# Shortest paths

24 maggio 2023

4.4 Shortest Paths in a Graph (only with positive costs):
Dijkstra Algorithm: Greedy

# Shortest Path Problem

Shortest path network.

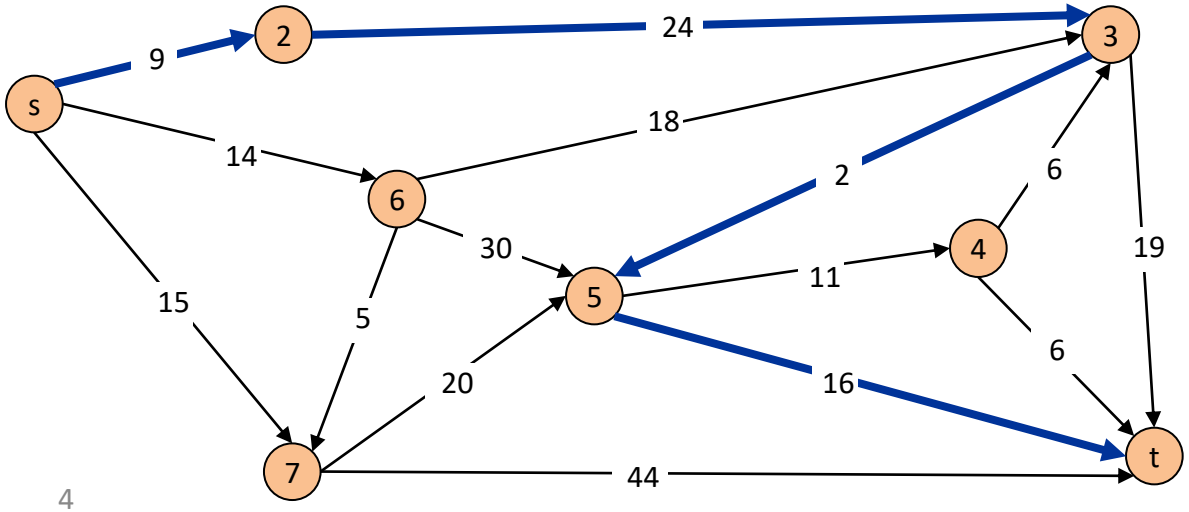
Directed graph G = (V, E).

Source s, destination t.

Length $\lambda_e$ = length/cost/weight of edge e.

Shortest path problem: find shortest (min cost) directed path from s to t.

cost of path = sum of edge costs in path




Cost of path s-2-3-5-t
= 9 + 24 + 2 + 16
= 51.

# Dijkstra's Algorithm (si pronuncia DAISTRA)

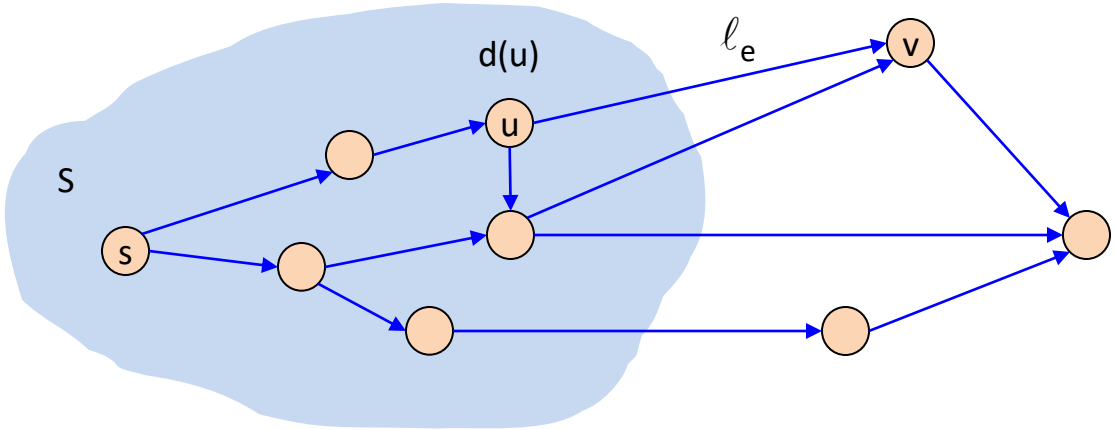
Dijkstra's algorithm (greedy approach).

- Maintain a set of explored nodes S for which we have determined the shortest path distance d(u) from s to u.
- Initialize S = { s }, d(s) = 0
- Repeatedly choose unexplored node v which minimizes

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

shortest path to some u in explored part, followed by a single edge (u, v)

- Add v to S and set d(v) = π(v).



# Implementazione Dijkstra

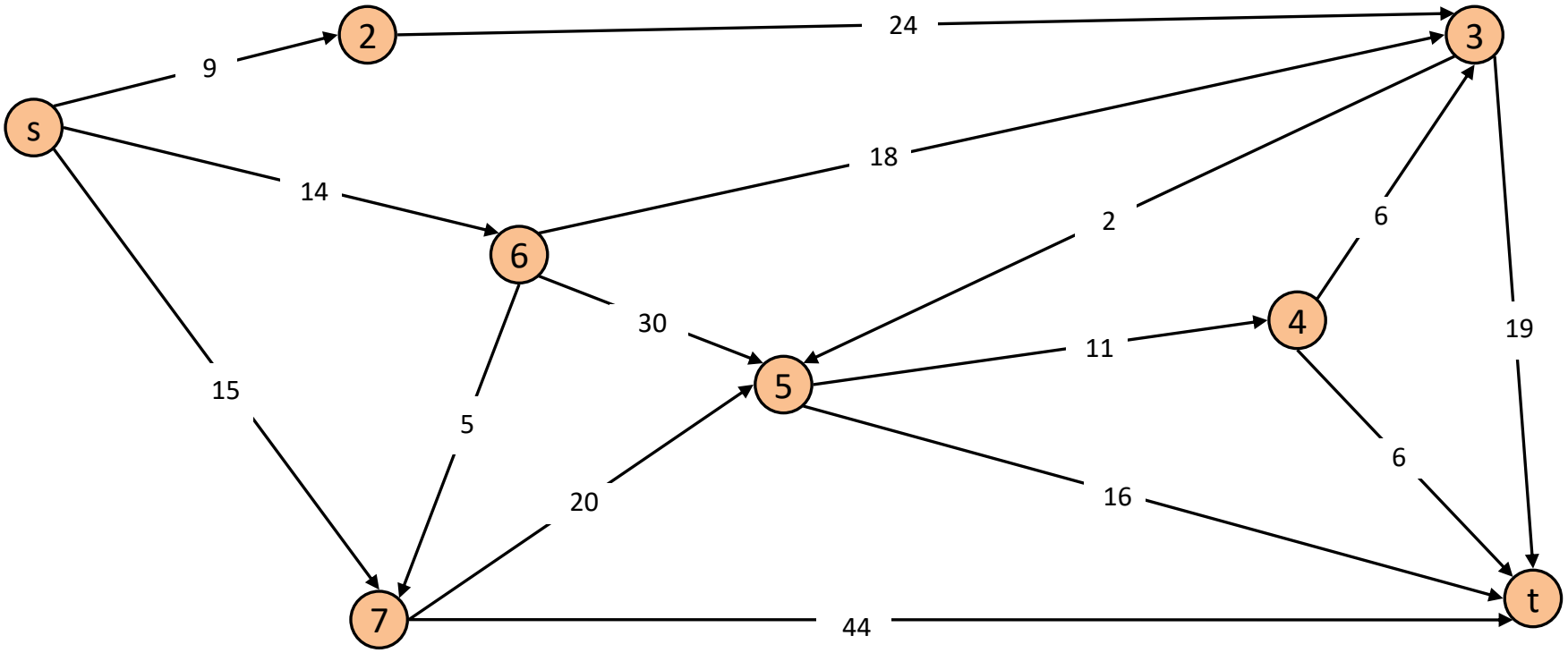
L'implementazione dell'algoritmo di Dijkstra è (molto) simile a quella dell'algoritmo di Prim (riportata qui sotto) ed è lasciata come esercizio.

```
Prim(G,c,s) {
    foreach (v ∈ V) a[v] ← ∞;   a[s]←0
    Initialize an empty priority queue Q
    foreach (v ∈ V) insert v onto Q with priority a[v]
    Initialize set of explored nodes S ← ϕ

    while (Q is not empty) {
        u ← delete min element from Q
        S ← S ∪ { u }
        foreach (edge e = (u, v) incident to u)
            if ((v ∉ S) and (c_e < a[v]))
                decrease priority a[v] to c_e
}
```

# Dijkstra's Shortest Path Algorithm

Find shortest path from s to t.

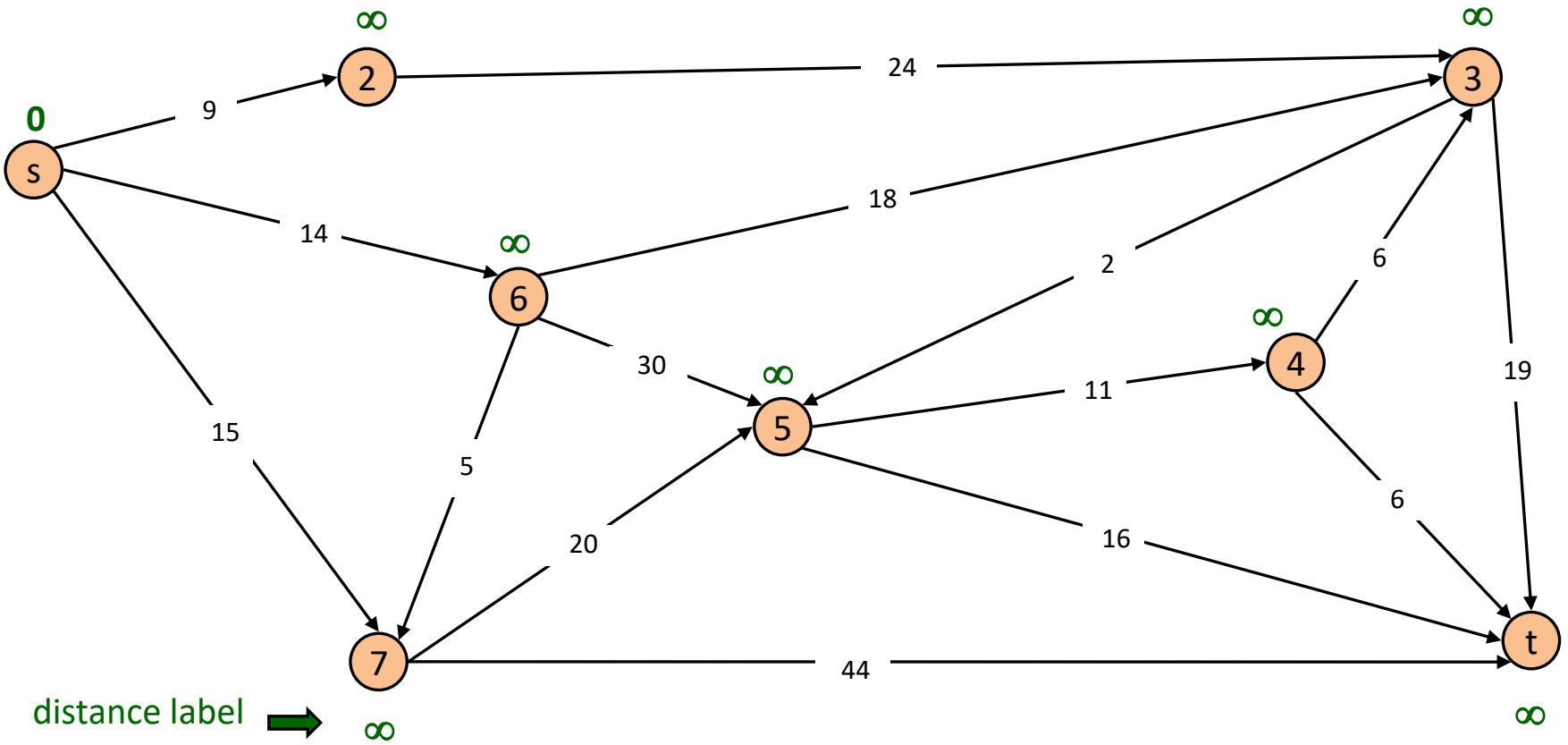


# Dijkstra's Shortest Path Algorithm

S = { }

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

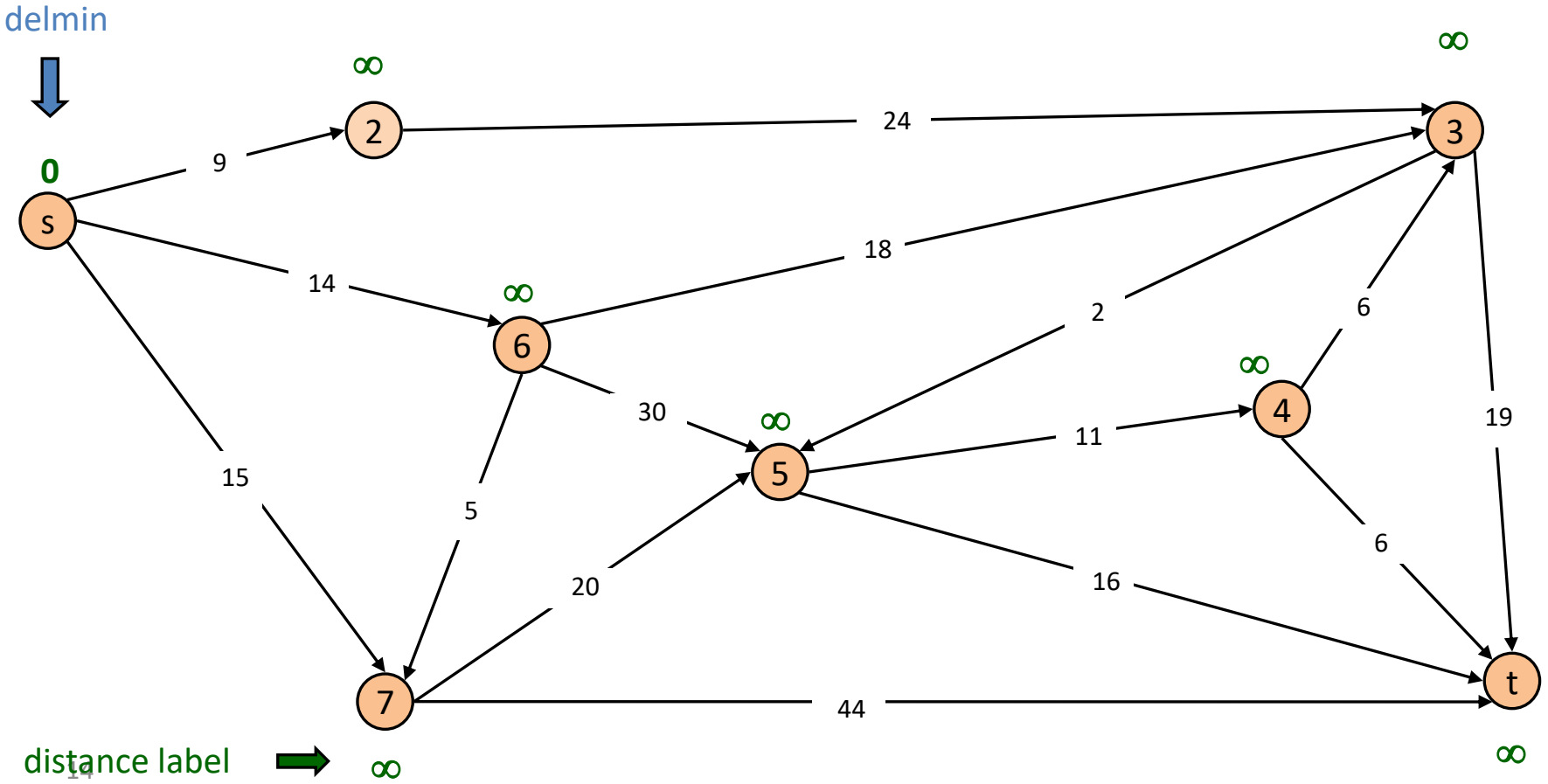
Nota: nell'ottica di una implementazione che mantenga per ogni nodo v non esplorato la priorità π(v) definita prima, useremo una coda a priorità PQ e un insieme S dei nodi esplorati, inizializzati come sopra in modo tale che dopo la prima iterazione si abbia S = { s }, d(s) = 0 come richiesto.



# Dijkstra's Shortest Path Algorithm

S = { }

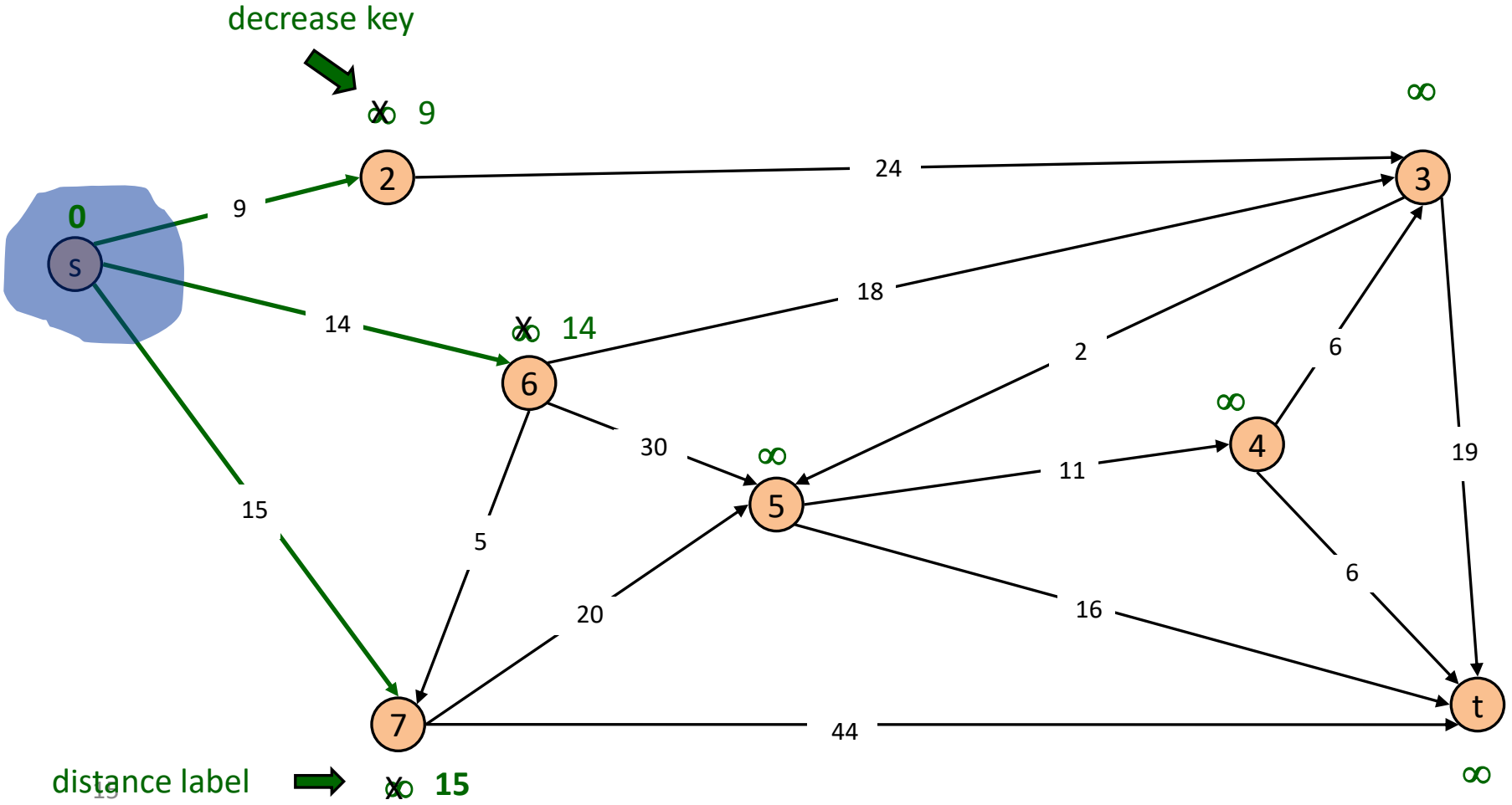
PQ = { (s,0), (2, ∞), (3, ∞), (4, ∞), (5, ∞), (6, ∞), (7, ∞), (t, ∞) }



# Dijkstra's Shortest Path Algorithm

S = { s }

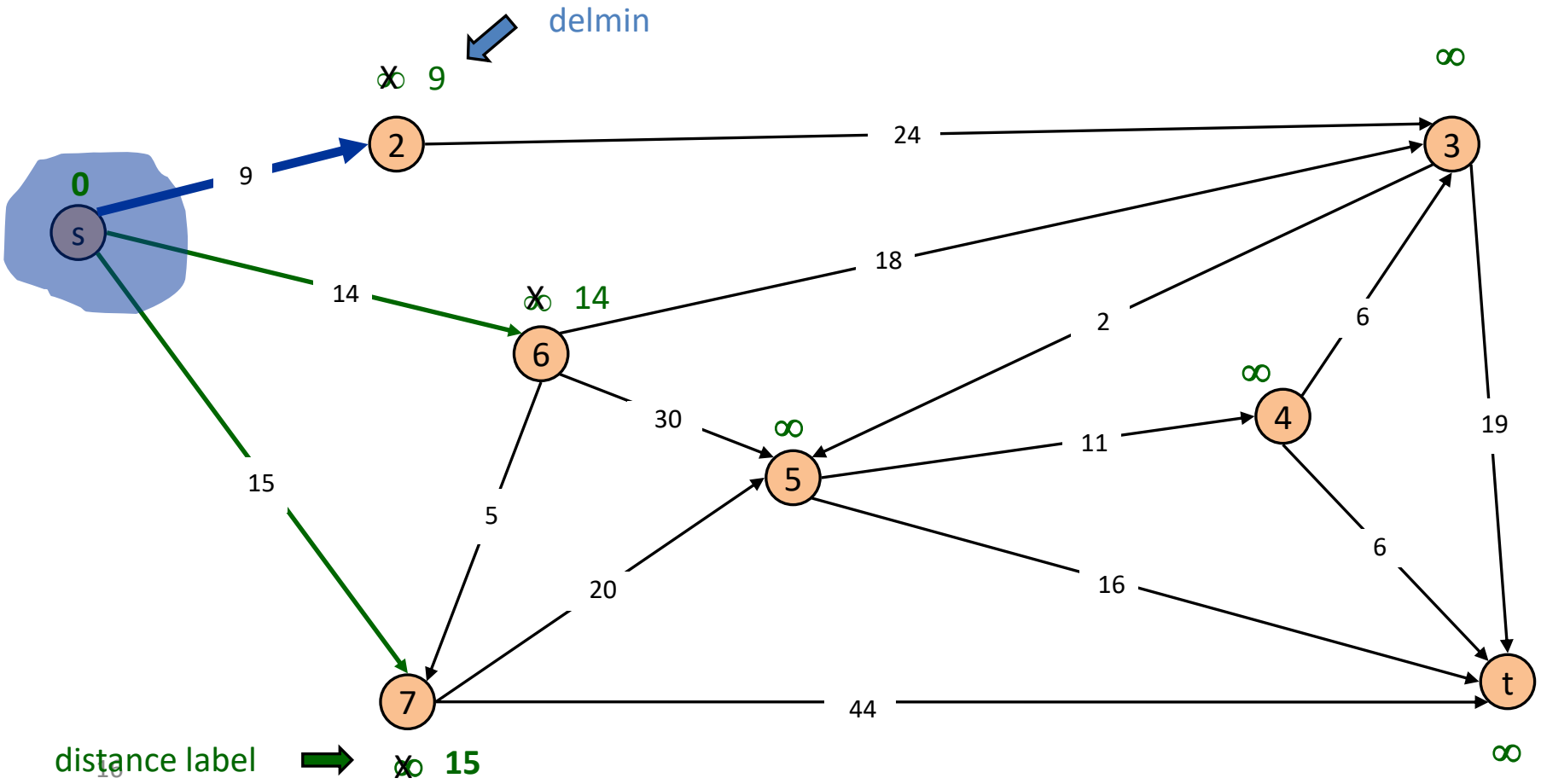
PQ = { (2, 9), (3, ∞), (4, ∞), (5, ∞), (6,14), (7,15), (t, ∞) }



# Dijkstra's Shortest Path Algorithm

S = { s }

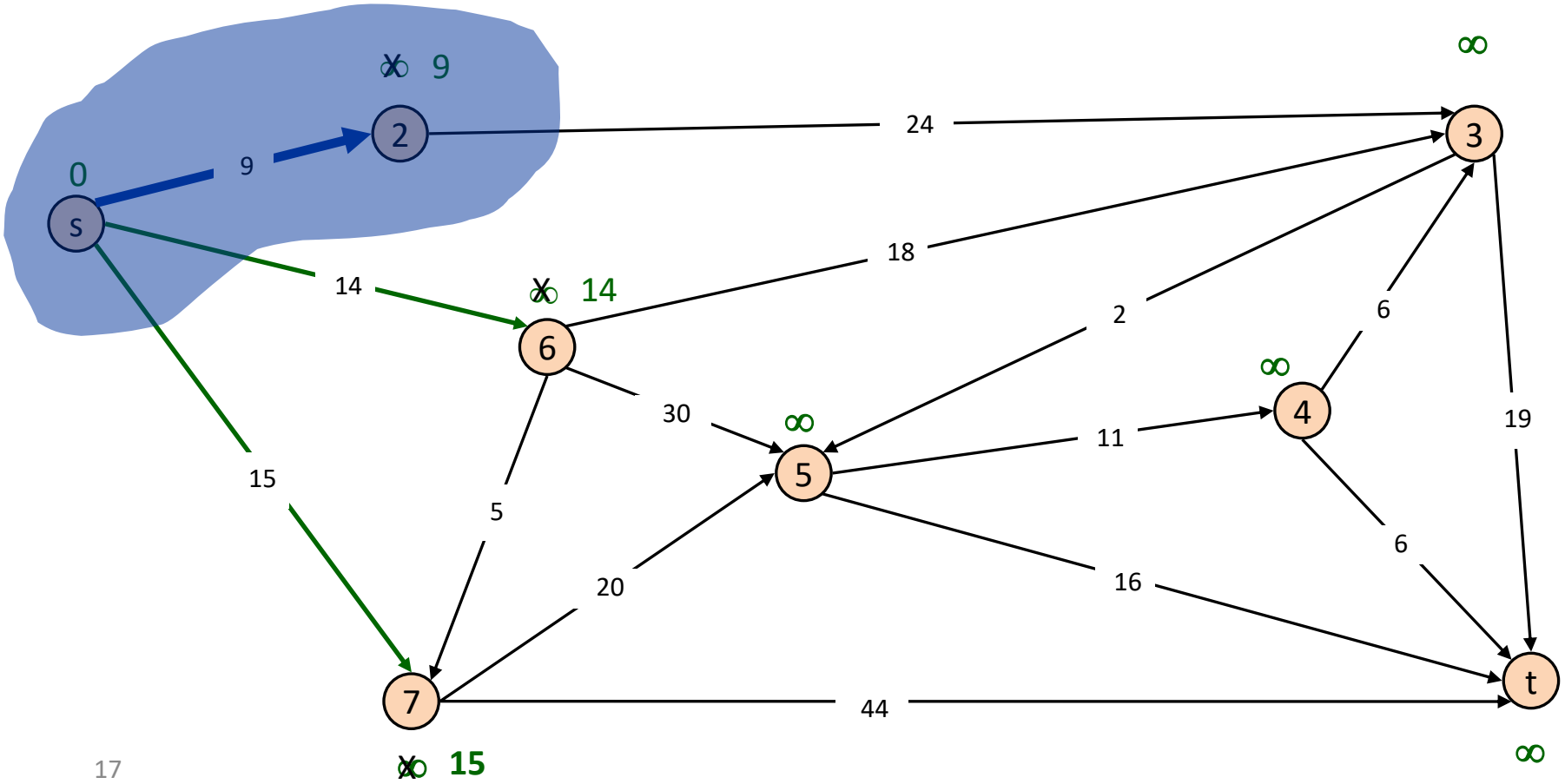
PQ = { (2, 9), (3, ∞), (4, ∞), (5, ∞), (6,14), (7,15), (t, ∞) }



# Dijkstra's Shortest Path Algorithm

S = { s }

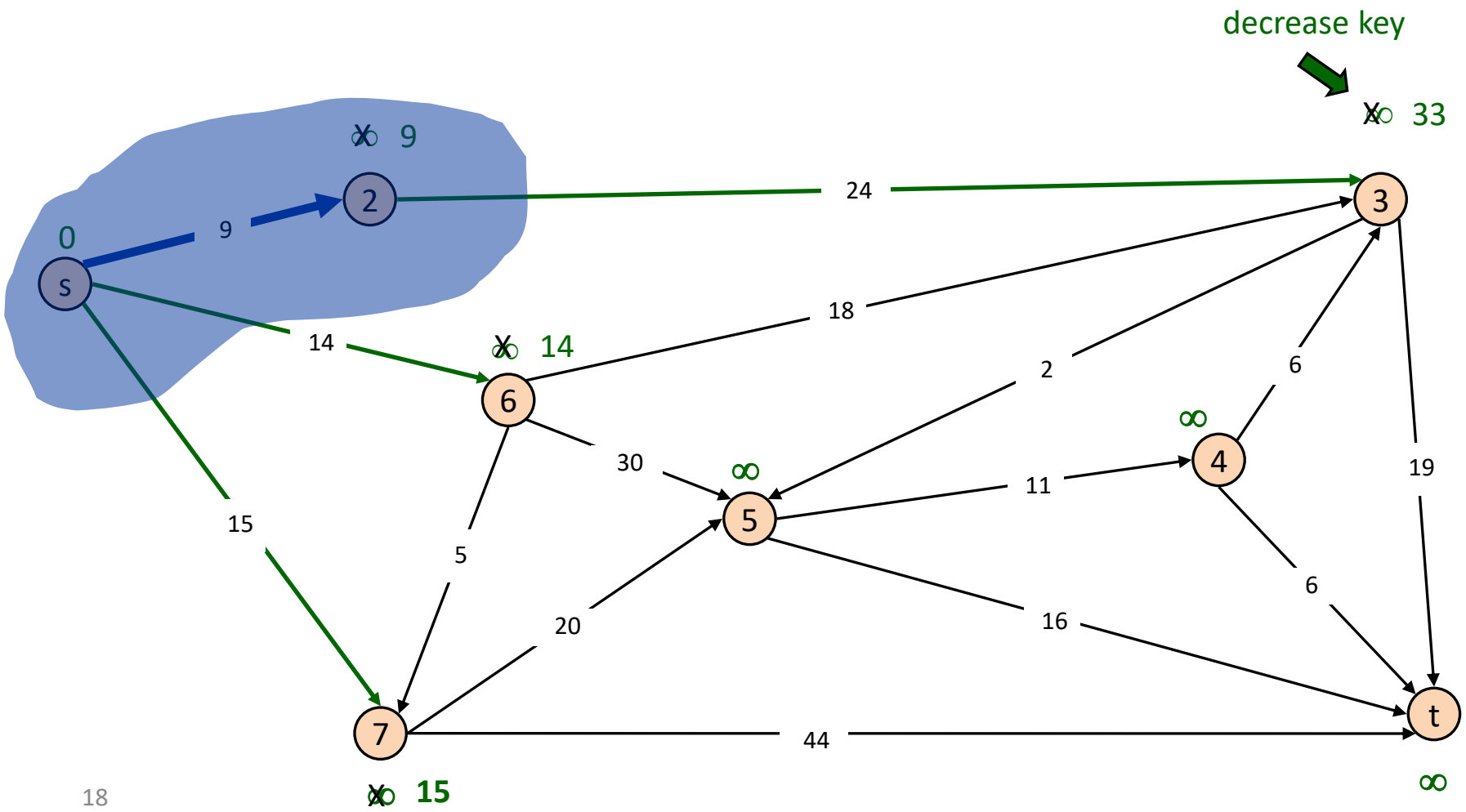
PQ = { (2, 9), (3, ∞), (4, ∞), (5, ∞), (6,14), (7,15), (t, ∞) }



# Dijkstra's Shortest Path Algorithm

S = { s, 2 }

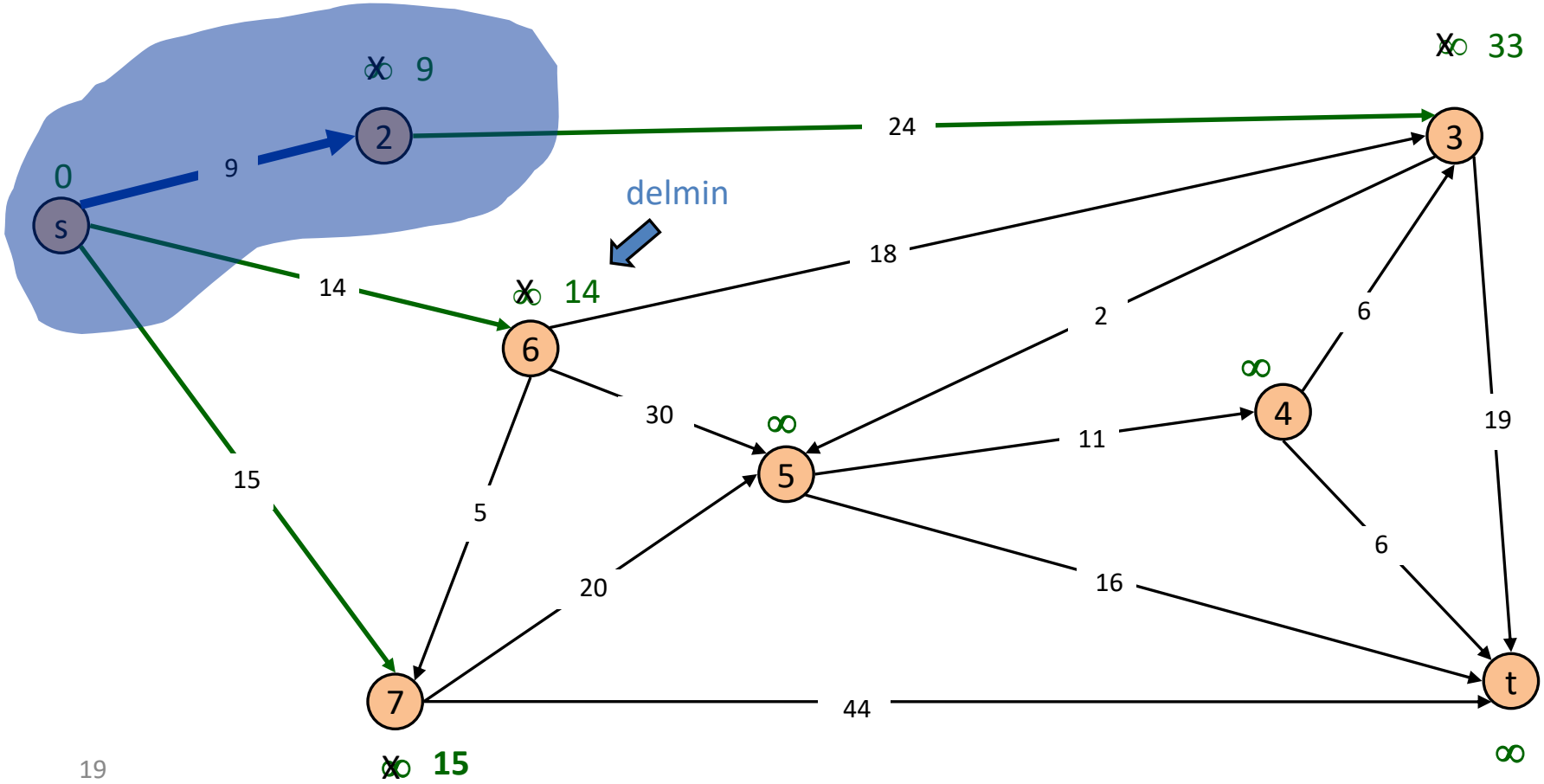
PQ = { (3,33), (4, ∞), (5, ∞), (6,14), (7,15), (t, ∞) }



# Dijkstra's Shortest Path Algorithm

S = { s, 2 }

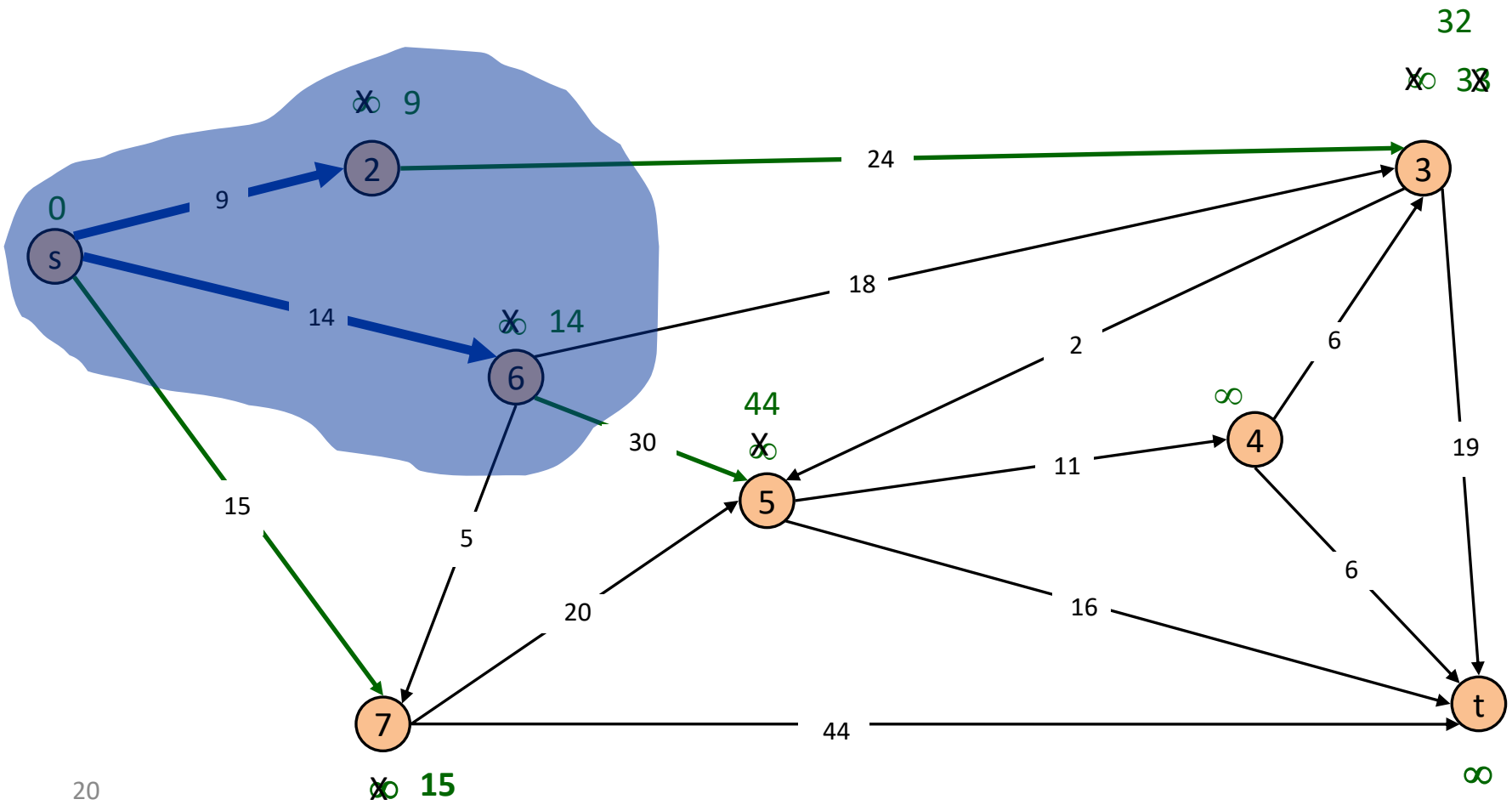
PQ = { (3,33), (4, ∞), (5, ∞), (6,14), (7,15), (t, ∞) }



# Dijkstra's Shortest Path Algorithm

S = { s, 2, 6 }

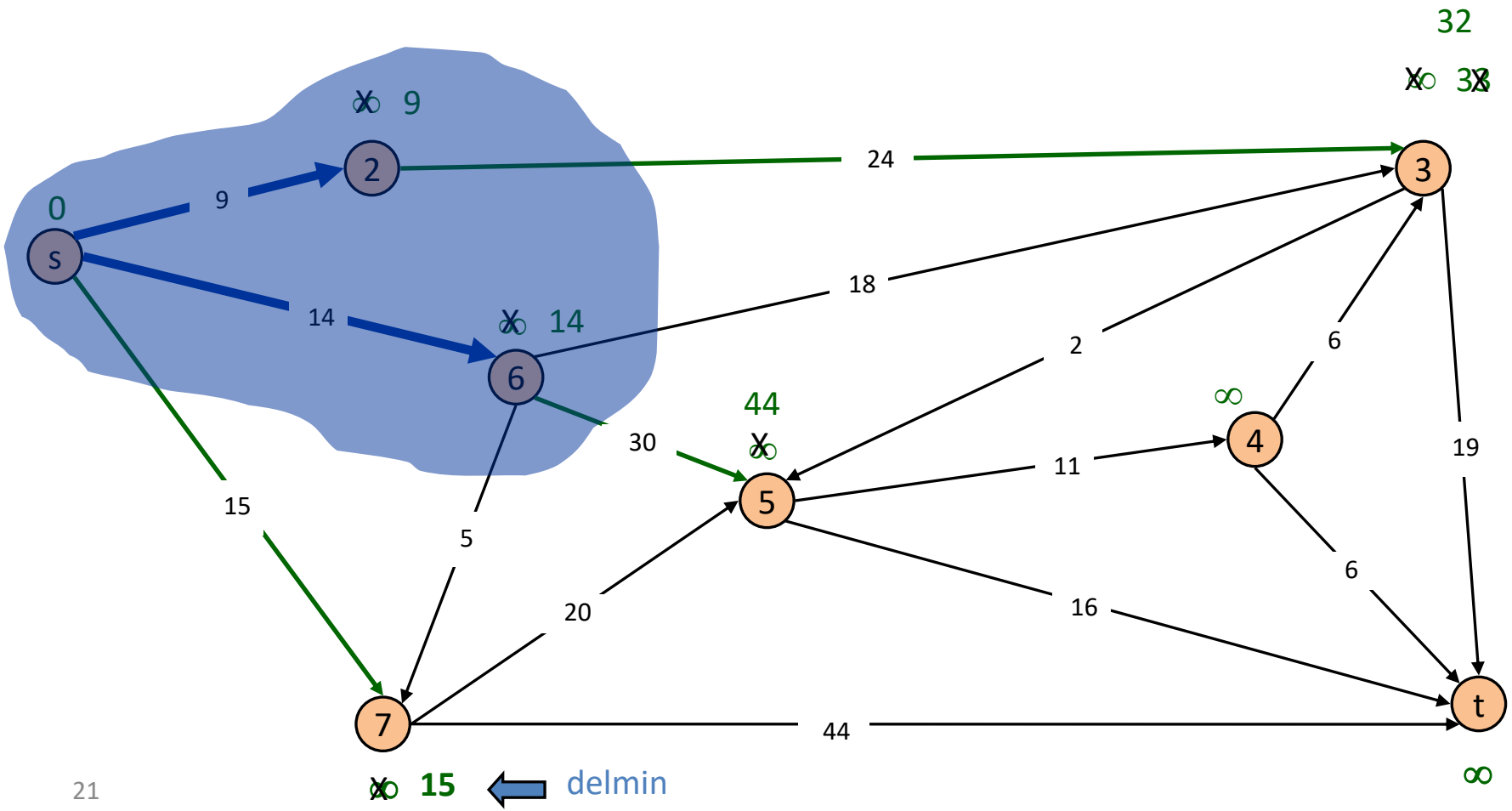
PQ = { (3,32), (4, ∞), (5,44), (7,15), (t, ∞) }



# Dijkstra's Shortest Path Algorithm

S = { s, 2, 6 }

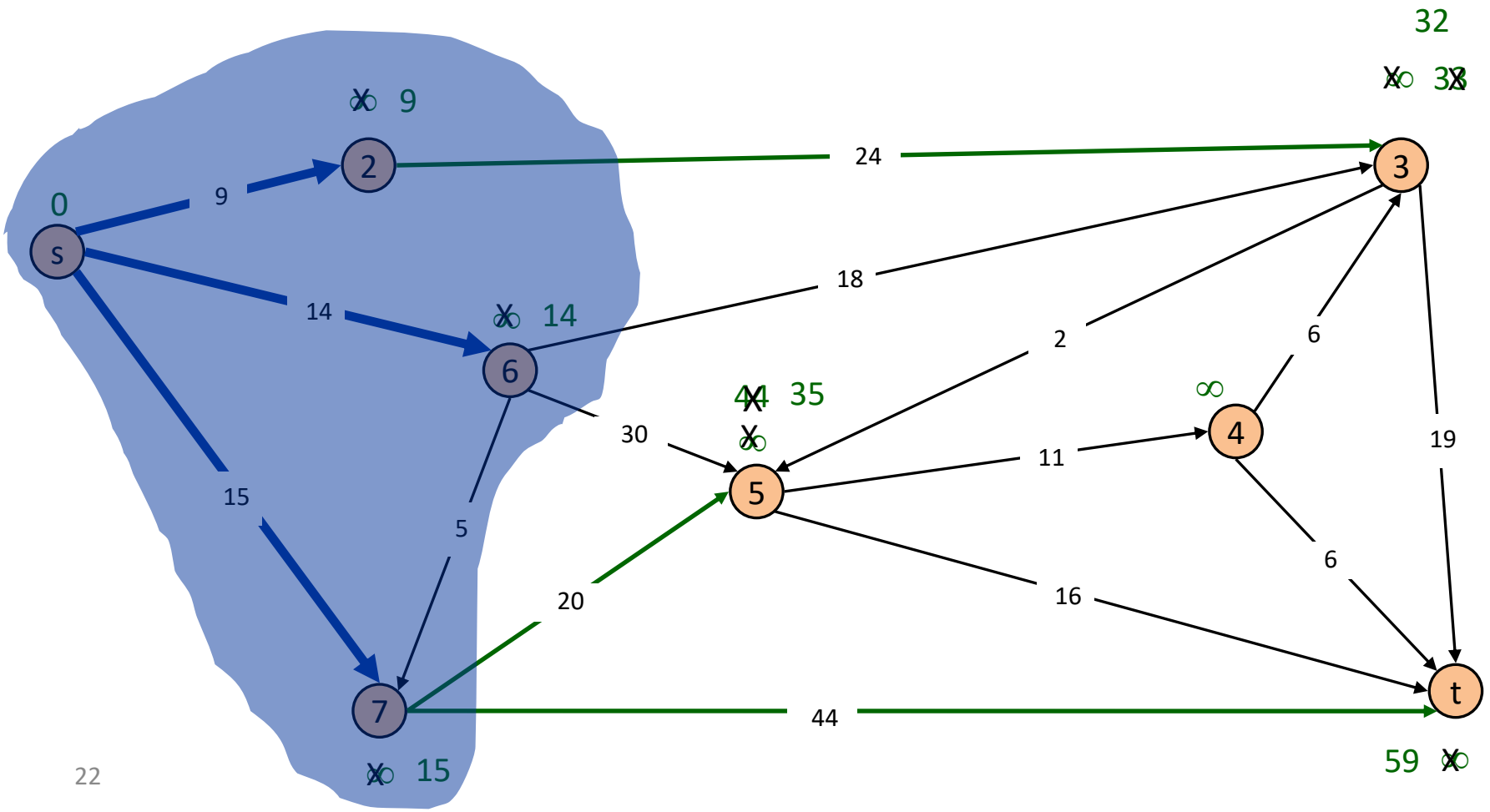
PQ = { (3,32), (4, ∞), (5,44), (7,15), (t, ∞) }



# Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7 }

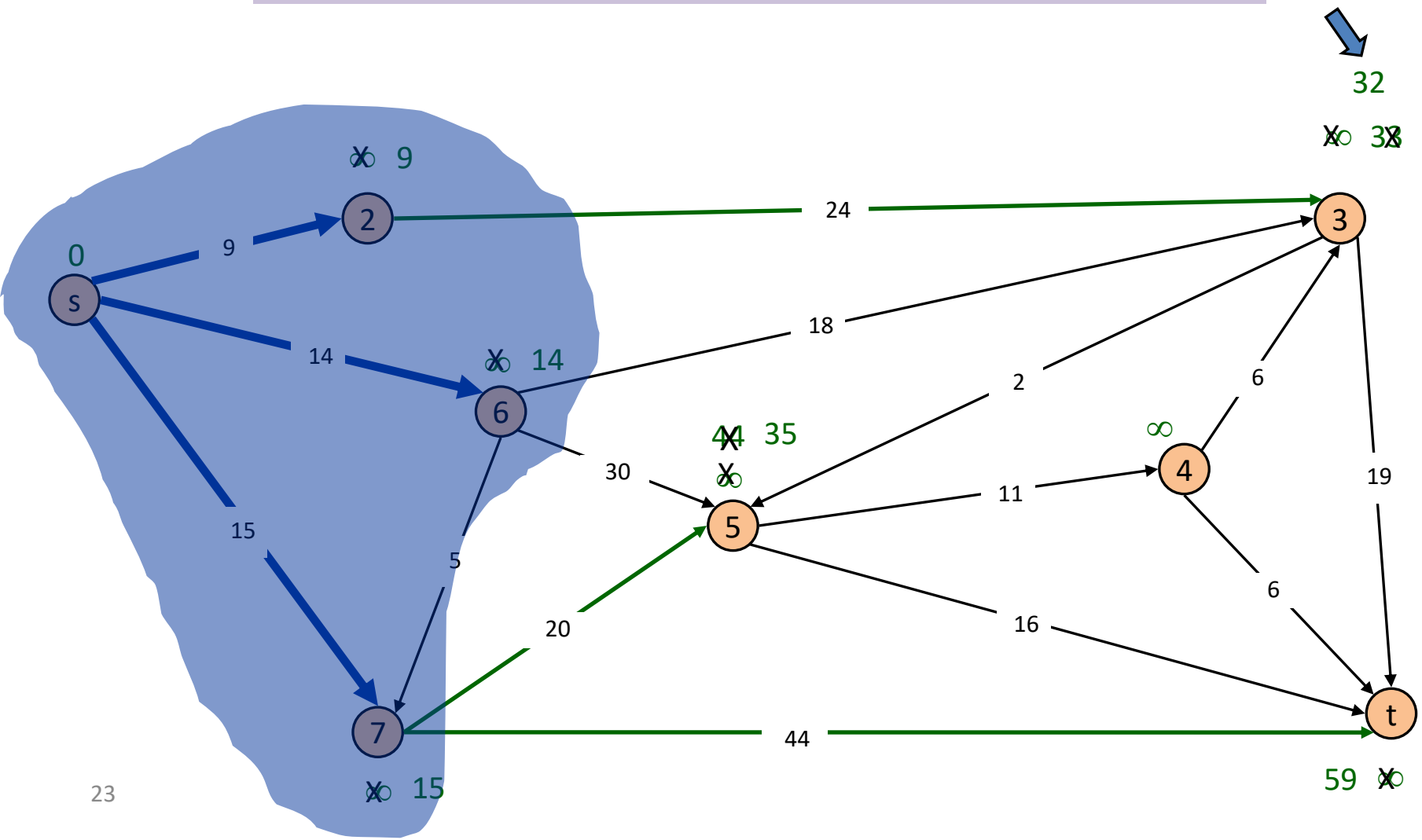
PQ = { (3,32), (4, ∞), (5,35), (t, 59) }



# Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7 }

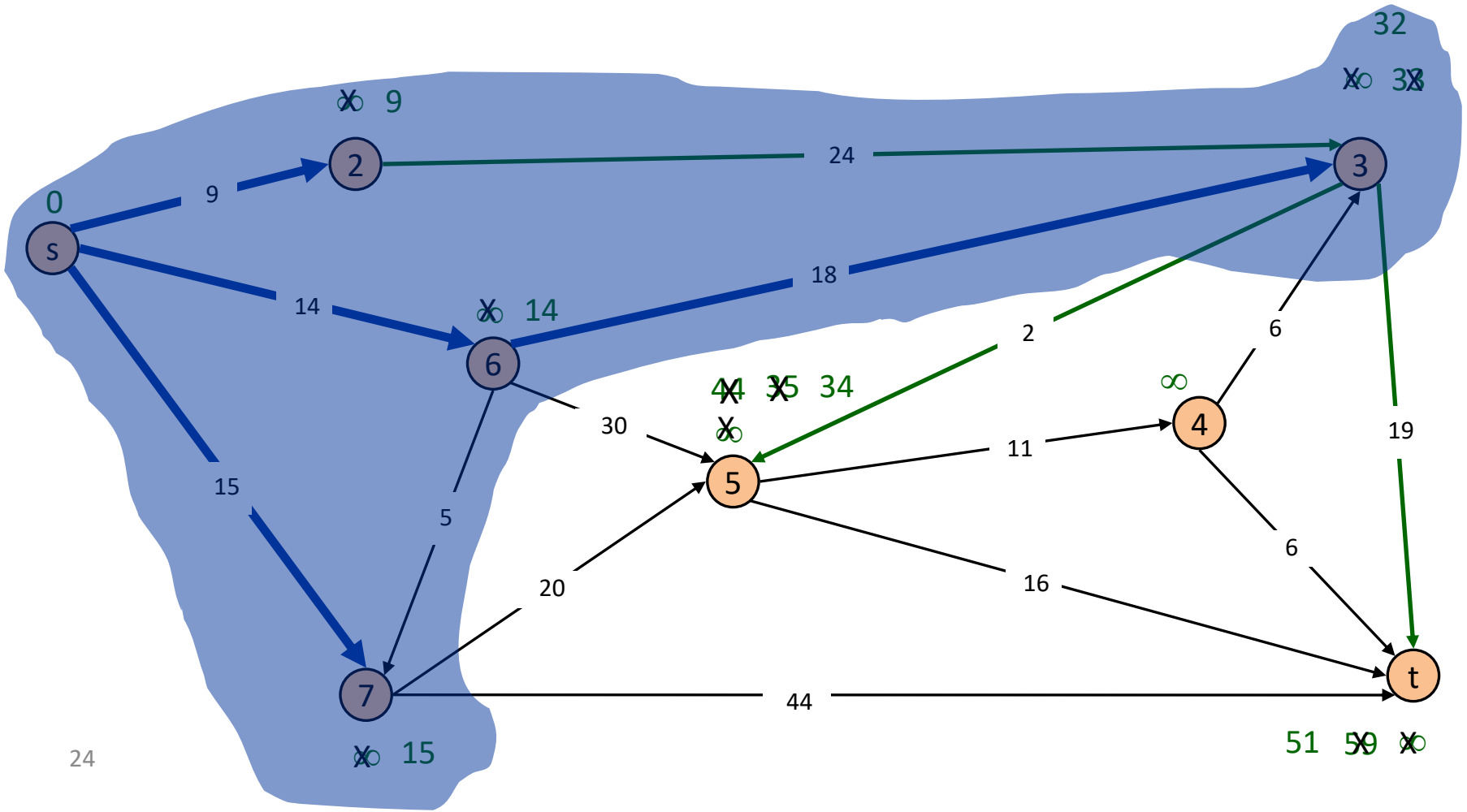
PQ = { (3,32), (4, ∞), (5,35), (t, 59) }



# Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7, 3}

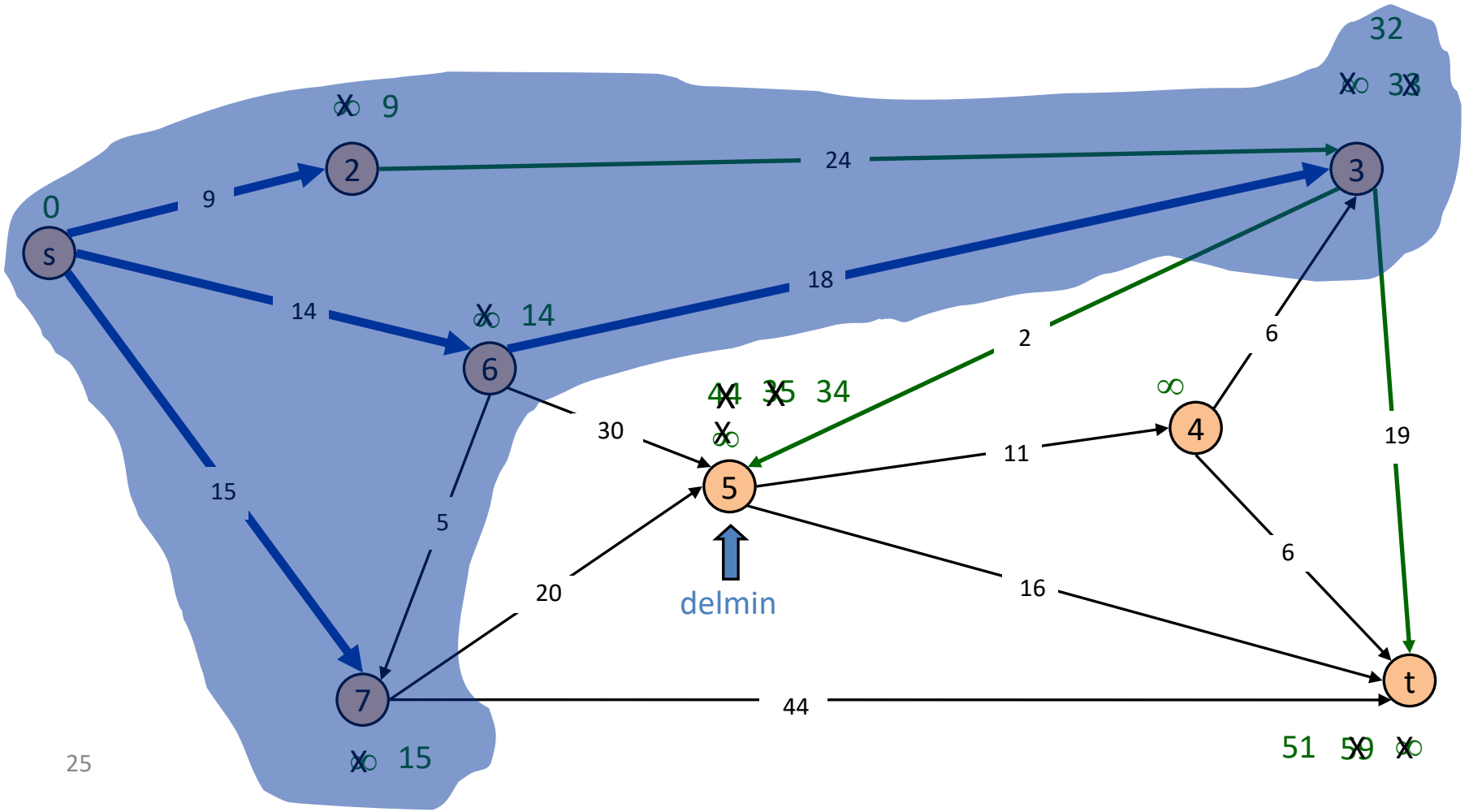
PQ = { (4, ∞), (5,34), (t, 51) }



# Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7, 3}

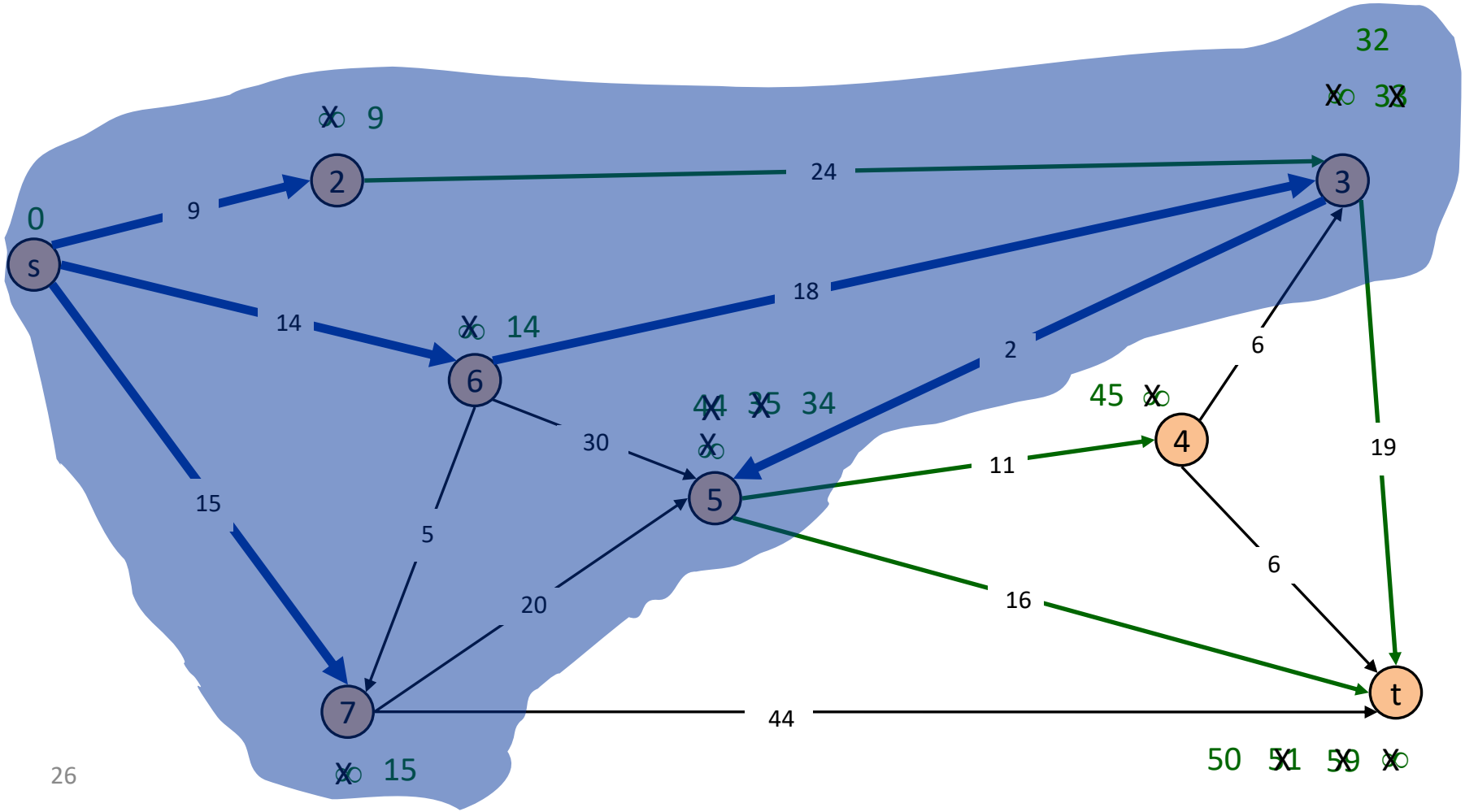
PQ = { (4, ∞), (5,34), (t, 51) }



# Dijkstra's Shortest Path Algorithm

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

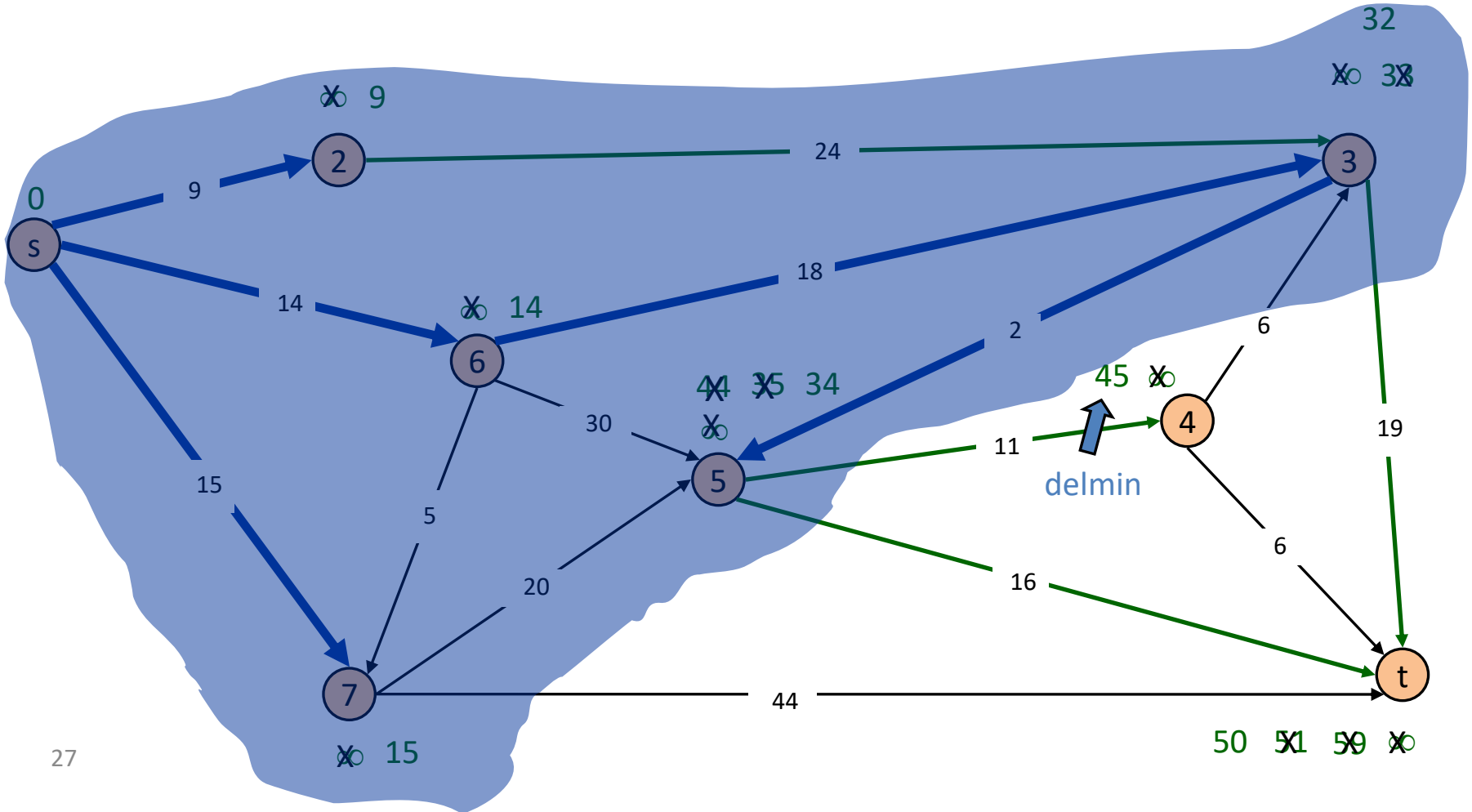
PQ = { (4, 45), (t, 50) }



# Dijkstra's Shortest Path Algorithm

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

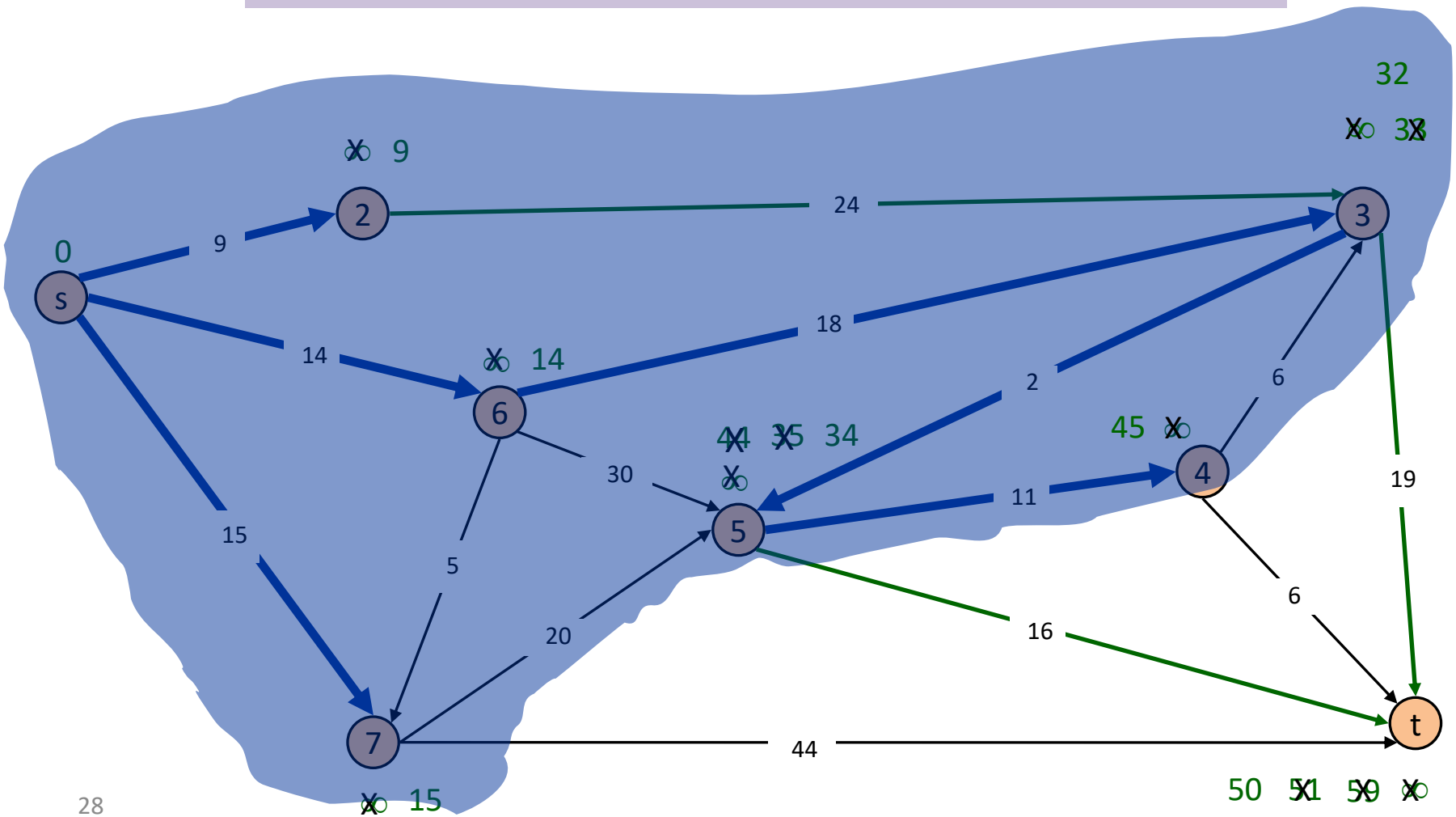
PQ = { (4, 45), (t, 50) }



# Dijkstra's Shortest Path Algorithm

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

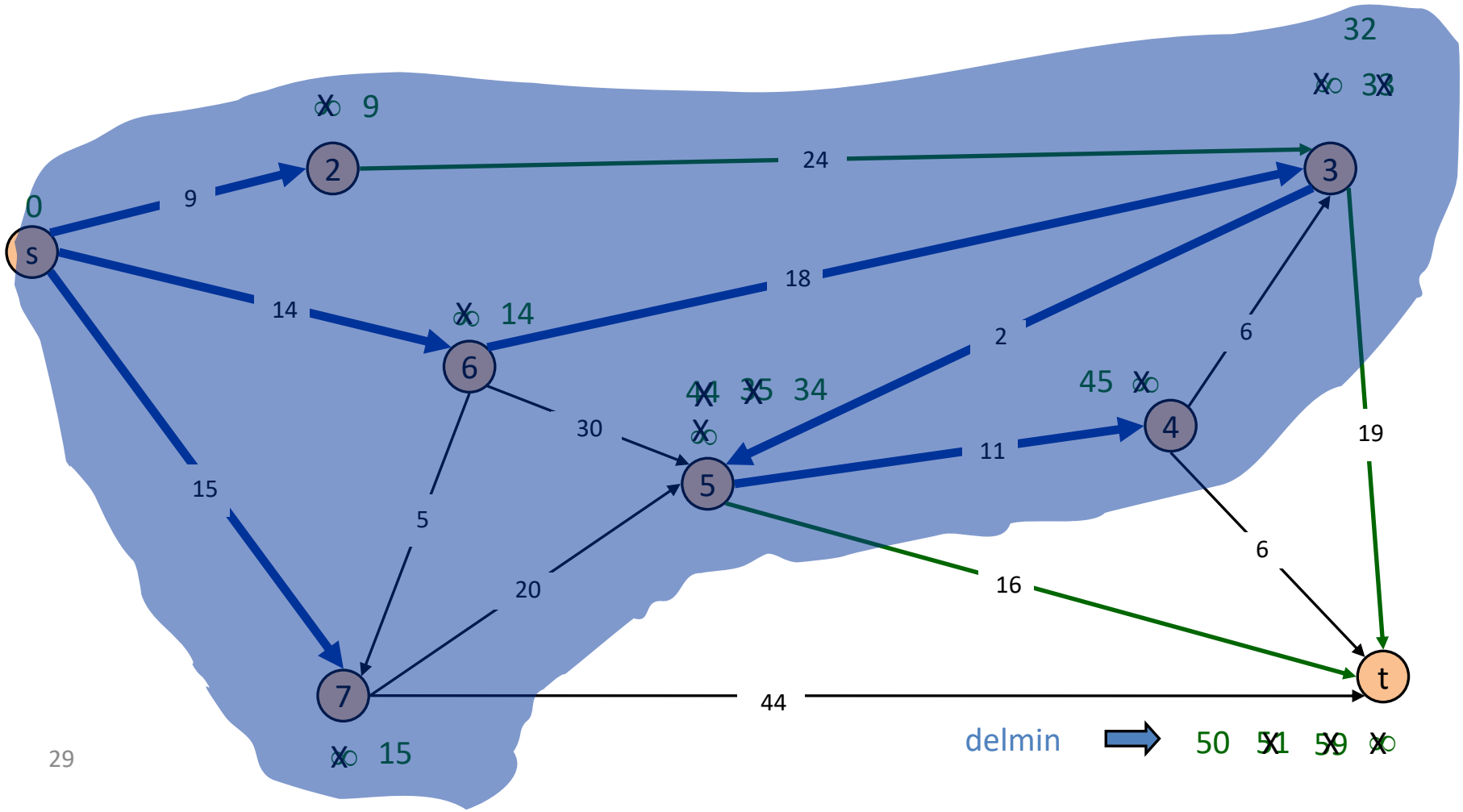
PQ = { (4, 45), (t, 50) }



# Dijkstra's Shortest Path Algorithm

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

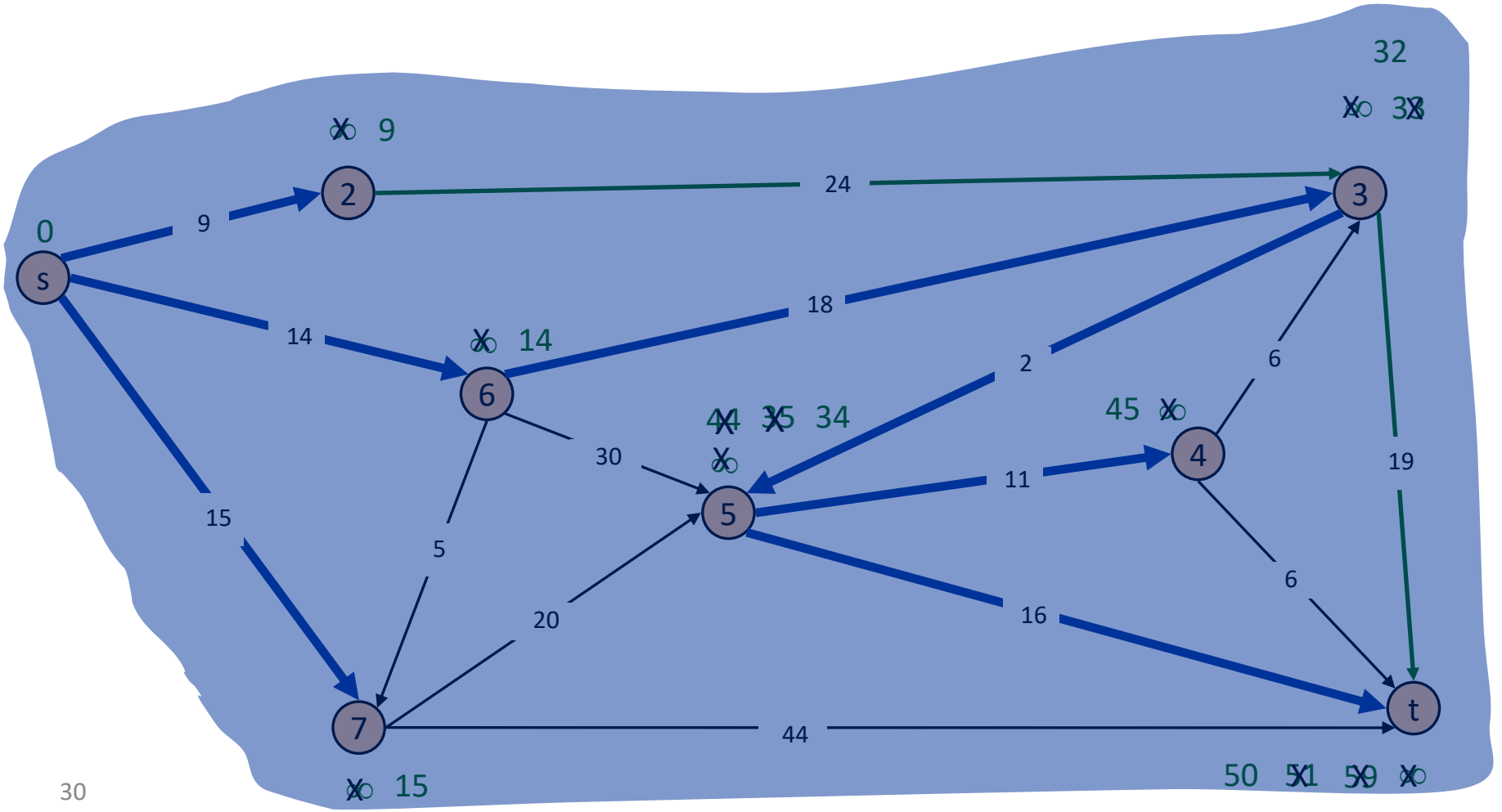
PQ = { (t, 50) }



# Dijkstra's Shortest Path Algorithm

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

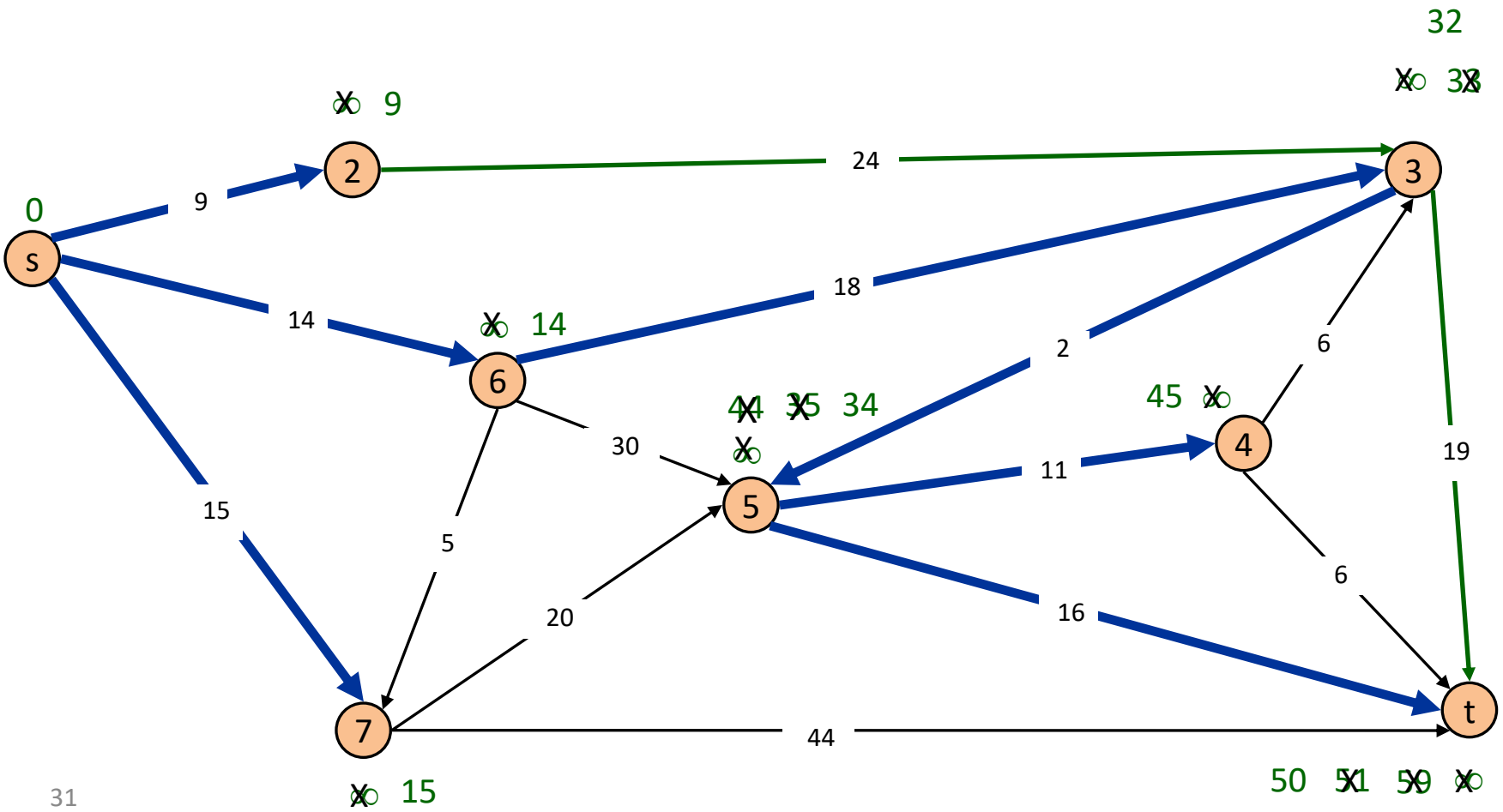
PQ = { }



# Dijkstra's Shortest Path Algorithm

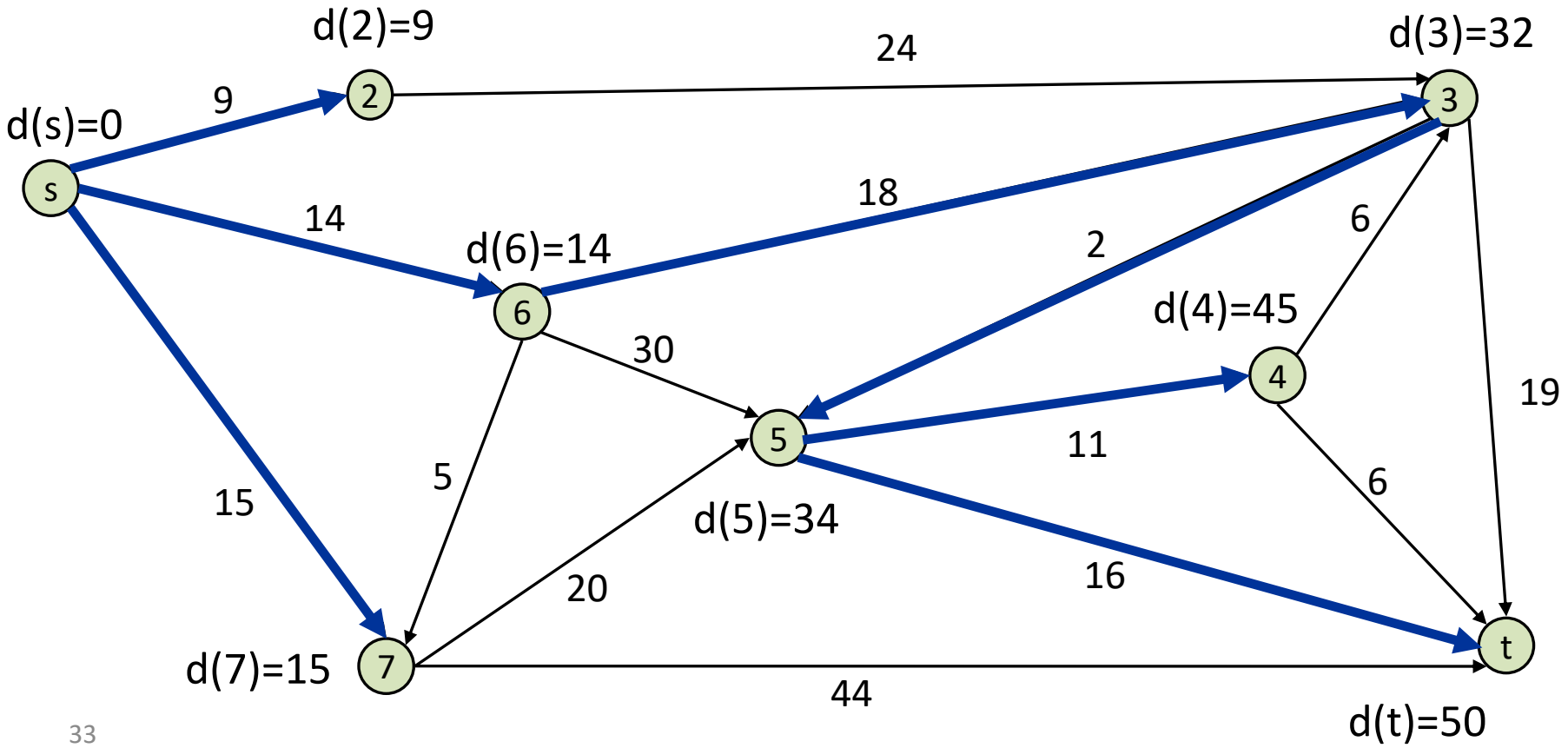
S = { s, 2, 3, 4, 5, 6, 7, t }

PQ = { }



# Dijkstra's Shortest Path Algorithm

## Risultato finale



# Applicabilità

Adesso vedremo che l'algoritmo di Dijkstra risolve correttamente il problema soltanto in caso di costi non-negativi.

L'algoritmo di Bellman-Ford, invece, risolve correttamente il problema anche in caso di costi negativi, ma senza cicli di costo negativo (altrimenti il costo minimo non sarebbe definito).

Questo è un esempio in cui la programmazione dinamica riesce a risolvere un problema in uno spettro di casi più ampio rispetto alla programmazione *greedy*.

# Dijkstra's Algorithm: Proof of Correctness (with non-negative weights only)

Invariant at each iteration (with non-empty S):

"For each $u \in S$, d(u) is the length of the shortest s-u path"

Pf. (by induction on $|S|$, with $1 \le |S| \le n$)

Base case: $|S| = 1$ is trivial.

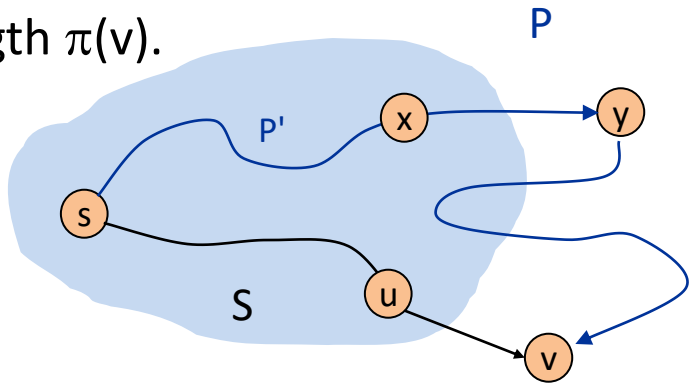
Inductive hypothesis: Assume true for $|S| = k \ge 1$ and show for k+1.

Let v be next node added to S and let u-v be the chosen edge.

The shortest s-u path plus (u, v) is an s-v path of length $\pi(v)$.

Consider any s-v path P. We'll see that it's no shorter than $\pi(v)$. Let x-y be the first edge in P that leaves S, and let P' be the subpath to x.

P is already too long as soon as it leaves S.




$$\ell(P) \ge \ell(P') + \ell(x,y) \ge d(x) + \ell(x, y) \ge \pi(y) \ge \pi(v)$$

**nonnegative weights**

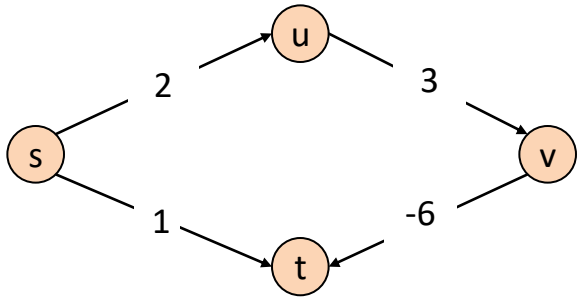
inductive hypothesis on x

defn of $\pi(y)$

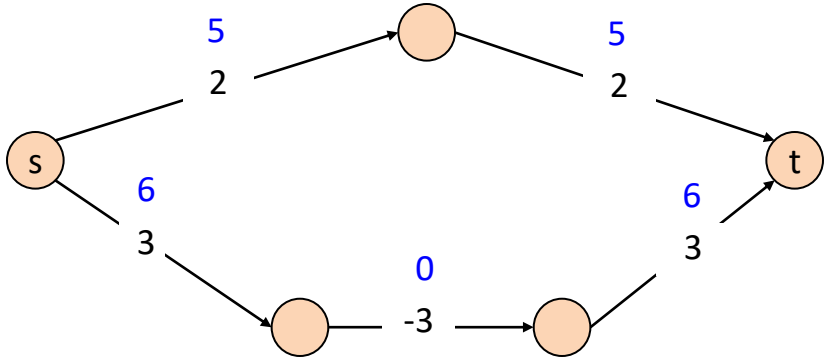
Dijkstra chose v instead of y

# Shortest Paths: Failed Attempts

Dijkstra. Can fail if negative edge costs /weights.




Re-weighting. Adding a constant to every edge cost can fail.



# Dijkstra's Algorithm

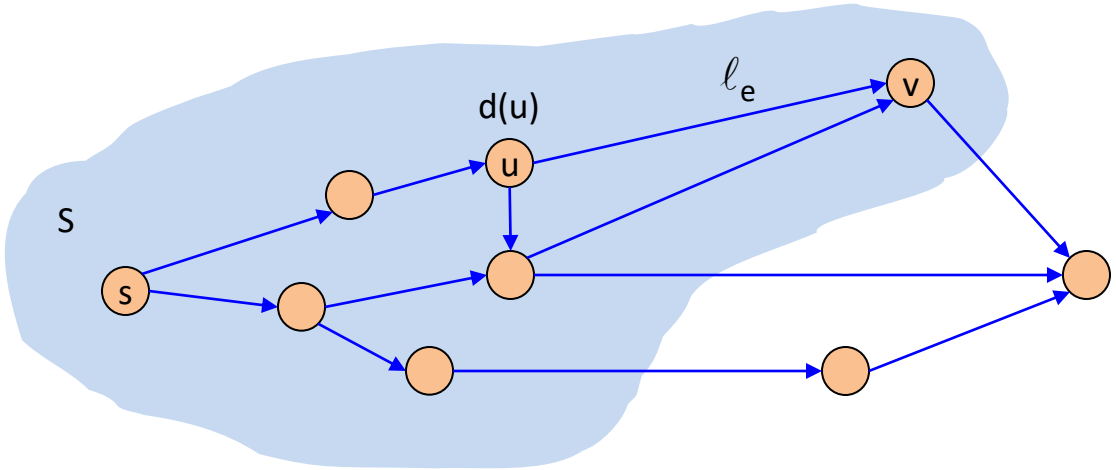
Dijkstra's algorithm.

- Maintain a set of explored nodes S for which we have determined the shortest path distance d(u) from s to u.
- Initialize S = { s }, d(s) = 0.
- Repeatedly choose unexplored node v which minimizes

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

shortest path to some u in explored part, followed by a single edge (u, v)

- Add v to S, and set d(v) = π(v).



# Dijkstra's Algorithm: Implementation

For each unexplored node, explicitly maintain

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

Next node to explore = node with minimum $\pi(v)$.

When exploring v, for each incident edge e = (v, w), update

$$\pi(w) = \min \{\ \pi(w),\ \pi(v) + \ell_e \}.$$

Efficient implementation. Maintain a priority queue of unexplored nodes, prioritized by $\pi(v)$.

Implemetazione simile all'algoritmo di Prim, per esercizio!

Priority Queue

| PQ Operation | Dijkstra | Array | Binary heap | d-way Heap | Fib heap † |
|---|---|---|---|---|---|
| Insert | n | 1 | log n | $d \log_d n$ | 1 |
| ExtractMin | n | n | log n | $d \log_d n$ | log n |
| ChangeKey | m | 1 | log n | $\log_d n$ | 1 |
| IsEmpty | n | 1 | 1 | 1 | 1 |
| Total | | $n^2$ | m log n | $m \log_{m/n} n$ | m + n log n |

† Individual ops are amortized bounds

# Esercizio (confronti)

Priority Queue

| PQ Operation | Dijkstra | Array | Binary heap | d-way Heap | Fib heap † |
|---|---|---|---|---|---|
| Insert | n | 1 | log n | $d \log_d n$ | 1 |
| ExtractMin | n | n | log n | $d \log_d n$ | log n |
| ChangeKey | m | 1 | log n | $\log_d n$ | 1 |
| IsEmpty | n | 1 | 1 | 1 | 1 |
| Total | | $n^2$ | m log n | $m \log_{m/n} n$ | m + n log n |

† Individual ops are amortized bounds

Quale implementazione della coda a priorità è preferibile (asintoticamente)?

In caso di costi non-negativi, quale algoritmo è preferibile (asintoticamente) fra quello greedy di Dijkstra e quello di PD di Bellman e Ford?

# Esercizio 1 (implementazione Dijkstra)

Scrivere un algoritmo che implementi l'algoritmo di Dijkstra tramite una coda a priorità e restituisca il costo minimo di un cammino da s a t.

*Suggerimento*: può essere utile considerare l'analoga implementazione dell'algoritmo di Prim:

```
Prim(G,c,s) {
    foreach (v ∈ V) a[v] ← ∞;  a[s]←0
    Initialize an empty priority queue Q
    foreach (v ∈ V) insert v onto Q
    Initialize set of explored nodes S ← ϕ

    while (Q is not empty) {
        u ← delete min element from Q
        S ← S ∪ { u }
        foreach (edge e = (u, v) incident to u)
            if ((v ∉ S) and (c_e < a[v]))
                decrease priority a[v] to c_e
}
```

# Esercizio 2

Dopo aver svolto l'esercizio 1 (implementazione Dijkstra), modificare lo pseudocodice in modo che l'algoritmo restituisca un cammino ottimale da s a t (non solo il suo costo minimo).

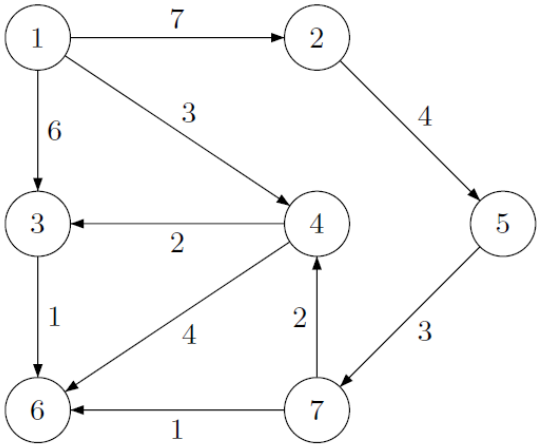
**Quesito 3** (Seconda prova intercorso del 31 maggio 2019)

a) Definire il **problema** risolto dall'algoritmo di Dijkstra.

b) Si consideri l'algoritmo di Dijkstra (**non** è richiesto di scriverne lo pseudocodice). Si supponga che ad una certa generica iterazione su un grafo G=(V,E) (come specificato al punto a)) con vertice di partenza s, sia noto l'insieme S dei nodi già considerati, e per ogni nodo x in S sia definito il valore d(x) che è uguale al costo minimo di un cammino in G da s a x. Si supponga che adesso l'algoritmo estragga dalla coda a priorità il nodo v con priorità $\pi(v)$. **Dimostrare** che qualsiasi cammino P in G da s a v ha un costo $\lambda(P)$, con $\lambda(P) \geq \pi(v)$, giustificandone i passaggi.

c) Mostrare **l'esecuzione** dell'algoritmo di Dijkstra sul seguente grafo G=(V,E) con vertice di partenza *s*=1, mettendo in evidenza gli aggiornamenti della struttura dati e il risultato finale ottenuto.

d) V**erificare** la dimostrazione fornita al punto b) in riferimento all'esecuzione dell'algoritmo sul grafo G=(V,E) (in basso) e vertice s=1, all'**iterazione** in cui **S** contiene **3 elementi**. Bisogna cioè, specificare l'insieme S, il prossimo nodo v scelto dall'algoritmo, considerare ogni cammino P da s a v e mostrare che il suo costo $\lambda(P)$ è $\lambda(P) \geq \pi(v)$. Specificare inoltre eventuali altri nodi e archi che sono stati utilizzati durante la dimostrazione.



# Cambio monete greedy (1, 5, 10)

Si consideri il problema di determinare il minimo numero di monete da 1, 5, 10 centesimi necessarie per scambiare un ammontare di denaro D, supponendo di poter utilizzare un numero illimitato di monete dello stesso taglio.

a) **Descrivere** ed **analizzare** un algoritmo *greedy* che risolve tale problema

b) Dimostrare la **correttezza** dell'algoritmo proposto

# Cambio monete greedy (1, 5, 10): correttezza (studente 2020)

Dimostriamo ora la correttezza dell'algoritmo dobbiamo provare che, preso un resto R, esso può essere dato con il minimo numero di monete considerando di volta in volta la massima moneta compatibile. Ci sono 3 casi:

1. -Se R < 5, allora il resto può semplicemente essere dato con R monete da 1, e non ci sono altre possibilità.
2. -Se 5 <= R < 10, allora se per assurdo il resto fosse dato nella soluzione ottimale prendendo in considerazione monete da 1 anziché da 5, finché possibile, allora potrei sostituire 5 monete da 1 con 1 da 5, ed otterrei un numero minore di monete rispetto alla soluzione ottimale, il che è assurdo.
3. -Se R >= 10, allora se per assurdo il resto fosse dato nella soluzione ottimale prendendo in considerazione monete da 1 e da 5 anziché da 10, finché possibile, allora potrei sostituire 10 monete da 1 con 1 da 10, oppure 2 monete da 5 con 1 da 10, ottenendo un numero di monete inferiore rispetto alla soluzione ottimale, il che è assurdo.

# **Lampioni** (Greedy)

Una società elettrica vuole piazzare dei lampioni lungo un viale rettilineo dove sono situate delle villette, in modo da illuminare tutte le villette. Ogni lampione riesce ad illuminare tutta la zona compresa nel raggio di 500 metri. Percorrendo il viale si incontrano, in ordine, le villette $v_1, v_2, \dots, v_n$. Per ogni $i = 1, \dots, n-1$, sia $d_{i,i+1}$ la distanza della villetta $v_i$ da $v_{i+1}$. Immaginiamo che le villette siano «puntiformi».

a) **Descrivere** ed **analizzare** un algoritmo che presi in input i valori $d_{i,i+1}$ restituisca il numero minimo di lampioni necessari per illuminare tutte le villette.
b) **Argomentare** la correttezza dell'algoritmo.

# Lampioni: correttezza

Una possibilità:

simile alla seconda dimostrazione della correttezza dello scheduling di attività, che riporto di seguito.

# Correttezza scheduling di intervalli: seconda dimostrazione

**Teorema.** L'algoritmo greedy è ottimale.

**Prova.** (per assurdo)

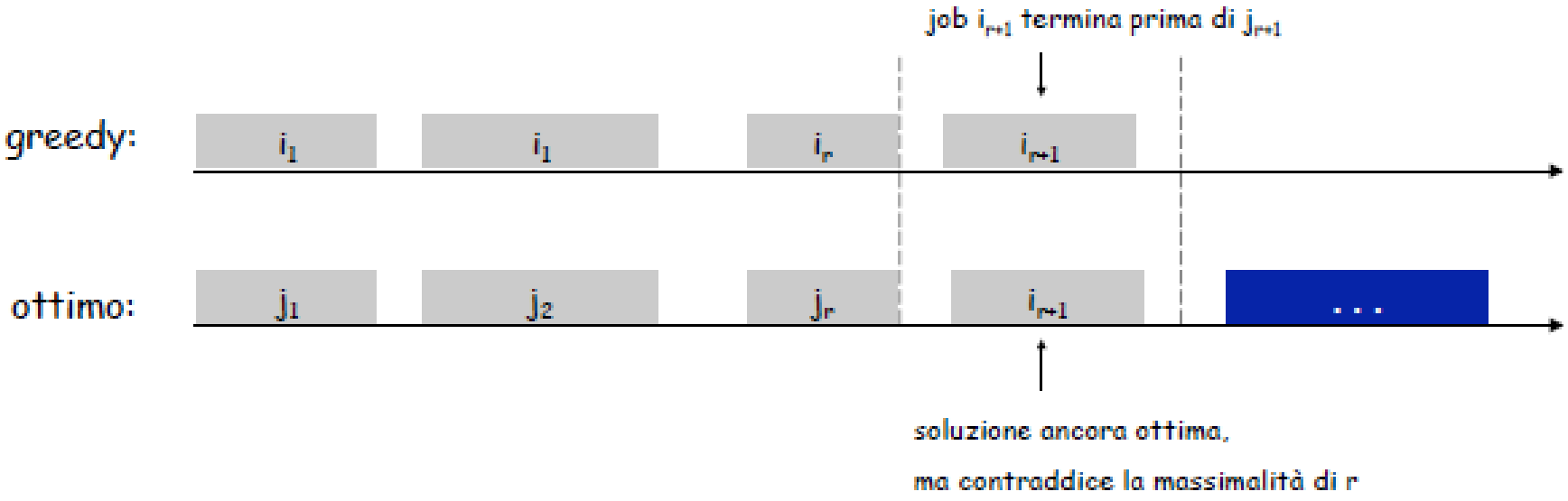
- Assumiamo che la scelta greedy non sia ottimale.
- Siano $i_1, i_2, \ldots i_k$ job scelti in modo greedy.
- Siano $j_1, j_2, \ldots j_m$ job in una soluzione ottimale con $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ dove r è il più grande valore possibile.

# Correttezza scheduling di intervalli: seconda dimostrazione

**Teorema.** L'algoritmo greedy è ottimale.

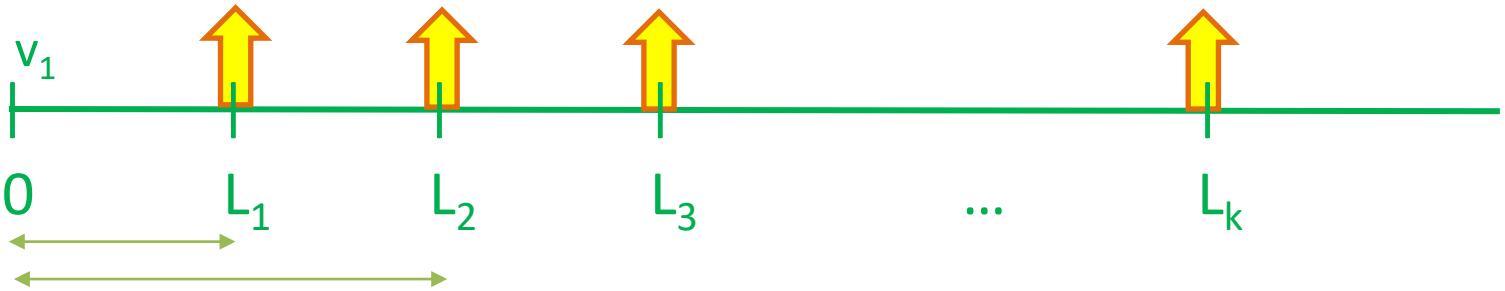
**Prova.** (per assurdo)

- Assumiamo che la scelta greedy non sia ottimale.
- Siano $i_1, i_2, \ldots i_k$ job scelti in modo greedy.
- Siano $j_1, j_2, \ldots j_m$ job in una soluzione ottimale con $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ dove r è il più grande valore possibile.



# Lampioni: correttezza

Sia $G = \{L_1, L_2, \dots, L_k\}$ la soluzione greedy, dove per ogni i=1, 2, … , k, $L_i$ è la distanza dell'i-esimo lampione dalla prima villetta $v_1$.



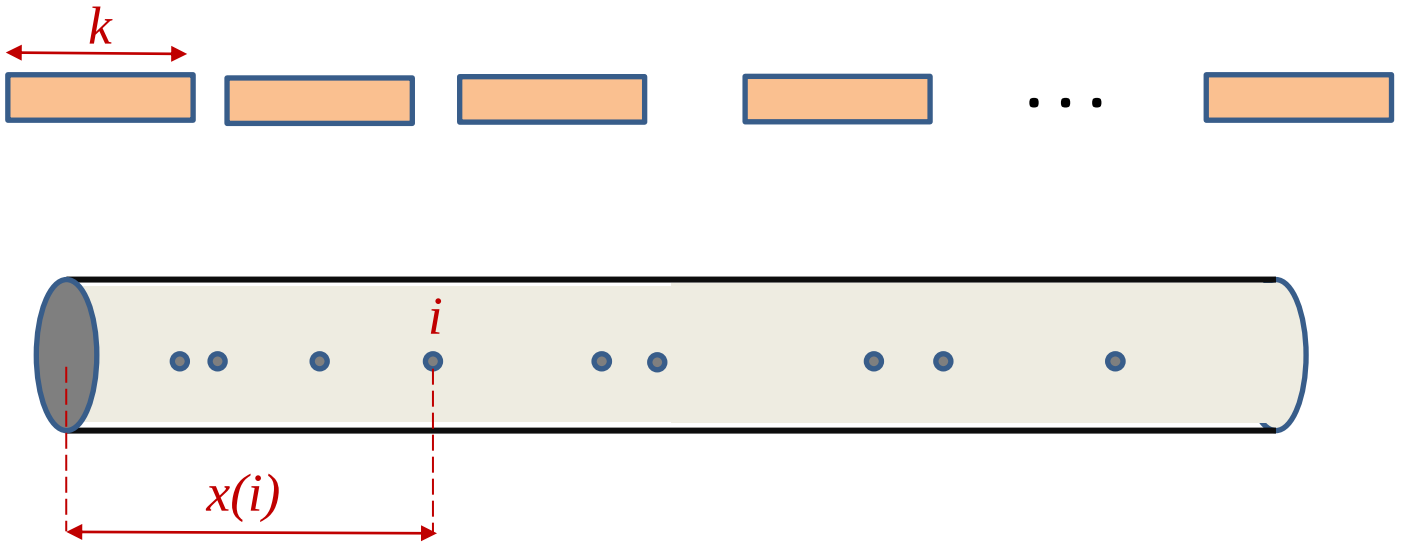

Supponiamo per assurdo che G non sia ottimale e ….. (to be continued)

**Quesito** (Svolto)

Si intende riparare un **tubo** di gomma che ha *n* fori allineati su di esso. Si indichi con *x(i)* la distanza in centimetri dell'*i*-esimo foro dalla estremità sinistra del tubo. La riparazione avviene attaccando pezzi di nastri adesivo al tubo, dove ciascun pezzo di nastro adesivo è lungo *k* centimetri, con *k* parametro dato. Il problema è quello di riparare il tubo, ovvero coprire tutti i fori, utilizzando il **minor** numero di pezzi di nastro adesivo.

**a)** **Descrivere** un algoritmo *greedy* che risolva il problema. (Nota: non è necessario scrivere lo pseudo-codice)

**b)** **Dimostrare** la correttezza dell'algoritmo proposto.



## Noleggio moto

Una comitiva di amici in vacanza in un'isola Covid-free, vuole noleggiare delle moto per visitare l'isola. Ognuno vuole effettuare il giro in un orario a sua scelta, indipendentemente dagli altri. Mario che non sopporta il sole, dalle 8 alle 11, Giovanna che è dormigliona, dalle 13 alle 15, Aurora vorrebbe fare un lungo giro dell'isola dalle 10 alle 14, anche Bea vorrebbe fare il giro con Aurora (ma su un'altra moto), Hans vorrebbe una moto dalle 9 alle 12 ed Elmo dalle 14 alle 16 . Poiché l'affitto delle moto si paga per l'intera giornata, decidono di organizzarsi in modo che ogni moto sia utilizzata da più persone nell'arco della giornata. E si chiedono: qual è il minimo numero di moto necessario per poter accontentare tutti?

a) Definire il problema computazionale e indicare se si tratta di uno dei problemi studiati

b) Descrivere verbalmente un algoritmo che risolva il problema (nella sua formulazione generale), indicandone la tecnica di progettazione utilizzata

c) Analizzarne la complessità di tempo

d) Dimostrarne la correttezza

e) Eseguire l'algoritmo che avete descritto al punto b) sui dati forniti nel testo sopra

## Scheduling che minimizza il ritardo (scambi)

Quesito 2 (appello 15/06/2020)

a) Definire il problema dello scheduling che minimizza il ritardo.

b) **Eseguire** l'algoritmo studiato su un insieme di attività {1, 2, 3, 4} con tempi di esecuzione rispettivamente t1 = 2, t2 = 3, t3 = 4, t4 = 3 e scadenze (deadline) d1 = 7, d2 = 6, d3 = 10, d4 = 4, specificando il ritardo dello scheduling ottenuto.

c) **Esibire** un'altra soluzione ottimale al problema precedente, che sia **diversa** da quella fornita al punto b).

d) In cosa consiste **la tecnica degli scambi successivi** che permette di trasformare una qualsiasi soluzione ottimale in quella fornita dall'algoritmo greedy? Applicare tale tecnica alla soluzione esibita al punto c).

## Scheduling che minimizza il ritardo (codice e correttezza)

a) **Definire** formalmente il problema dello Scheduling che minimizza il ritardo.

b) Considerate la **variante** al problema in cui la **soluzione** cercata, anziché lo scheduling, è il suo ritardo. **Descrivere** tramite pseudo-codice un algoritmo che risolve tale variante al problema.

c) Dimostrarne la **correttezza**.

## Zaino semplice

a) **Definire** formalmente il **problema** dello zaino.

b) **Descrivere** un algoritmo *greedy* che risolva il problema dello zaino, nelle ipotesi che tutti gli oggetti abbiamo lo **stesso valore**. Fornirne un **esempio** di esecuzione.

c) Dimostrare che l'algoritmo descritto risolve **correttamente** il problema.

# Tempo medio di attesa

**Quesito** 3 (36 punti)

**Un** processore deve eseguire **n** processi a partire dall'ora **0**. Ogni processo ha una sua **durata** prevista. Il problema è quello di pianificare l'esecuzione di **tutti** i processi in modo da **minimizzare** il **tempo medio di attesa** (definito come la media aritmetica dei tempi di attesa), dove il **tempo di attesa** di ogni processo è pari al tempo in cui sarà stabilito l'inizio della sua esecuzione.

a) Definire il **problema** computazionale (specificandone i dati in ingresso e in uscita). Si tratta di un problema studiato?

b) Descrivere un **algoritmo *greedy*** per risolvere il problema. Si potrà avere il massimo della votazione se l'algoritmo è descritto tramite **pseudocodice** ed è **commentato** per chiarirne il funzionamento.

c) Dimostrare la **correttezza** dell'algoritmo proposto.

# Calendario

**Lezione 34** *(giovedì 25 maggio- **ore 14-16 in S6**)*: Algoritmi intelligenti di ricerca esaustiva: backtracking e branch-and-bound ([DPV] par. 9.1). Fine del programma!

**Lezione 35** *(venerdì 26 maggio)***:** Esercitazione

**Lezione 36** *(mercoledì 31 maggio):* Esercitazione

**Lezione Plus** *(giovedì 1 giugno)*

**Seconda prova intercorso**: **8 giugno ore 15, aula P3+P4** in concomitanza col pre-appello.

La prova verterà su tutto il programma, **in particolare** sugli argomenti svolti dopo la prima prova.

Chi ha totalizzato almeno **40/100** è ammesso alla seconda prova. Supporrò che chi è ammesso alla seconda prova vi partecipi. Se invece qualcuno volesse **rinunciare** alla prima prova per partecipare ad un appello completo, è pregato di comunicarmelo. In ogni caso, chi parteciperà alla seconda prova dovrà **iscriversi** al pre-appello su Esse3, per poter registrare l'esito delle prove.