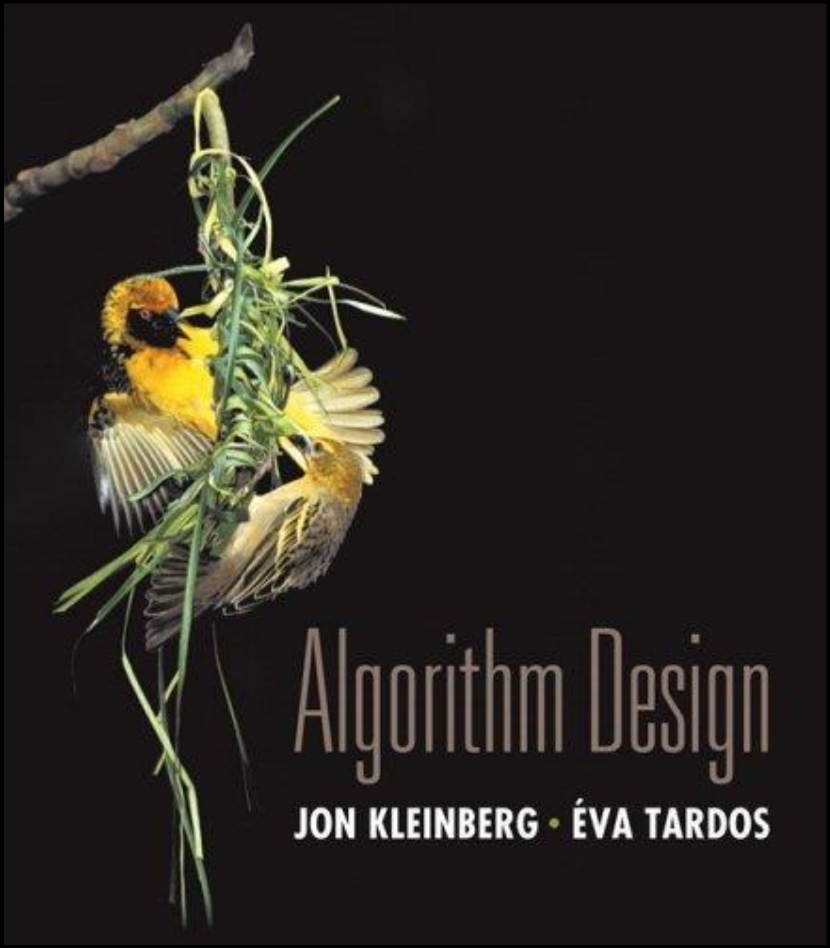



# Chapter 4

## Greedy Algorithms, Part 2

Slides by Kevin Wayne.

# Goals

Understand that sometimes greed is ~~good~~ optimal!

Be able to **analyze** whether a greedy algorithm is optimal

- show it "stays ahead" of any other algorithm
- inductively
- lower bound the optimal solution, show that greedy achieves this bound
- exchangability and other problem structure

Problems:

- Interval scheduling
- Coin changing
- Optimal caching
- Shortest path
- Minimum spanning tree

# 4.2 Scheduling to Minimize Lateness

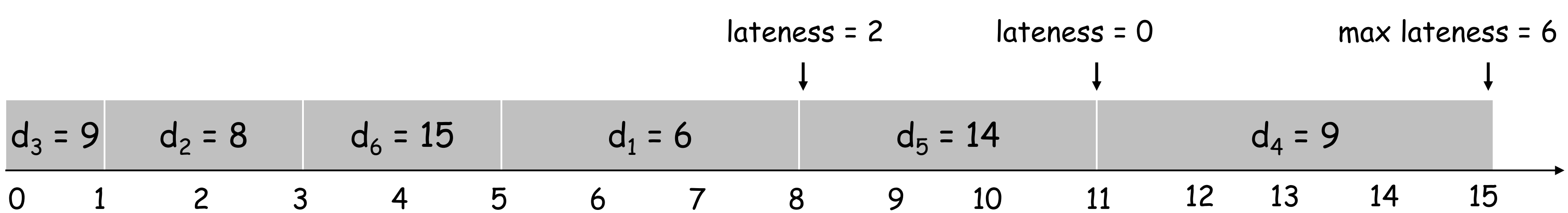
# Scheduling to Minimizing Lateness

## Minimizing lateness problem.

- Single resource processes one job at a time.
- Job j requires $t_j$ units of processing time and is due at time $d_j$.
- If j starts at time $s_j$, it finishes at time $f_j = s_j + t_j$.
- Lateness: $\ell_j = \max\{0, f_j - d_j\}$.
- Goal: schedule all jobs to minimize maximum lateness $L = \max \ell_j$.

Ex:

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| $t_j$ | 3 | 2 | 1 | 4 | 3 | 2 |
| $d_j$ | 6 | 8 | 9 | 9 | 14 | 15 |



# Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time $t_j$.

- [Earliest deadline first] Consider jobs in ascending order of deadline $d_j$.

- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

# Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time $t_j$.

| | 1 | 2 |
|---|---|---|
| $t_j$ | 1 | 10 |
| $d_j$ | 100 | 10 |

counterexample

- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

| | 1 | 2 |
|---|---|---|
| $t_j$ | 1 | 10 |
| $d_j$ | 2 | 10 |

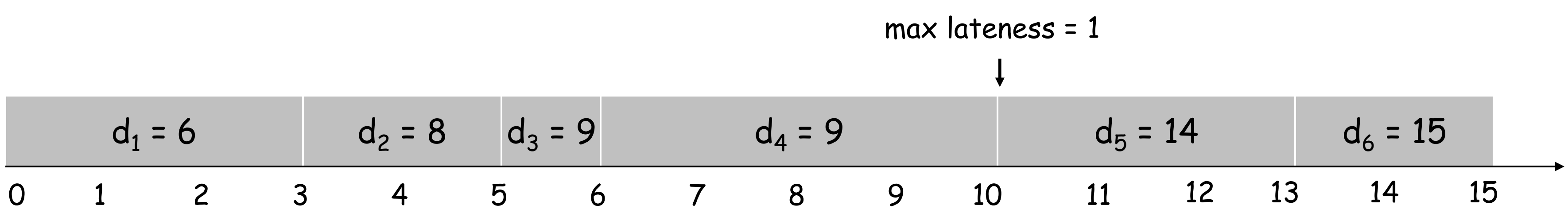
counterexample

# Minimizing Lateness: Greedy Algorithm

**Greedy algorithm.** Earliest deadline first.

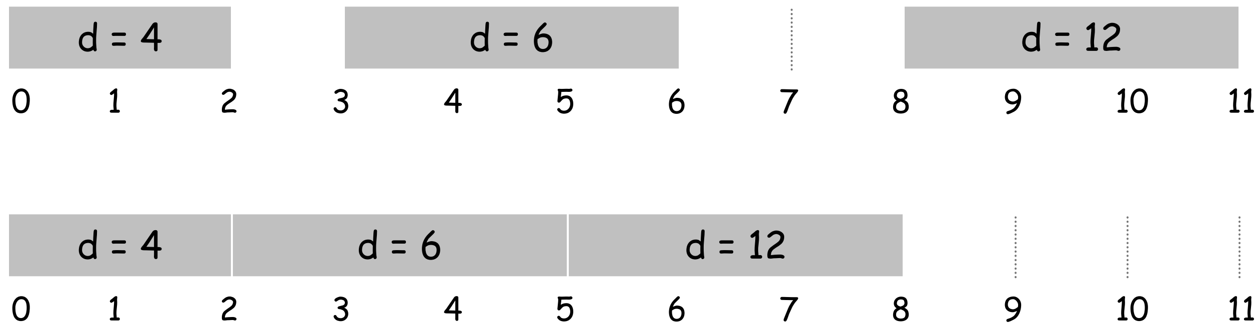
```
Sort n jobs by deadline so that d_1 ≤ d_2 ≤ … ≤ d_n

t ← 0
for j = 1 to n
   Assign job j to interval [t, t + t_j]
   s_j ← t, f_j ← t + t_j
   t ← t + t_j
output intervals [s_j, f_j]
```



# Minimizing Lateness: No Idle Time

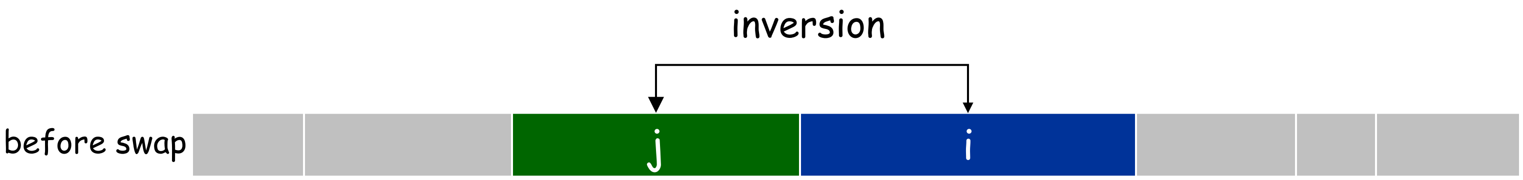
Observation. There exists an optimal schedule with no idle time.




Observation. The greedy schedule has no idle time.

# Minimizing Lateness: Inversions

Def. An inversion in schedule S is a pair of jobs i and j such that: $i < j$ but j scheduled before i.

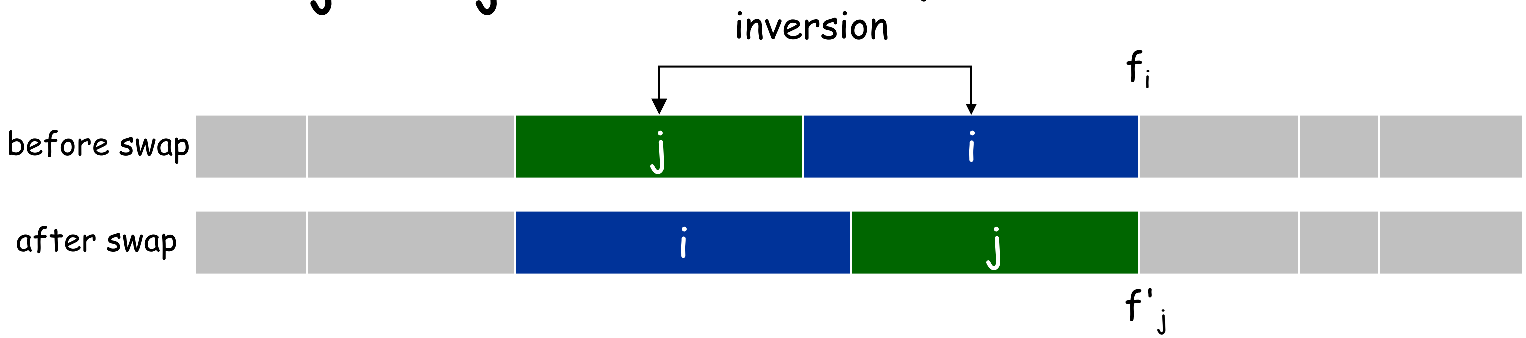



Observation. Greedy schedule has no inversions.

Observation. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.

# Minimizing Lateness: Inversions

Def. An inversion in schedule S is a pair of jobs i and j such that: i < j but j scheduled before i.




Claim. Swapping two adjacent, inverted jobs reduces the number of inversions by one and does not increase the max lateness.

Pf. Let $\ell$ be the lateness before the swap, and let $\ell'$ be it afterwards.

- $\ell'_k = \ell_k$ for all $k \neq i, j$
- $\ell'_i \leq \ell_i$
- If job j is late:

$$
\begin{aligned}
\ell'_j &= f'_j - d_j && \text{(definition)} \\
&= f_i - d_j && (j \text{ finishes at time } f_i) \\
&\leq f_i - d_i && (i < j) \\
&\leq \ell_i && \text{(definition)}
\end{aligned}
$$

# Minimizing Lateness: Analysis of Greedy Algorithm

Theorem. Greedy schedule S is optimal.

Pf. (contradiction) Define S* to be an optimal schedule that has the fewest number of inversions, and let's see what happens.

- Can assume S* has no idle time.
- If S* has no inversions, then S = S*.
- If S* has an inversion, let i-j be an adjacent inversion.
  - swapping i and j does not increase the maximum lateness and strictly decreases the number of inversions
  - this contradicts definition of S* ▪

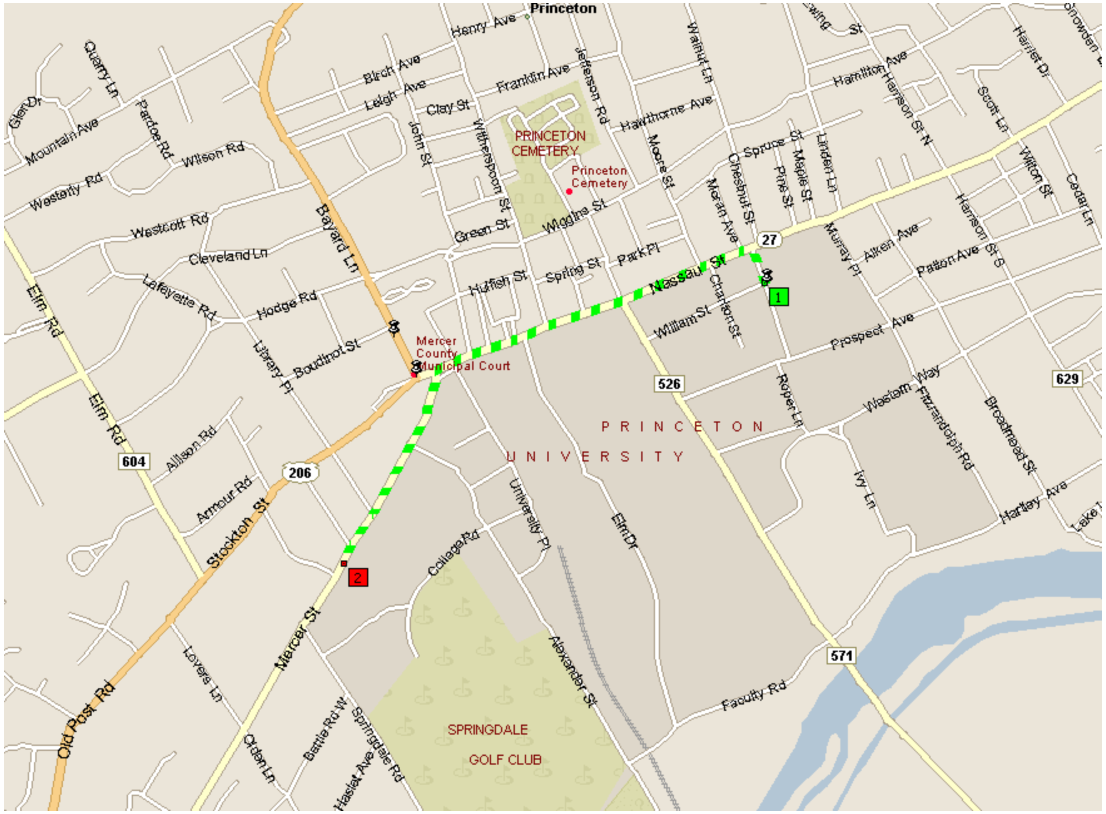
# Greedy Analysis Strategies

Greedy algorithm stays ahead. Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.

Exchange argument. Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

Structural. Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.

# 4.4 Shortest Paths in a Graph




shortest path from Princeton CS department to Einstein's house

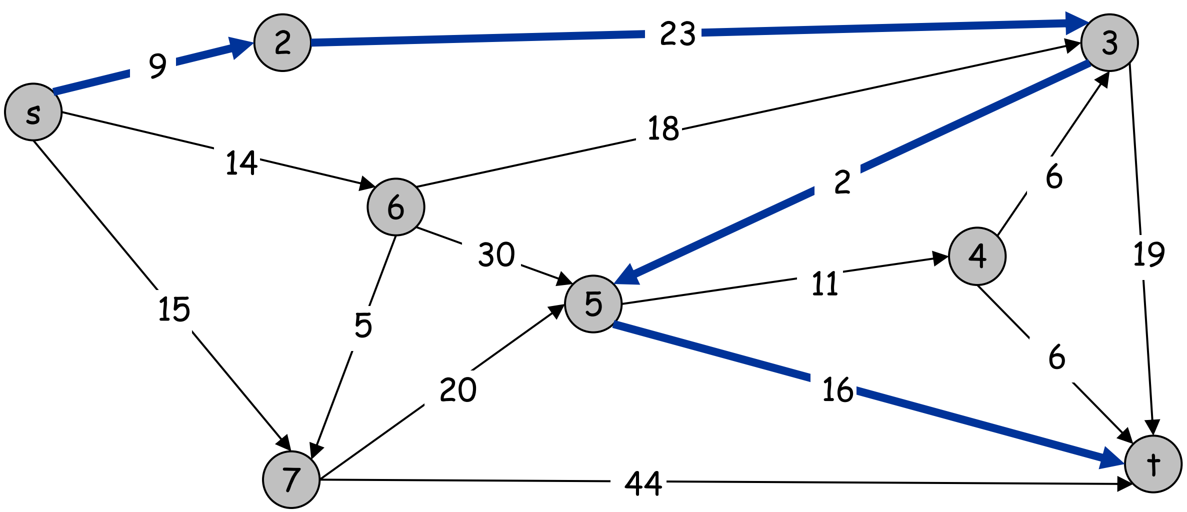
# Shortest Path Problem

Shortest path network.

- Directed graph $G = (V, E)$.
- Source $s$, destination $t$.
- Length $\ell_e$ = length of edge $e$.

Shortest path problem: find shortest directed path from $s$ to $t$.

cost of path = sum of edge costs in path




Cost of path s-2-3-5-t
= 9 + 23 + 2 + 16
= 48.

# Dijkstra's Algorithm

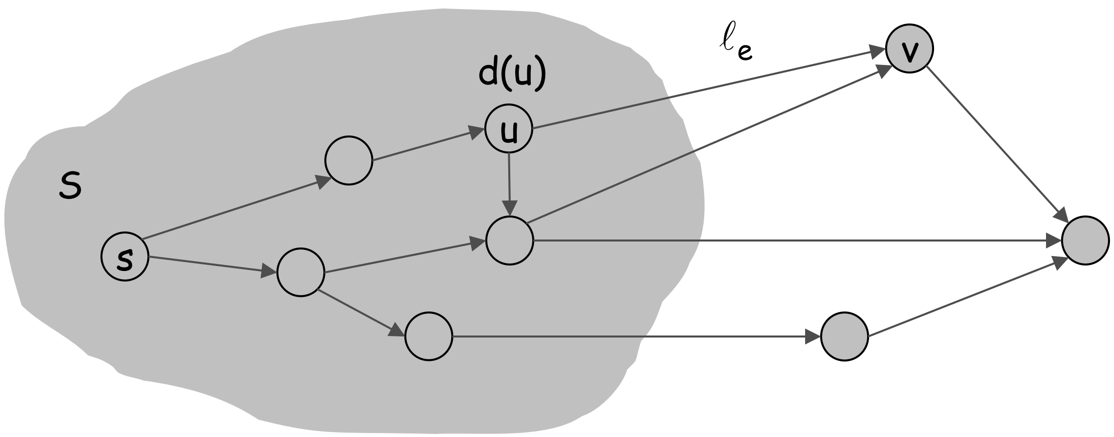
Dijkstra's algorithm.

- 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) = $\pi$(v).



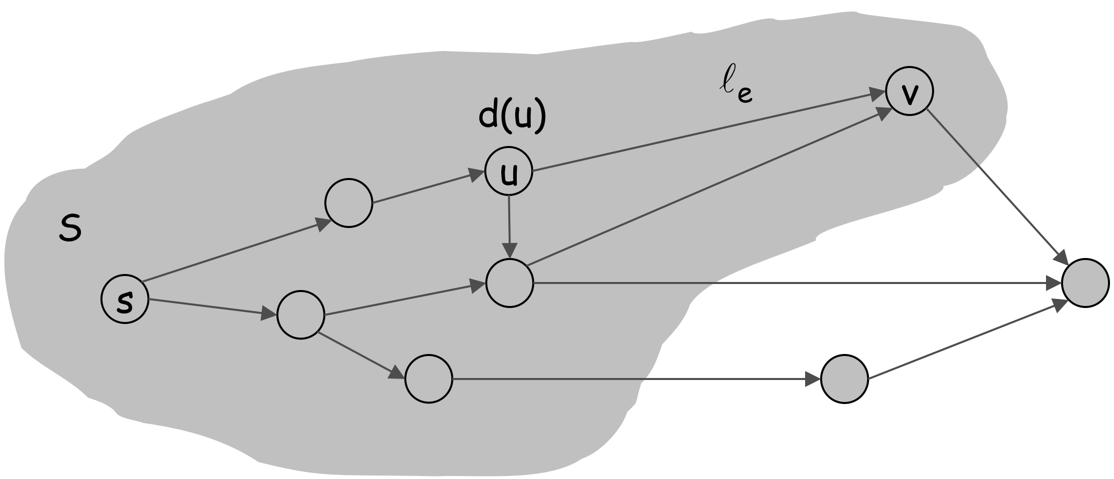
# Dijkstra's Algorithm

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

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

add v to S, and set d(v) = π(v).

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

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

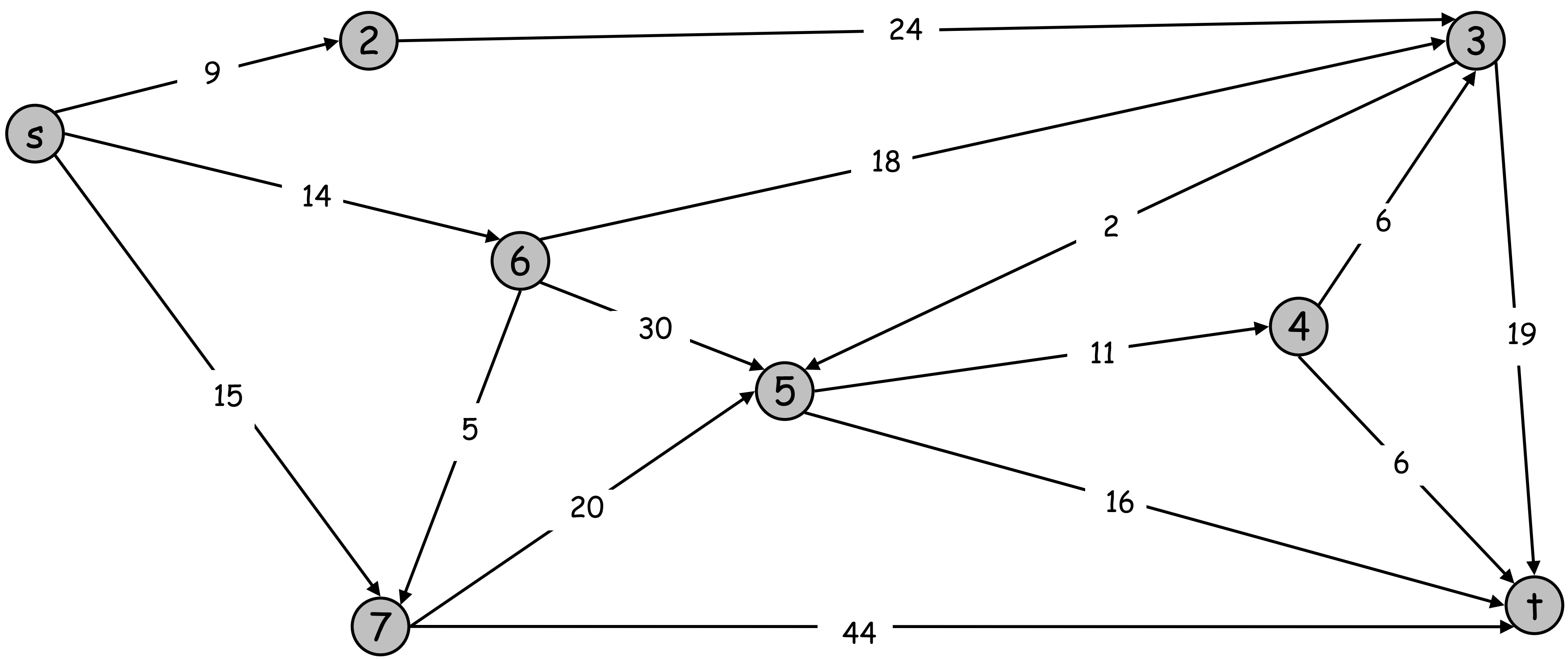
Efficient implementation. Maintain a priority queue of unexplored nodes, prioritized by $\pi(v)$.

| PQ Operation | Dijkstra | Array | Binary heap | d-way Heap | Fib heap † |
|---|---|---|---|---|---|
| Insert | n | n | 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

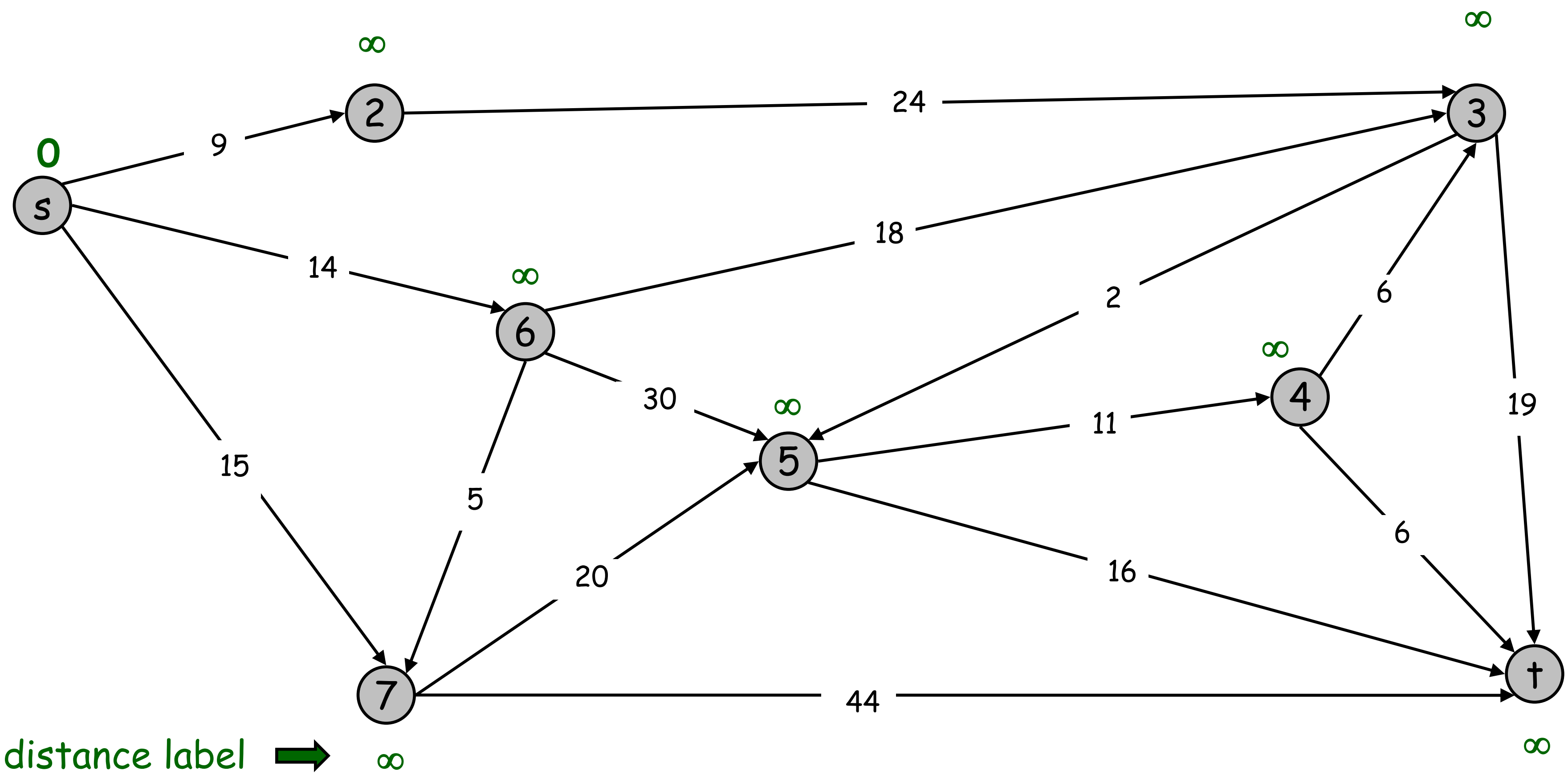
# Dijkstra's Shortest Path Algorithm

Find shortest path from s to t.



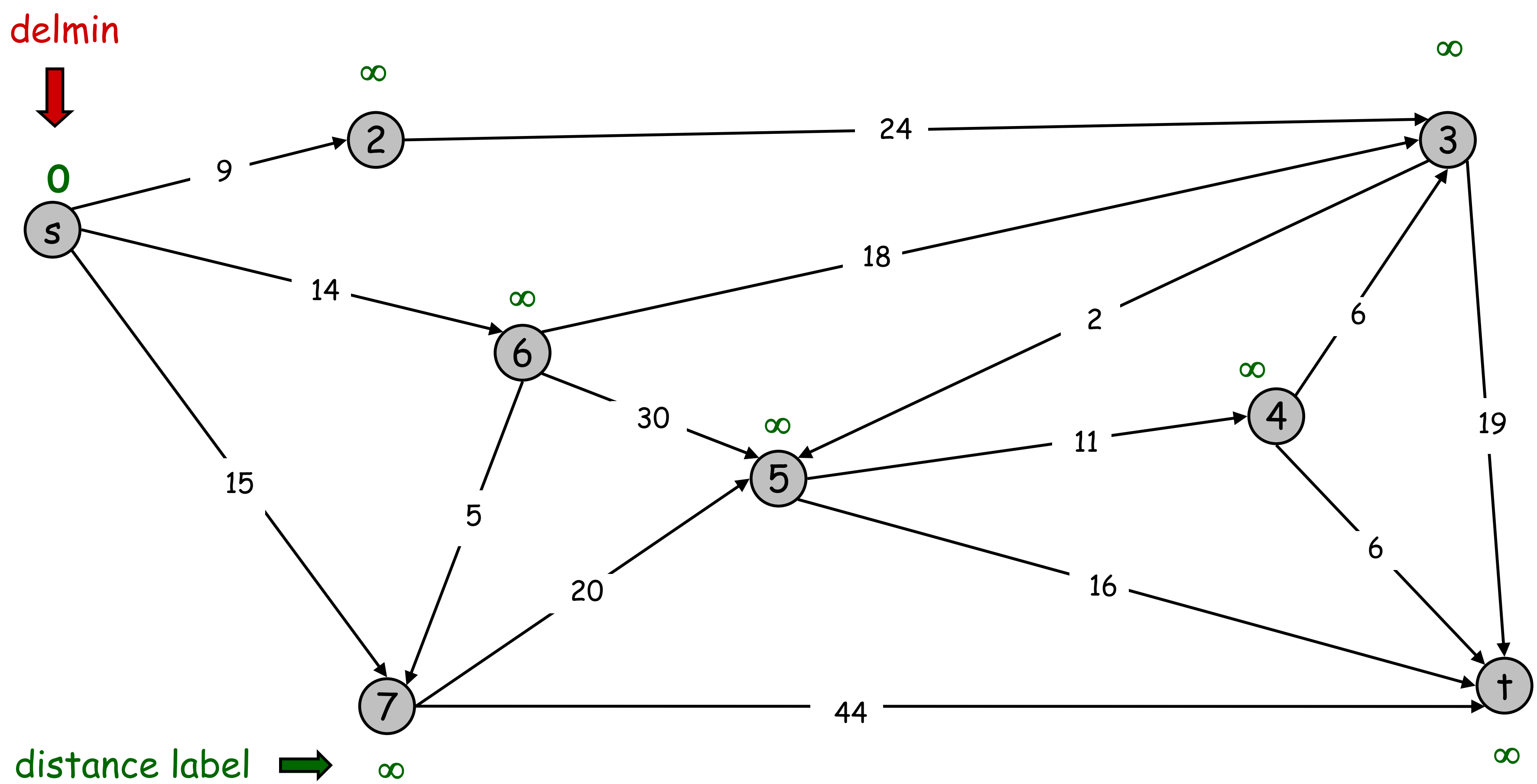
# Dijkstra's Shortest Path Algorithm

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



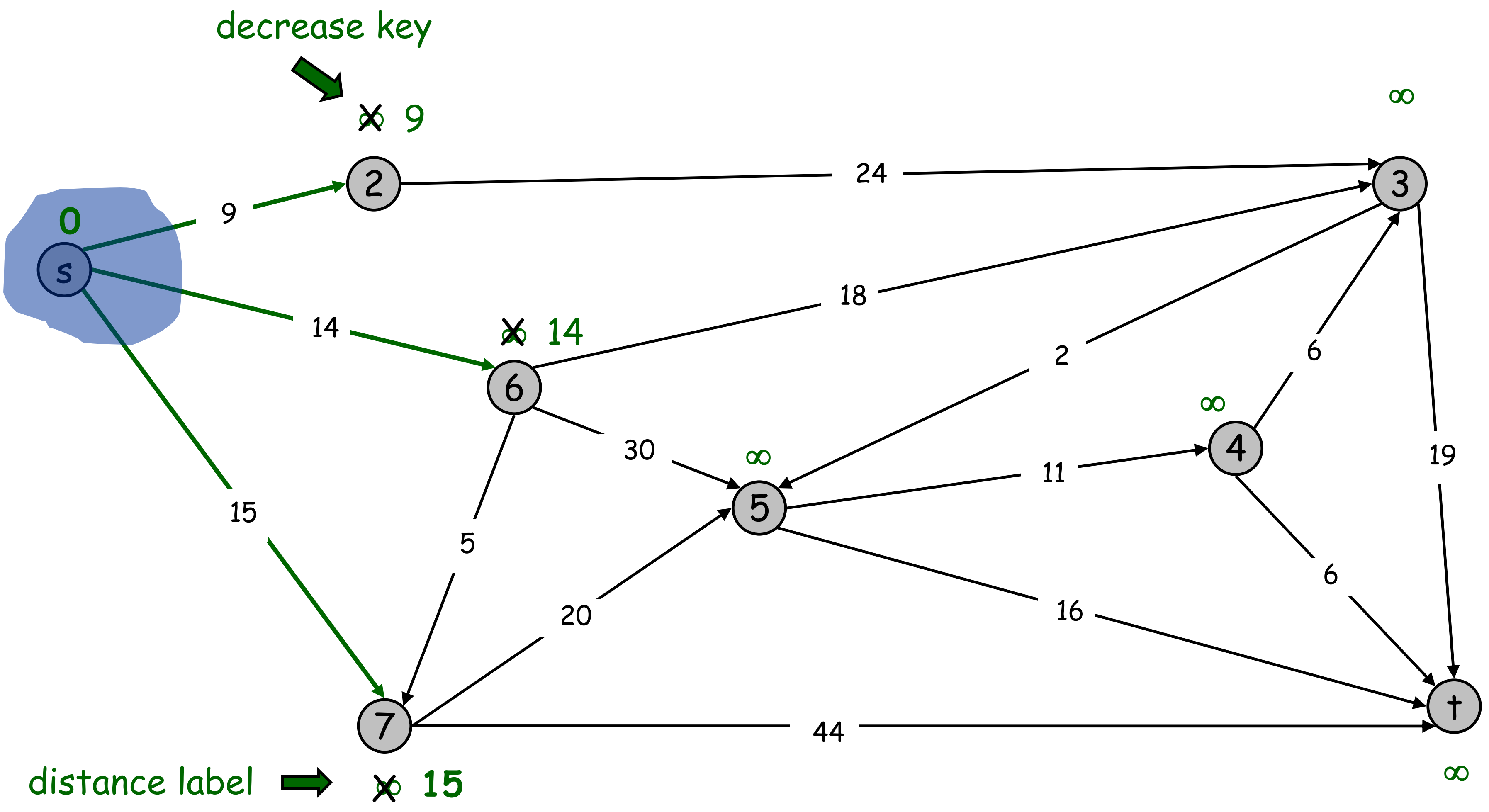
# Dijkstra's Shortest Path Algorithm

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



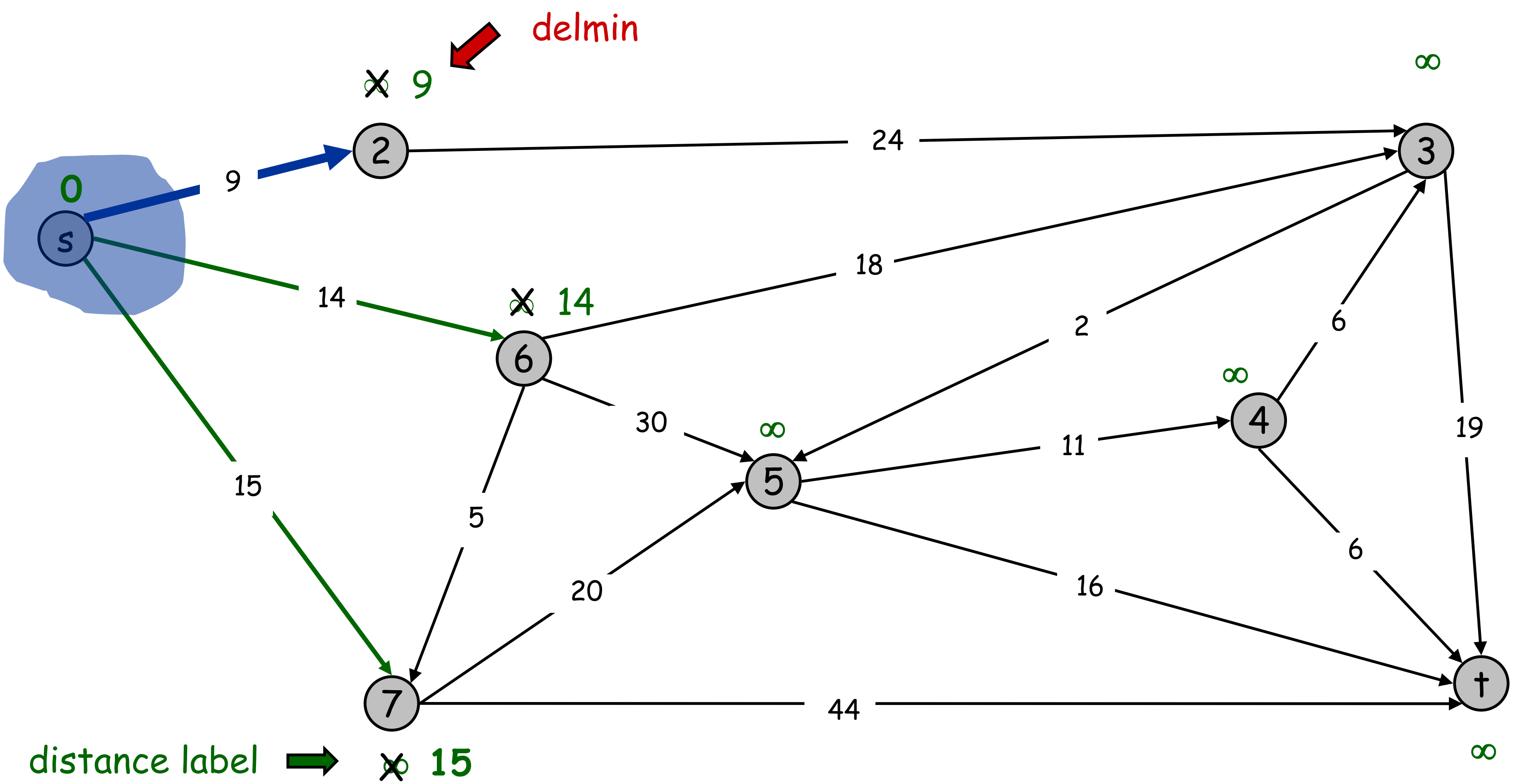
# Dijkstra's Shortest Path Algorithm

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



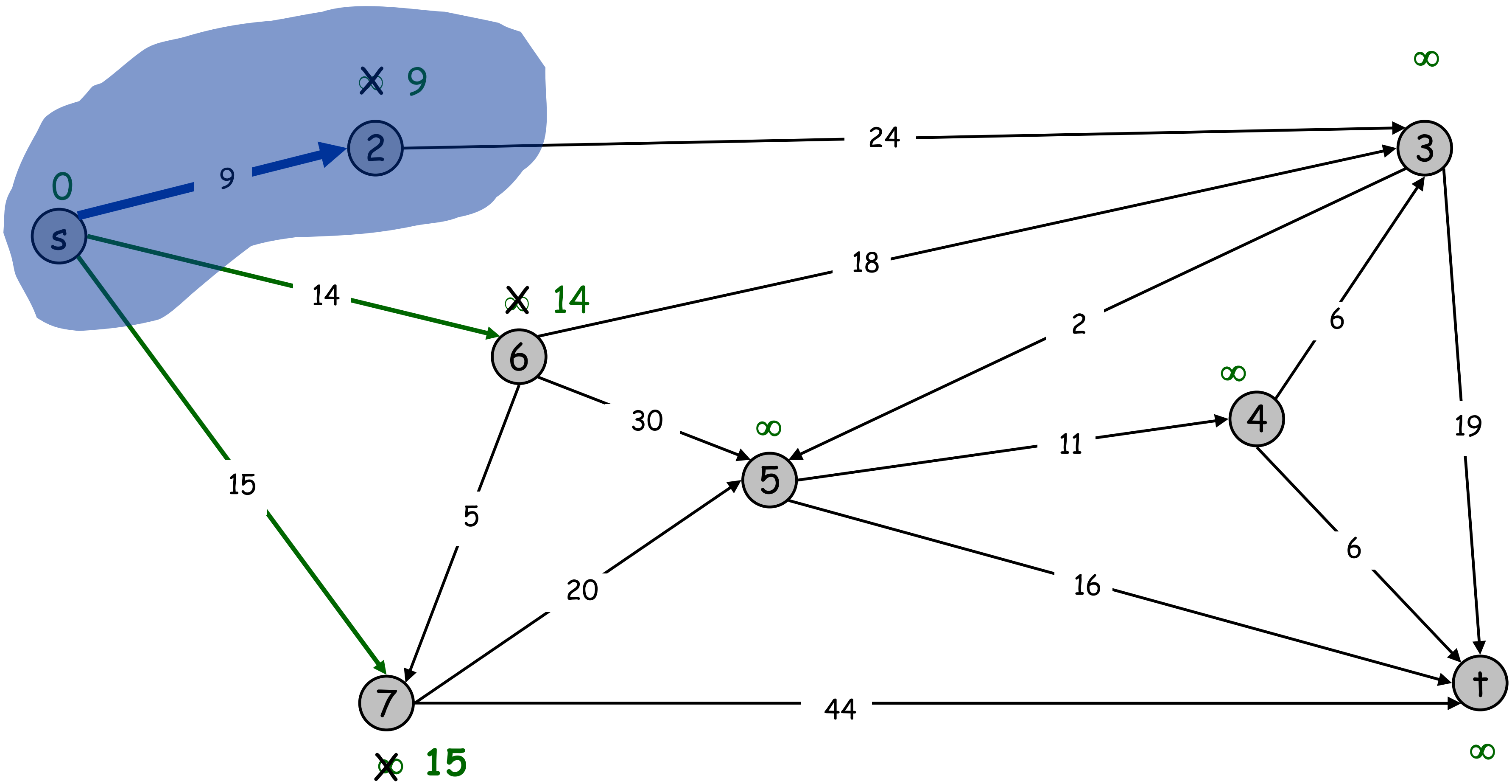
# Dijkstra's Shortest Path Algorithm

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



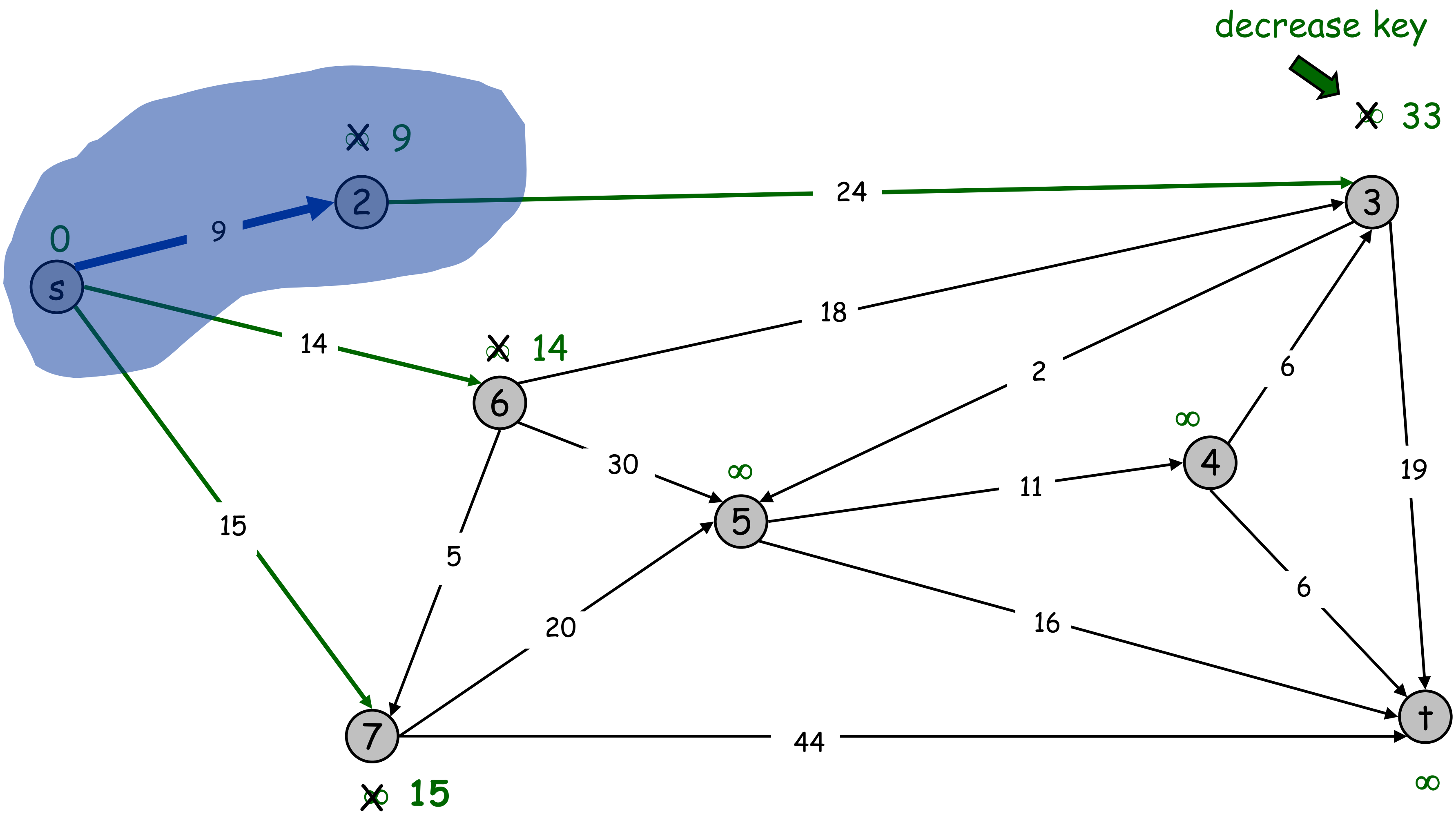
# Dijkstra's Shortest Path Algorithm

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



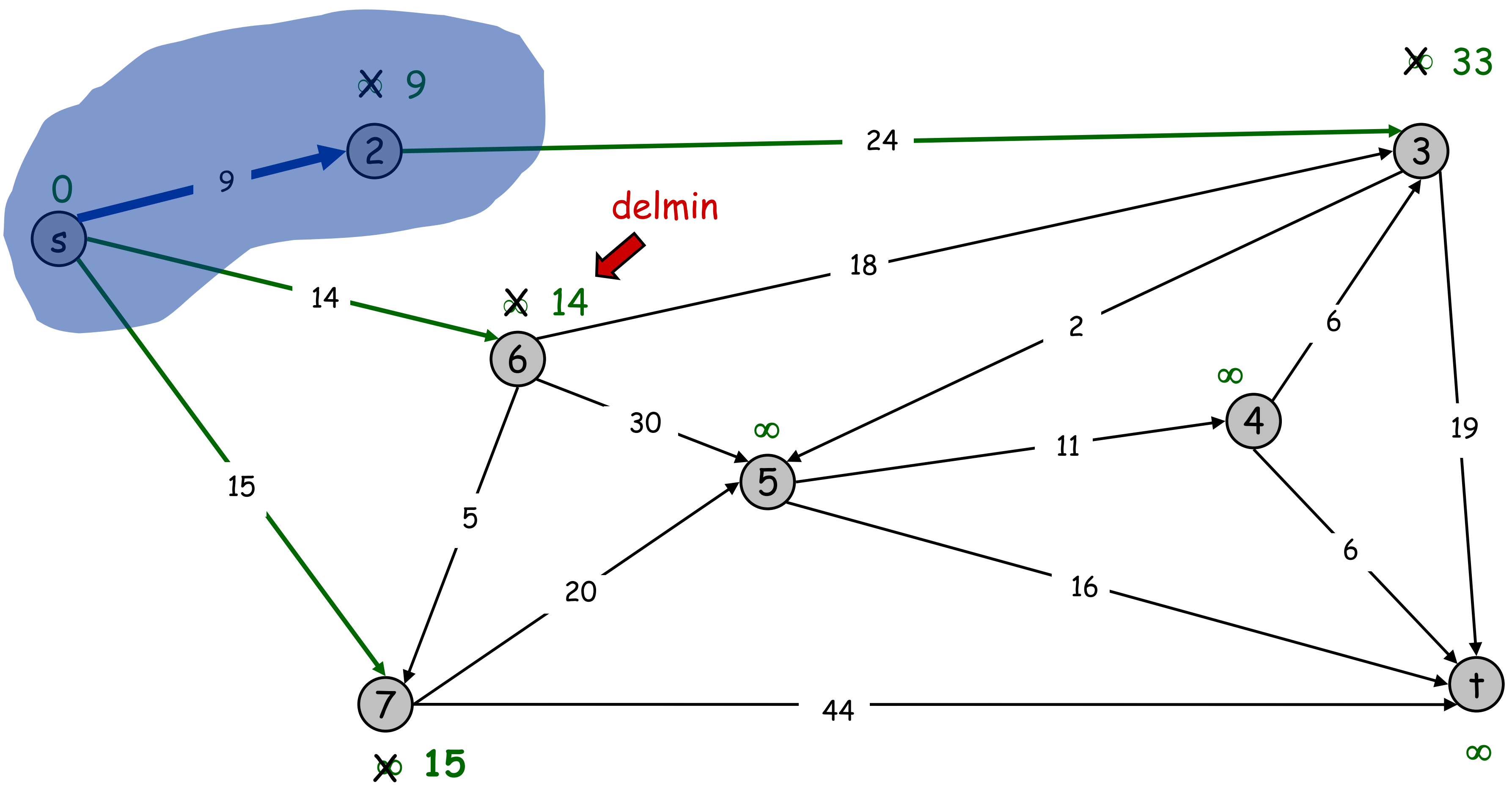
# Dijkstra's Shortest Path Algorithm

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



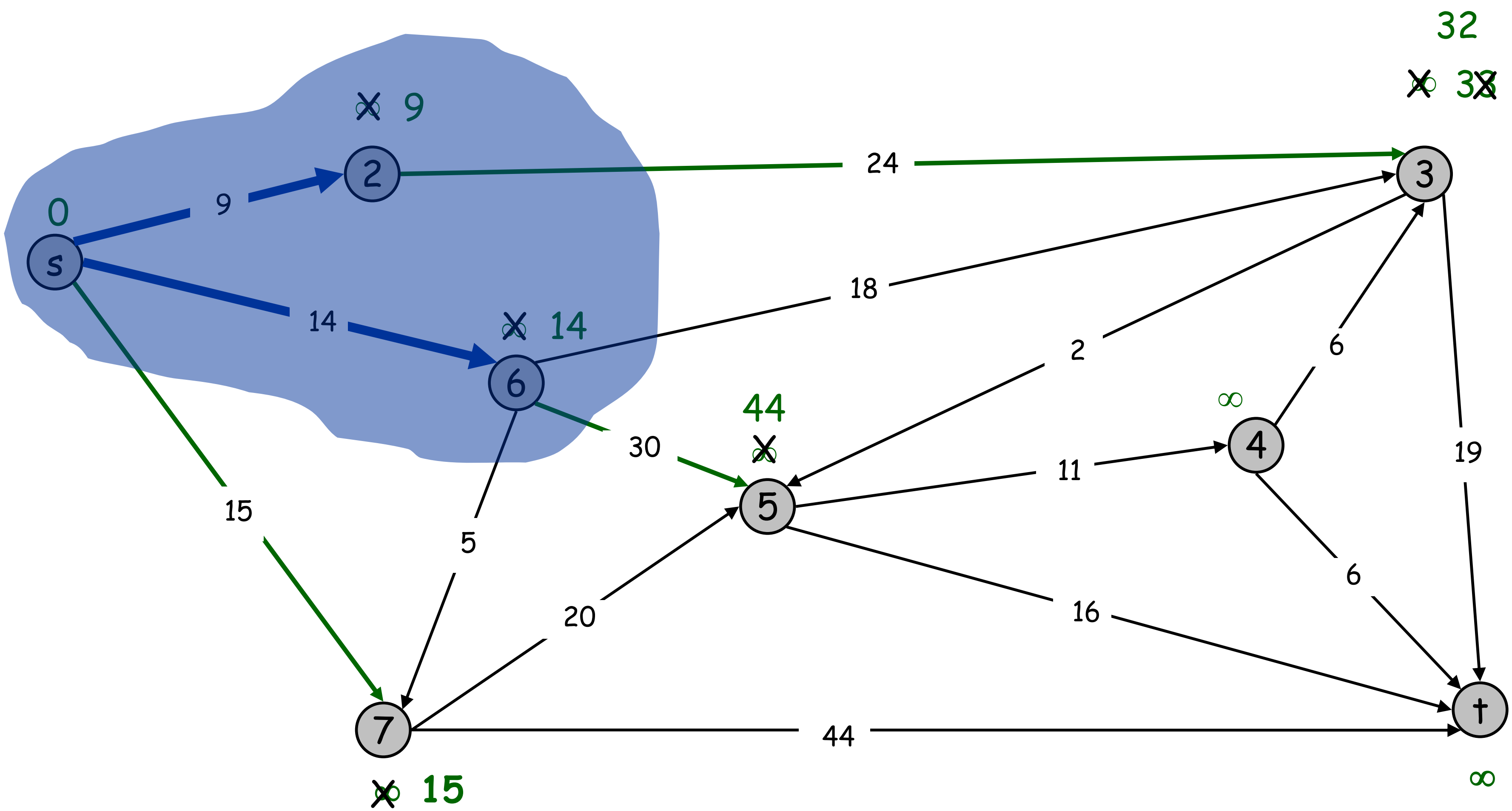
# Dijkstra's Shortest Path Algorithm

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



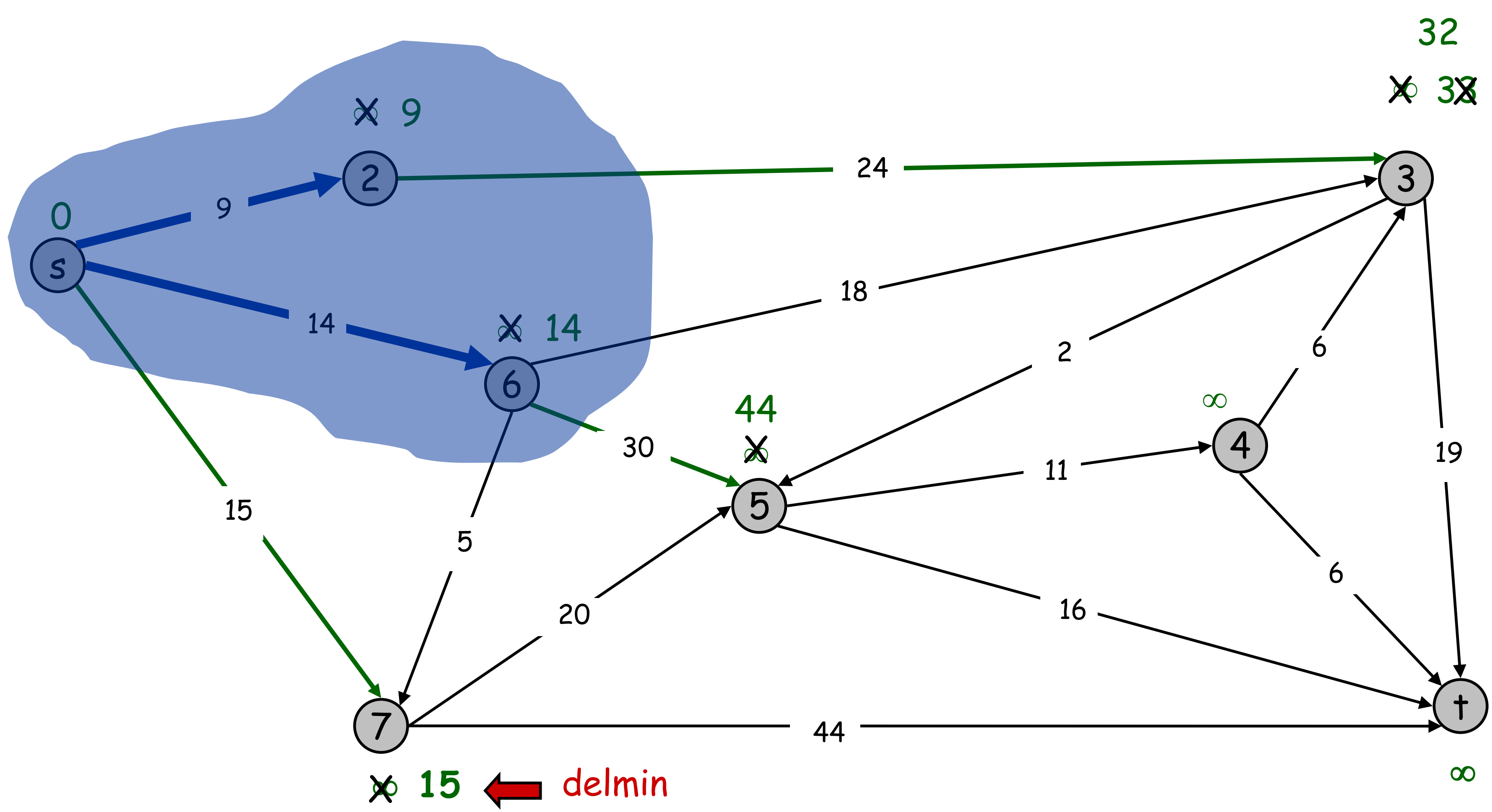
# Dijkstra's Shortest Path Algorithm

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



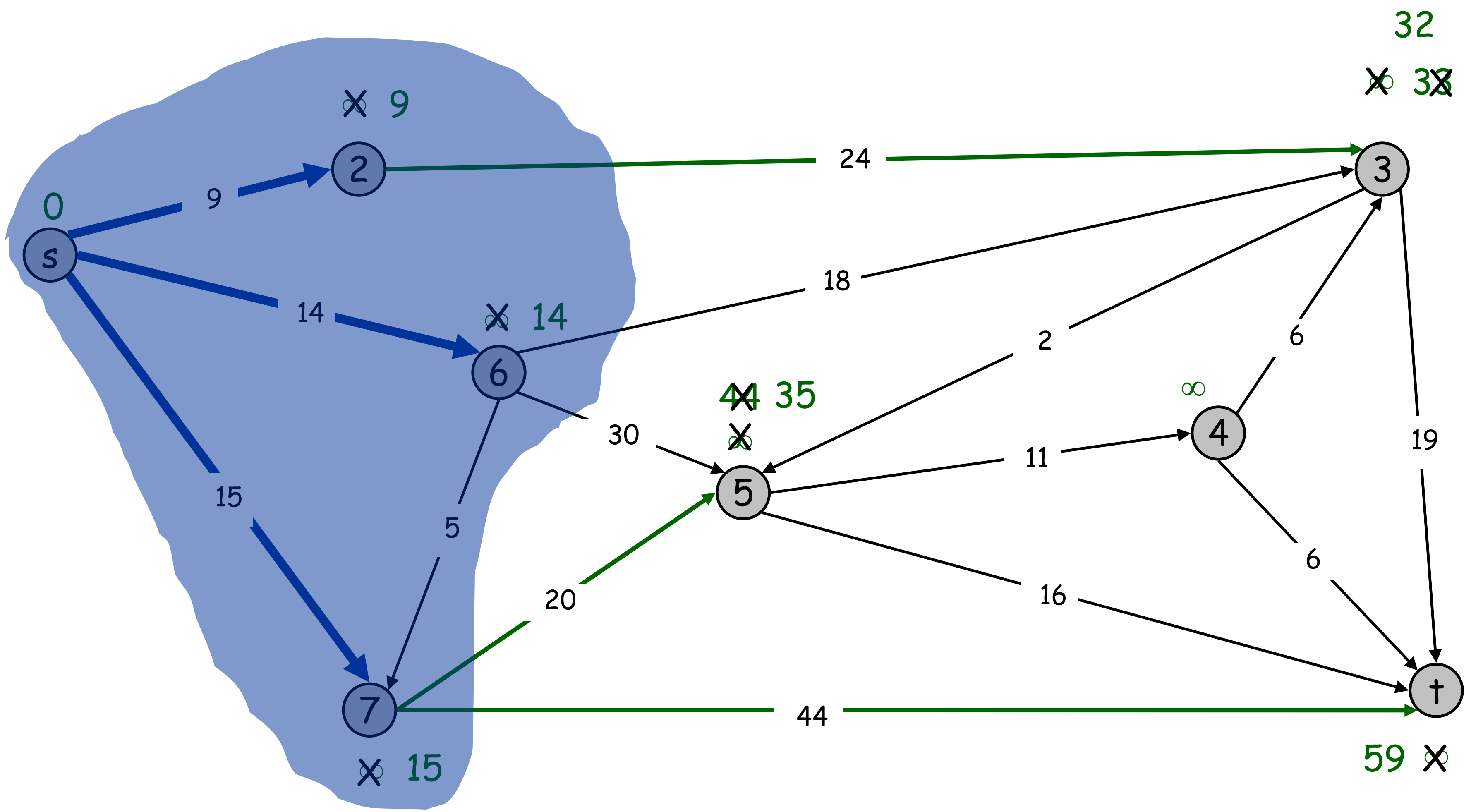
# Dijkstra's Shortest Path Algorithm

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



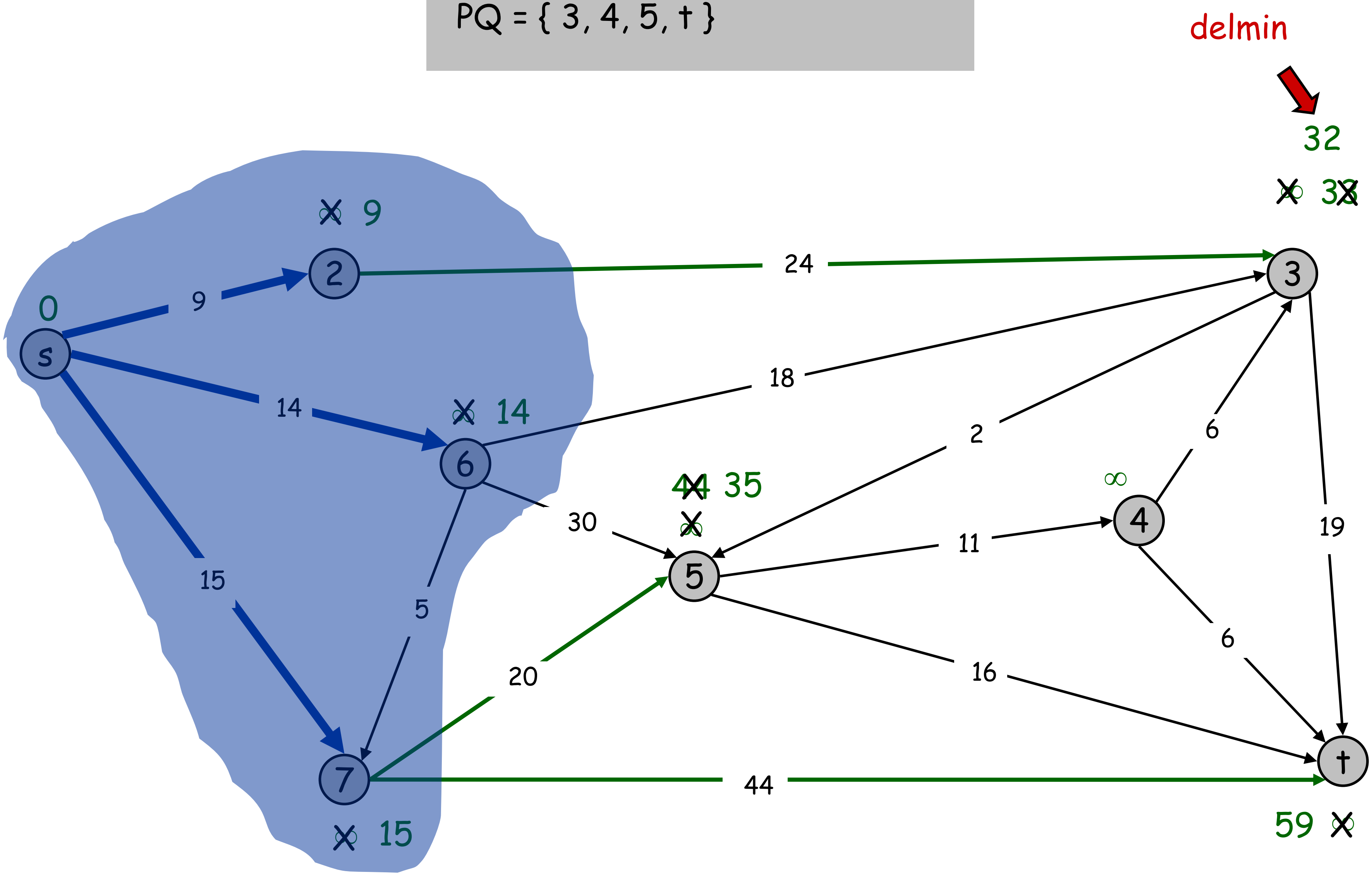
# Dijkstra's Shortest Path Algorithm

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



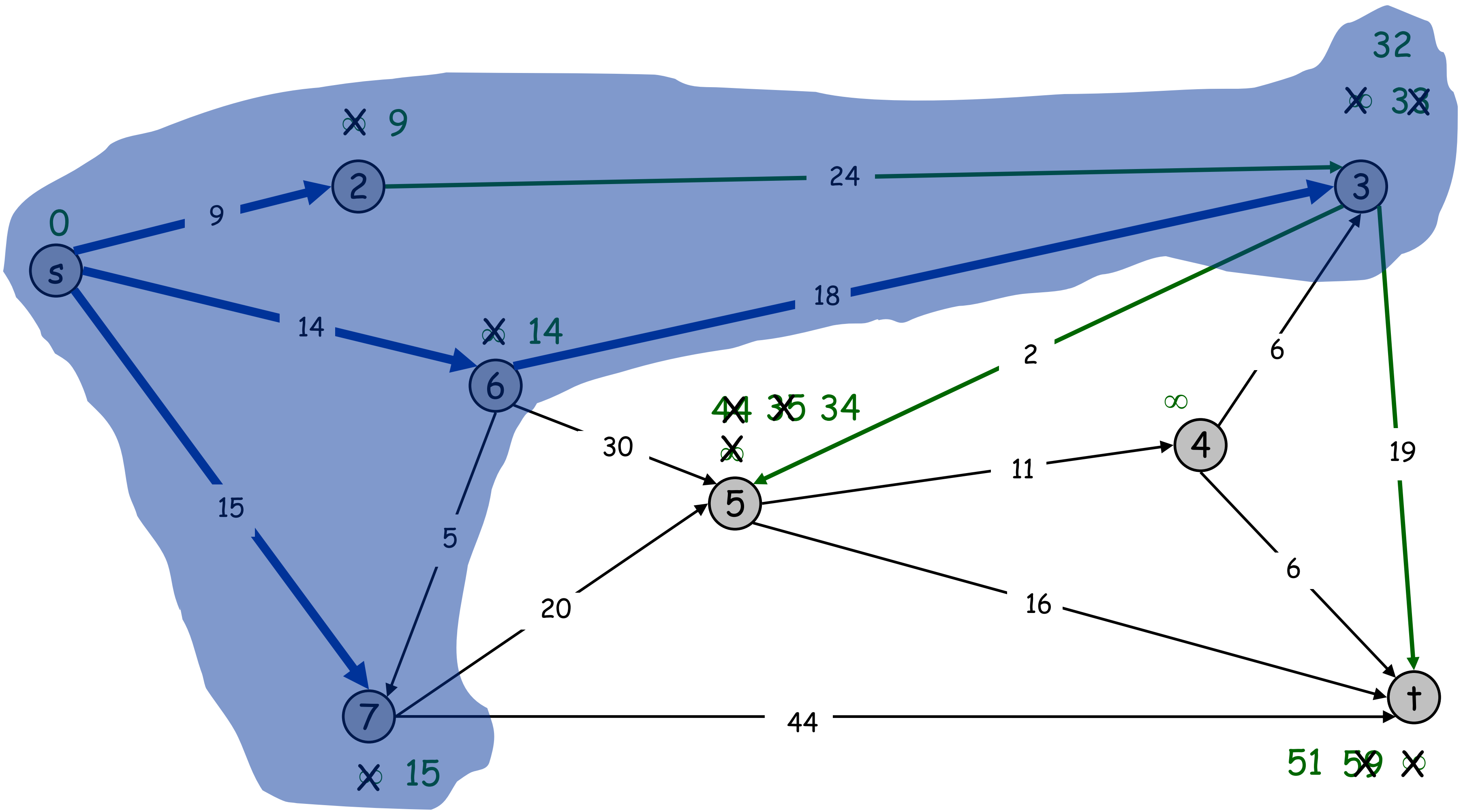
# Dijkstra's Shortest Path Algorithm

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



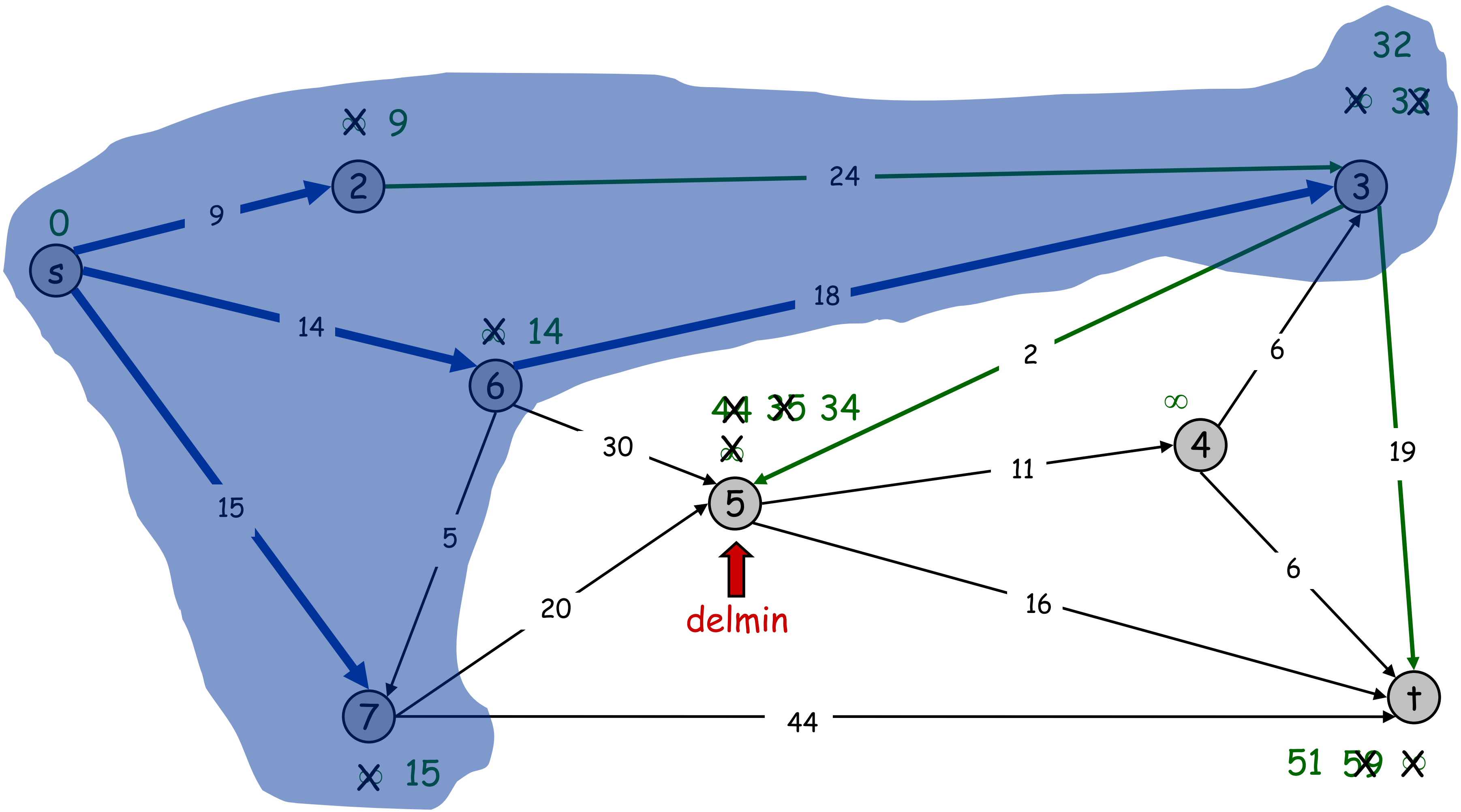
# Dijkstra's Shortest Path Algorithm

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



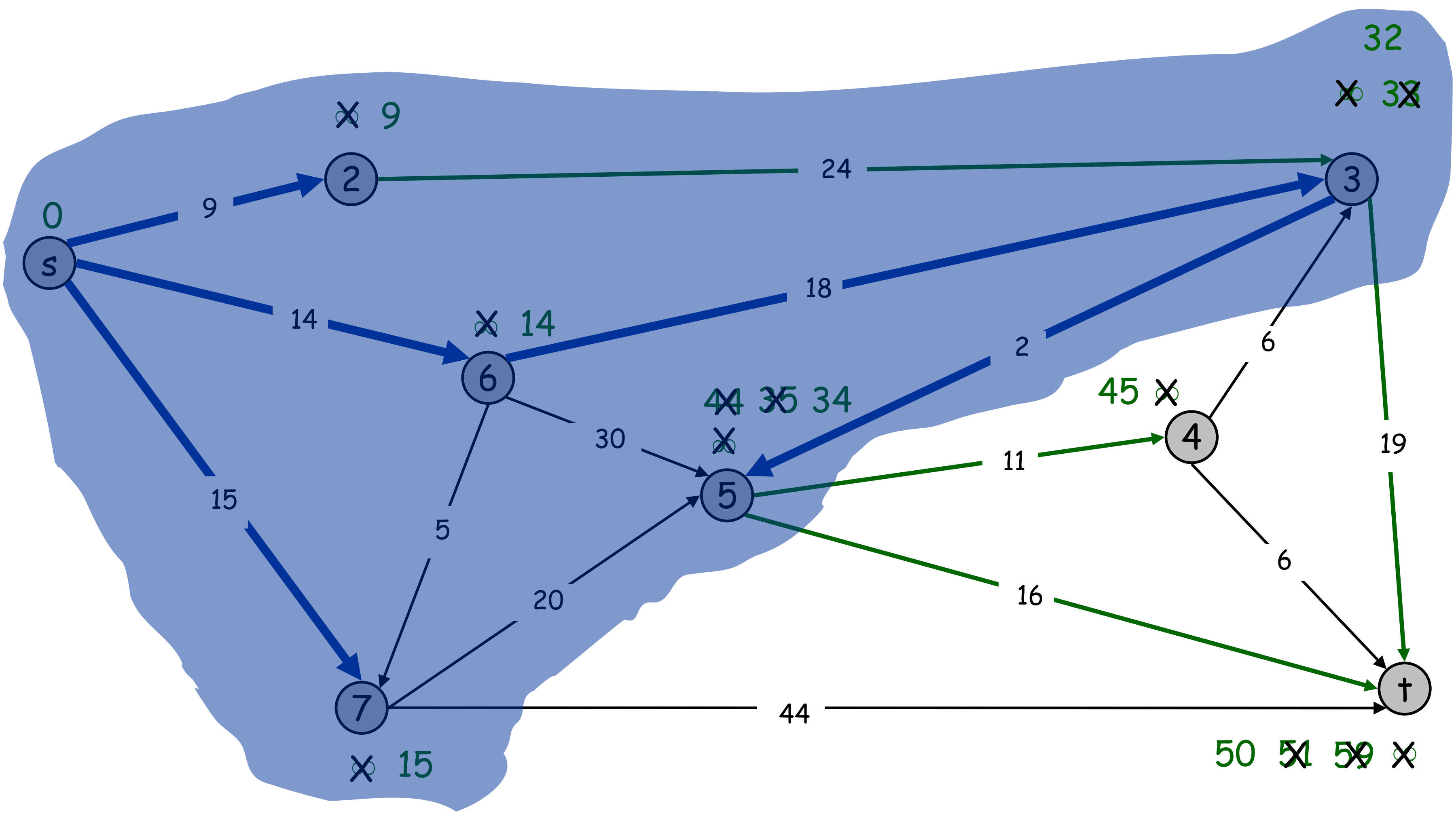
# Dijkstra's Shortest Path Algorithm

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



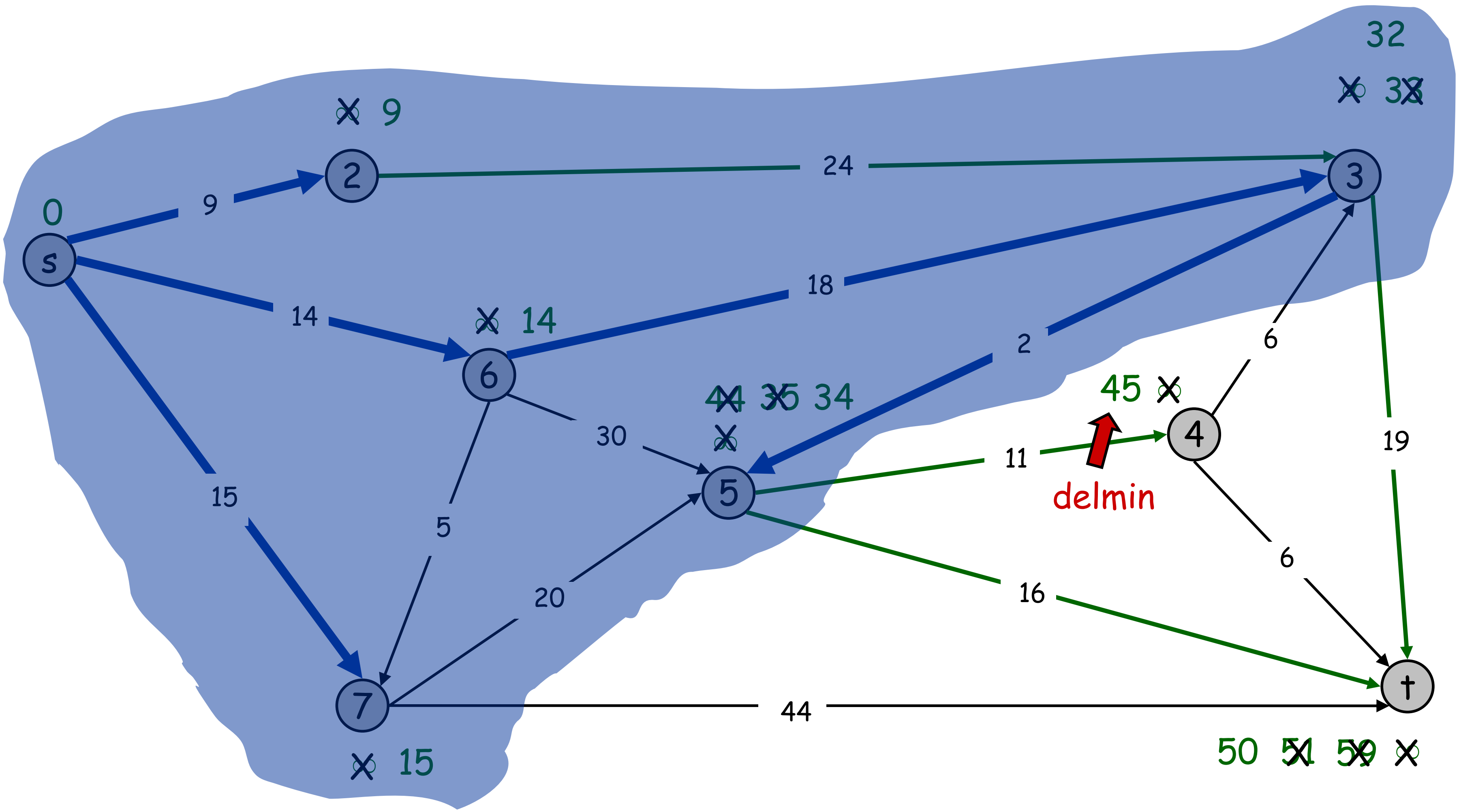
# Dijkstra's Shortest Path Algorithm

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



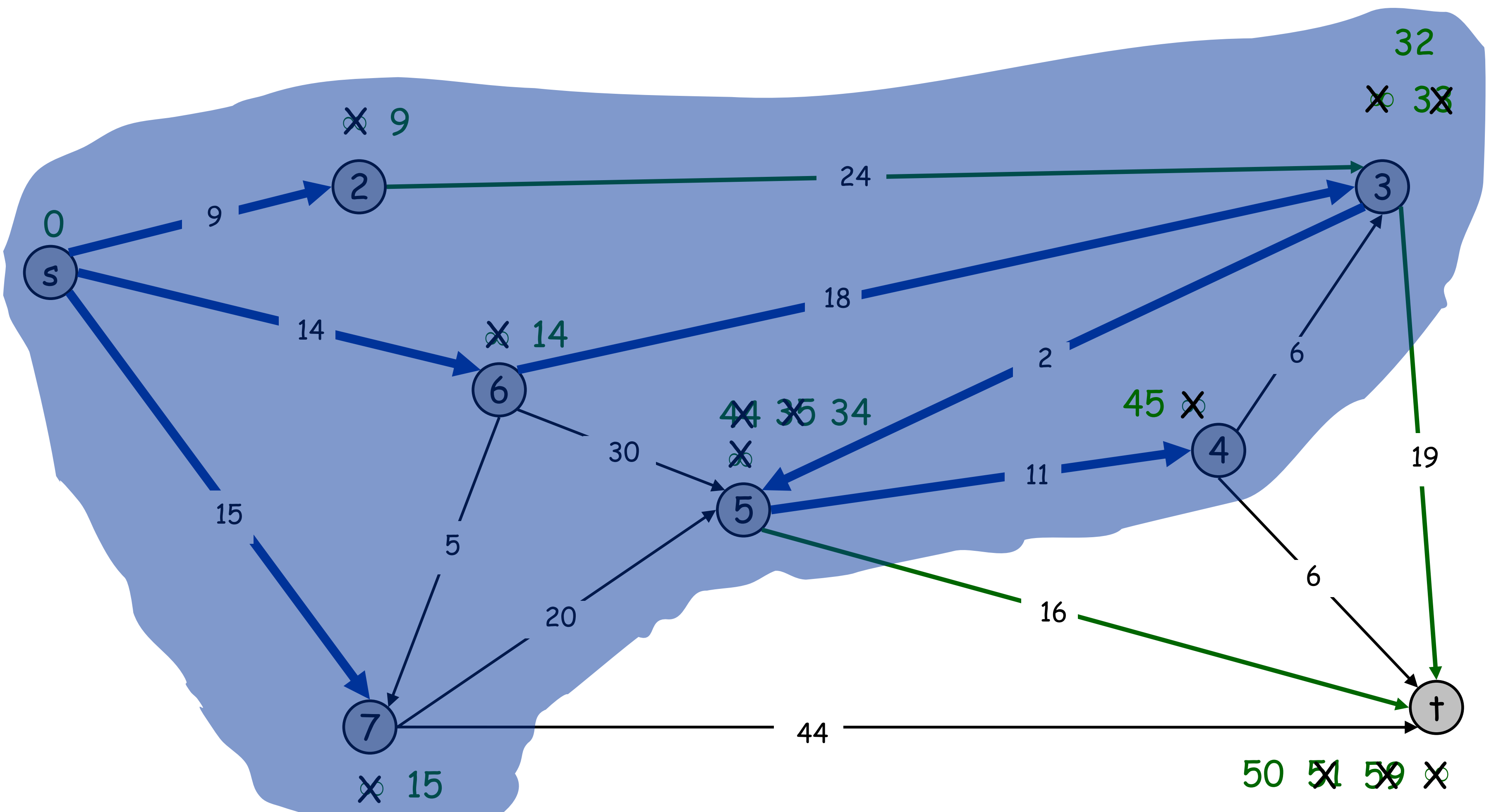
# Dijkstra's Shortest Path Algorithm

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



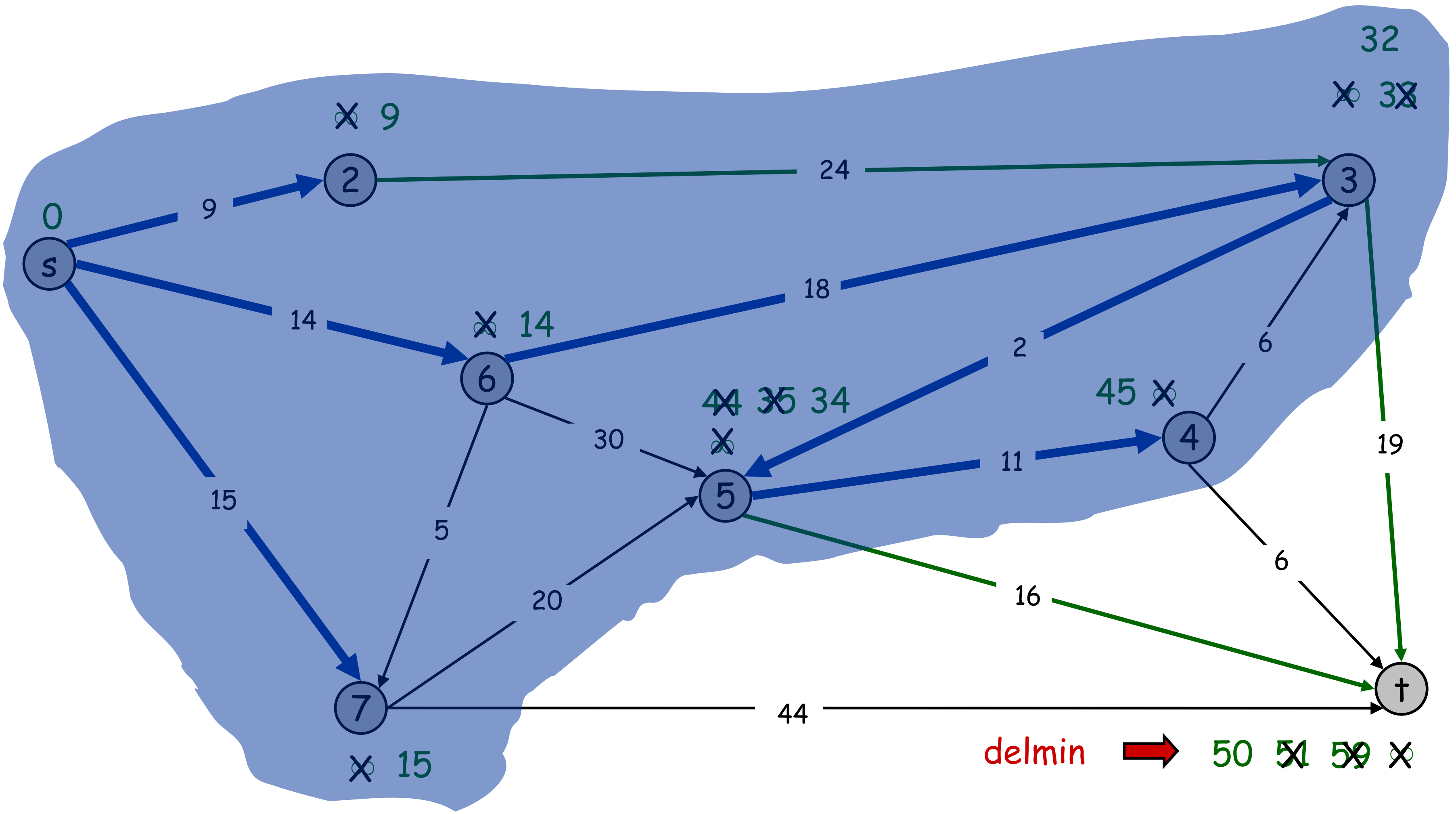
# Dijkstra's Shortest Path Algorithm

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



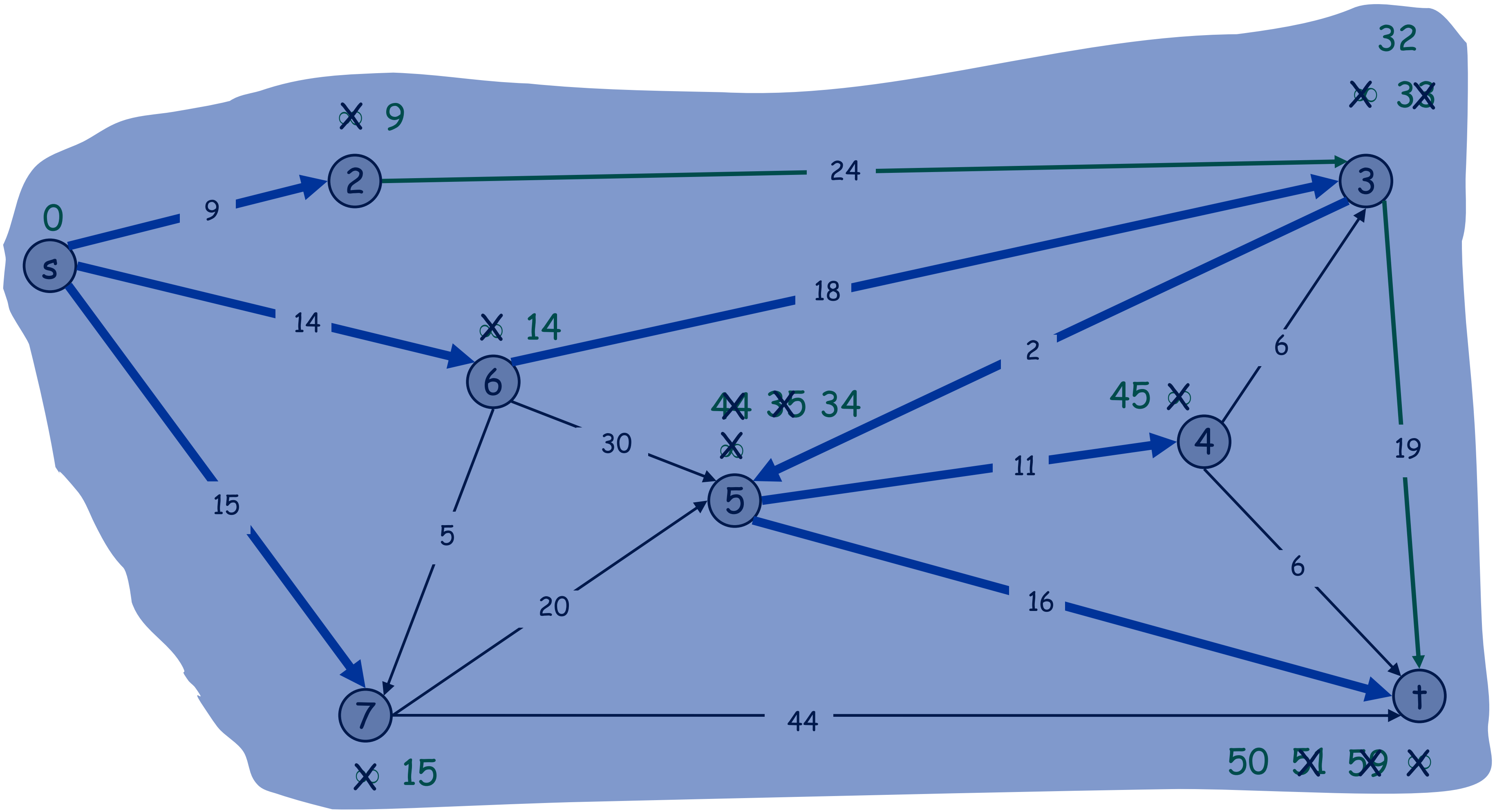
# Dijkstra's Shortest Path Algorithm

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



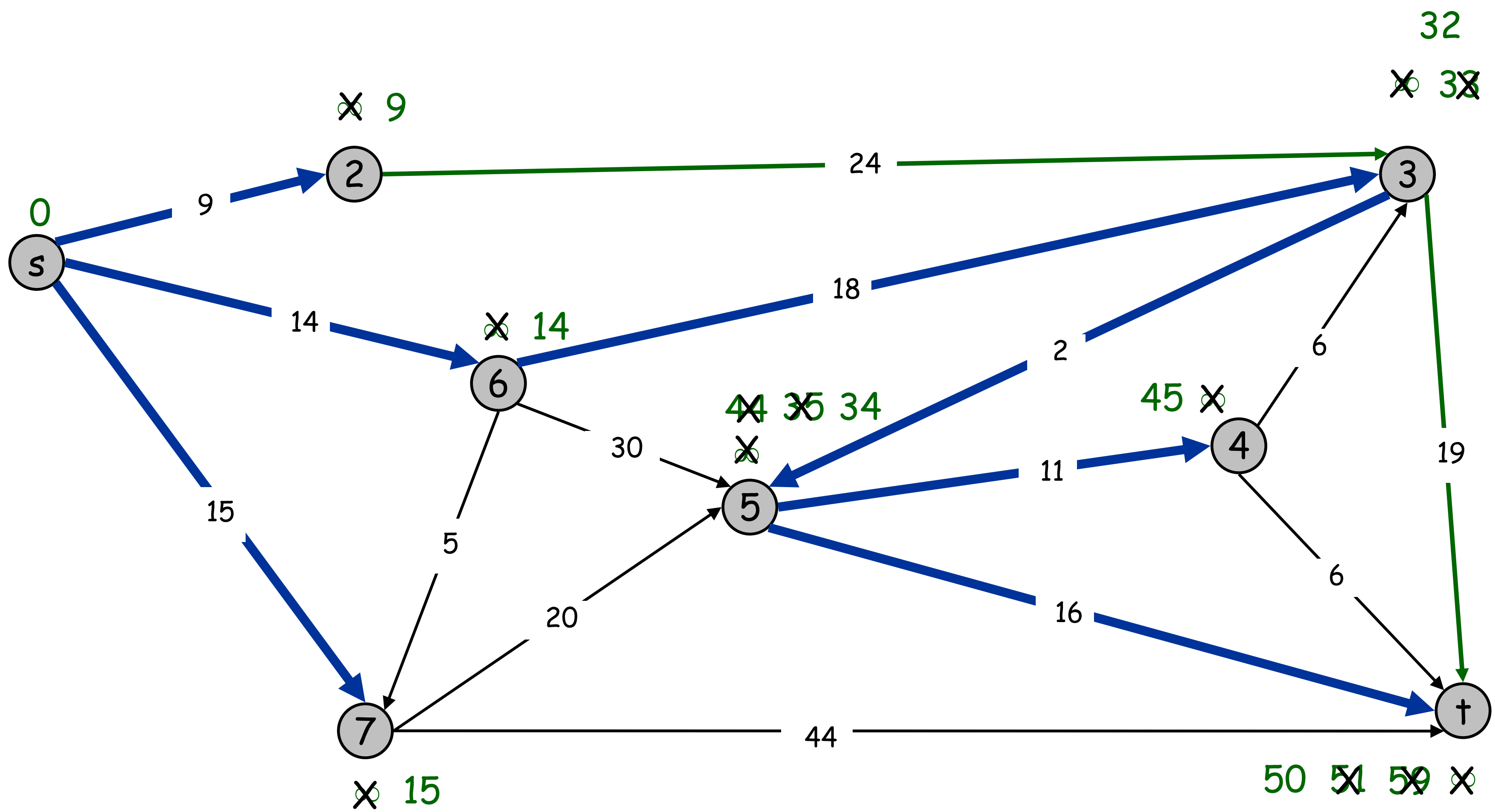
# Dijkstra's Shortest Path Algorithm

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



# Dijkstra's Shortest Path Algorithm

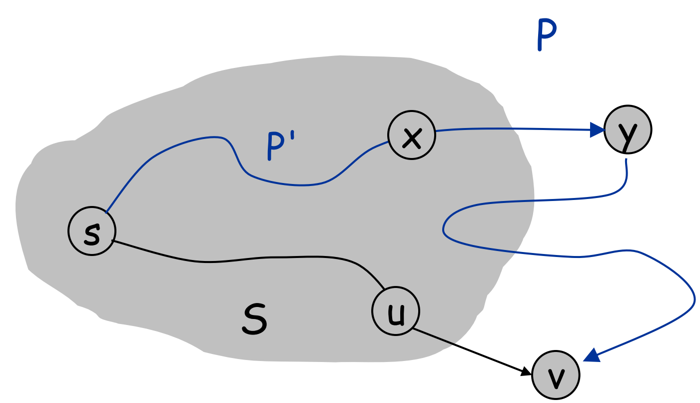
S = { s, 2, 3, 4, 5, 6, 7, t }
PQ = { }



# Dijkstra's Algorithm: Proof of Correctness

Invariant. For each node $u \in S$, $d(u)$ is the length of the shortest s-u path.
Pf. (by induction on $|S|$)
Base case: $|S| = 1$ is trivial.
Inductive hypothesis: Assume true for $|S| = k \geq 1$.

- 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.




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

- $\ell(P) \geq \ell(P') + \ell(x,y)$: nonnegative weights
- $\ell(P') + \ell(x,y) \geq d(x) + \ell(x, y)$: inductive hypothesis
- $d(x) + \ell(x, y) \geq \pi(y)$: defn of $\pi(y)$
- $\pi(y) \geq \pi(v)$: Dijkstra chose v instead of y