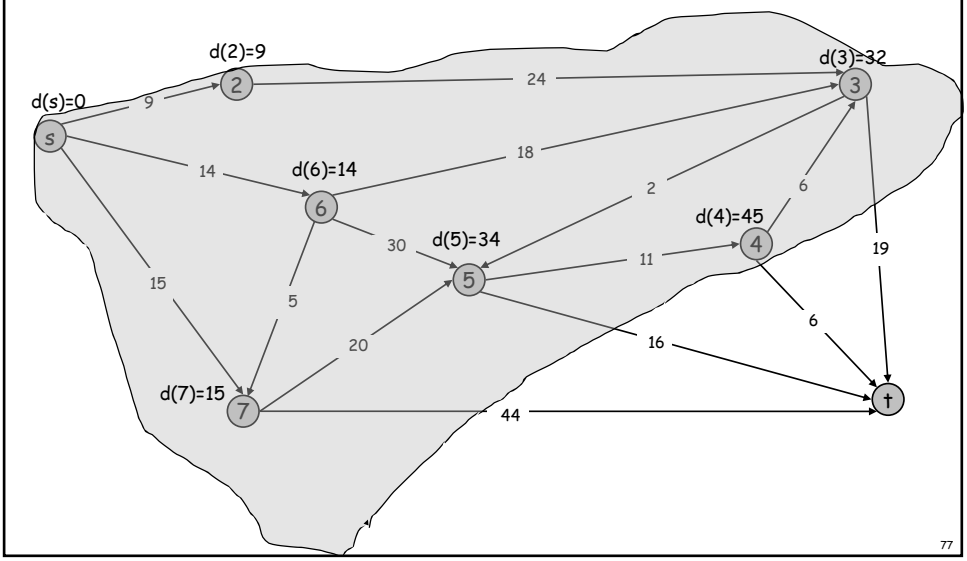


76

76

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

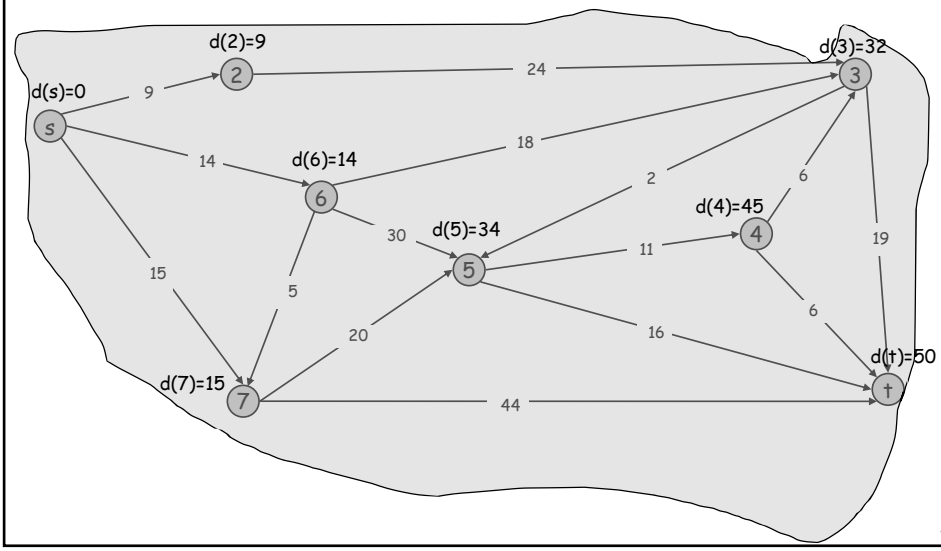


77

77

Algoritmo di Dijkstra

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



78

78

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)

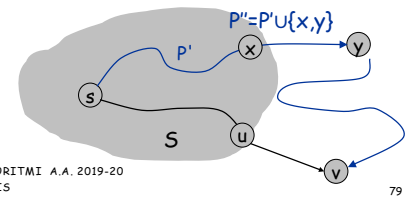
Base: |S| = 1. In questo caso S={s} e d(s)=0 per cui la tesi vale banalmente.

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

$$d'(v) = \min_{e=(u,v):\, u \in S} d(u) + \ell_e ,$$

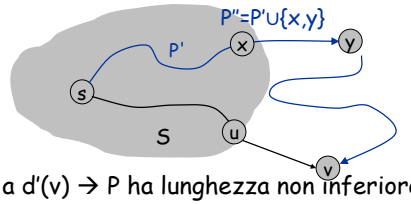
- Consideriamo il percorso di lunghezza d'(v), cioé quello formato dal percorso più corto da s ad u più l'arco (u,v)
- Consideriamo un **qualsiasi** altro percorso P da s a v. Dimostriamo che P non è più corto di d'(v)
- Sia (x,y) il primo arco di P che esce da S.
- Sia P' il sottocammino di P fino a x.
- P' più l'arco (x,y) ha già lunghezza non più piccola di d'(v) altrimenti l'algoritmo non avrebbe scelto v.



79

Algortimo di Dijksta: Correttezza

- Dimostriamo che il percorso P'' formato da P' più l'arco (x,y) ha già lunghezza non più piccola di d'(v).
- ℓ(P') + ℓ((x,y)) ≥ d(x) + ℓ((x,y)) siccome d(x) è per ipotesi induttiva uguale alla lunghezza del percorso più corto da s a x
- d(x) + ℓ((x,y)) ≥ d'(y) siccome d'(y)= min $_{(u,y):\ u\in S}$ d(u)+ ℓ((u,y))
- d'(y) ≥ d'(v) altrimenti alla k-esima iterazione l'algoritmo non avrebbe inserito v in S



- P'' ha lunghezza non inferiore a d'(v) → P ha lunghezza non inferiore a d'(v)
  - l'implicazione è vera perchè P'' è il sottocammino di P cha va da s a y e il sottocammino di P che va da y a v non può avere costo negativo.

80

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.

81

Algoritmo di Dijkstra: come renderlo più 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:

1. per ogni x ∉ S, teniamo traccia dell'ultimo valore computato d'(x) (se non è mai stato calcolato poniamo d'(x)= ∞) e
2. ad ogni iterazione del while: per ogni z ∉ S, ricomputiamo il valore d'(z) solo se è stato appena aggiunto ad S un nodo u per cui esiste l'arco (u,z)

   per calcolare il nuovo valore di d'(z) basta calcolare min{d'(z), d(u)+ℓ_e }, dove e=(u,z) è l'arco che congiunge z al nodo u appena introdotto in S. Possiamo farlo perchè teniamo traccia dell'ultimo valore precedentemente computato d'(z).
3. Se per ogni x∉ S, memorizziamo (d'(x),x) in una coda a priorità, d'(v)=min$_{x \notin S}${d'(x)} può essere ottenuto invocando la funzione Min o direttamente ExtractMin che va anche a rimuovere l'entrata (d'(v),v). L'aggiornamento di d'(z) al punto 1 può essere fatto con ChangeKey.

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(v),v)<-- ExtractMin(Q)
  Add v to S
  For each edge e=(v,z)
   If  z not in S && d(v)+ ℓe< d'(z)
     ChangeKey(Q,z, d(v)+ 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

- Tempo inizializzazione: 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

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: 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: $O(n^2)$

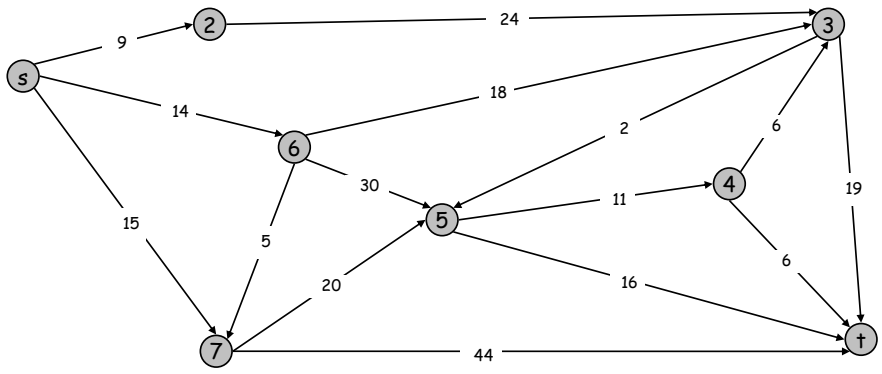
Se la coda è implementata mediante un heap binario:
Inizializzazione: O(n) con costruzione bottom up oppure O(n log n) con n inserimenti
While: O(n log n) per le n ExtractMin; O(m log n) per le m ChangeKey
Tempo algoritmo: O(n log n+ m log n)

84

84

Algoritmo di Dijkstra

Trova i percorsi più corti

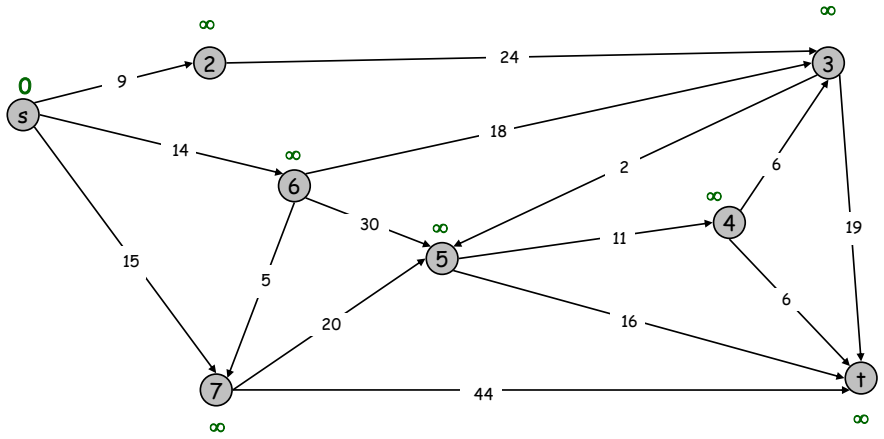


85

85

Algoritmo di Dijkstra

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

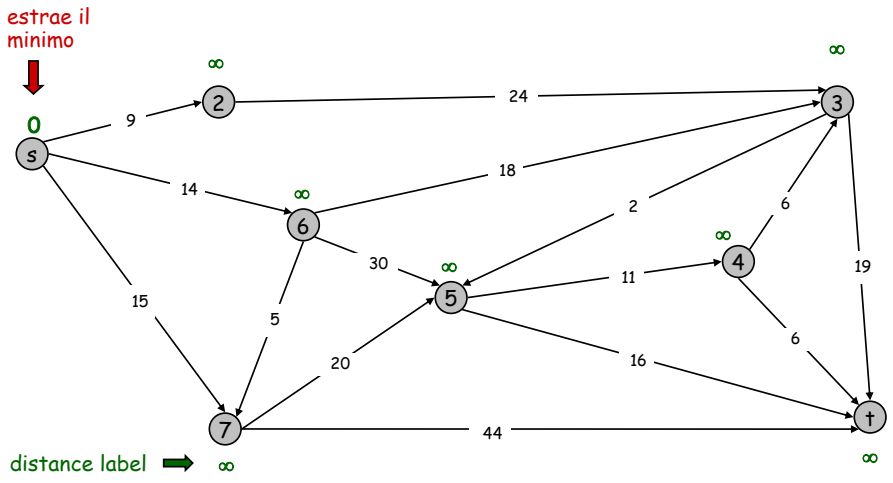


86

86

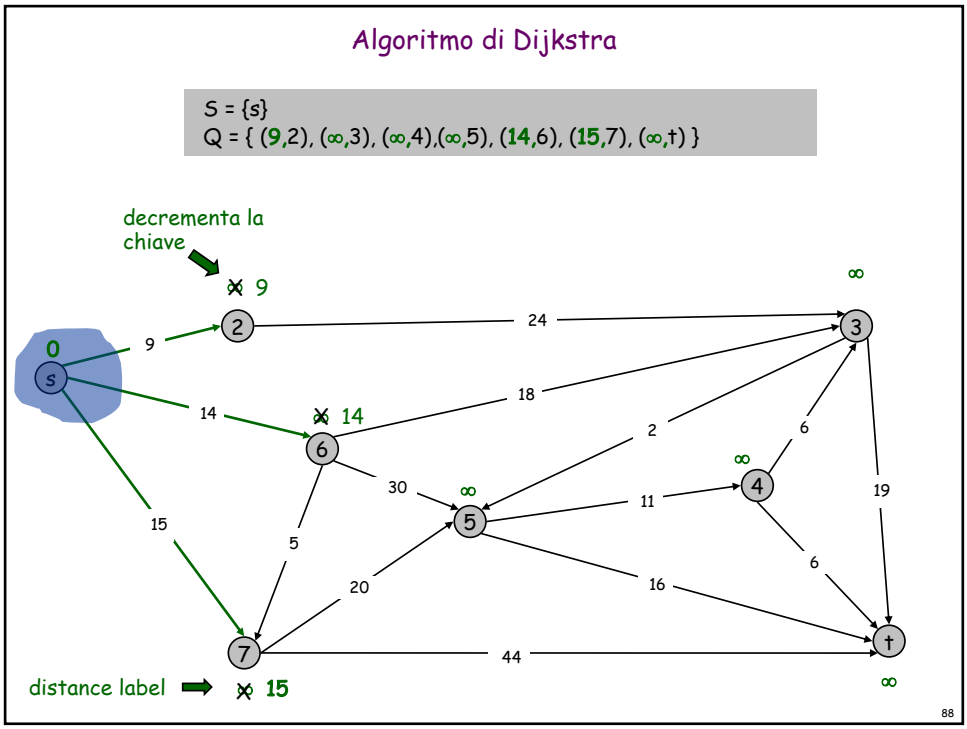
Algoritmo di Dijkstra

S = { }
Q = { (0,s), (∞,2), (∞,3), (∞,4),(∞,5), (∞,6), (∞,7), (∞,t) }

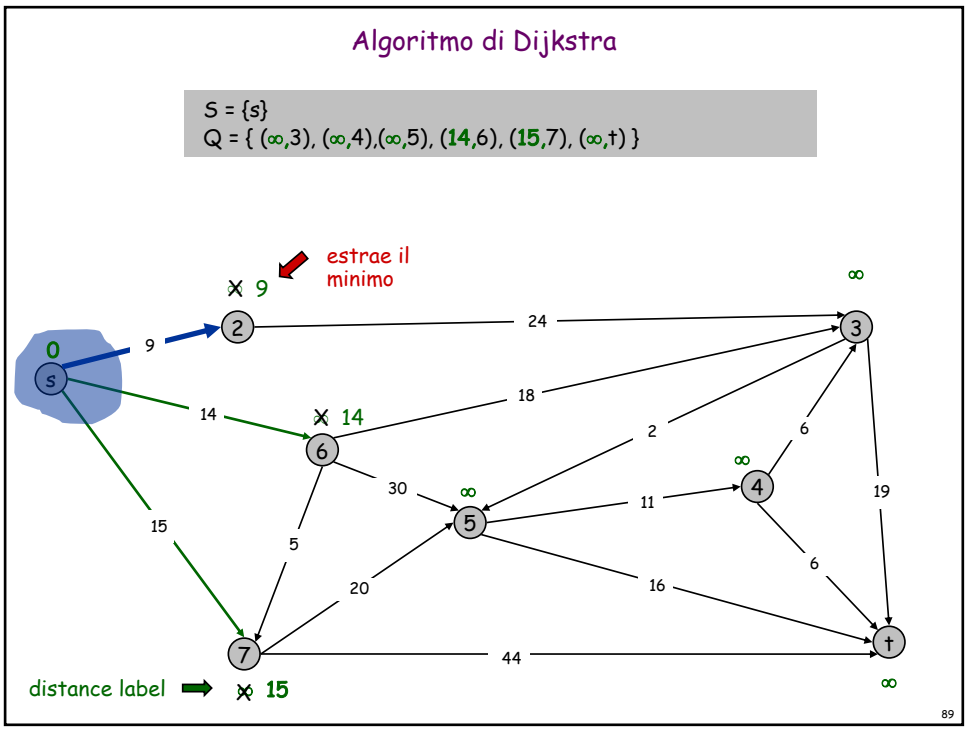


87

87


Algoritmo di Dijkstra
S = {s}
Q = { (9,2), (∞,3), (∞,4),(∞,5), (14,6), (15,7), (∞,t) }
decrementa la chiave
9
14
15
distance label
15
0
s
2
3
4
5
6
7
t
24
18
2
6
30
11
19
5
20
16
6
44
∞
88

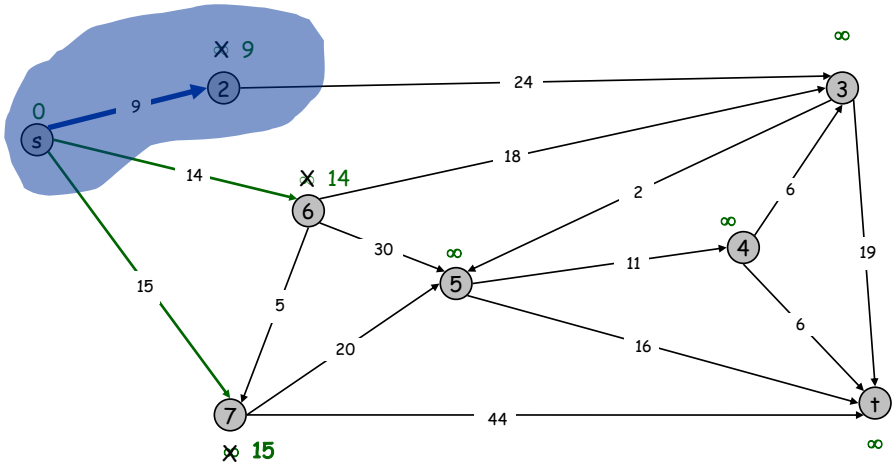
88


Algoritmo di Dijkstra
S = {s}
Q = { (∞,3), (∞,4),(∞,5), (14,6), (15,7), (∞,t) }
estrae il minimo
9
14
15
distance label
15
0
s
2
3
4
5
6
7
t
24
18
2
6
30
11
19
5
20
16
6
44
∞
89

89

Algoritmo di Dijkstra

S = {s,2}
Q = { (∞,3), (∞,4),(∞,5), (14,6), (15,7), (∞,t) }

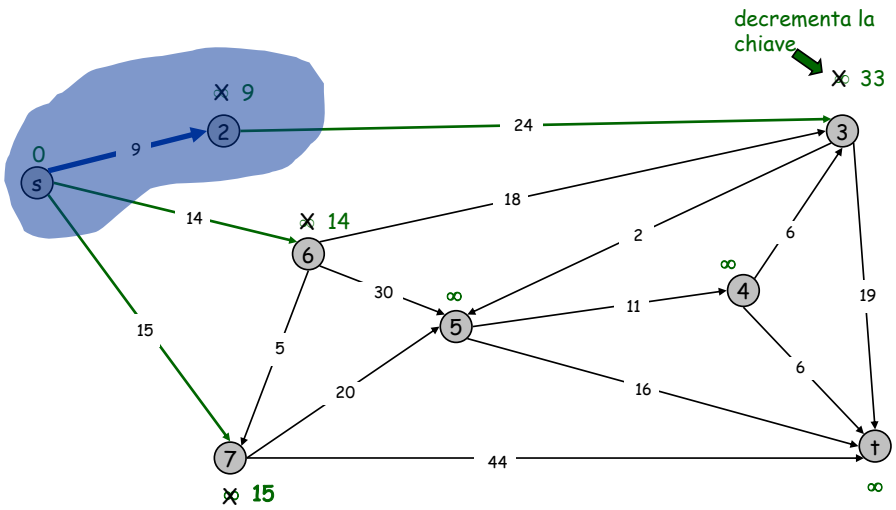


90

90

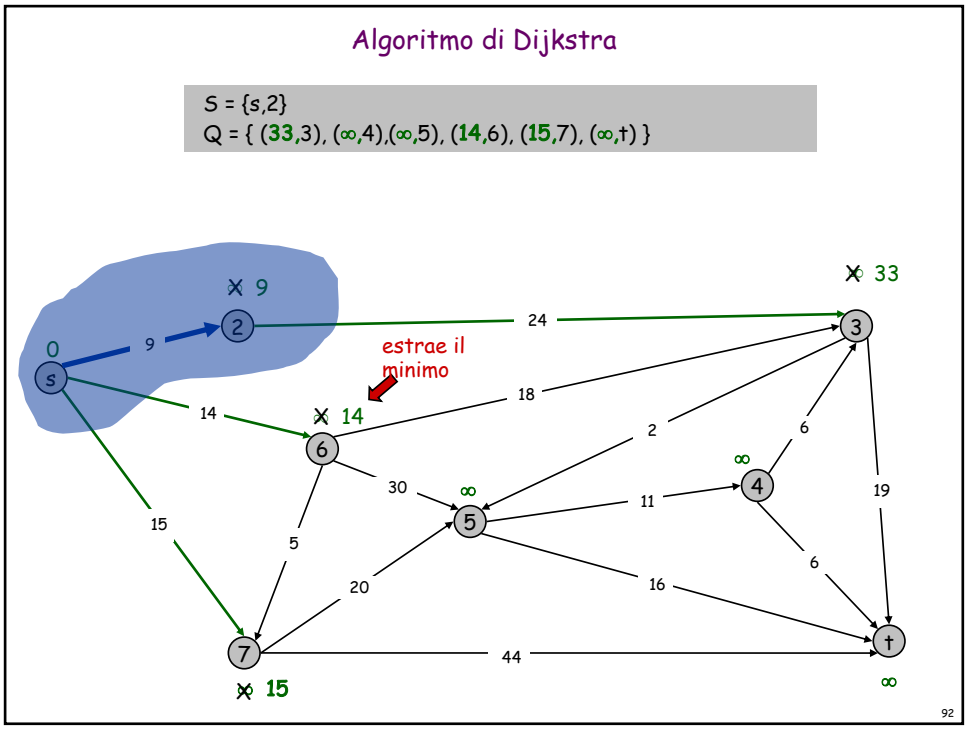
Algoritmo di Dijkstra

S = {s,2}
Q = { (33,3), (∞,4),(∞,5), (14,6), (15,7), (∞,t) }

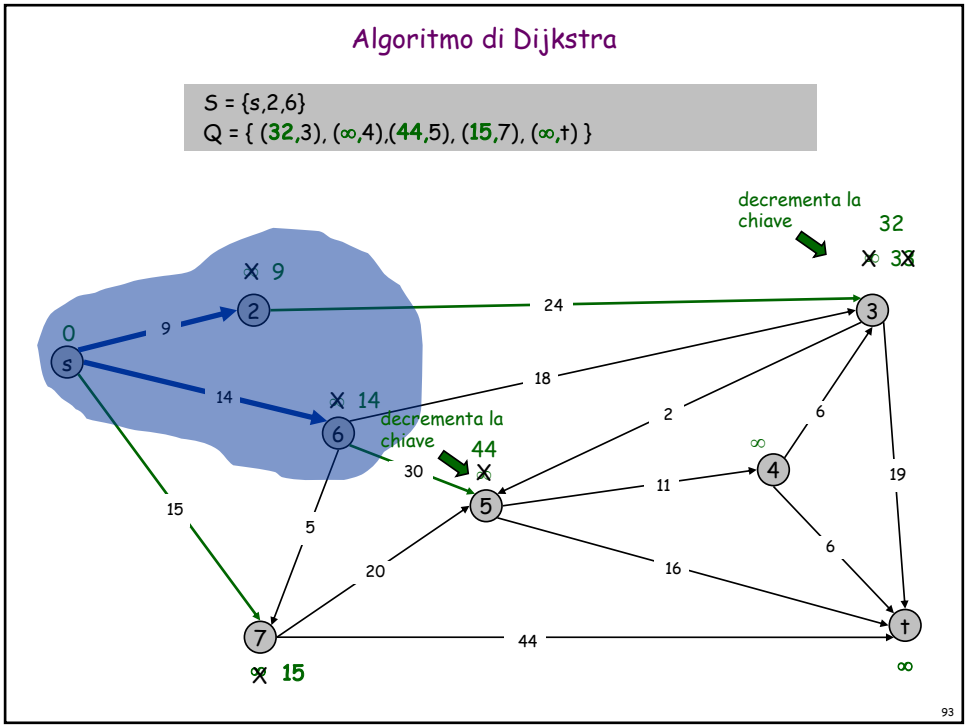


91

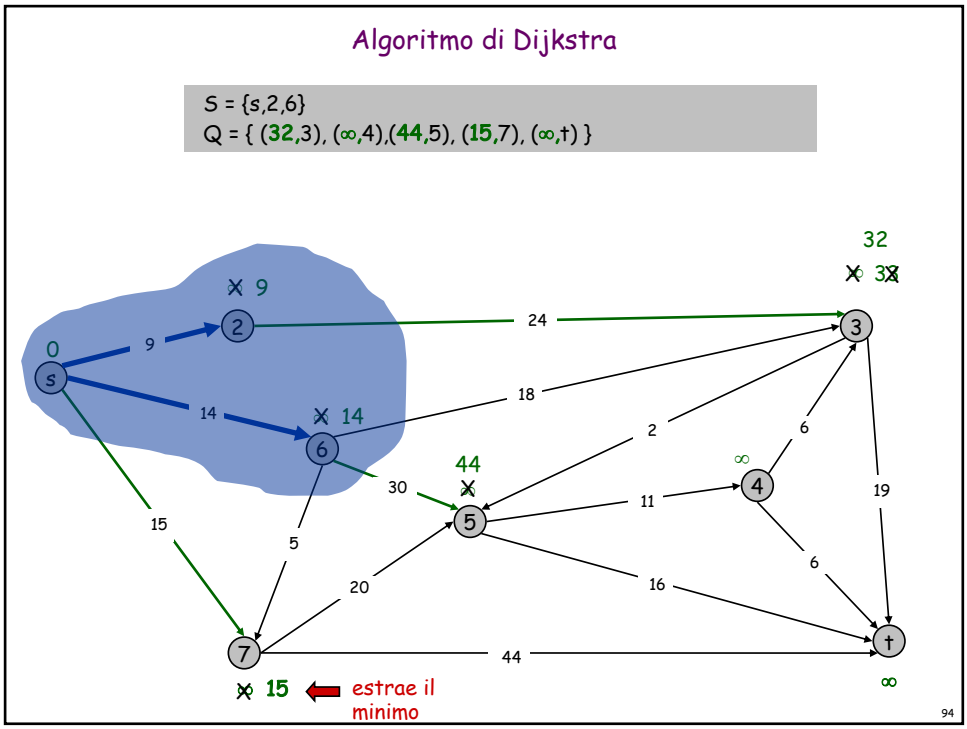
91


Algoritmo di Dijkstra
S = {s,2}
Q = { (33,3), (∞,4),(∞,5), (14,6), (15,7), (∞,t) }
9
33
estrae il minimo
14
15
∞
∞
∞
0
s
2
3
4
5
6
7
t
24
9
14
18
2
6
30
11
19
5
20
16
6
44
15
∞
92

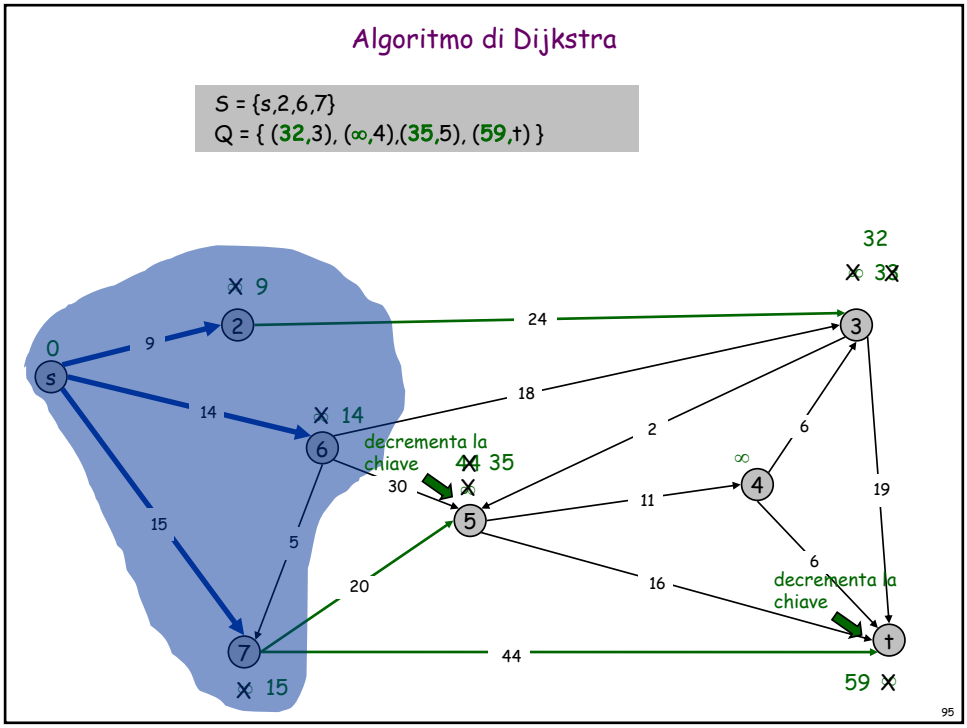
92


Algoritmo di Dijkstra
S = {s,2,6}
Q = { (32,3), (∞,4),(44,5), (15,7), (∞,t) }
decrementa la chiave
32
9
14
decrementa la chiave
44
15
∞
∞
0
s
2
3
4
5
6
7
t
24
9
14
18
2
6
30
11
19
5
20
16
6
44
15
∞
93

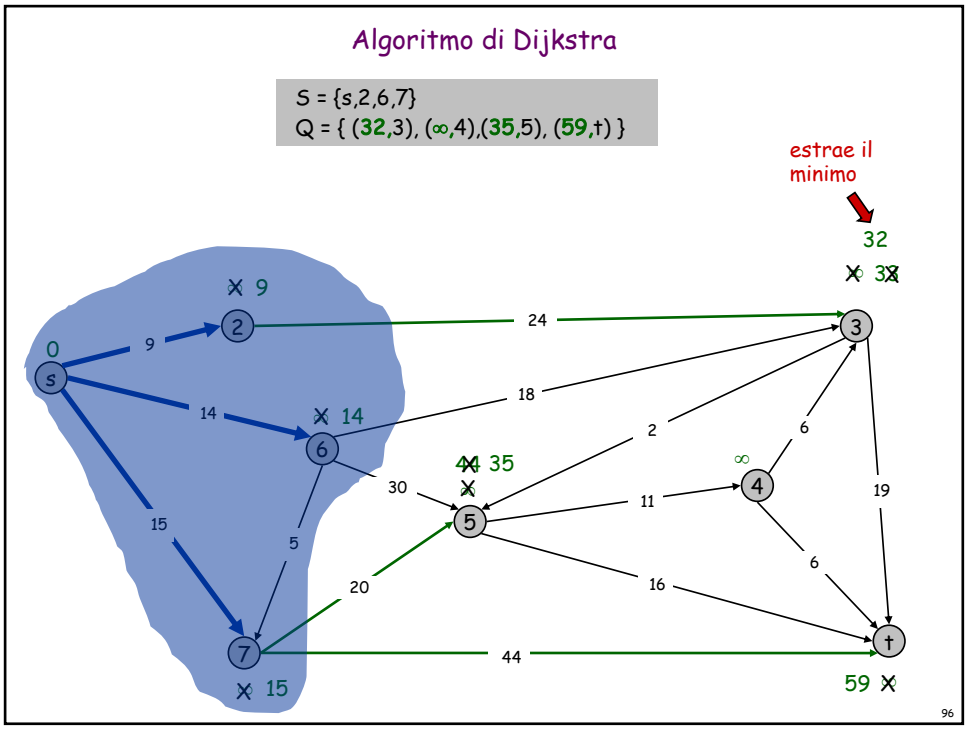
93


Algoritmo di Dijkstra
S = {s,2,6}
Q = { (32,3), (∞,4),(44,5), (15,7), (∞,t) }
0
s
9
2
24
3
32
18
14
6
14
2
6
∞
4
44
30
5
11
19
15
5
20
16
6
7
44
t
∞
15
estrae il minimo
94

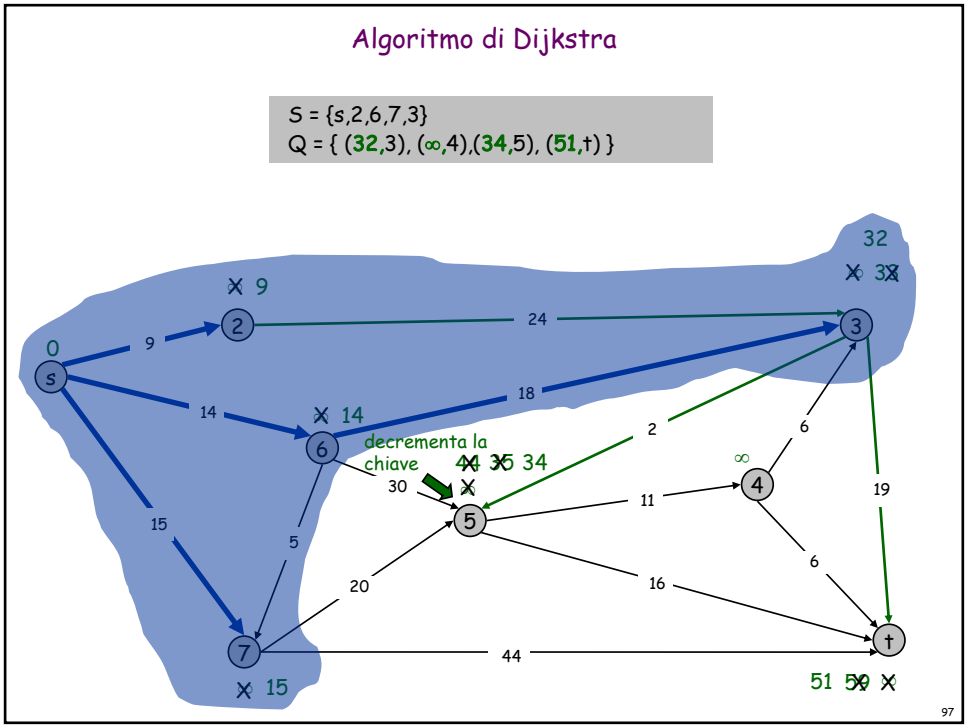
94


Algoritmo di Dijkstra
S = {s,2,6,7}
Q = { (32,3), (∞,4),(35,5), (59,t) }
0
s
9
2
24
3
32
18
14
6
14
decrementa la chiave
2
6
∞
4
44 35
30
5
11
19
15
5
20
16
6
decrementa la chiave
7
44
t
15
59
95

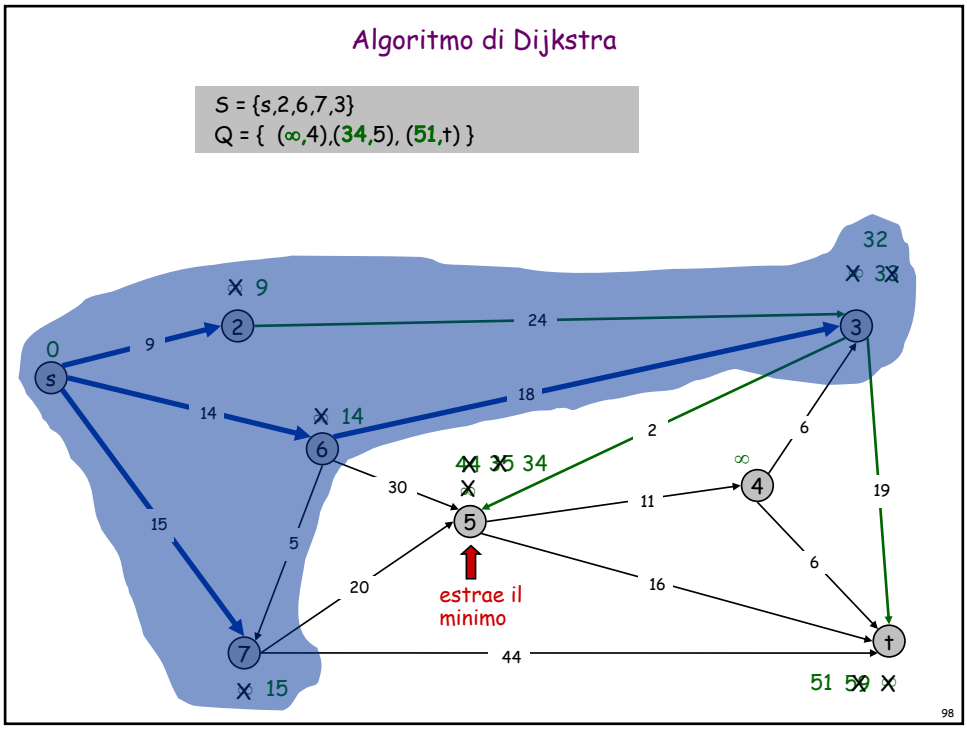
95


Algoritmo di Dijkstra
S = {s,2,6,7}
Q = { (32,3), (∞,4),(35,5), (59,t) }
estrae il minimo
32
9
14
35
15
59
s
2
3
4
5
6
7
t
0
9
24
18
14
2
6
30
11
19
5
20
16
6
44
96

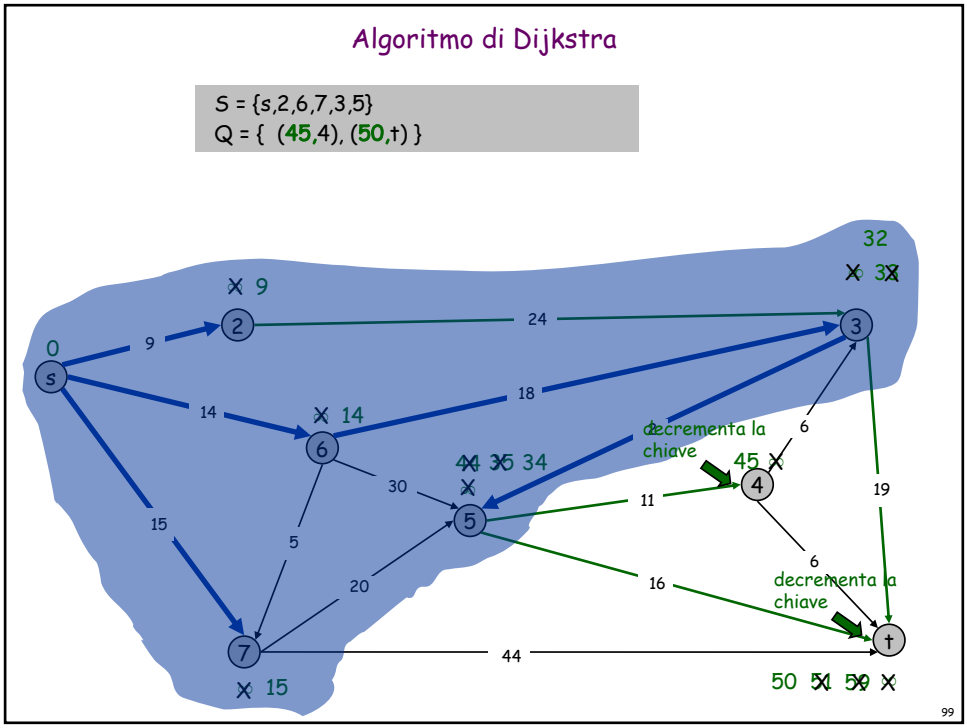
96


Algoritmo di Dijkstra
S = {s,2,6,7,3}
Q = { (32,3), (∞,4),(34,5), (51,t) }
32
9
14
decrementa la chiave
34
15
51
59
s
2
3
4
5
6
7
t
0
9
24
18
14
2
6
30
11
19
5
20
16
6
44
97

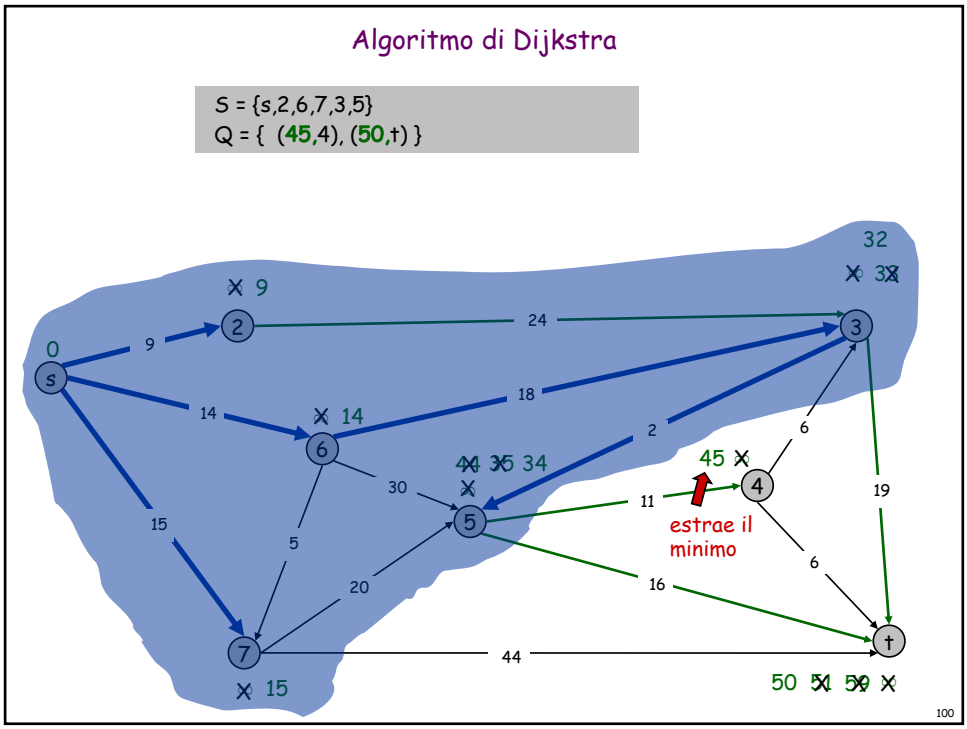
97


Algoritmo di Dijkstra
S = {s,2,6,7,3}
Q = { (∞,4),(34,5), (51,t) }
0
s
9
2
24
3
14
6
18
30
5
15
7
20
estrae il minimo
5
2
11
4
6
19
16
6
44
t
98

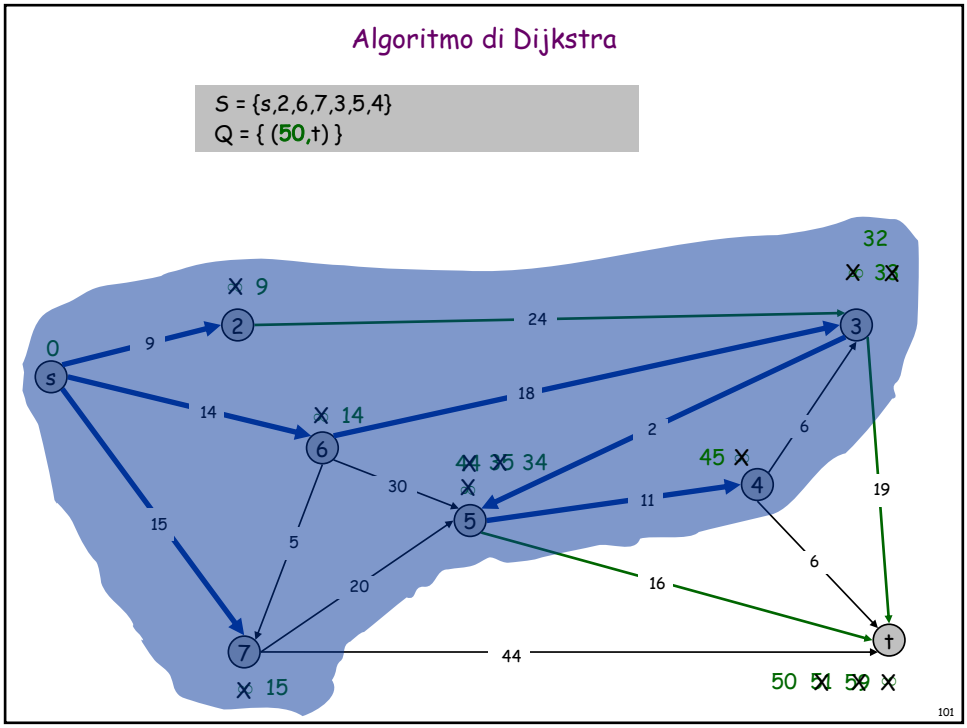
98


Algoritmo di Dijkstra
S = {s,2,6,7,3,5}
Q = { (45,4), (50,t) }
0
s
9
2
24
3
14
6
18
30
5
15
7
20
decrementa la chiave
5
2
11
4
6
19
16
6
decrementa la chiave
44
t
99

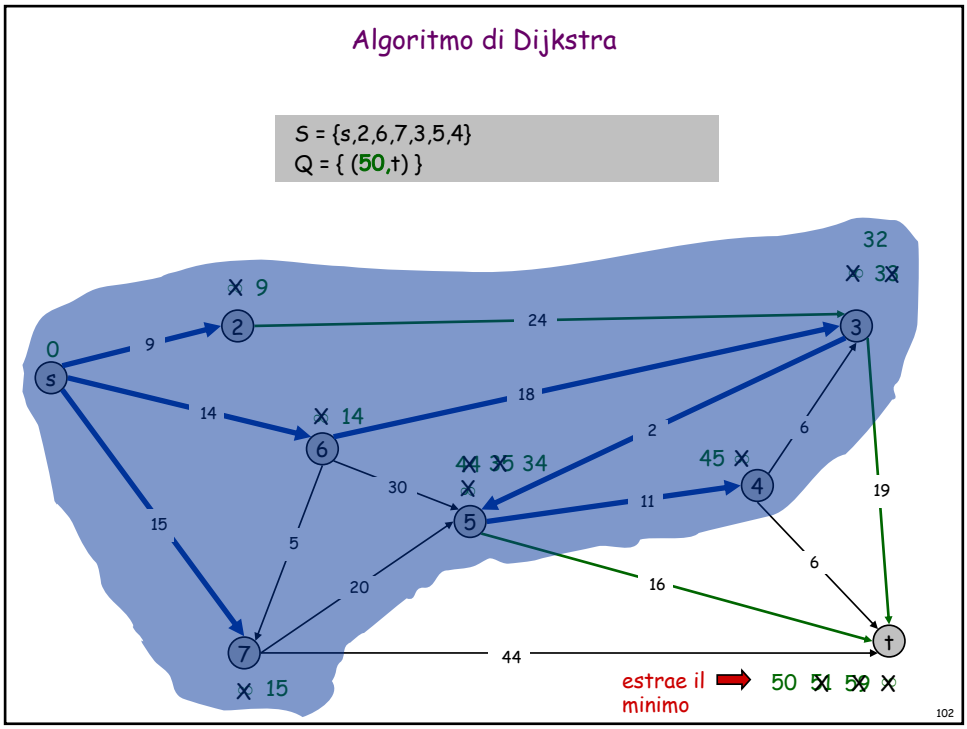
99


Algoritmo di Dijkstra
S = {s,2,6,7,3,5}
Q = { (45,4), (50,t) }
0
s
9
14
15
24
18
2
6
19
30
5
20
11
16
44
45
estrae il
minimo
50
100

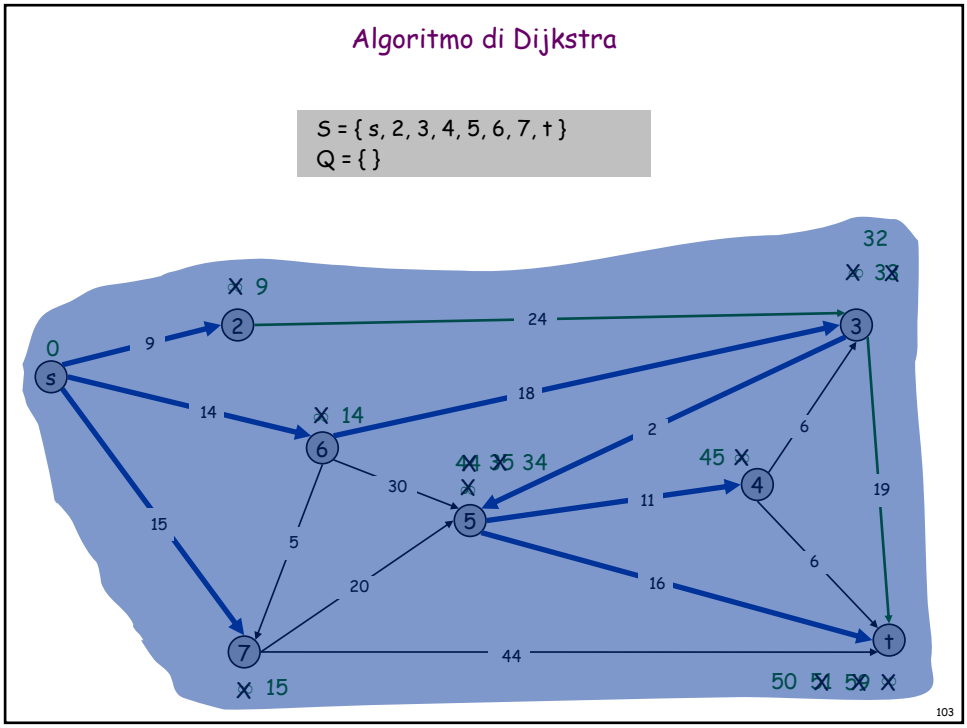
100


Algoritmo di Dijkstra
S = {s,2,6,7,3,5,4}
Q = { (50,t) }
0
s
9
14
15
24
18
2
6
19
30
5
20
11
16
44
45
50
101

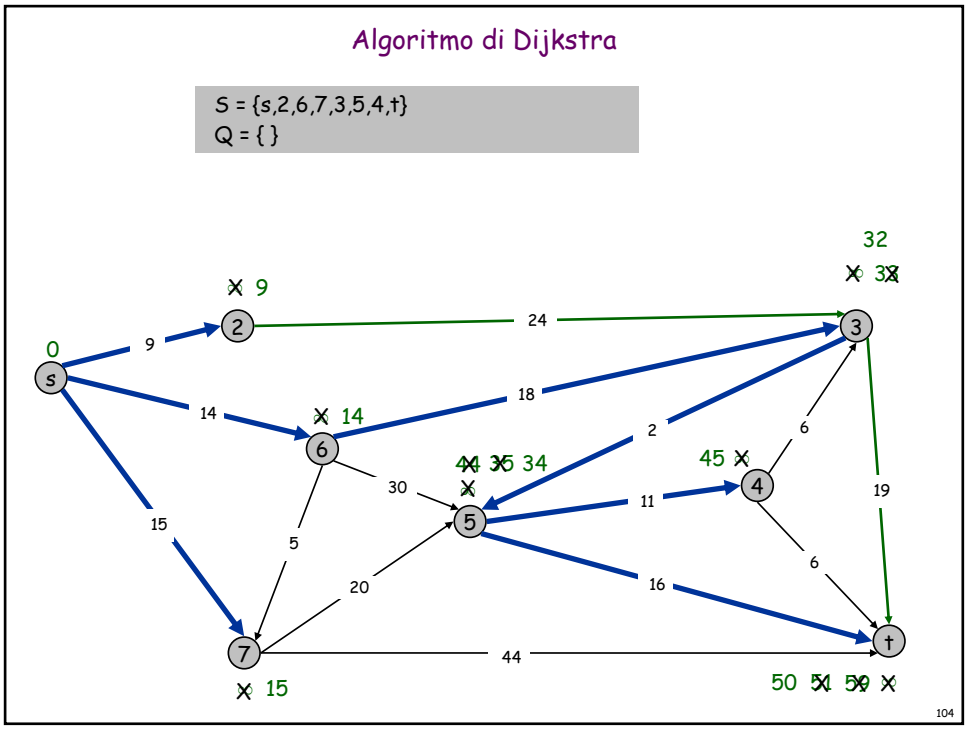
101


Algoritmo di Dijkstra
S = {s,2,6,7,3,5,4}
Q = { (50,t) }
0
s
9
14
15
24
18
30
5
20
2
6
11
16
19
44
9
14
34
45
32
15
50
estrae il minimo
102

102


Algoritmo di Dijkstra
S = { s, 2, 3, 4, 5, 6, 7, t }
Q = { }
0
s
9
14
15
24
18
30
5
20
2
6
11
16
19
44
9
14
34
45
32
15
50
103

103


Algoritmo di Dijkstra
S = {s,2,6,7,3,5,4,t}
Q = { }
0
9
14
15
24
18
2
6
19
11
30
5
20
16
44
6
9
14
34
45
32
15
50
104

104