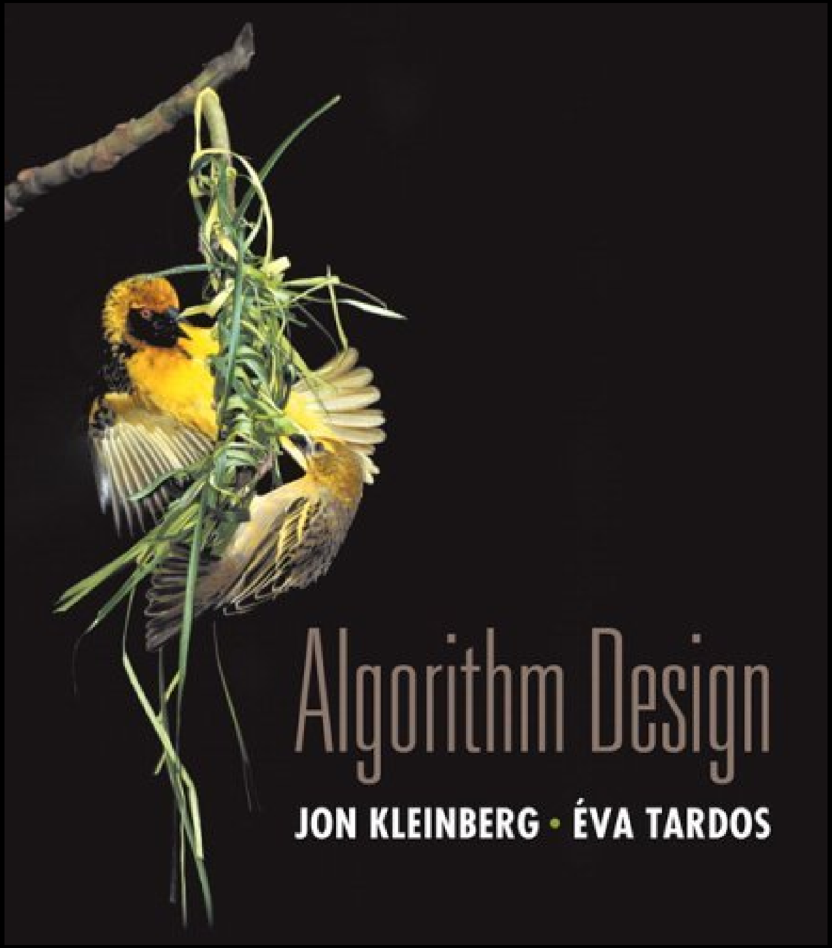
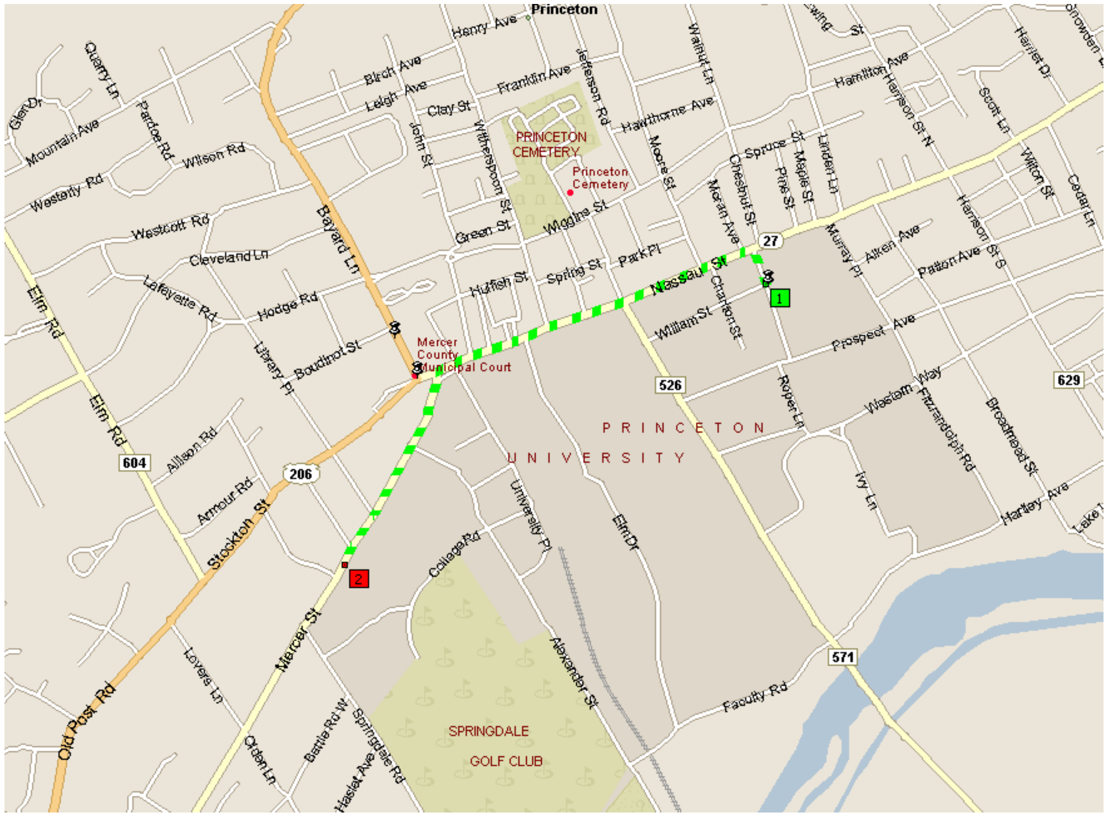


Chapter 4

Shortest Path Greedy Algorithms

Shortest Paths in a Graph



shortest path from Princeton CS department to Einstein's house

Shortest Path Problem

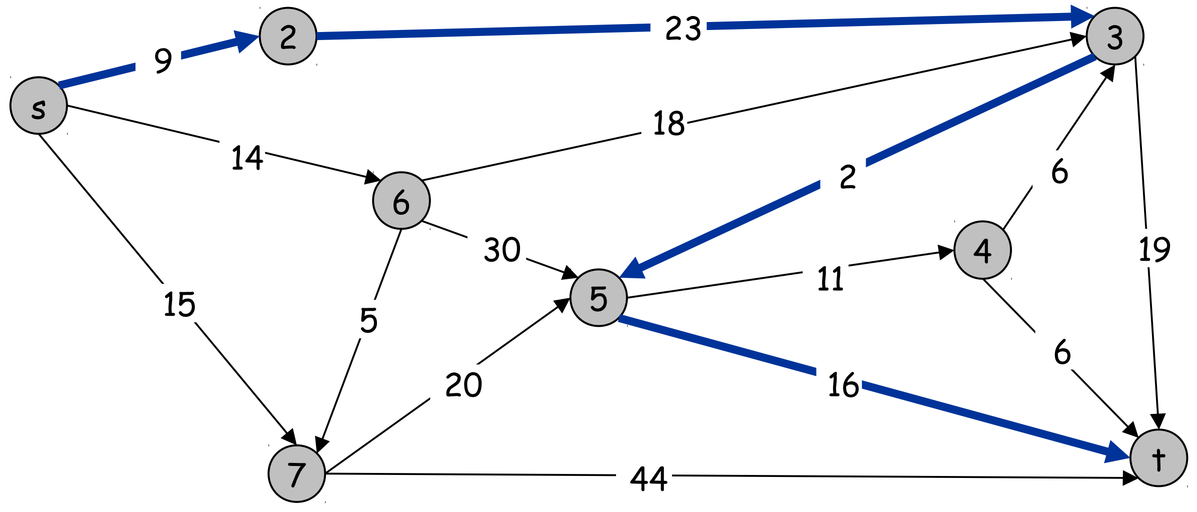
Shortest path network.

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

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

$\uparrow$

cost of path = sum of edge costs in path

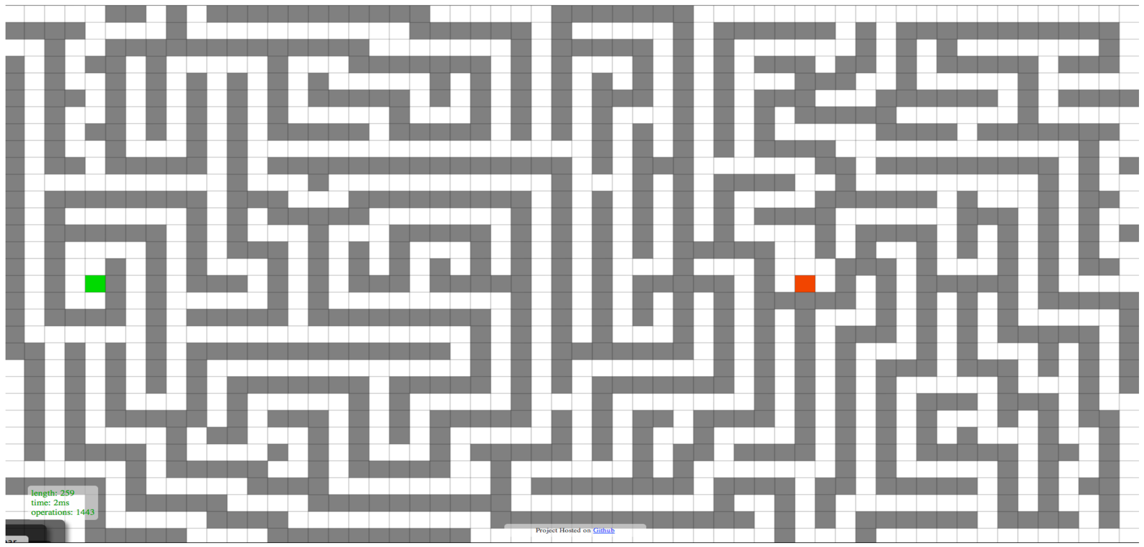


Cost of path s-2-3-5-t
= 9 + 23 + 2 + 16
= 48.

Shortest path applications

Applications of shortest path include but are not limited to:

- Generating directions for maps (Google Maps / GPS)
- Finding solutions to puzzles with states (Rubik's Cube)
- Optimal routing in a network of computers
- Finding arbitrage opportunities in currency exchange
- Robot navigation
- Pathfinding in computer games


length: 259
time: 2ms
operations: 1443

Dijkstra's Algorithm

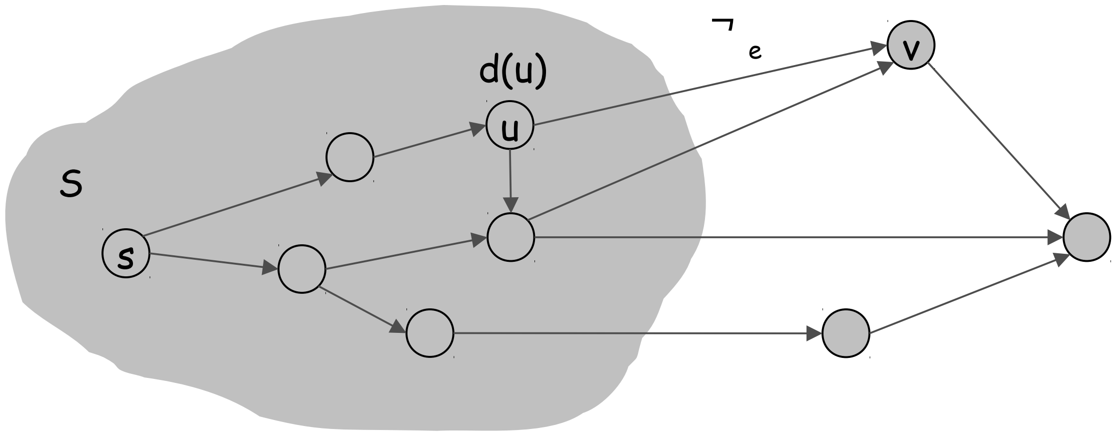
Dijkstra's algorithm.

- 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

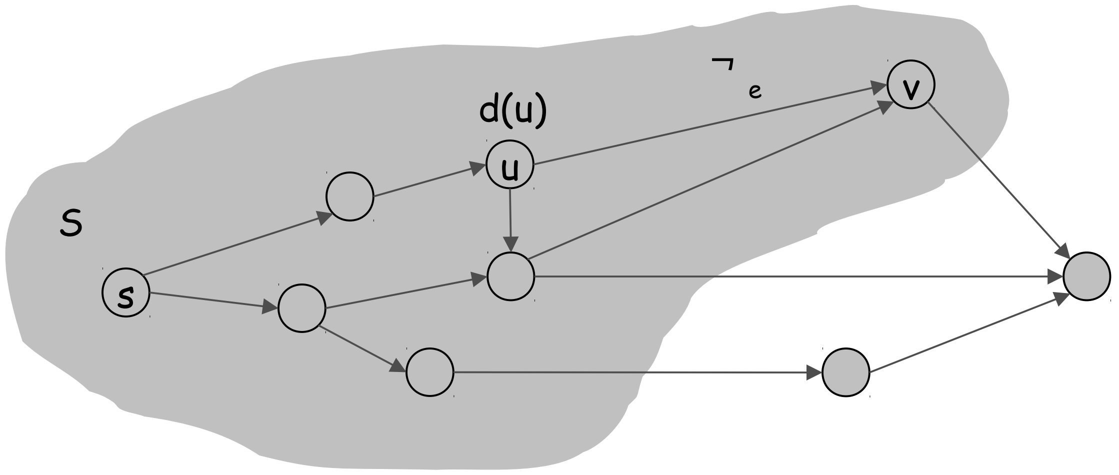
Dijkstra's algorithm.

- 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)

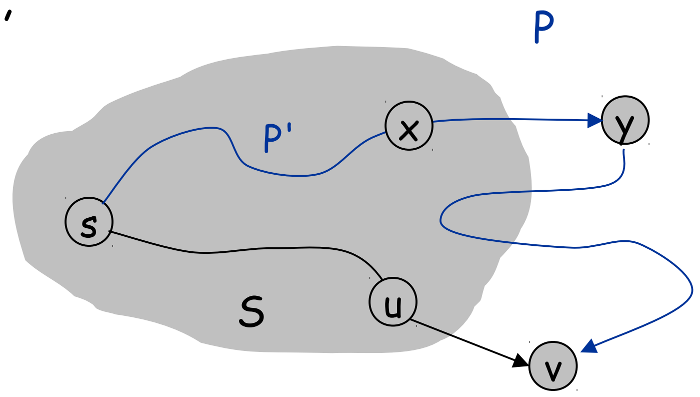
add v to S, and set d(v) = π(v).



Dijkstra's Algorithm: 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.



$$\neg\ (P) \underset{\substack{\uparrow \\ \text{nonnegative} \\ \text{weights}}}{\geq} \neg\ (P') + \neg\ (x,y) \underset{\substack{\uparrow \\ \text{inductive} \\ \text{hypothesis}}}{\geq} d(x) + \neg\ (x, y) \underset{\substack{\uparrow \\ \text{defn of } \pi(y)}}{\geq} \pi(y) \underset{\substack{\uparrow \\ \text{Dijkstra chose v} \\ \text{instead of y}}}{\geq} \pi(v)$$

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 π(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 π(v).

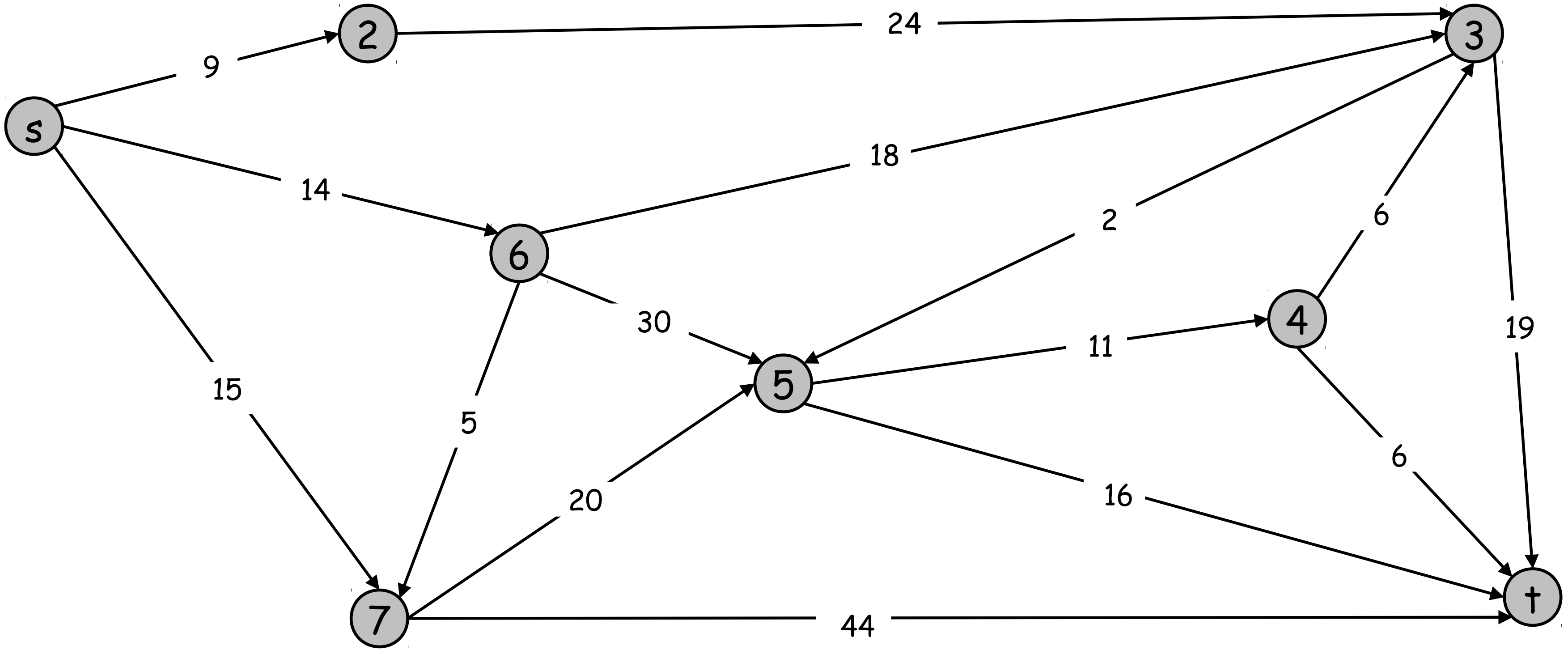
PQ Operation	Dijkstra	Array	Binary heap	d-way Heap	Fib heap †
Insert	n	n	log n	$d \log_d n$	1
ExtractMin	n	n	log n	$d \log_d n$	log n
ChangeKey	m	1	log n	$\log_d n$	1
IsEmpty	n	1	1	1	1
Total		n^2	m log n	$m \log_{m/n} n$	m + n log n

† Individual ops are amortized bounds

Dijkstra Example

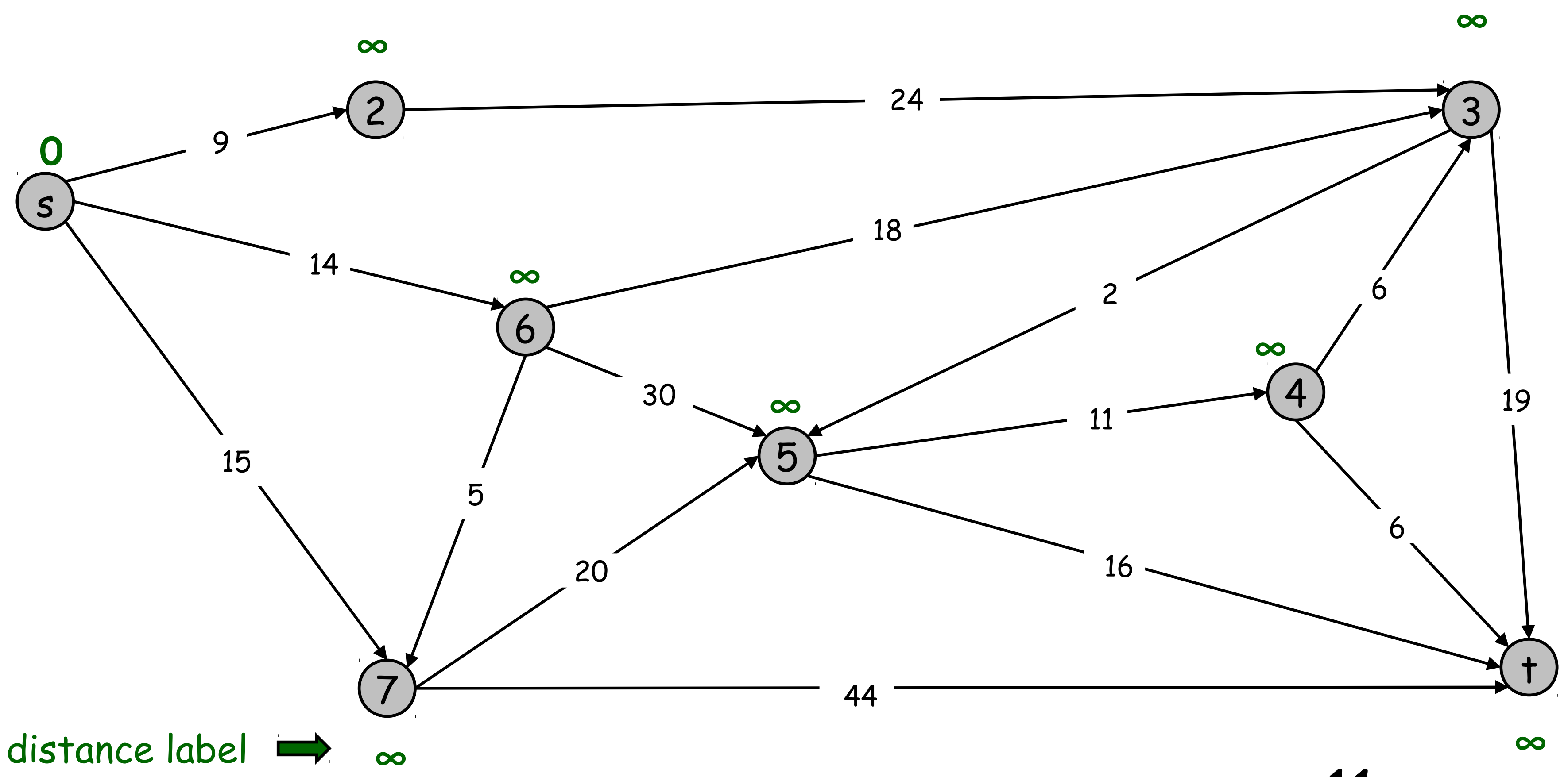
Dijkstra's Shortest Path Algorithm

Find shortest path from s to t.



Dijkstra's Shortest Path Algorithm

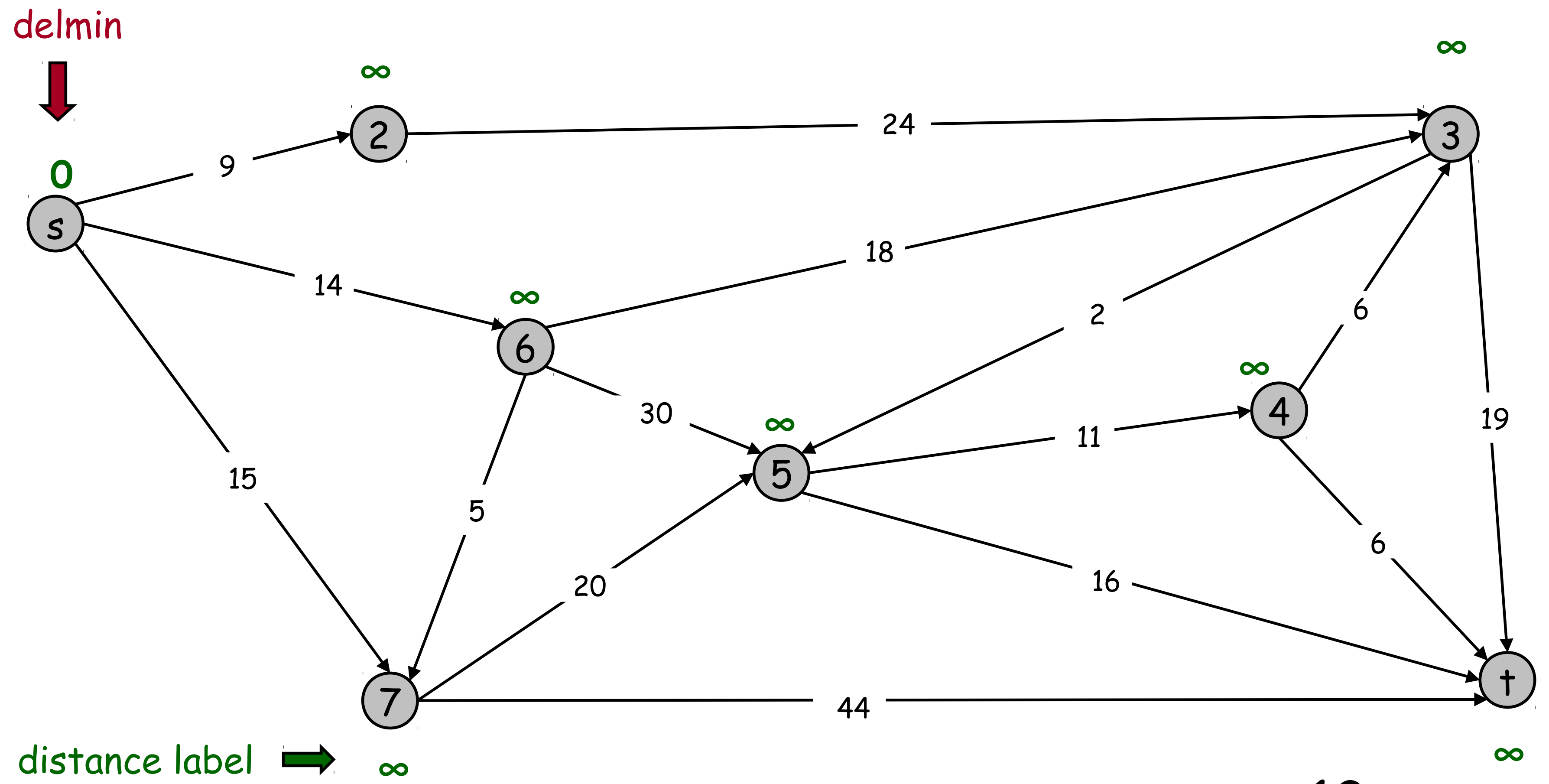
S = { }
PQ = { s, 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

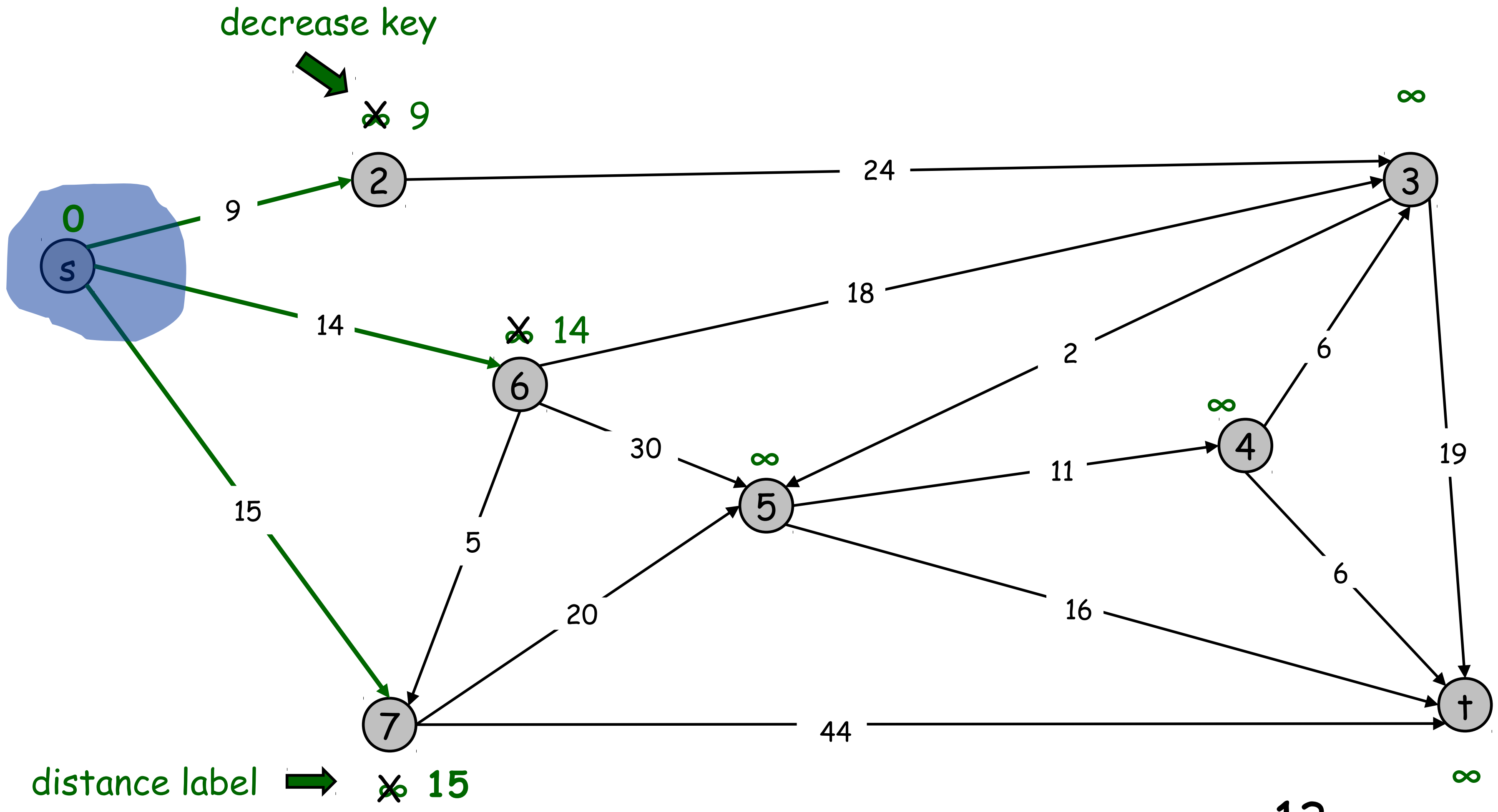
S = { }

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



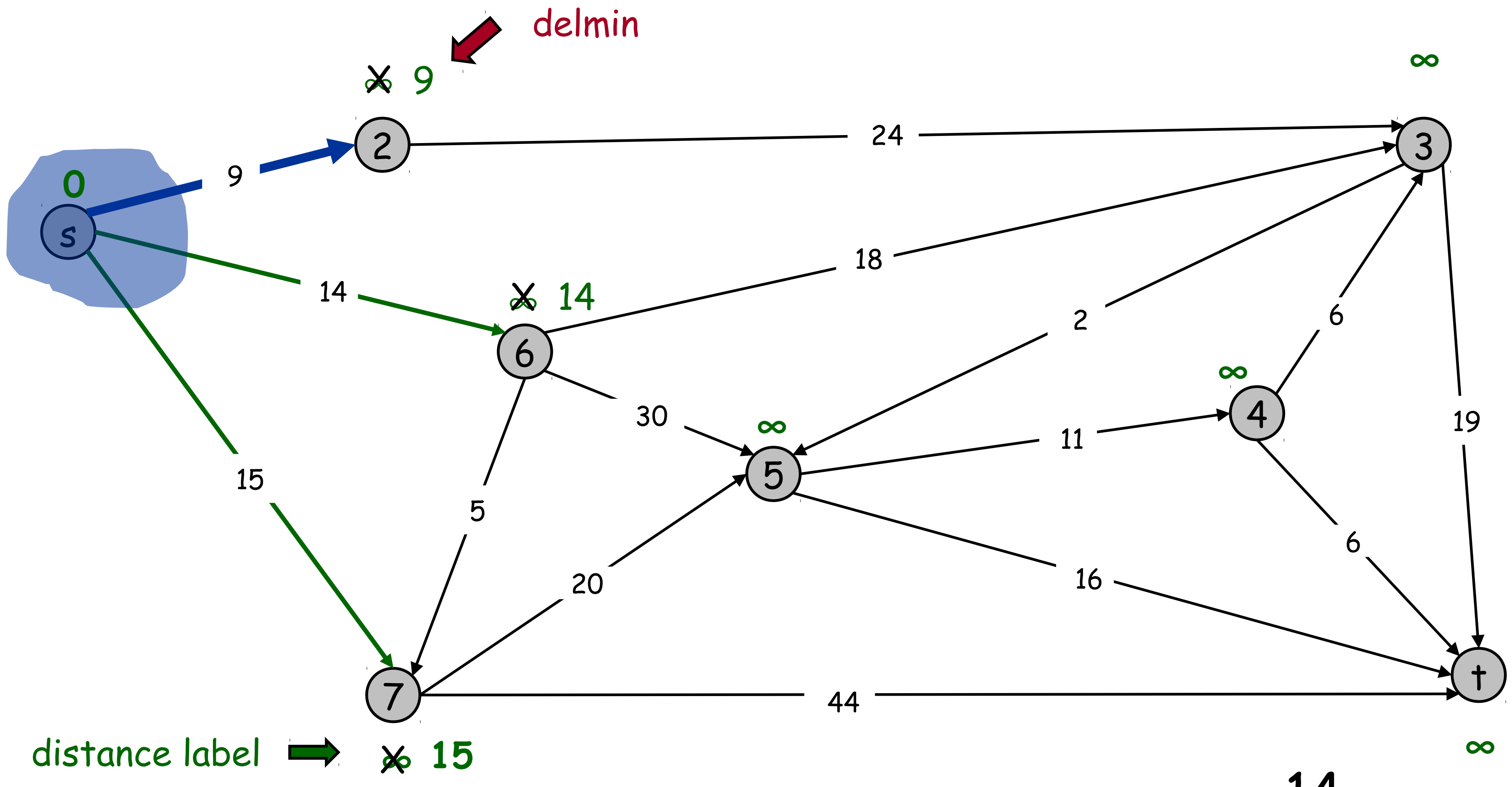
Dijkstra's Shortest Path Algorithm

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



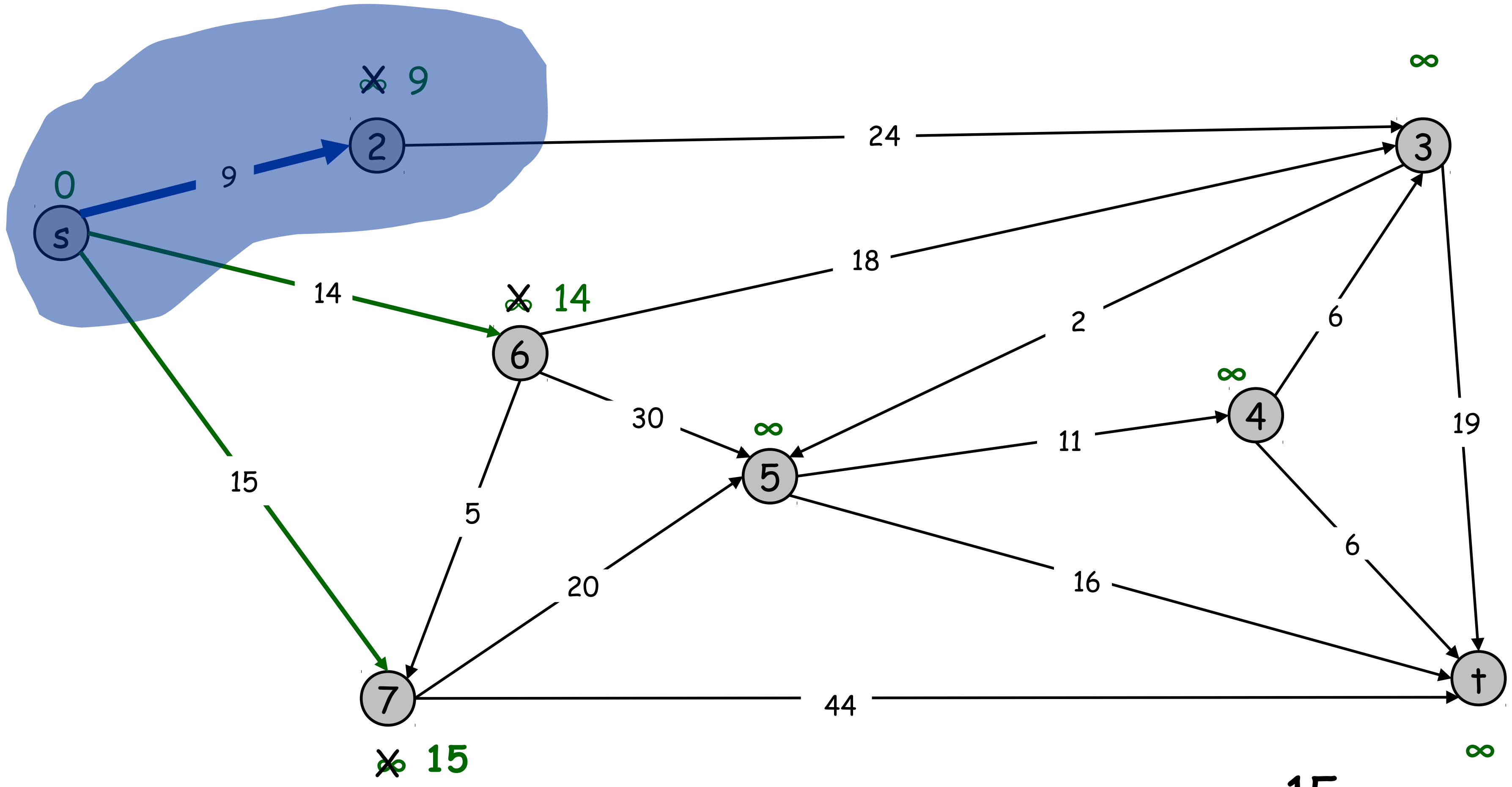
Dijkstra's Shortest Path Algorithm

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



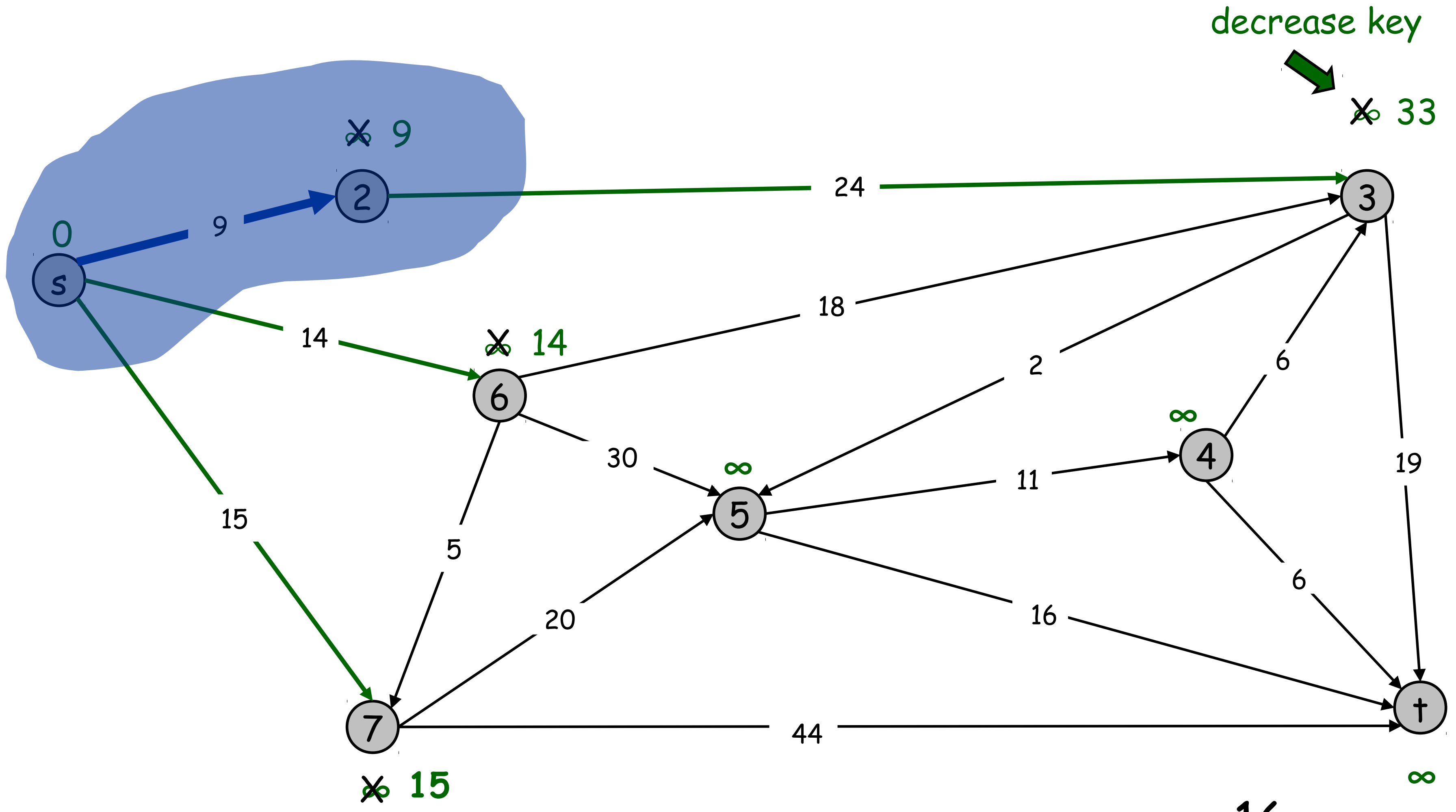
Dijkstra's Shortest Path Algorithm

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



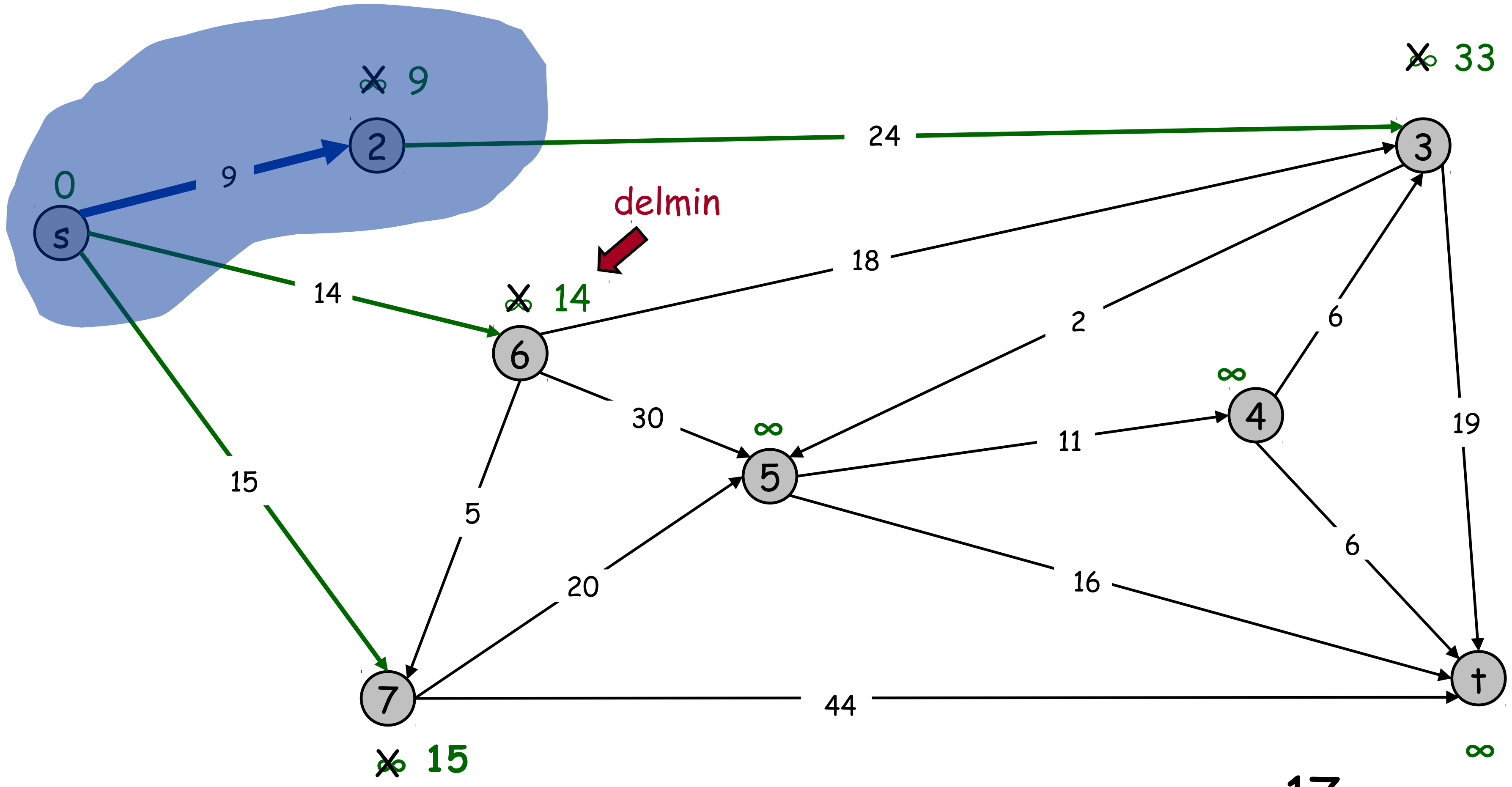
Dijkstra's Shortest Path Algorithm

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



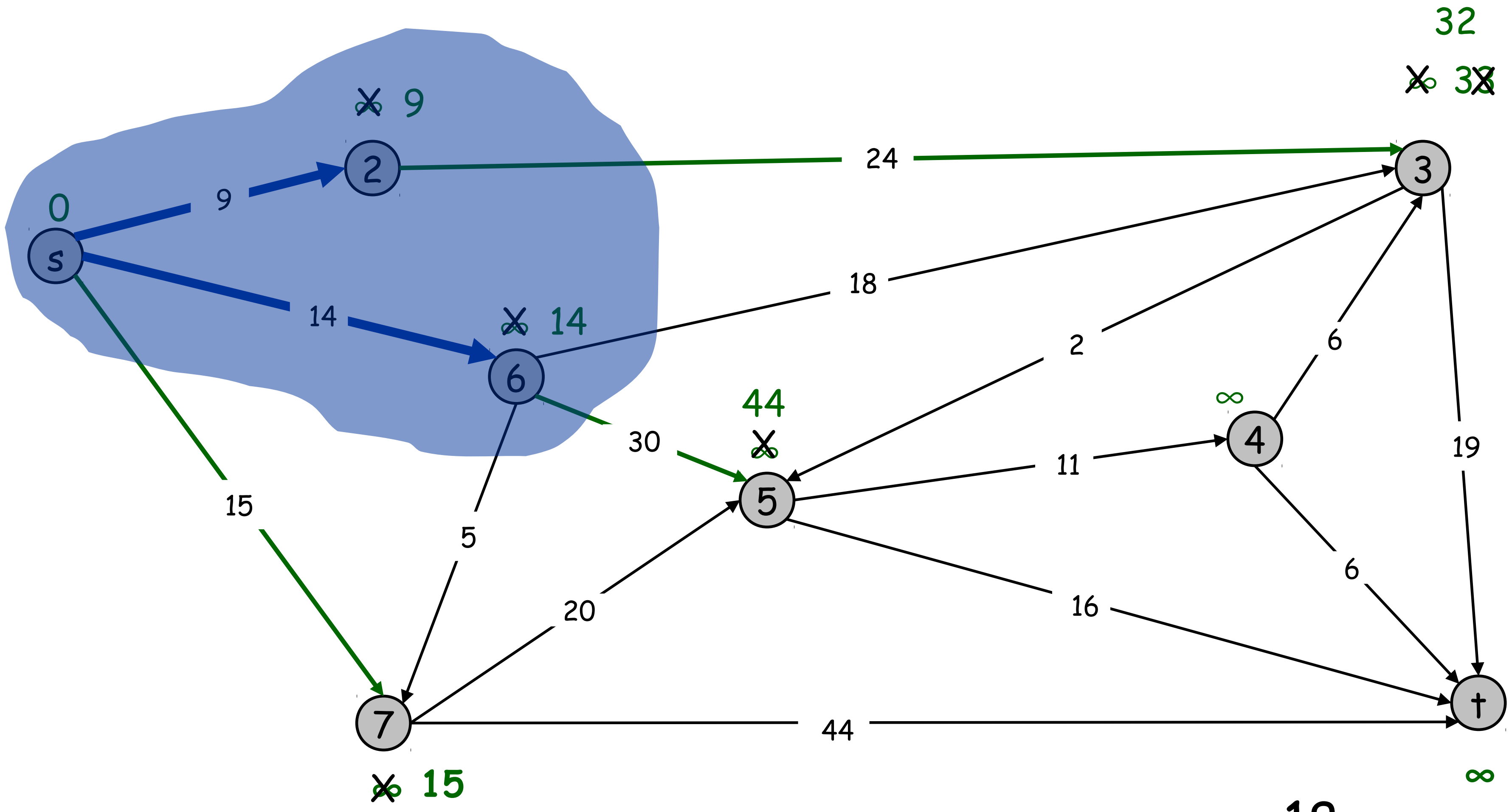
Dijkstra's Shortest Path Algorithm

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



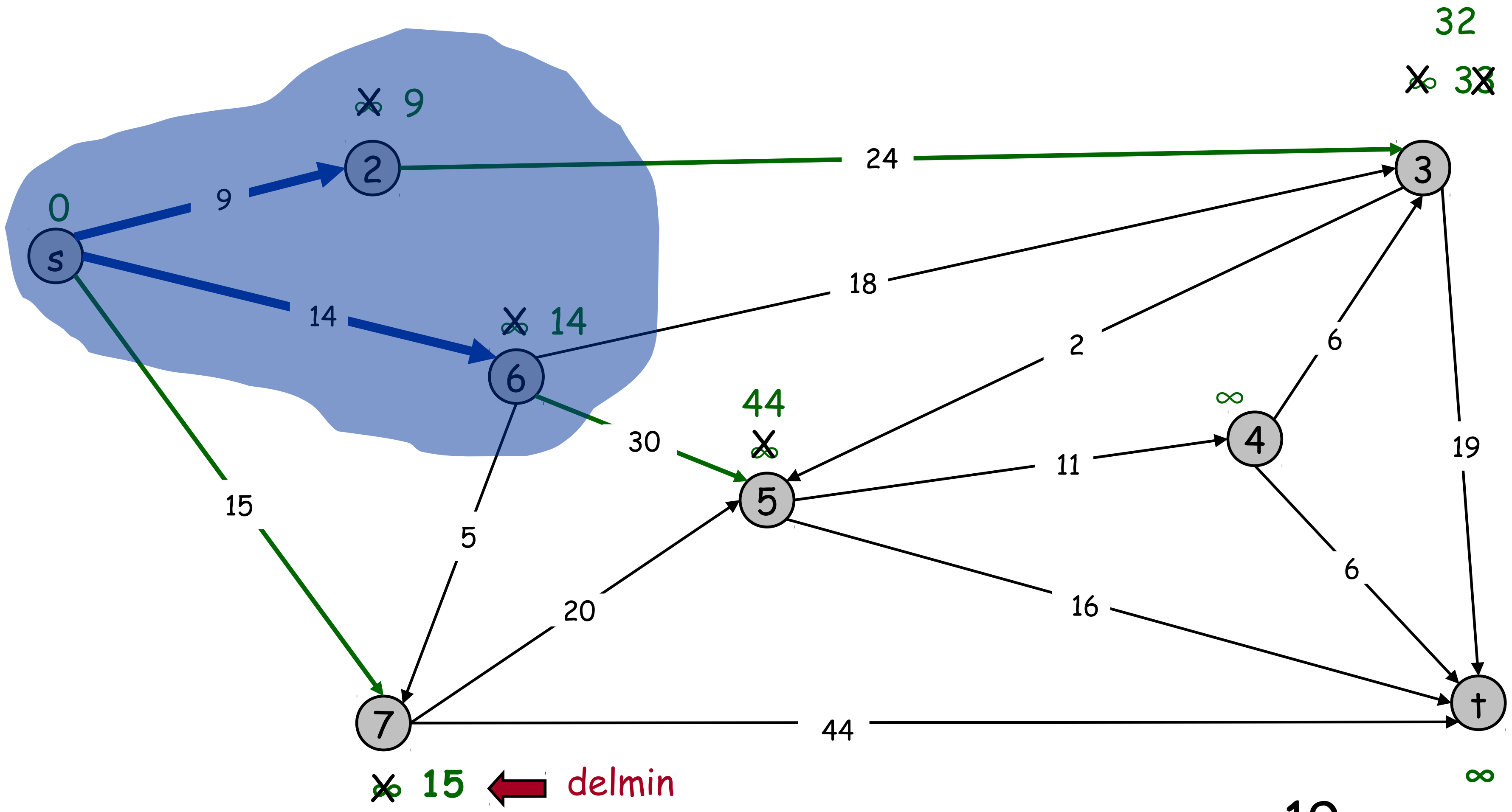
Dijkstra's Shortest Path Algorithm

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



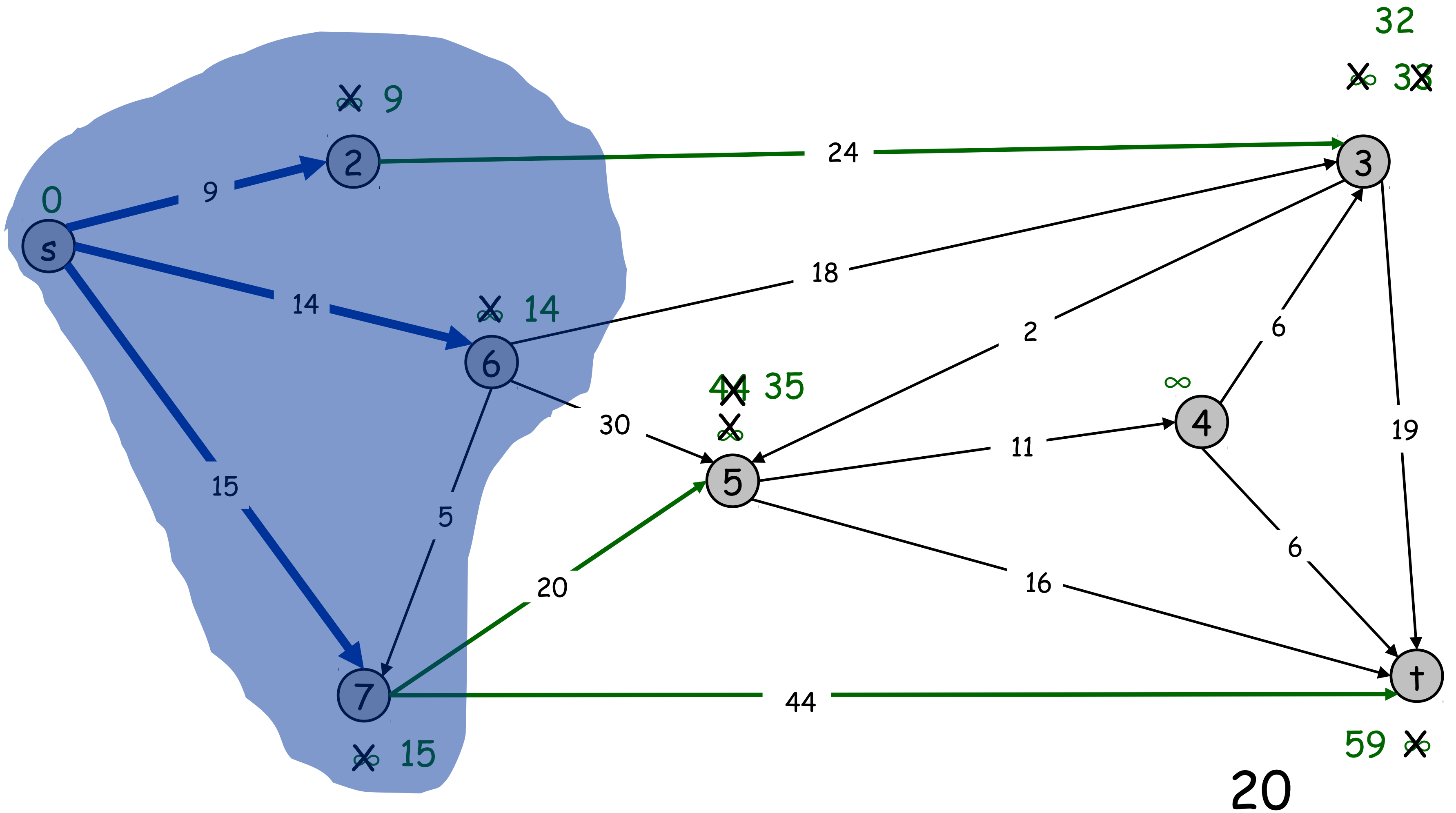
Dijkstra's Shortest Path Algorithm

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



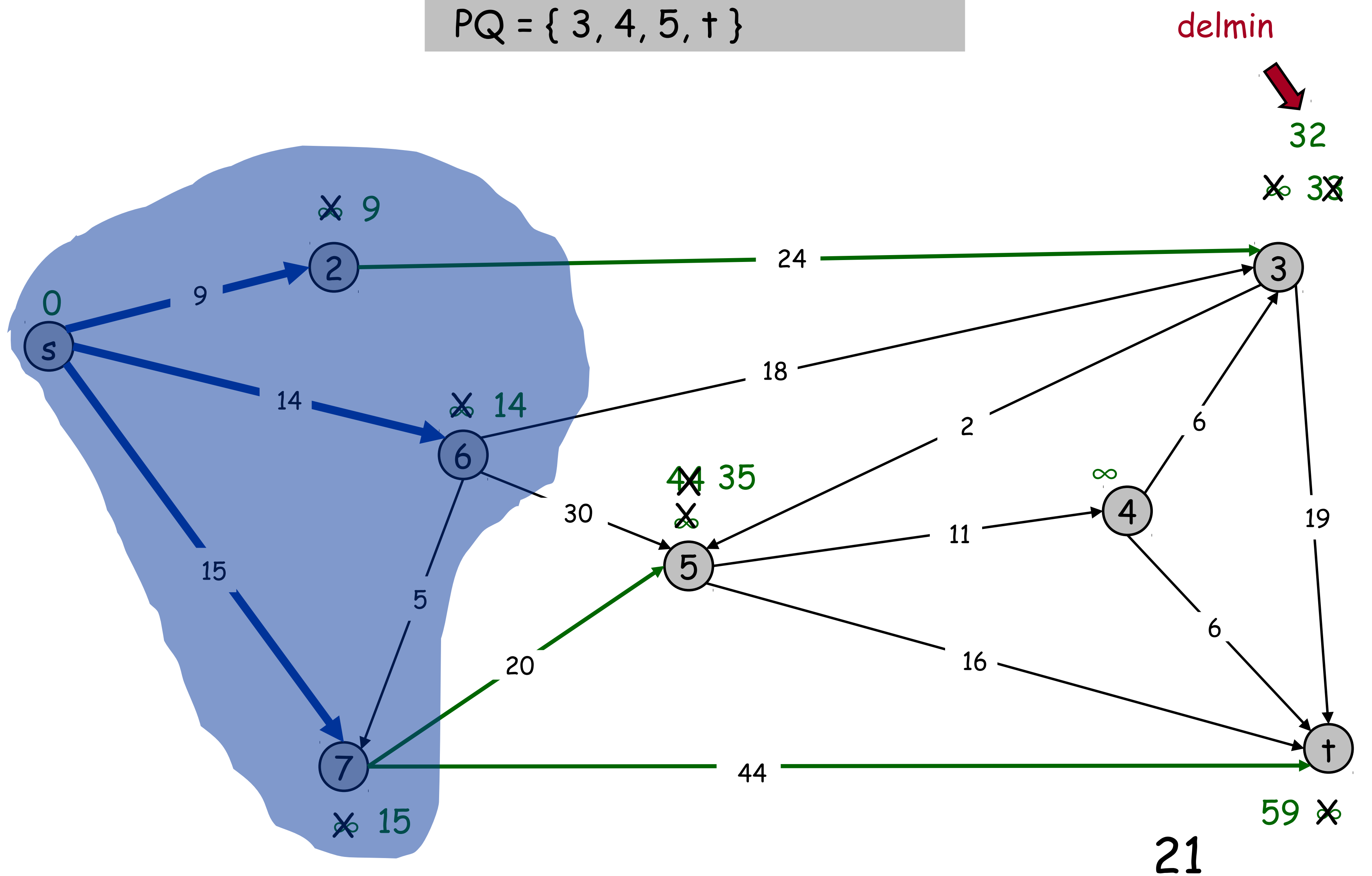
Dijkstra's Shortest Path Algorithm

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



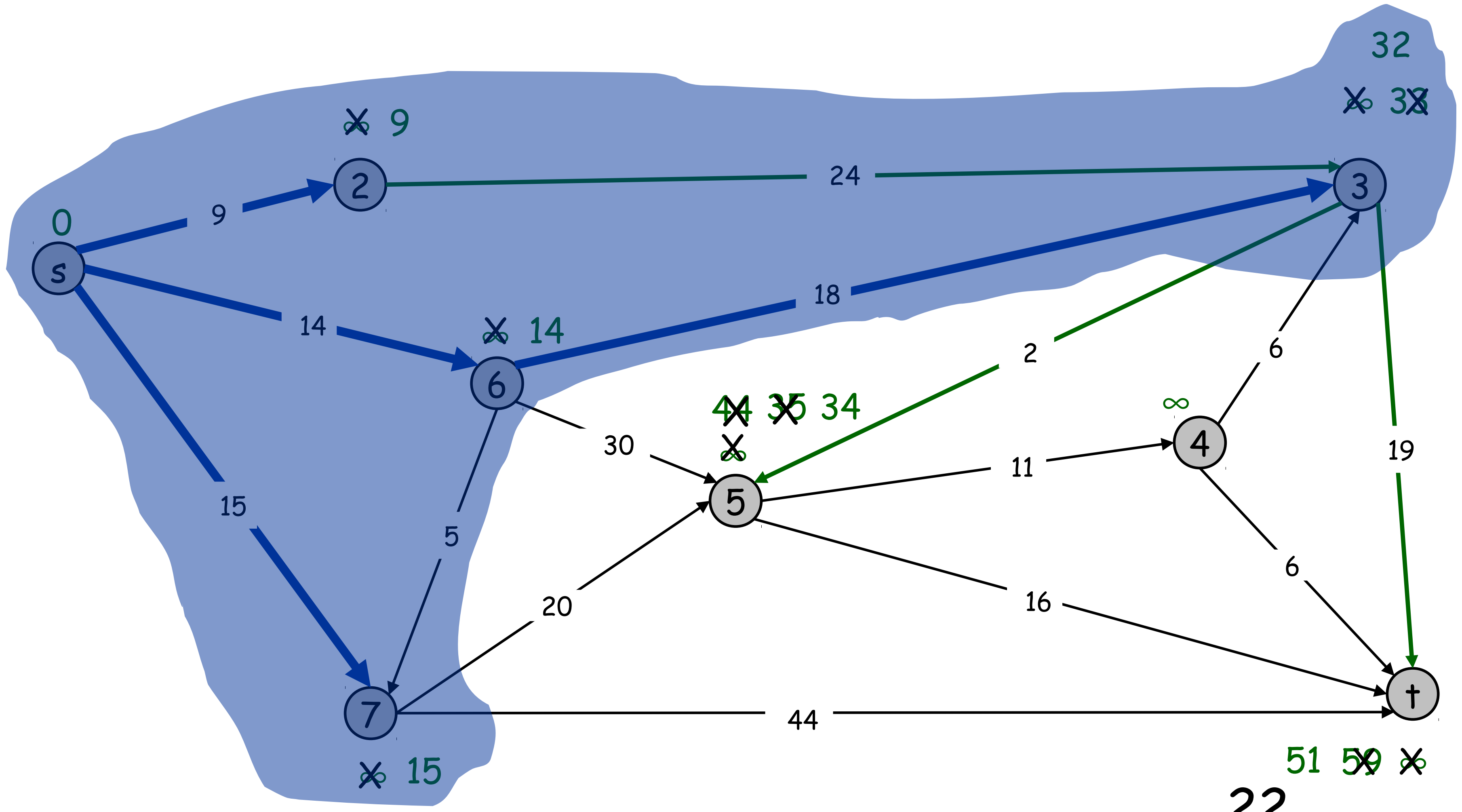
Dijkstra's Shortest Path Algorithm

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



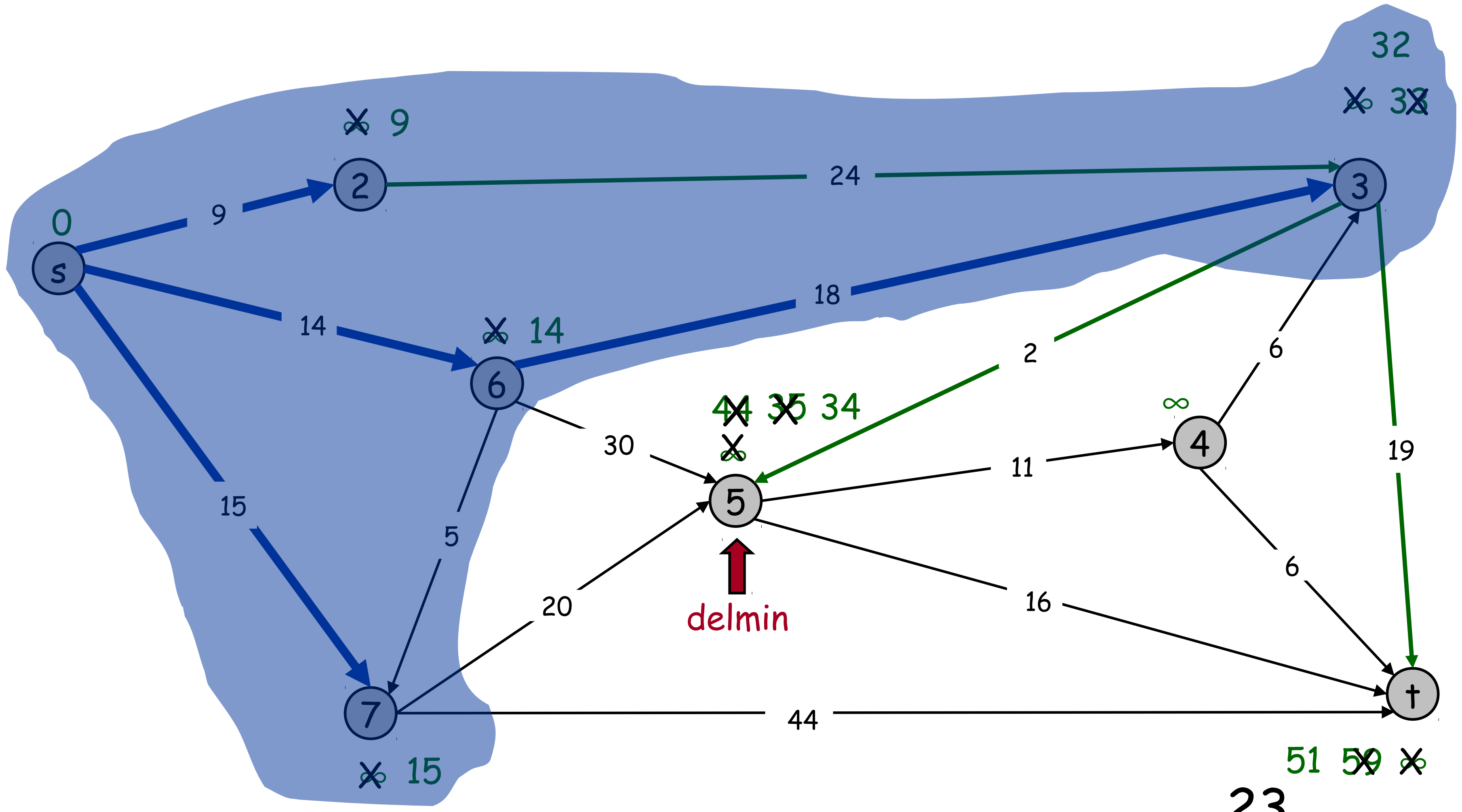
Dijkstra's Shortest Path Algorithm

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



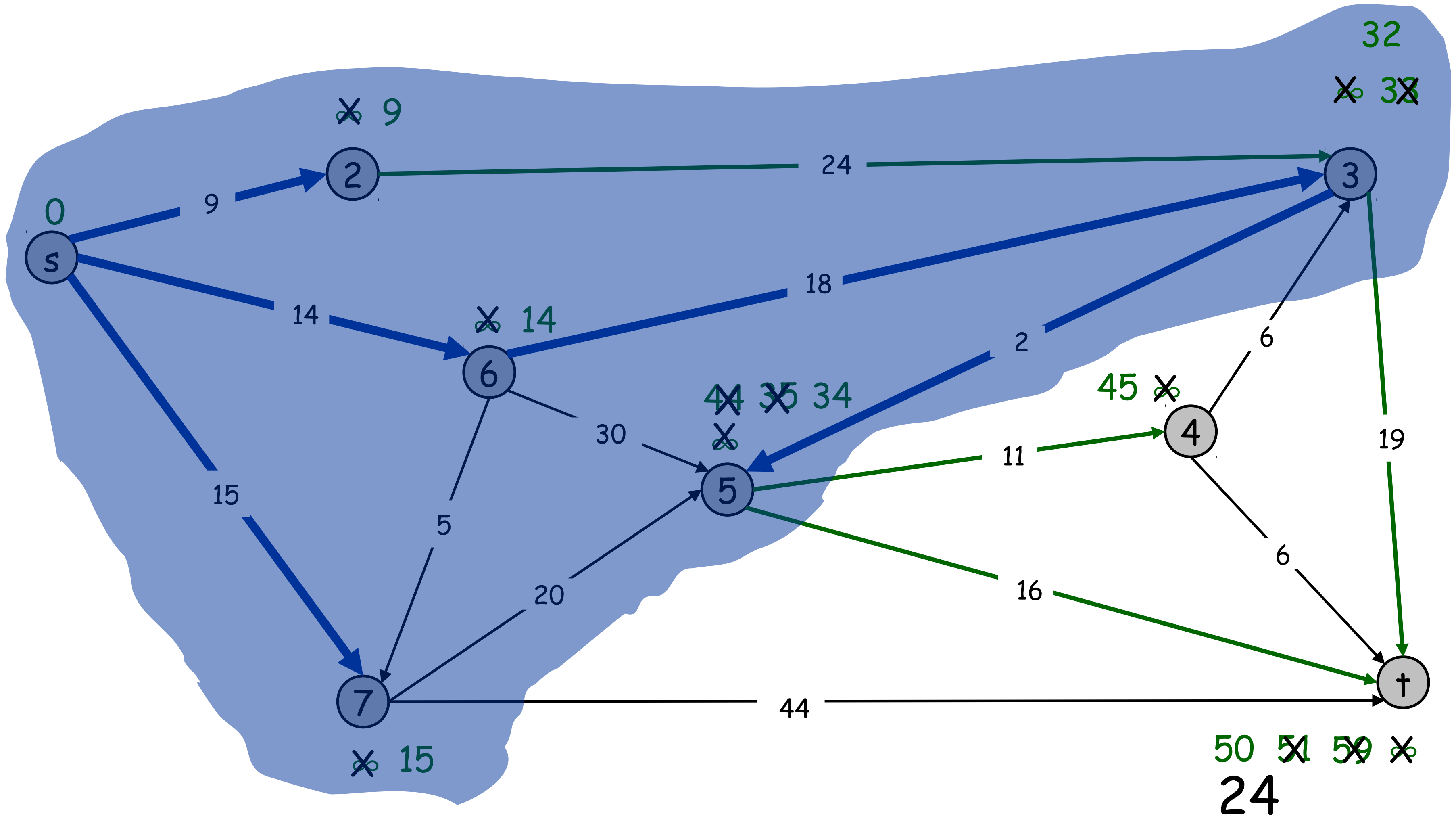
Dijkstra's Shortest Path Algorithm

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



Dijkstra's Shortest Path Algorithm

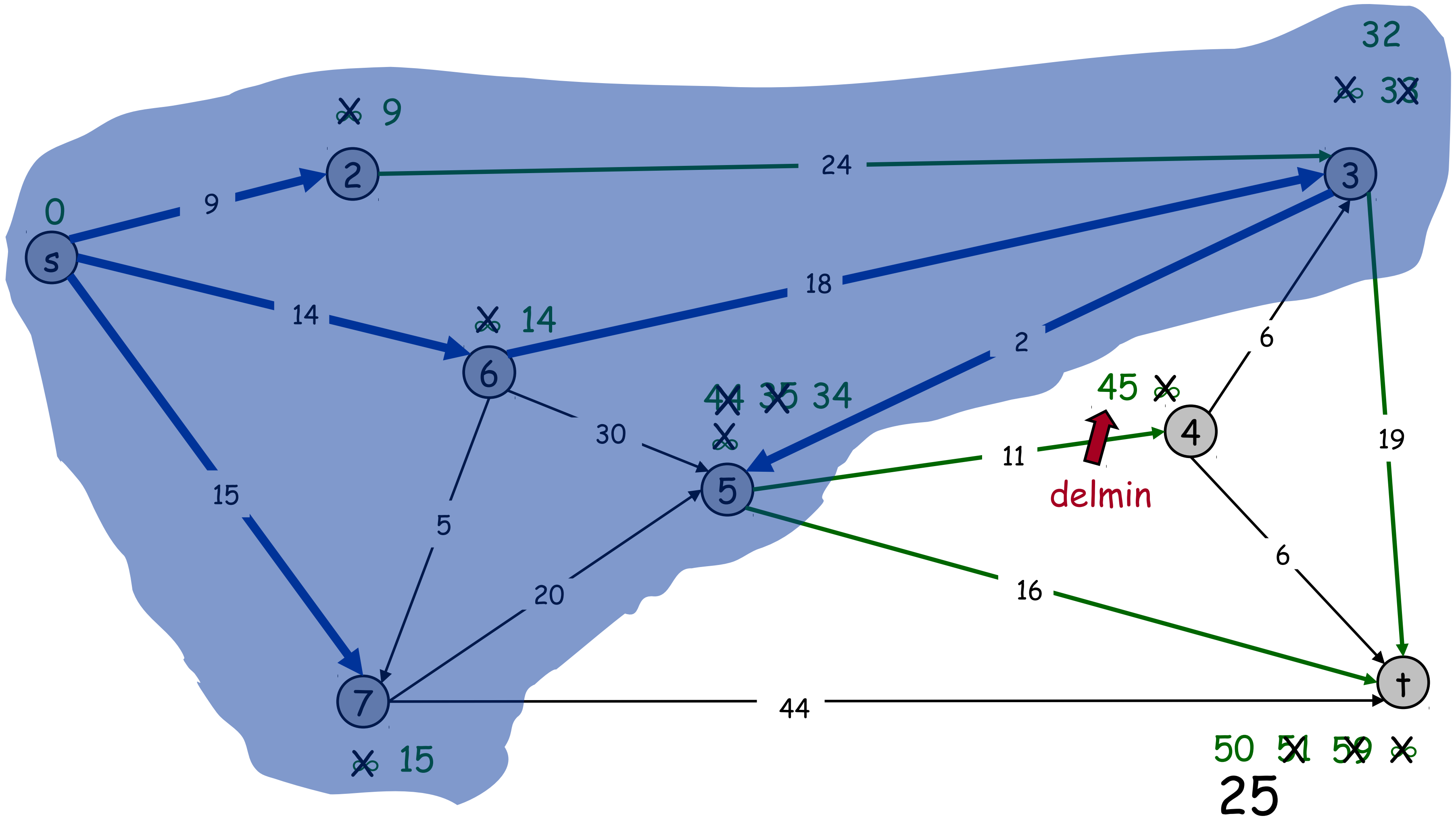
S = { s, 2, 3, 5, 6, 7 }
PQ = { 4, t }



24

Dijkstra's Shortest Path Algorithm

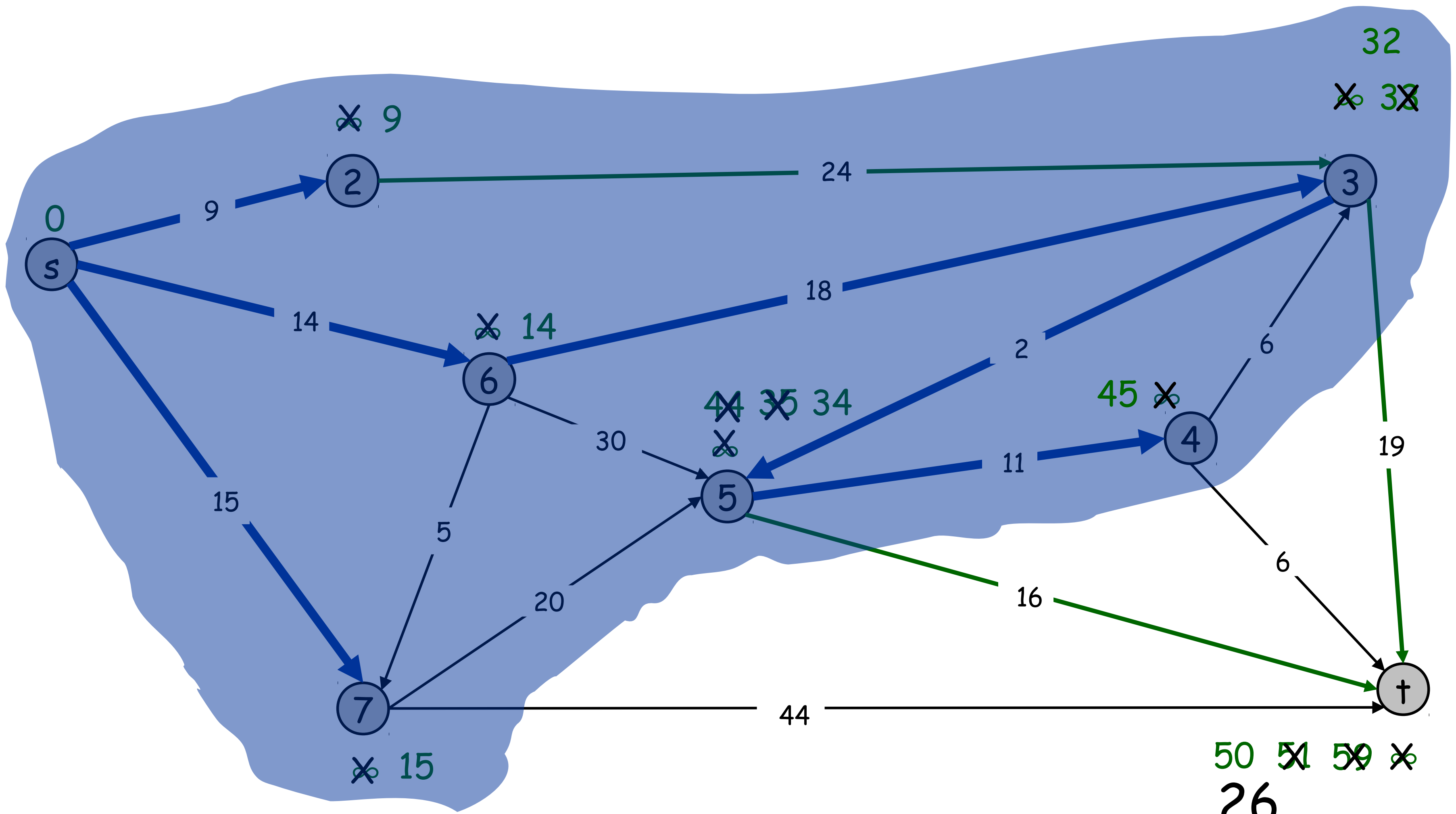
S = { s, 2, 3, 5, 6, 7 }
PQ = { 4, t }



25

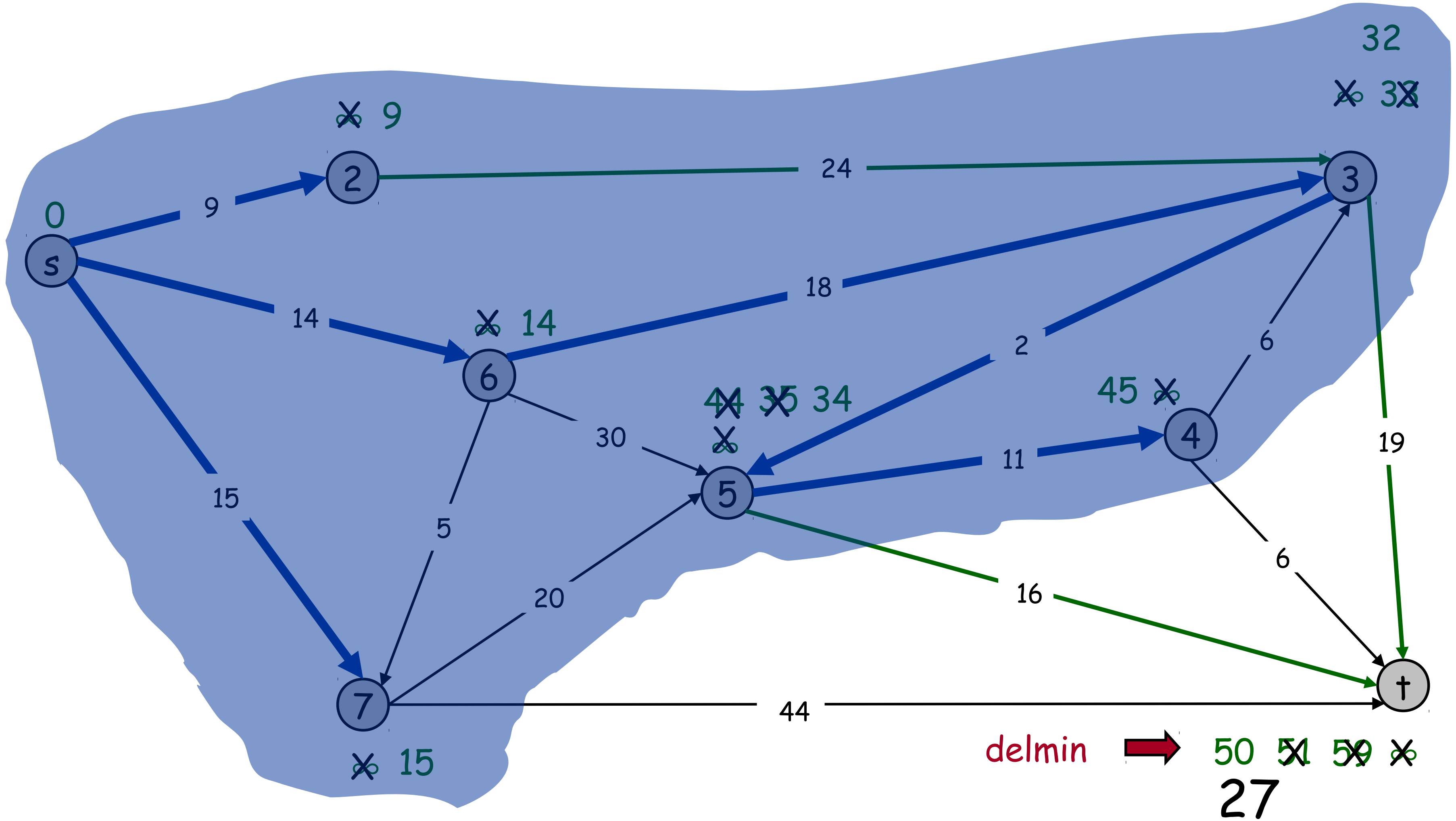
Dijkstra's Shortest Path Algorithm

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



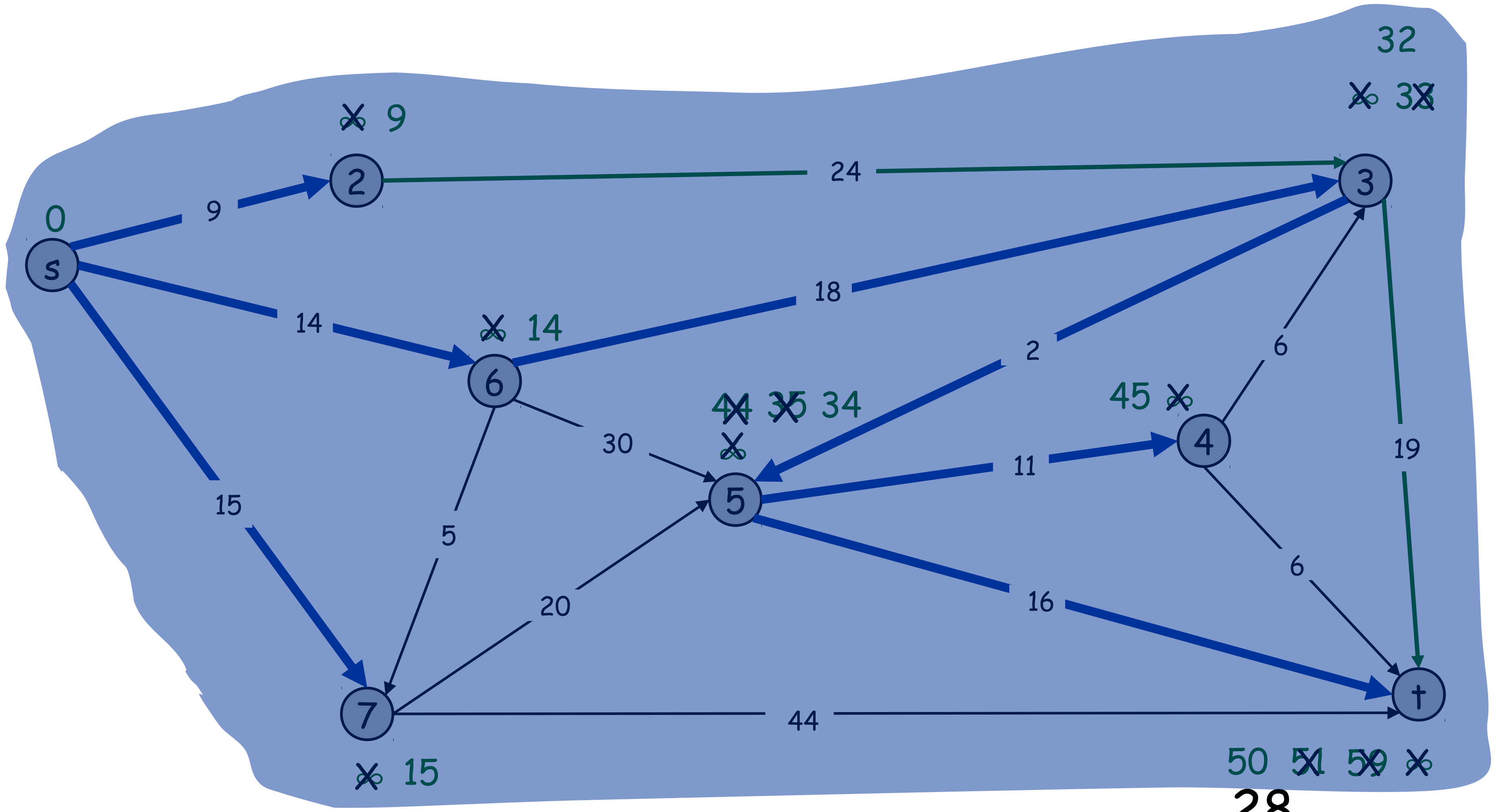
Dijkstra's Shortest Path Algorithm

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



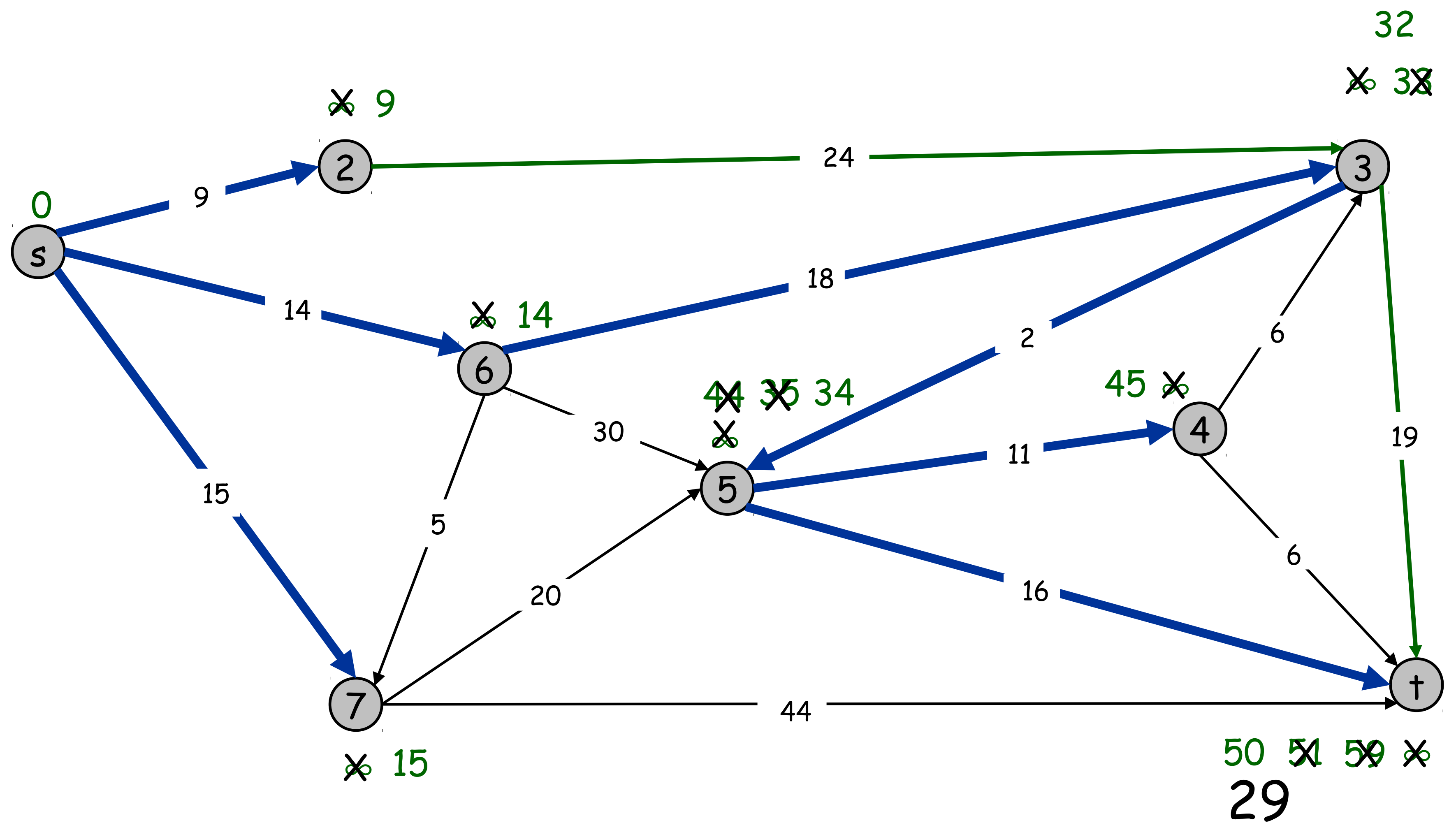
Dijkstra's Shortest Path Algorithm

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



Dijkstra's Shortest Path Algorithm

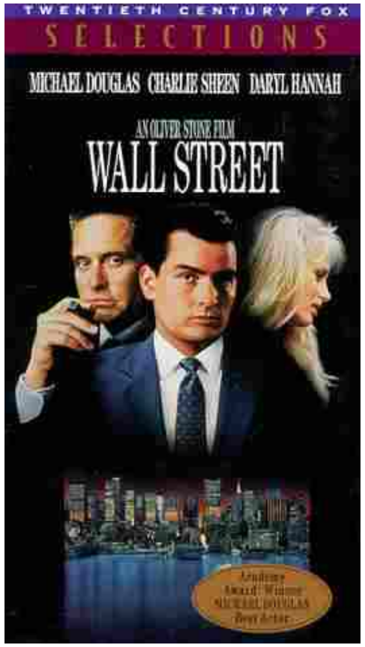
S = { s, 2, 3, 4, 5, 6, 7, t }
PQ = { }



Coin Changing

Greed is good. Greed is right. Greed works.
Greed clarifies, cuts through, and captures the essence of the evolutionary spirit.

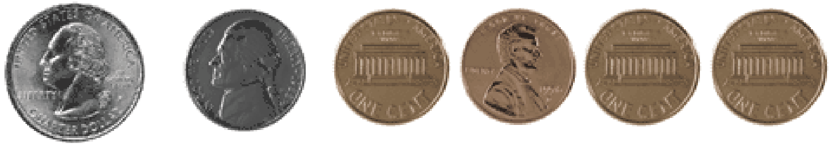
- Gordon Gecko (Michael Douglas)



Coin Changing

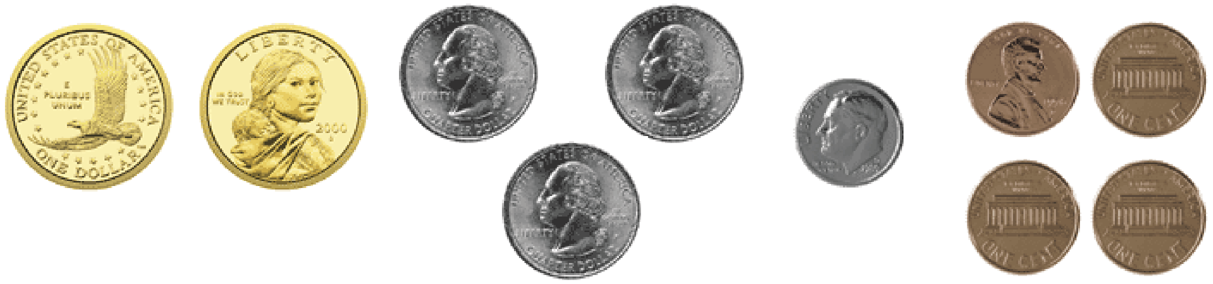
Goal. Given currency denominations: 1, 5, 10, 25, 100, devise a method to pay amount to customer using fewest number of coins.

Ex: 34¢.



Cashier's algorithm. At each iteration, add coin of the largest value that does not take us past the amount to be paid.

Ex: $2.89.



Coin-Changing: Greedy Algorithm

Cashier's algorithm. At each iteration, add coin of the largest value that does not take us past the amount to be paid.

```
Sort coins denominations by value: $c_1 < c_2 < … < c_n$.

  ↙ coins selected
S ← $\phi$
while ($x \neq 0$) {
   let k be largest integer such that $c_k \leq x$
   if (k = 0)
      return "no solution found"
   $x \leftarrow x - c_k$
   $S \leftarrow S \cup \{k\}$
}
return S
```

Q. Is cashier's algorithm optimal?

Coin-Changing: Analysis of Greedy Algorithm

Theorem. Greed is optimal for U.S. coinage: 1, 5, 10, 25, 100.
Pf. (by induction on x)

- Consider optimal way to change $c_k \leq x < c_{k+1}$: greedy takes coin k.
- We claim that any optimal solution must also take coin k.
  - if not, it needs enough coins of type $c_1, \ldots, c_{k-1}$ to add up to x
  - table below indicates no optimal solution can do this
- Problem reduces to coin-changing $x - c_k$ cents, which, by induction, is optimally solved by greedy algorithm. ▪

k	c_k	All optimal solutions must satisfy	Max value of coins 1, 2, …, k-1 in any OPT
1	1	$P \leq 4$	-
2	5	$N \leq 1$	4
3	10	$N + D \leq 2$	4 + 5 = 9
4	25	$Q \leq 3$	20 + 4 = 24
5	100	no limit	75 + 24 = 99

Coin-Changing: Analysis of Greedy Algorithm

Observation. Greedy algorithm is sub-optimal for US postal denominations: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

Counterexample. 140¢.

- Greedy: 100, 34, 1, 1, 1, 1, 1, 1.
- Optimal: 70, 70.

Edsger W. Dijkstra

The question of whether computers can think is like the question of whether submarines can swim.

Do only what only you can do.

In their capacity as a tool, computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind.

The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence.

APL is a mistake, carried through to perfection. It is the language of the future for the programming techniques of the past: it creates a new generation of coding bums.