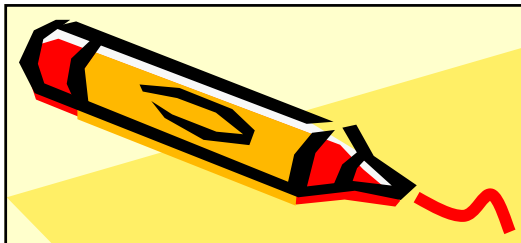
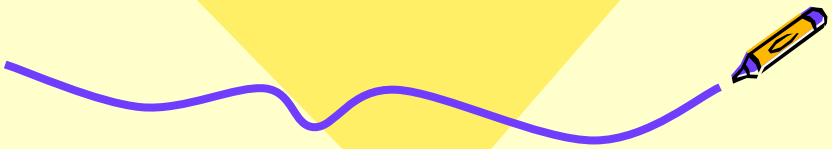
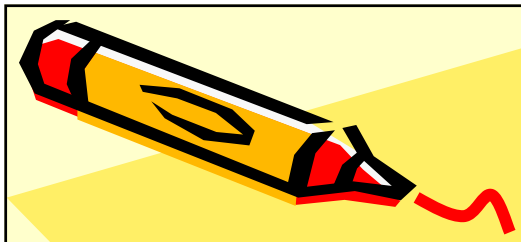


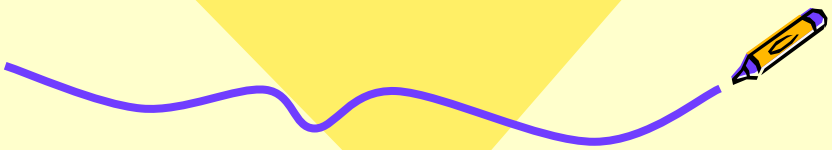
Laboratorio di Algoritmi e Strutture Dati

Aniello Murano
http://people.na.infn.it/~murano/





Algoritmi per il calcolo di percorsi minimi su un grafo



Un semplice problema

- Problema: Supponiamo che un motociclista voglia raggiungere Genova partendo da Napoli. Avendo a disposizione una mappa dell'Italia in cui per ogni collegamento diretto tra città è segnata la sua lunghezza, come può il motociclista trovare il percorso minimo?

Soluzione del problema

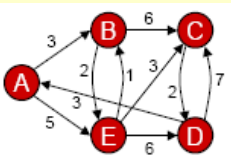
- Una soluzione è quella di numerare tutti i possibili cammini da Napoli a Genova, per ognuno calcolare la lunghezza complessiva e poi selezionare il più breve
- Questa soluzione non è la più efficiente perché ci sono milioni di cammini da analizzare.
- In questa lezione vediamo come risolvere questo problema in modo efficiente.
- In pratica, modellando la cartina dell'Italia come un grafo orientato pesato G=(V, E), dove ciascun vertice rappresenta una città, ogni arco (u,v) rappresenta una strada diretta da u a v ed ogni peso w(u,v) corrispondente ad un arco (u,v) rappresenta la distanza tra u e v, il problema da risolvere è quello di trovare il cammino minimo che collega il vertice corrispondente a Napoli con quello corrispondente a Genova.

Definizione di Shortest path (SP)

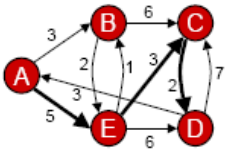
- Dato un grafo pesato orientato G=(V,E), il peso di un cammino $p=(v_0v_1,\ldots,v_k)$ è dato dalla somma dei pesi degli archi che lo costituiscono, cioè

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

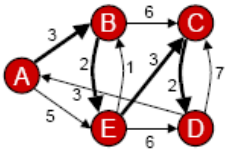
- Uno **shortest path** (cammino minimo) dal nodo u al nodo v di V è un cammino $p = (u,v_1,v_2,\ldots,v)$ tale che w(p) è minimo
- Il costo del cammino minimo da u a v è denotato con $\delta(u, v)$.
- Se non esiste un cammino da u a v allora $\delta(u, v) = \infty$



Grafo pesato orientato



Uno SP da A a D
w(p) = 10



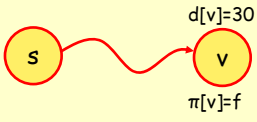
Un altro SP da A a D
w(p) = 10

Principio di ottimalità

- Dato un grafo pesato orientato G=(V,E) e uno shortest path $p = (v_0,v_1,\ldots,v_k)$ da v_0 a v_k, qualsiasi sottocammino $p' = (v_i,v_{i+1},\ldots,v_j)$ contenuto in p è anch'esso uno shortest path tra v_i e v_j

Algoritmi per il calcolo dello SP

- Dato un grafo pesato connesso orientato G=(V,E) e un nodo sorgente s di V, esistono diversi algoritmi per trovare uno SP da s verso ogni altro nodo di V (**single-source shortest path problem**)
- Dall'esecuzione di tali algoritmi si ottiene, per ogni nodo v di V, uno SP p e si calcola
  - d[v] = distanza del nodo v dal nodo sorgente s lungo lo SP p
  - π[v] = predecessore del nodo v lungo lo SP p
- Inizializzazione: per ogni nodo v di V
  - d[v] = ∞ se v ≠ s, altrimenti d[s] = 0
  - π[v] = Ø
- L'idea è ad ogni passo d[v] tale che d[v] = δ(s, v)
- Durante l'esecuzione si usa la tecnica del **rilassamento** (relaxation) di un generico arco (u,v) di E, che serve a migliorare la nostra stima per d.
- Gli algoritmi si differenziano sulla modalità di eseguire il rilassamento
  - Algoritmo di **Dijkstra** $O(E + V \log V)$
  - Algoritmo di **Bellman-Ford** $O(E\ V)$

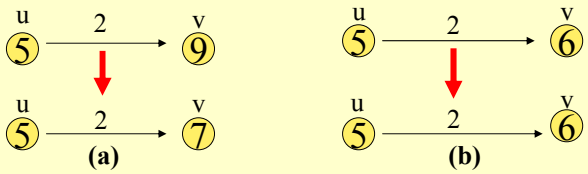


Rilassamento di un arco

- Il rilassamento di un arco (u,v) di E, consiste nel valutare se, utilizzando u come predecessore di v, si può migliorare il valore corrente della distanza d[v] e, in tal caso, si aggiornano d[v] e π[v]
- Procedura relax(u,v):

  se **d[v] > d[u] + w(u,v);** allora

  d[v] = d[u] + w(u,v); e **π[v] = u;**



- In (a), d[v] > d[u] + w(u,v). Quindi il valore di d[v] decresce
- In (b), d[v] ≤ d[u] + w(u,v). Quindi d[v] non viene modificato

Algoritmo di Dijkstra

- L'algoritmo di Dijkstra risolve il problema di cammini minimi con sorgente singola su un grafo orientato e pesato G = (V, E) nel caso in cui tutti i pesi degli archi siano non negativi.
- Assumeremo quindi che il peso $w(u, v) \geq 0$ per ogni arco (u, v) di E.
- L'algoritmo di Dijkstra mantiene un insieme S che contiene i vertici il cui peso di cammino minimo dalla sorgente s è già stato determinato.
- Inizialmente S viene inizializzato vuoto (inizializzazione).
- L'algoritmo poi seleziona ripetutamente un vertice u di S'=V – S con la minima stima di cammino minimo, inserisce u in S e rilassa tutti gli archi uscenti da u.
- Viene usata una coda con priorità Q che contiene tutti i vertici in S'
- L'algoritmo termina quando S=V

Inizializzazione

For ogni vertice v **di V**
do d[v] $\leftarrow \infty$
$\pi[v] \leftarrow$ **NIL**
d[s] $\leftarrow$ **0**

DIJKSTRA(G,s)

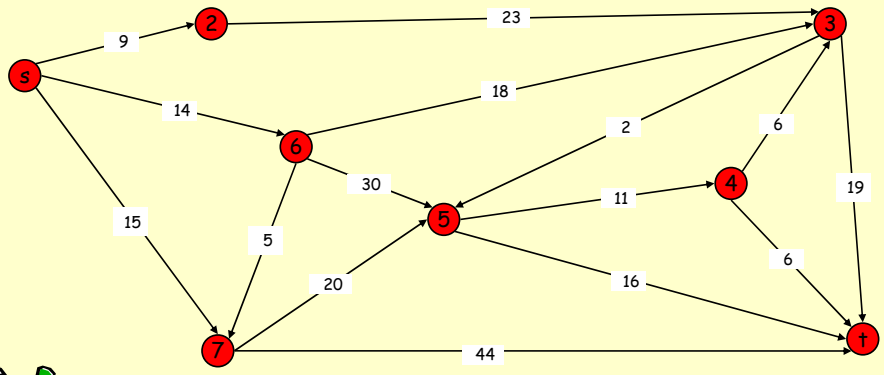
```
1.  INIZIALIZE-SINGLE-SOURCE(G ,s)
2.  S ← Ø
3.  Q ← V[G]
4.  while Q ≠ Ø
5.        do u ← EXTRACT-MIN(Q)
6.              S ← S unito {u}
7.                    for ogni vertice v di Adj[u]
8.                    do RELAX(u, v)
```

Tratto da:
Introduzione agli algoritmi
Di H.Cormen

- La linea 1 esegue l'inizializzazione,
- la linea 2 inizializza l'insieme S con l'insieme vuoto.
- La linea 3 inizializza la coda con priorità Q con tutti i vertici in V-S.
- Ad ogni esecuzione del ciclo while un vertice u viene estratto da Q e viene inserito in S (la prima volta u = s).
- Infine le linee 7-8 rilassano ogni arco (u, v) che esce da u, aggiornando la stima d[v] ed il predecessore π[v] se il cammino minimo per v può essere migliorato passando per u.
- Si osservi che ogni vertice viene estratto da Q ed inserito in S una sola volta;
- Quindi il ciclo while viene ripetuto |V| volte.

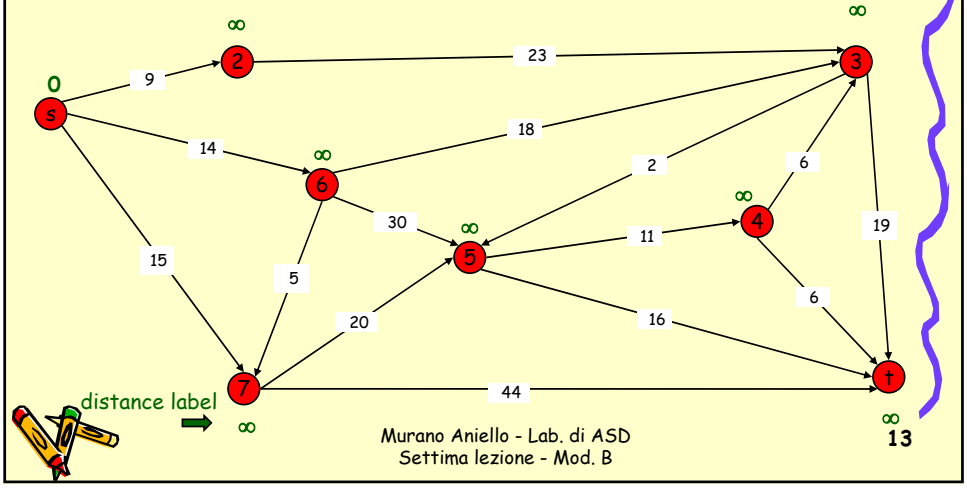
Dijkstra's Shortest Path Algorithm

Consideriamo il seguente grafo e il problema di trovare il cammino minimo da s a t



Dijkstra's Shortest Path Algorithm

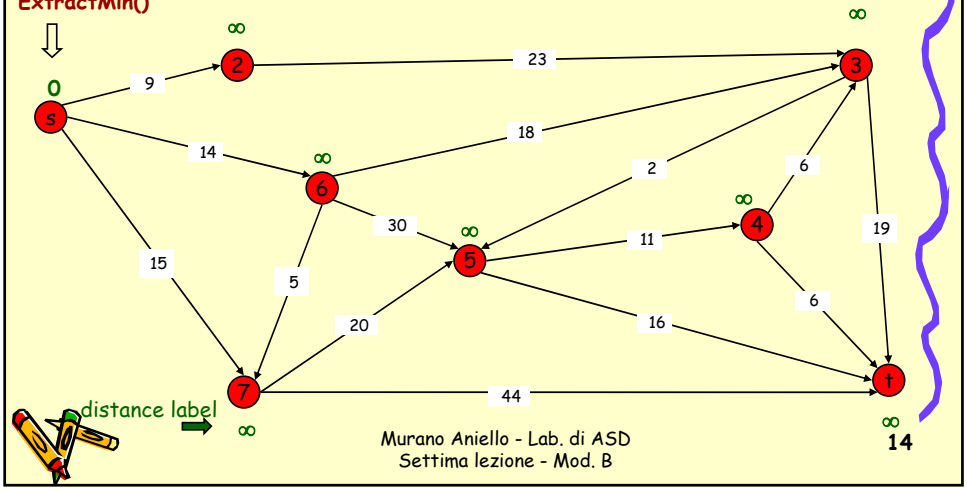
S = { }
Q = { s, 2, 3, 4, 5, 6, 7, t }

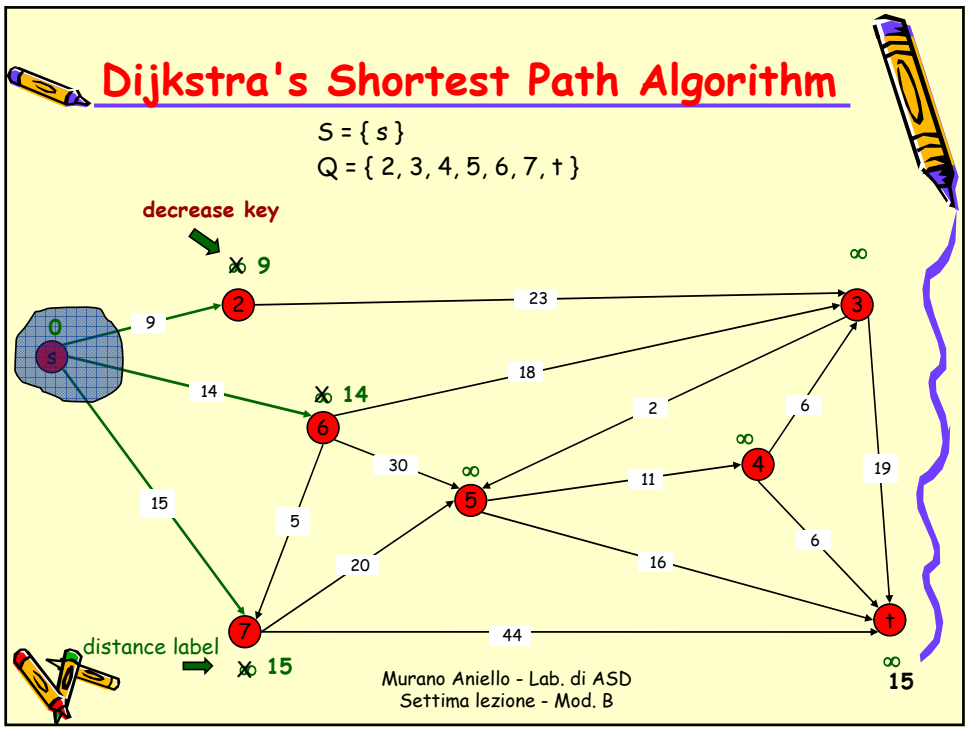


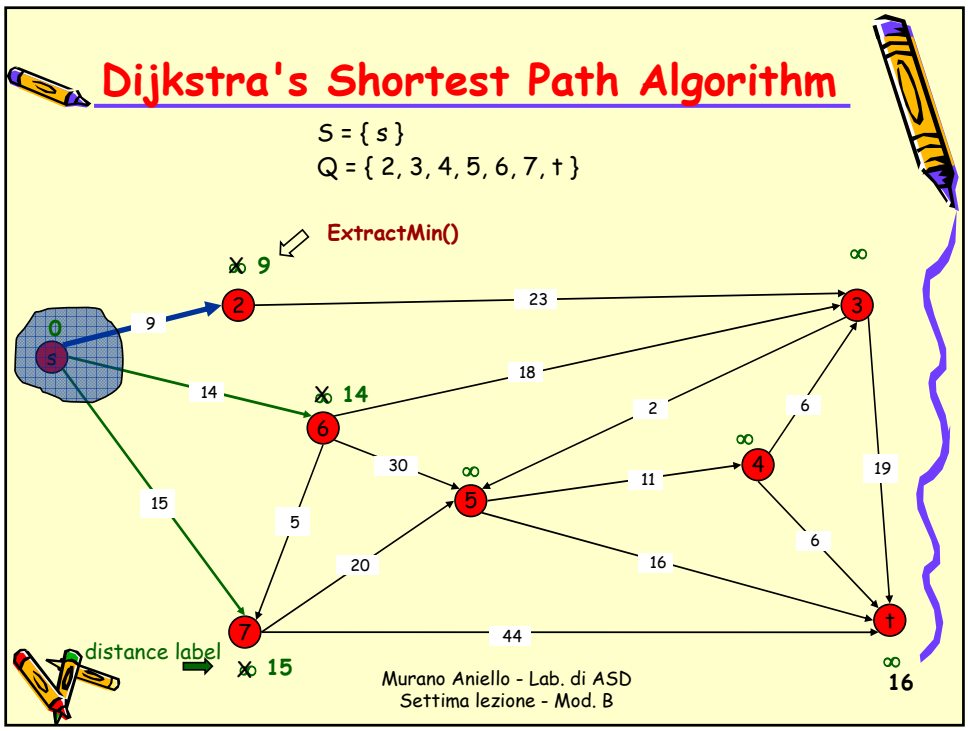
Dijkstra's Shortest Path Algorithm

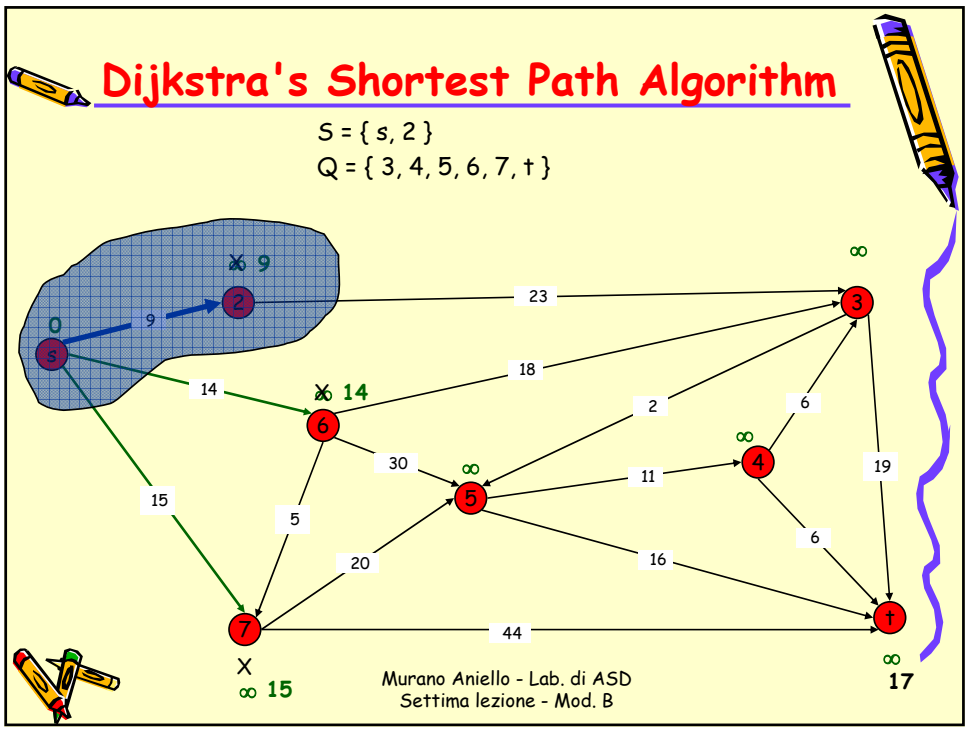
S = { }
Q = { s, 2, 3, 4, 5, 6, 7, t }

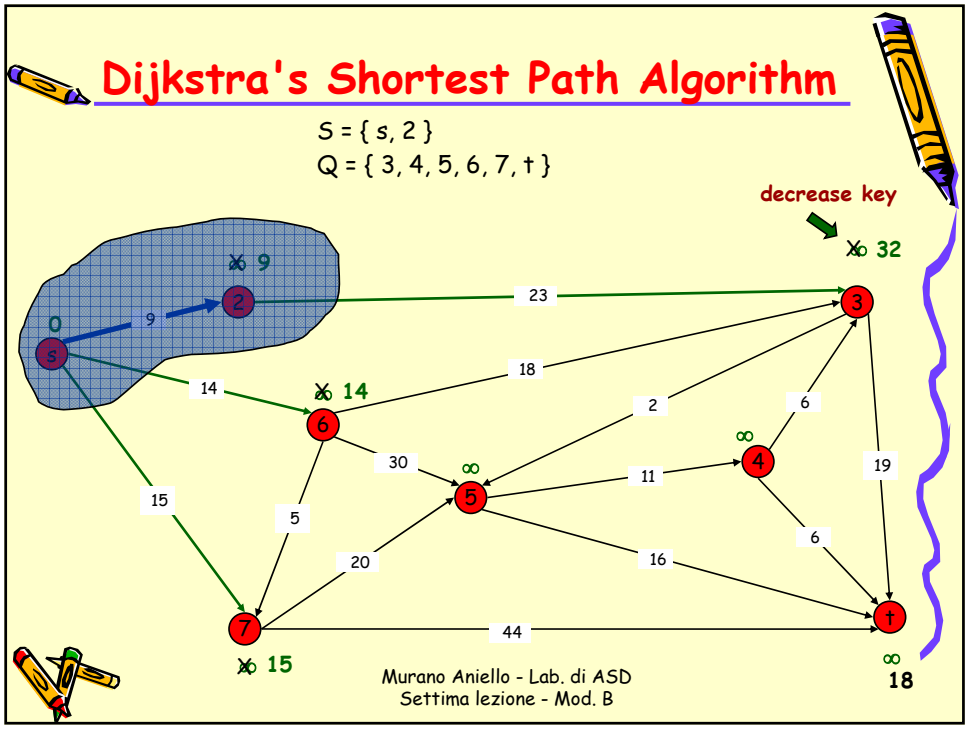
ExtractMin()

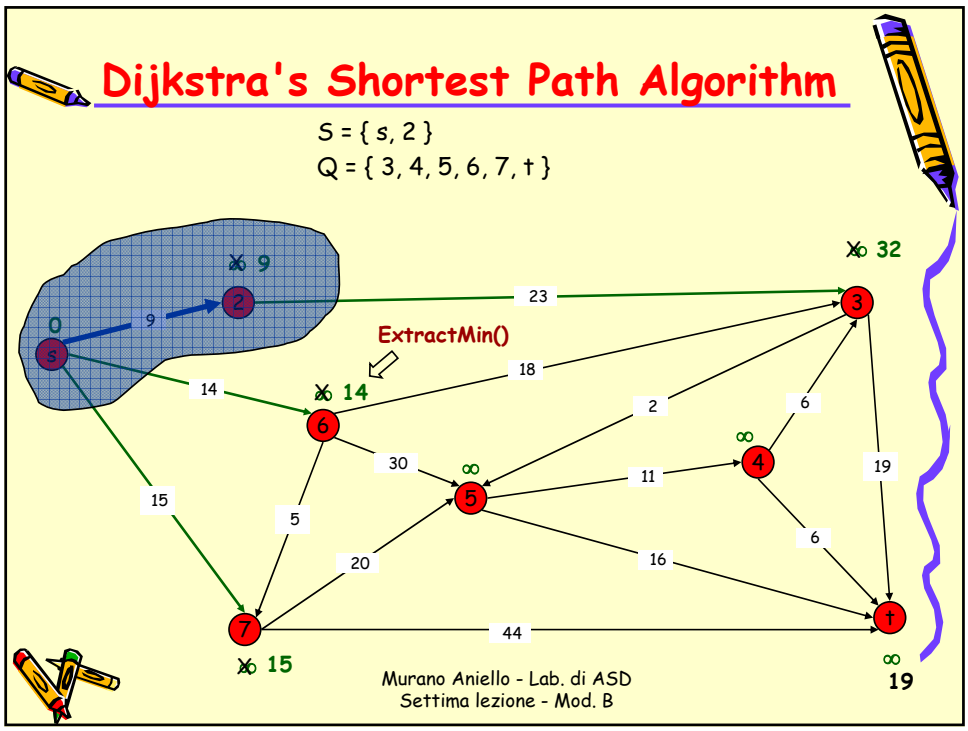


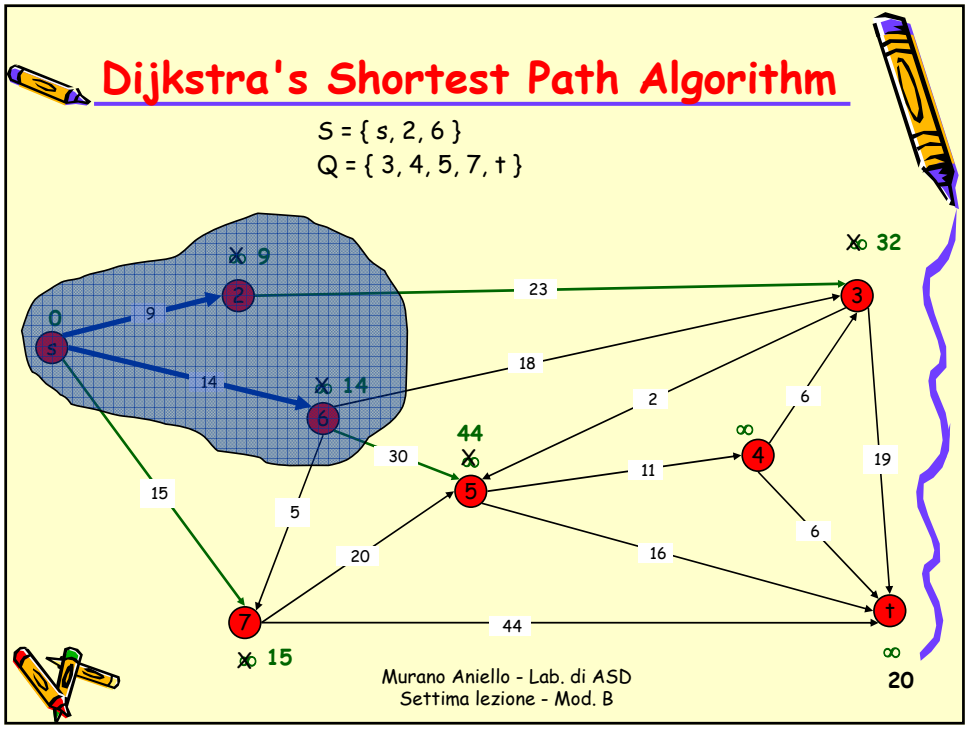

Dijkstra's Shortest Path Algorithm
S = { s }
Q = { 2, 3, 4, 5, 6, 7, t }
decrease key
∞ 9
∞ 14
distance label
∞ 15
Murano Aniello - Lab. di ASD
Settima lezione - Mod. B
∞ 15

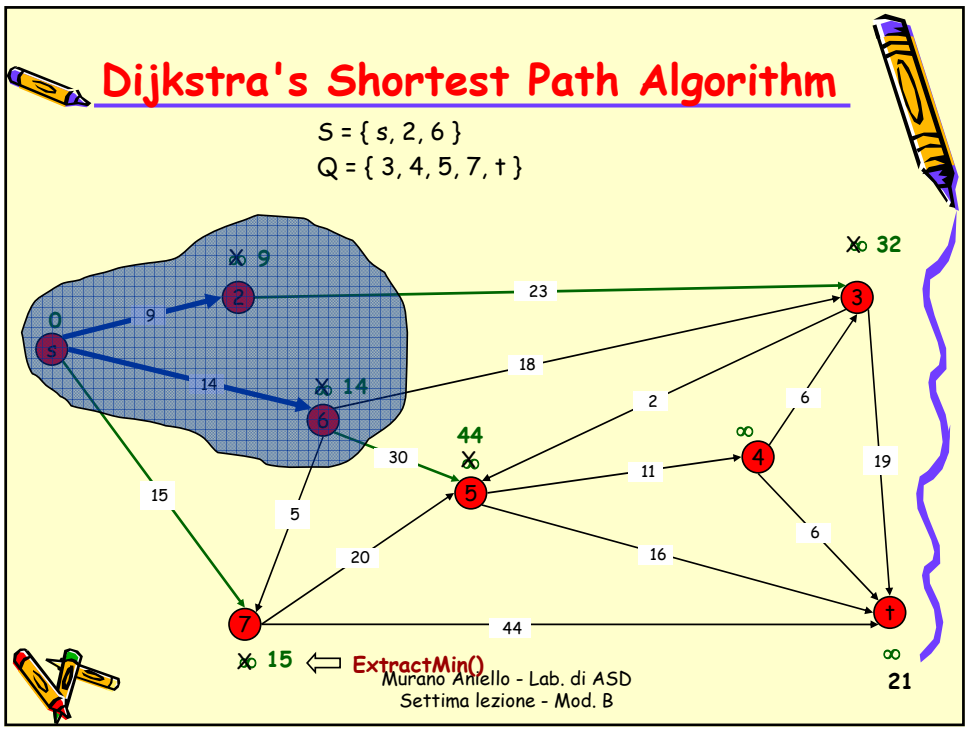

Dijkstra's Shortest Path Algorithm
S = { s }
Q = { 2, 3, 4, 5, 6, 7, t }
ExtractMin()
∞ 9
∞ 14
distance label
∞ 15
Murano Aniello - Lab. di ASD
Settima lezione - Mod. B
∞ 16

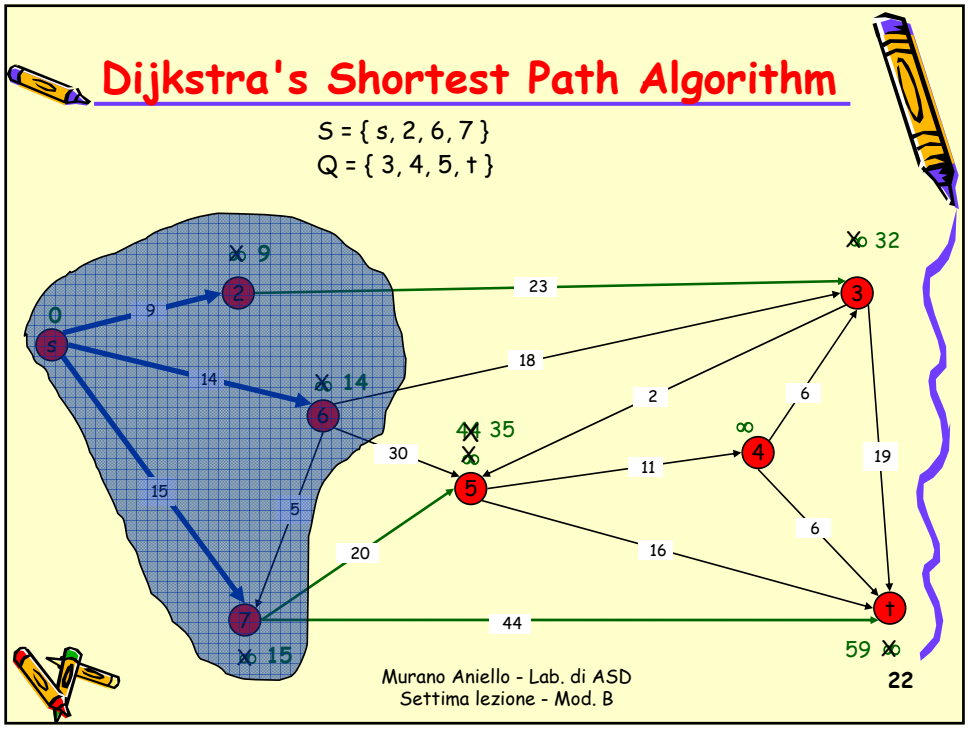

Dijkstra's Shortest Path Algorithm
S = { s, 2 }
Q = { 3, 4, 5, 6, 7, t }
∞ 9
0
s
9
2
23
∞
3
14
18
∞ 14
6
2
6
30
∞
5
∞
4
11
19
15
5
6
20
16
7
44
t
X
∞ 15
∞
17
Murano Aniello - Lab. di ASD
Settima lezione - Mod. B

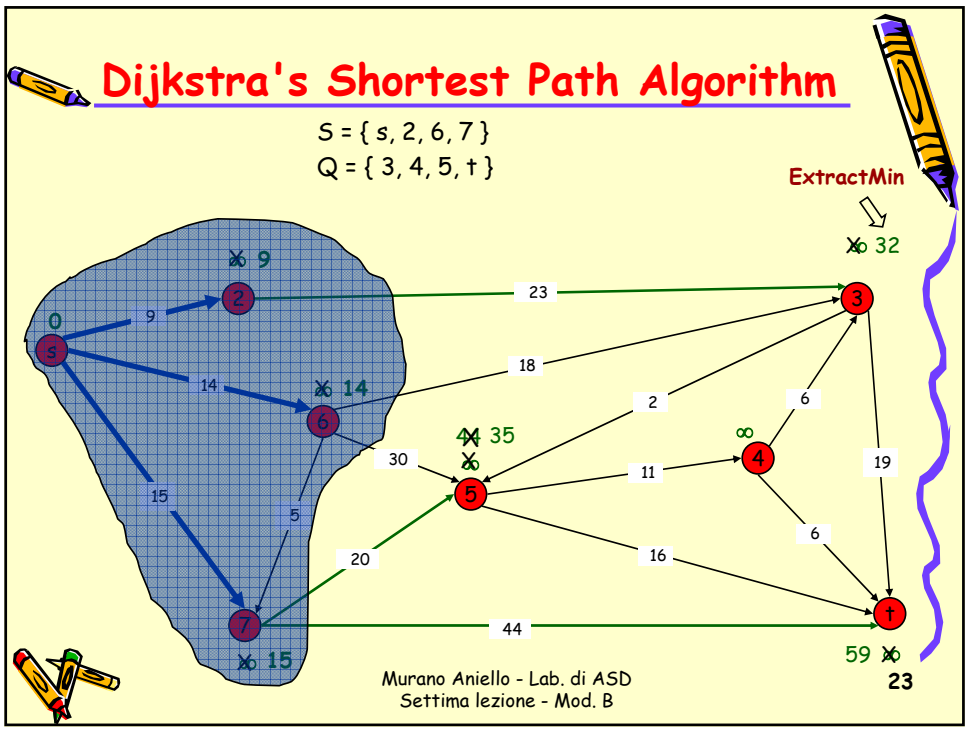

Dijkstra's Shortest Path Algorithm
S = { s, 2 }
Q = { 3, 4, 5, 6, 7, t }
decrease key
∞ 32
∞ 9
0
s
9
2
23
3
14
18
∞ 14
6
2
6
30
∞
5
∞
4
11
19
15
5
6
20
16
7
44
t
∞ 15
∞
18
Murano Aniello - Lab. di ASD
Settima lezione - Mod. B

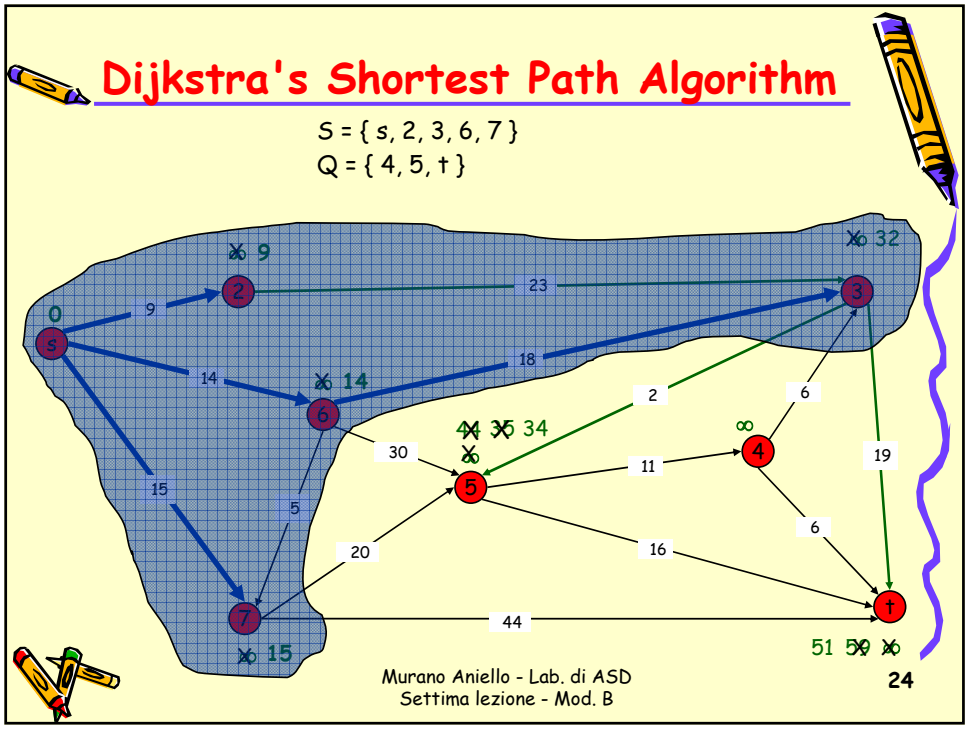

Dijkstra's Shortest Path Algorithm
S = { s, 2 }
Q = { 3, 4, 5, 6, 7, t }
ExtractMin()
9
32
14
15
19
Murano Aniello - Lab. di ASD
Settima lezione - Mod. B

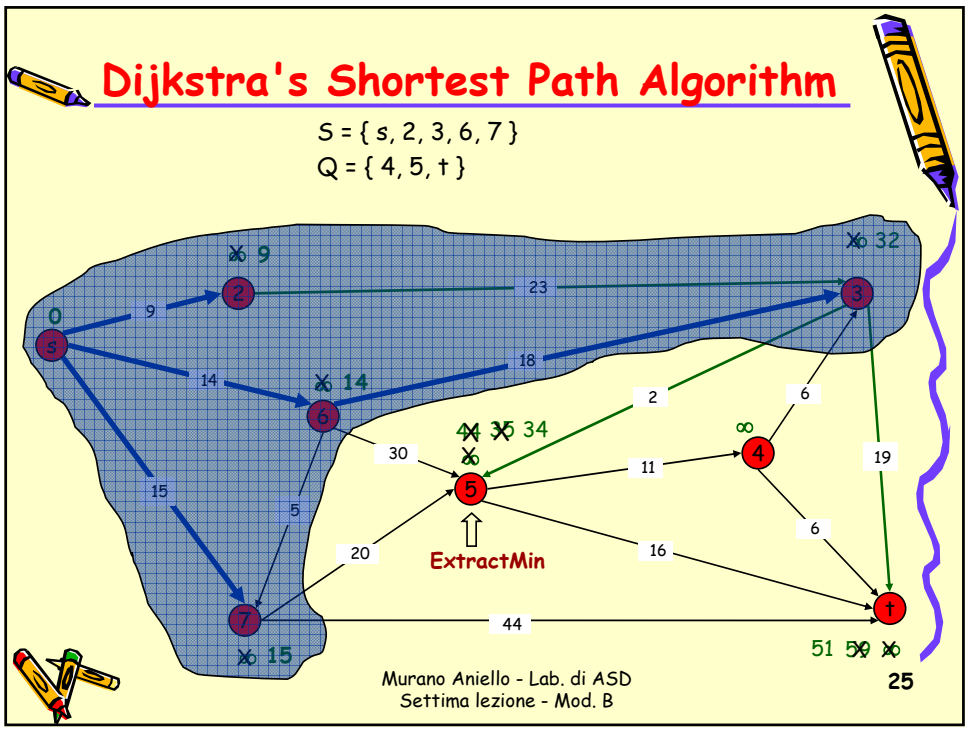

Dijkstra's Shortest Path Algorithm
S = { s, 2, 6 }
Q = { 3, 4, 5, 7, t }
9
32
14
44
15
20
Murano Aniello - Lab. di ASD
Settima lezione - Mod. B

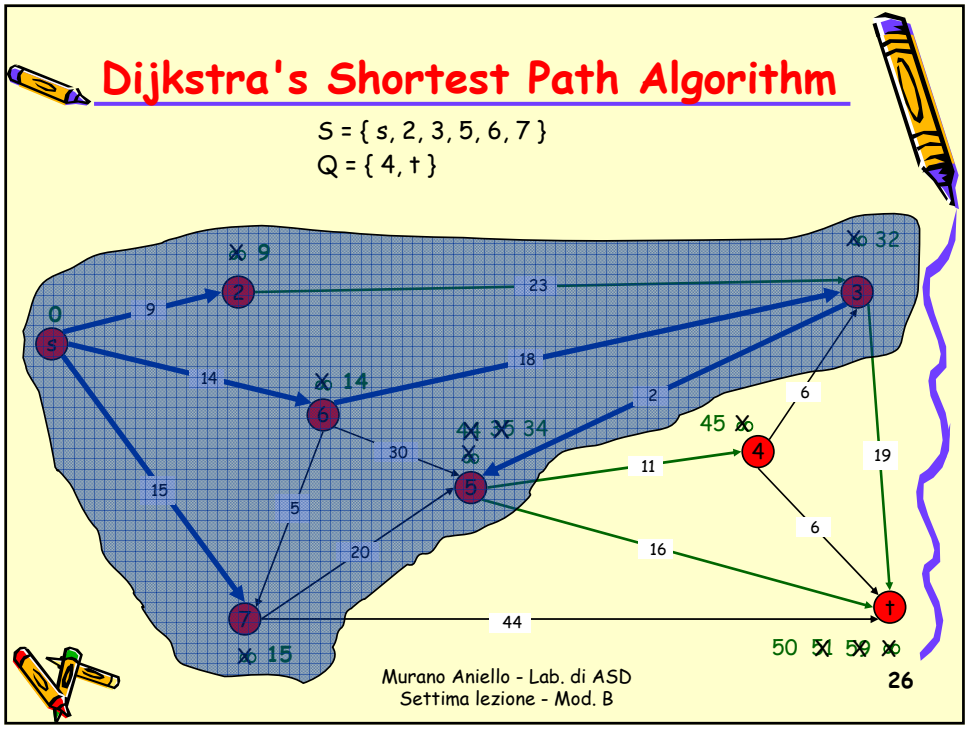

Dijkstra's Shortest Path Algorithm
S = { s, 2, 6 }
Q = { 3, 4, 5, 7, t }
9
32
14
44
15
ExtractMin()
21
Murano Aniello - Lab. di ASD
Settima lezione - Mod. B

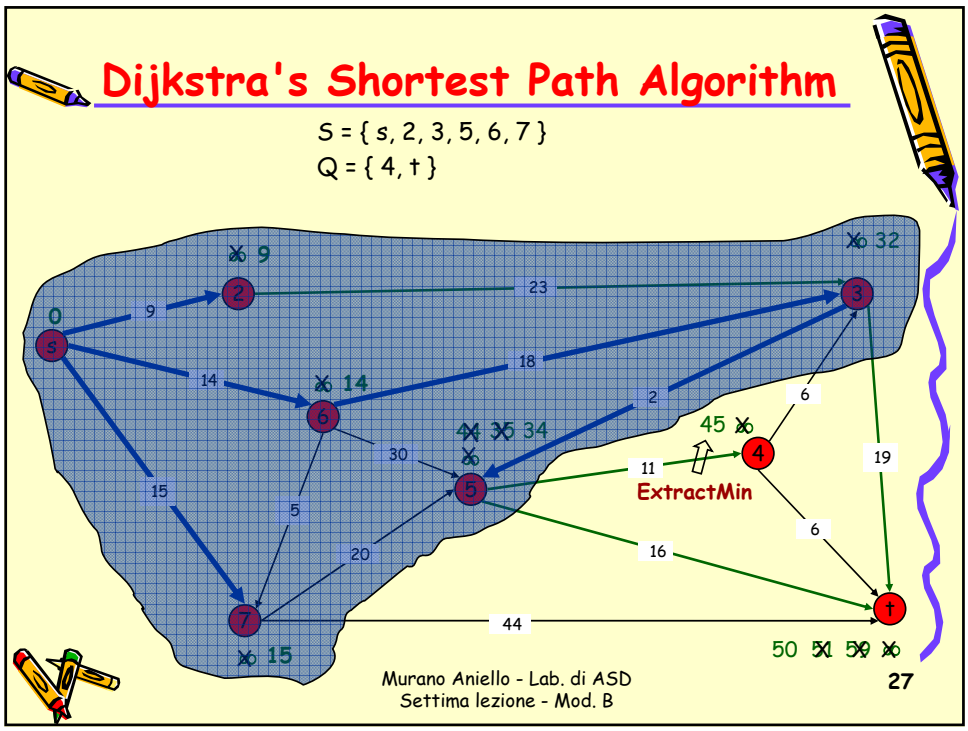

Dijkstra's Shortest Path Algorithm
S = { s, 2, 6, 7 }
Q = { 3, 4, 5, t }
9
32
14
35
15
59
22
Murano Aniello - Lab. di ASD
Settima lezione - Mod. B

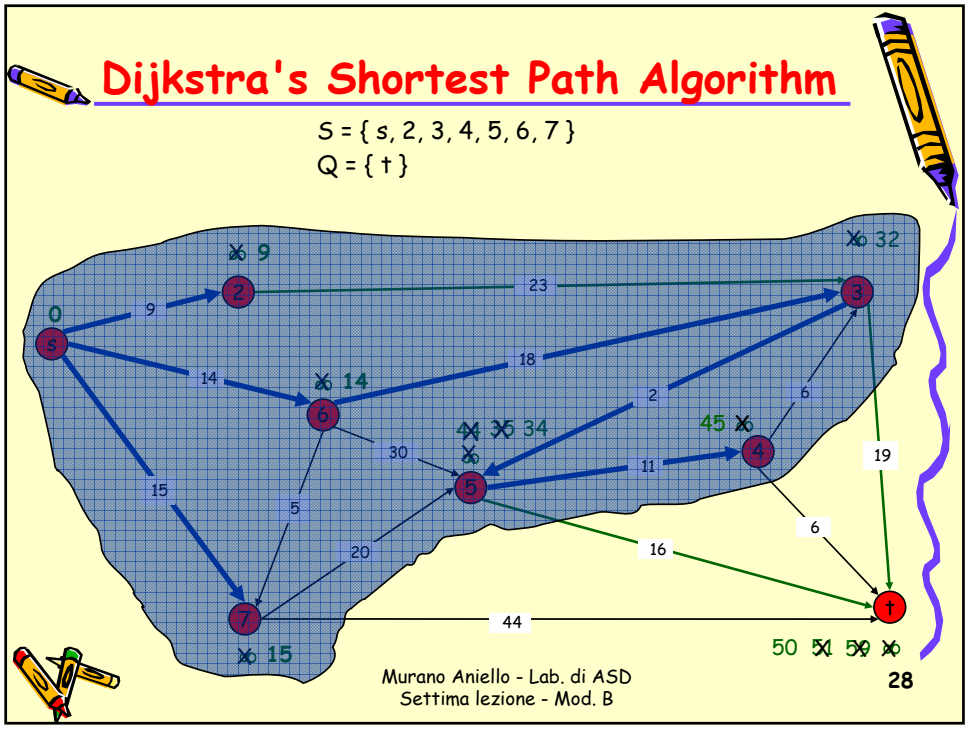

Dijkstra's Shortest Path Algorithm
S = { s, 2, 6, 7 }
Q = { 3, 4, 5, t }
ExtractMin
∞ 32
∞ 9
∞ 14
44 35
59 ∞
∞ 15
0
9
14
15
5
23
18
2
6
30
11
∞
19
20
16
44
Murano Aniello - Lab. di ASD
Settima lezione - Mod. B
23

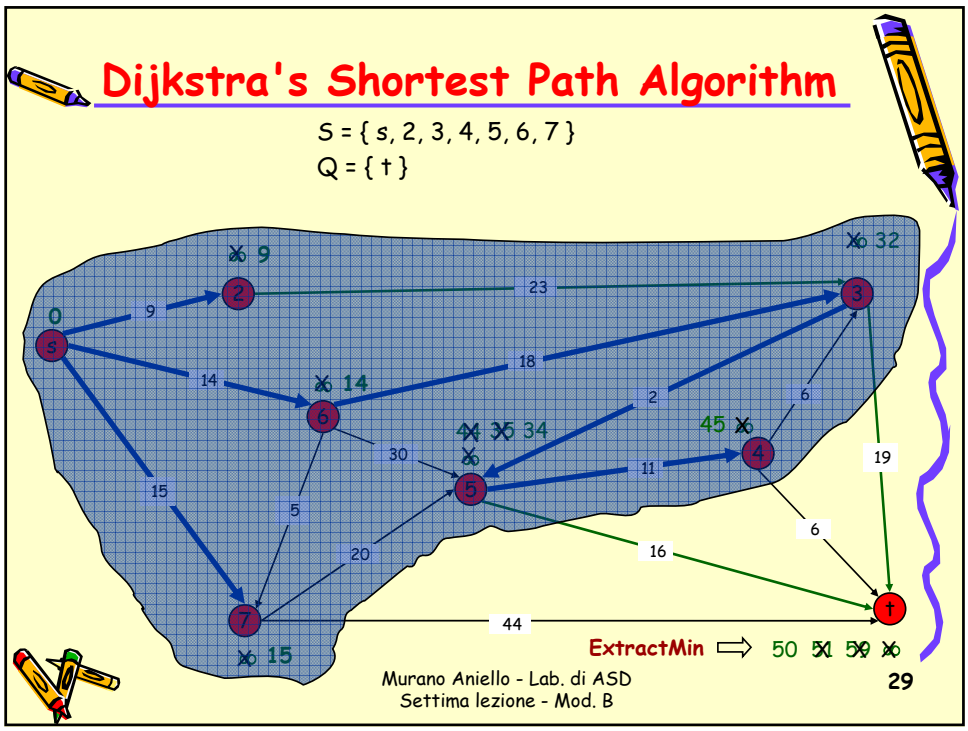

Dijkstra's Shortest Path Algorithm
S = { s, 2, 3, 6, 7 }
Q = { 4, 5, t }
∞ 32
∞ 9
∞ 14
44 35 34
51 50 ∞
∞ 15
0
9
14
15
5
23
18
2
6
30
11
∞
19
20
16
44
Murano Aniello - Lab. di ASD
Settima lezione - Mod. B
24

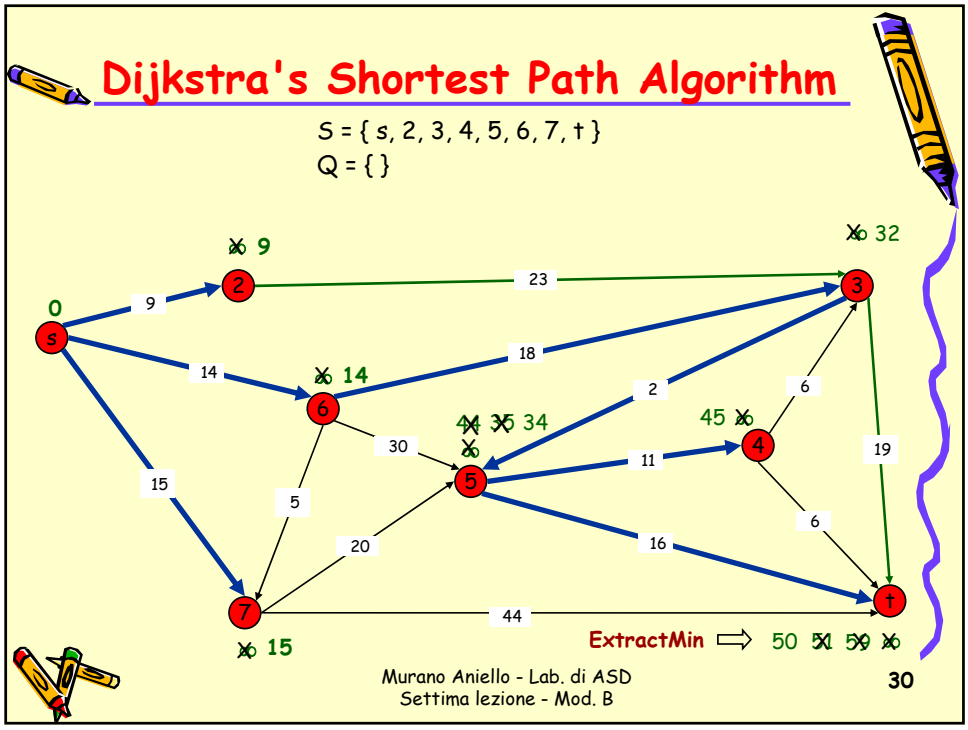

Dijkstra's Shortest Path Algorithm
S = { s, 2, 3, 6, 7 }
Q = { 4, 5, t }
0
s
9
2
23
3
14
6
18
2
6
30
5
11
4
19
15
5
20
ExtractMin
16
6
7
44
t
Murano Aniello - Lab. di ASD
Settima lezione - Mod. B
25


Dijkstra's Shortest Path Algorithm
S = { s, 2, 3, 5, 6, 7 }
Q = { 4, t }
0
s
9
2
23
3
14
6
18
2
6
30
5
11
4
19
15
5
20
16
6
7
44
t
Murano Aniello - Lab. di ASD
Settima lezione - Mod. B
26


Dijkstra's Shortest Path Algorithm
S = { s, 2, 3, 5, 6, 7 }
Q = { 4, t }
ExtractMin
Murano Aniello - Lab. di ASD
Settima lezione - Mod. B
27

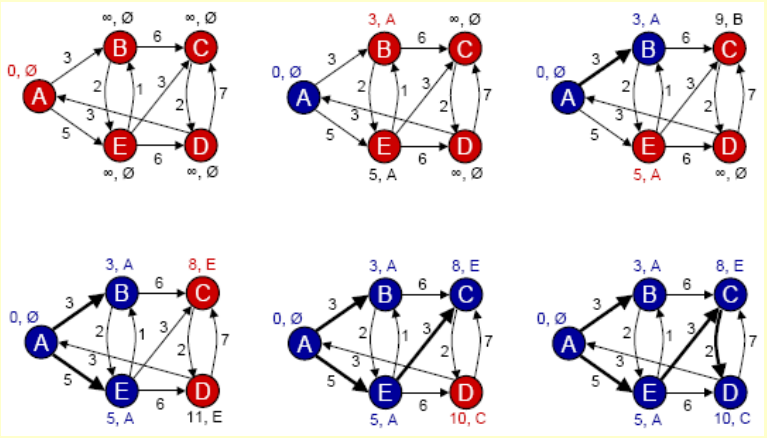

Dijkstra's Shortest Path Algorithm
S = { s, 2, 3, 4, 5, 6, 7 }
Q = { t }
Murano Aniello - Lab. di ASD
Settima lezione - Mod. B
28


Dijkstra's Shortest Path Algorithm
S = { s, 2, 3, 4, 5, 6, 7 }
Q = { t }
0
9
32
14
34
45
15
44
ExtractMin
Murano Aniello - Lab. di ASD
Settima lezione - Mod. B
29


Dijkstra's Shortest Path Algorithm
S = { s, 2, 3, 4, 5, 6, 7, t }
Q = { }
0
9
32
14
34
45
15
44
ExtractMin
Murano Aniello - Lab. di ASD
Settima lezione - Mod. B
30

Un altro esempio

Supponiamo di voler calcolare il cammino minimo da A a D



Complessità (1/2)

1. INIZIALIZE-SINGLE-SOURCE(G ,s)
2. S ← Ø // Inizializzazione: θ(V) //
3. Q ← V[G] // Per costruire la coda a priorità: θ(V) //
4. **while** Q ≠ Ø // eseguito |V| volte //
5. **do** u ← EXTRACT-MIN(Q)
6. S ← S unito {u}
7. **for** ogni vertice v di Adj[u] // |E| volte //
8. **do** RELAX(u, v)

- Ciclo "while" eseguito |V| volte
- |V| chiamate a EXTRACT-MIN
- ciclo interno su archi fatto |E| volte
- Al più |E| chiamate a Relax
- Tempo totale:
- $\Theta(V + V \times T_{EXTRACT\text{-}MIN} + E \times T_{RELAX})$
- Dunque, la complessità dipende molto da come è implementata la coda di priorità

Complessità (2/2)

- Usando un array non ordinato per implementare la coda:
- EXTRACT-MIN in tempo Θ(n), Relax in Θ(1)
- Tempo totale: $\Theta(V + V V + E) = \Theta(V^2)$
- In un grafo non fortemtente conesso conviene usare un heap binario invece di una coda di priorità
- Usando un heap, la complessità diventa: θ((V+E) logV)
  - Per costruire un heap: θ (V)
  - ExtractMin prende tempo θ(lgV) (se si pensa ad un heap con minimo nella radice) e questa operazione viene eseguita |V| volte
  - Il costo di relax è O(lgV) e questo viene effettuato |E| volte.