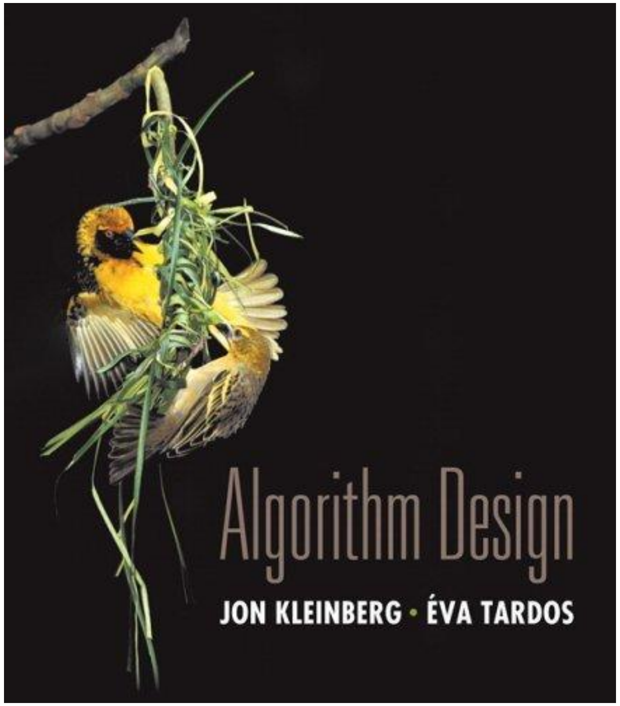
Today slide credits

- Best algorithms book:
  - Slides c/o Kevin Wayne
    - With slight changes

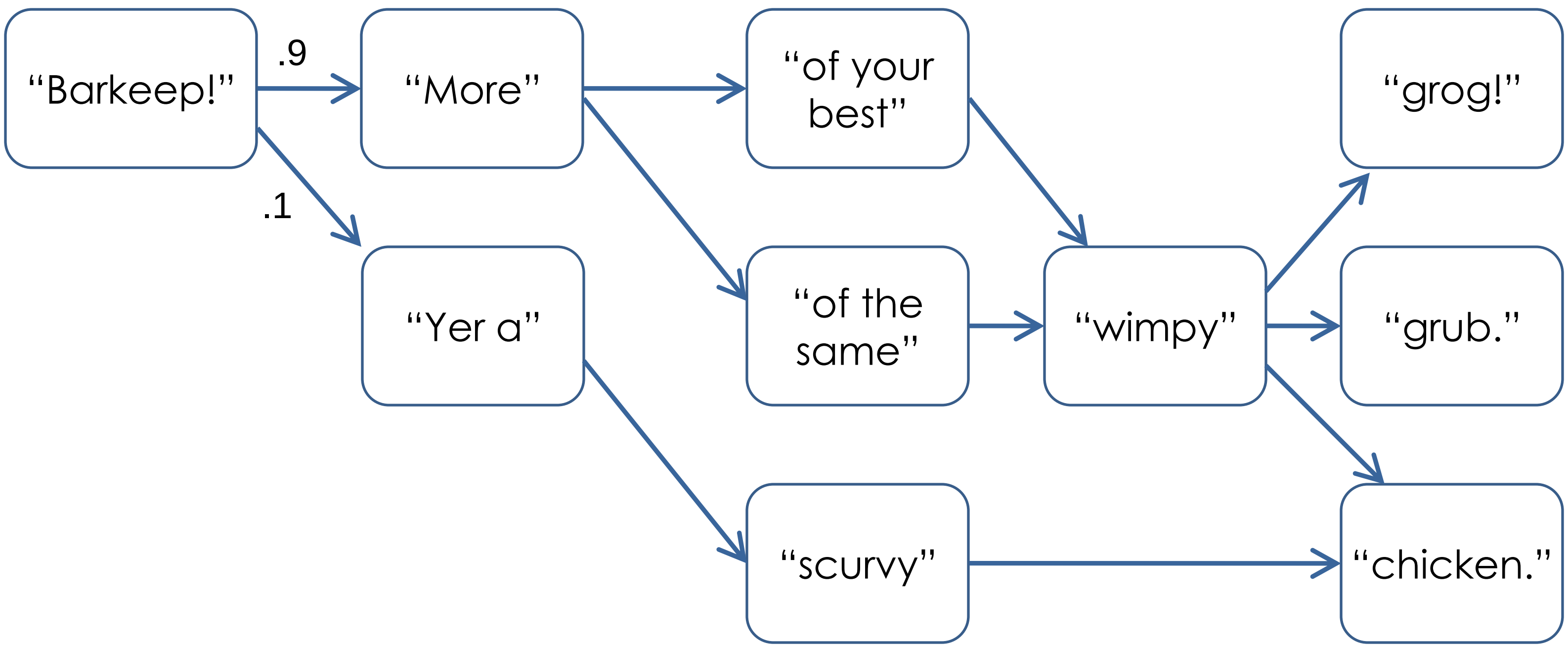


Two basic algorithms

- Exhaustive search: try everything
  - Always works. Always slow.
- Greedy method: act locally
  - Sometimes works. Always fast.
- Today: a triumph of greed
  - Plus a nice induction proof
- First: pirate grammar

Pirate grammar

- What is a pirate's favorite sentence?



Problem: shortest paths

- Underlying problem for examples
  - Not completely obvious
    - Pirate favorite sentence?
    - Photoshopping images??
- General version: given a graph with edge weights, a starting node s and a target t, find shortest path from s to t
- Claim: this problem is impossible
  - Proof?

Cycles

- Consider a cycle A-B-C-A
  - Where the weight sum is negative
- Go around this multiple times
  - Always makes an even shorter path!
- Does the presence of a negative weight cycle imply no shortest path?
  - Almost, but not quite
- Let's assume positive edge weights
  - Can detect negative cycles

Key property

- Suppose the shortest path from s to t goes via v
  – I.e., s … v … t

    shortest s-v path

  – Otherwise, we would take that "shortcut" instead, and create an even shorter path
  – Parse this statement carefully!
- When considering s-v-t paths, we only need the shortest s-v path
  – Don't need to try everything!

Idea: Dijkstra (1959)

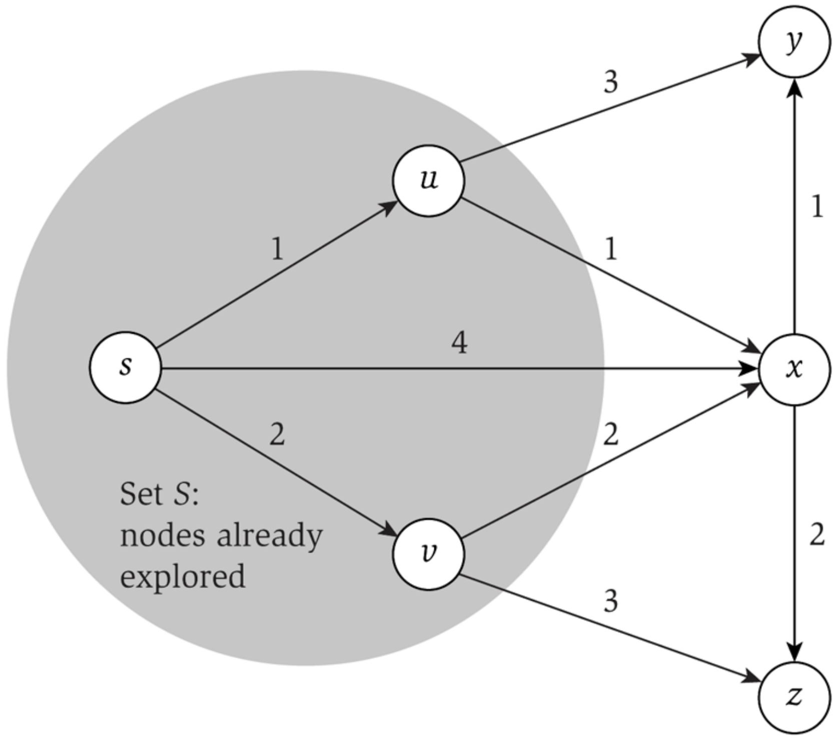
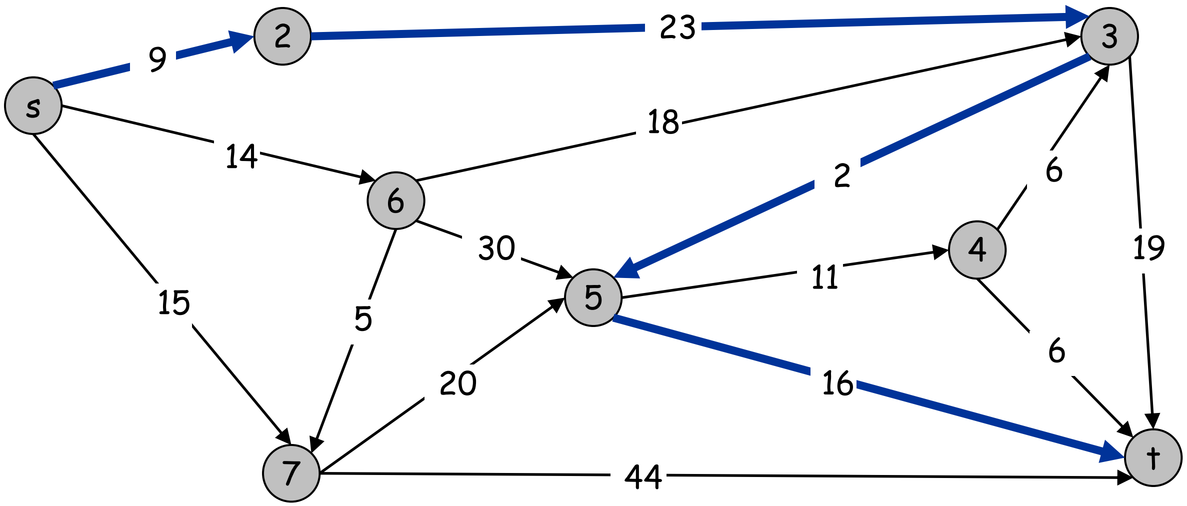


Figure 4.7 A snapshot of the execution of Dijkstra's Algorithm. The next node that will be added to the set S is x, due to the path through u.

- Can think of expanding a ball
  - Actually a variant of BFS!

Shortest path example



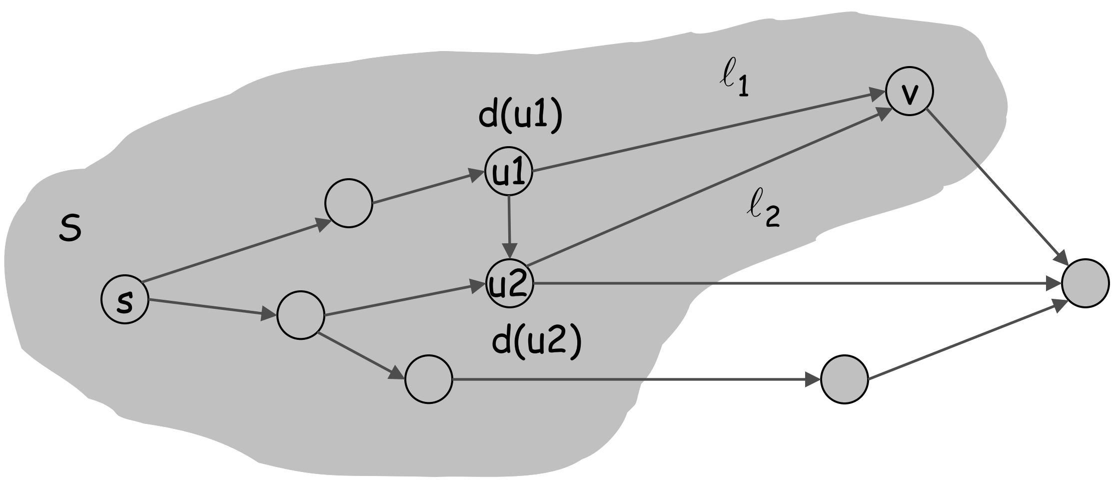
Cost of path s-2-3-5-t
= 9 + 23 + 2 + 16
= 48.

Dijkstra’s algorithm

- On each recursive call we will have an explored set S
  - For each node u in S we hold the **shortest** path from s to u, write this as d(u)
    - Both the distance and the actual path
    - Easiest to just think about the distance d(u)
      - Can easily extend this to add path
  - Add an unexplored node v to S
    - But, which one to choose?
    - Adjacent to S, so we add just one edge

Choice of edge for a node

- The new node v can be adjacent to several nodes in S
  - v is at the "fringe" of the set S
  - If we choose to add v, we need to pick the right node in S to connect it to



$d(u1) + \ell_1$
versus
$d(u2) + \ell_2$

Choice of node

- If we pick v to add to S, we will connect it to the u in S that minimizes d(u) + the length of the (u,v) edge
  - Call this shortest path length $\pi(v)$
  - But can we pick an arbitrary v to add?
- Can prove that this would break our invariant about S!
- Need to pick v with smallest $\pi(v)$, then add it to S with $d(v) = \pi(v)$

Algorithm

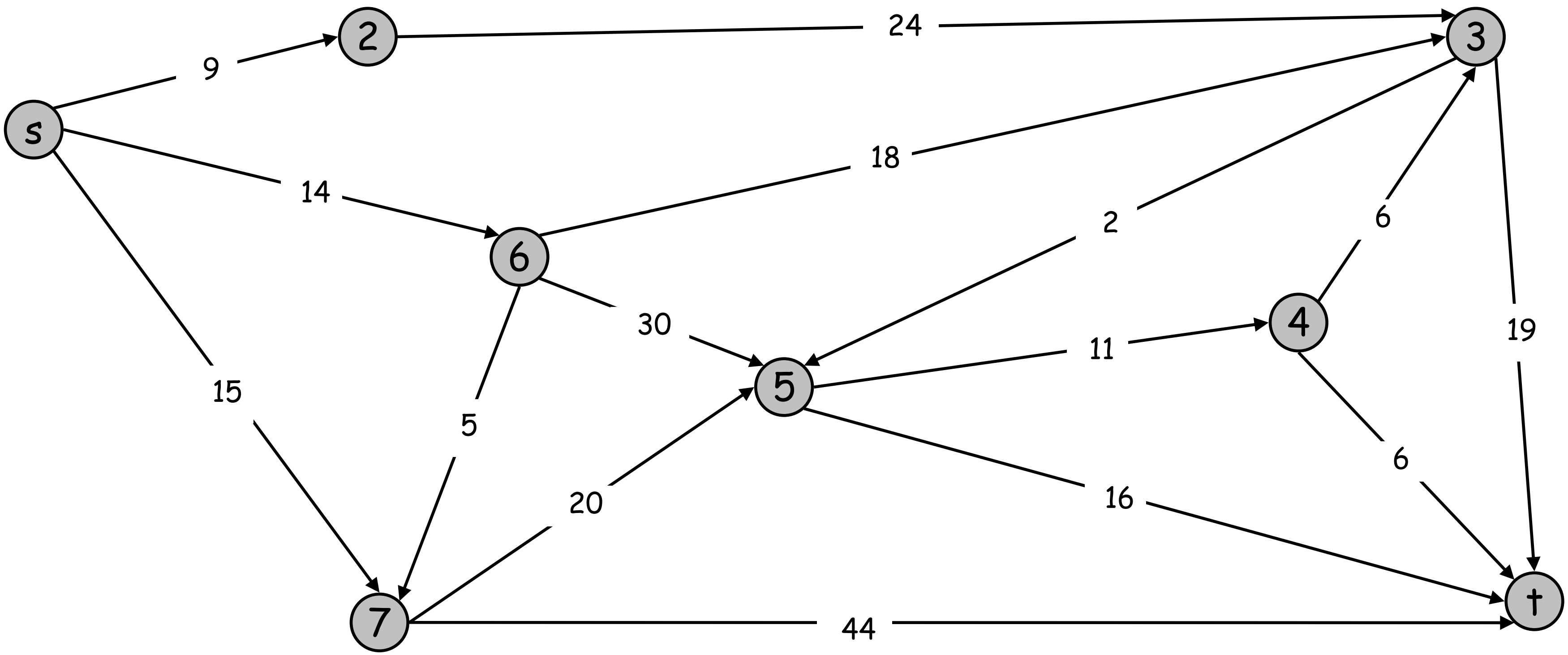
- Start with S={s}, all other nodes in Q
  - d(s) = 0, else d(v) = ∞ (i.e. upper bound)
- Pick v on fringe of S that minimizes π(v)
  - I.e., a v in Q with a neighbor in S
- On recursive call, we will have
  - d(v) = π(v)
  - v is in S, and no longer in Q
- Done when we pick target t
  - Computes more than shortest s-t path!

Dijkstra's Shortest Path Algorithm

Find shortest path from s to t.

Blue arrows are shortest path to a node within S.

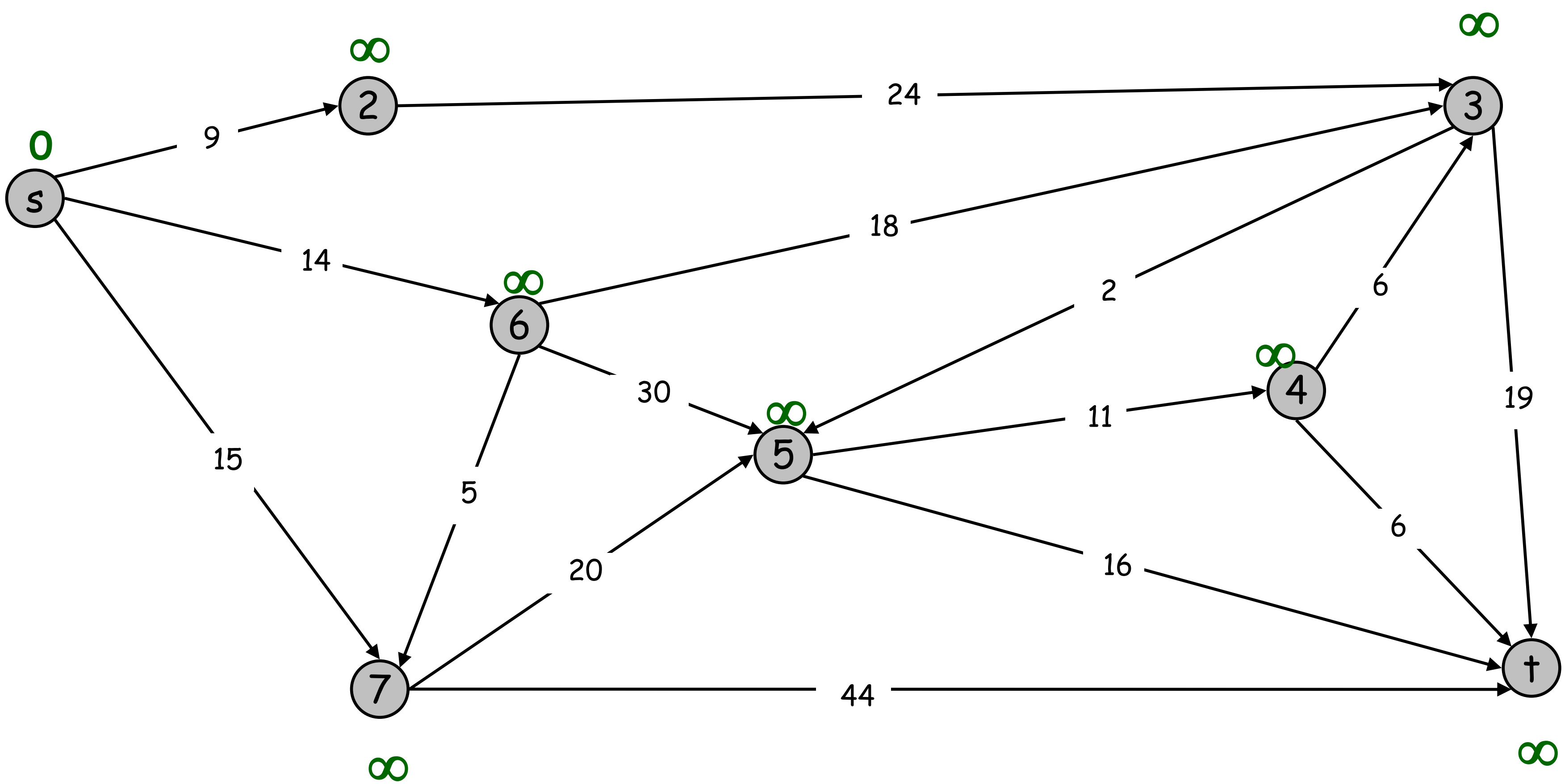
Green arrows are how we would add for each vertex.



Dijkstra's Shortest Path Algorithm

S = { s }

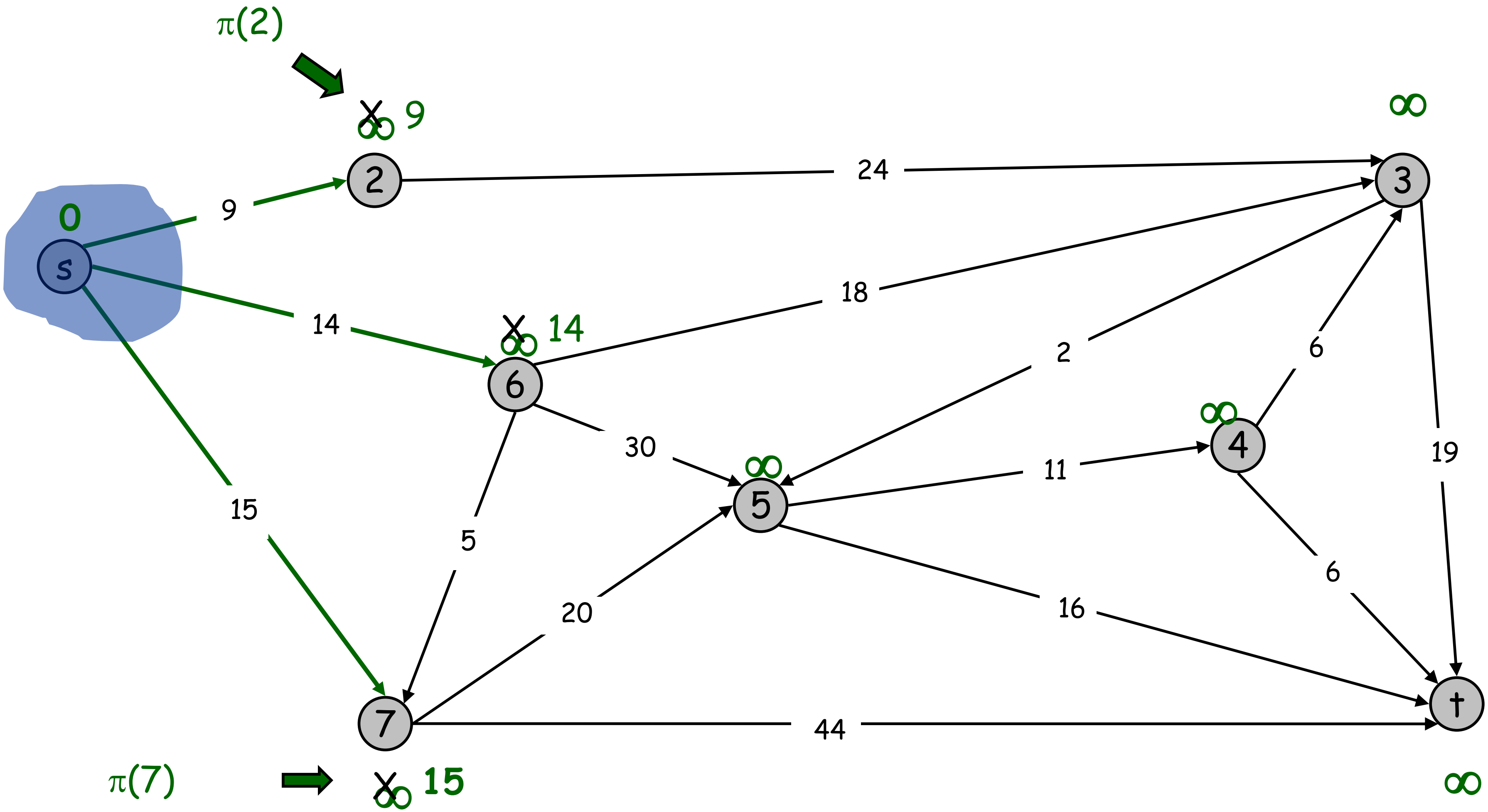
Q = { 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

$S = \{ s \}$

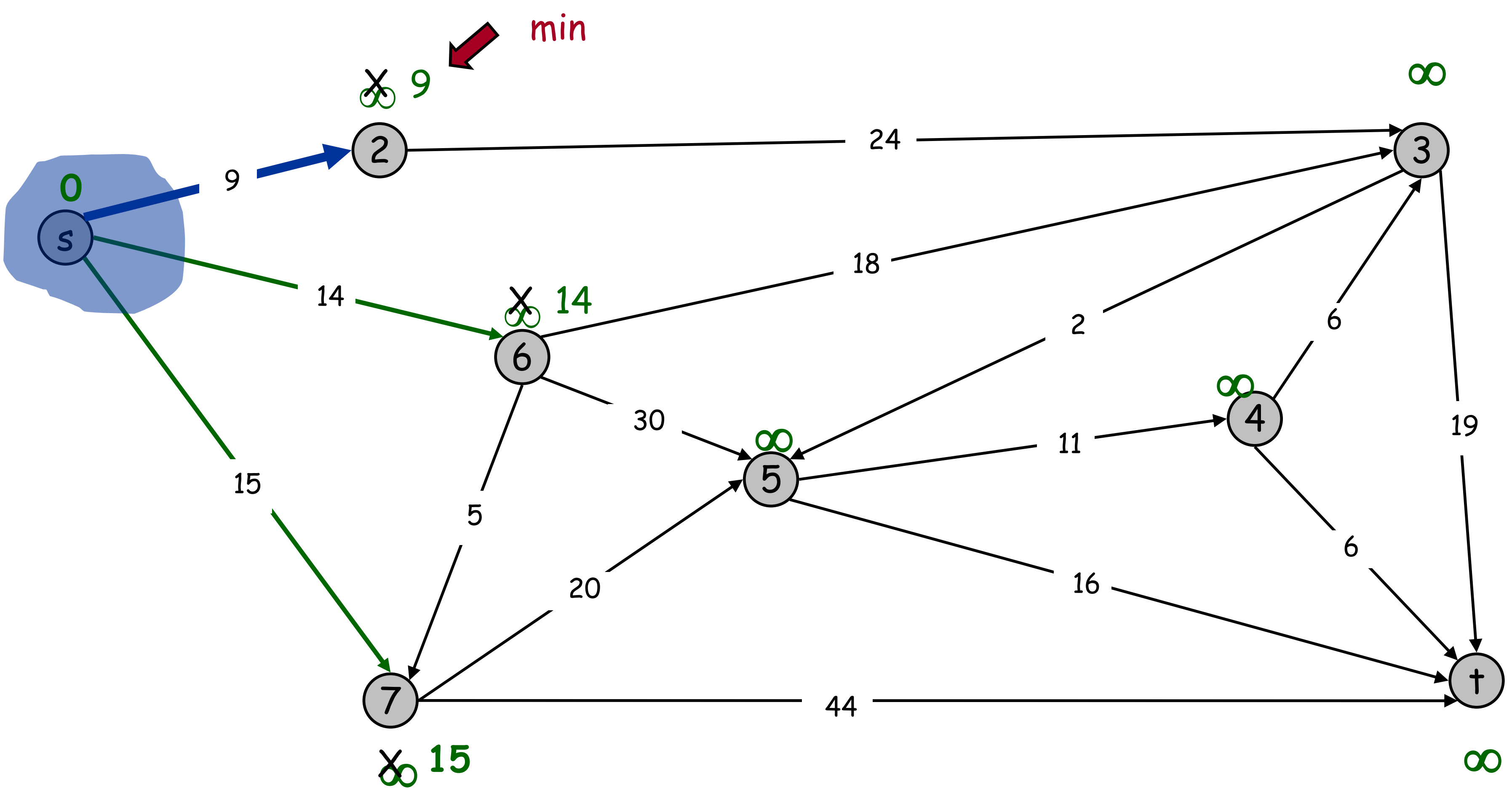
$Q = \{ 2, 3, 4, 5, 6, 7, t \}$



Dijkstra's Shortest Path Algorithm

S = { s }

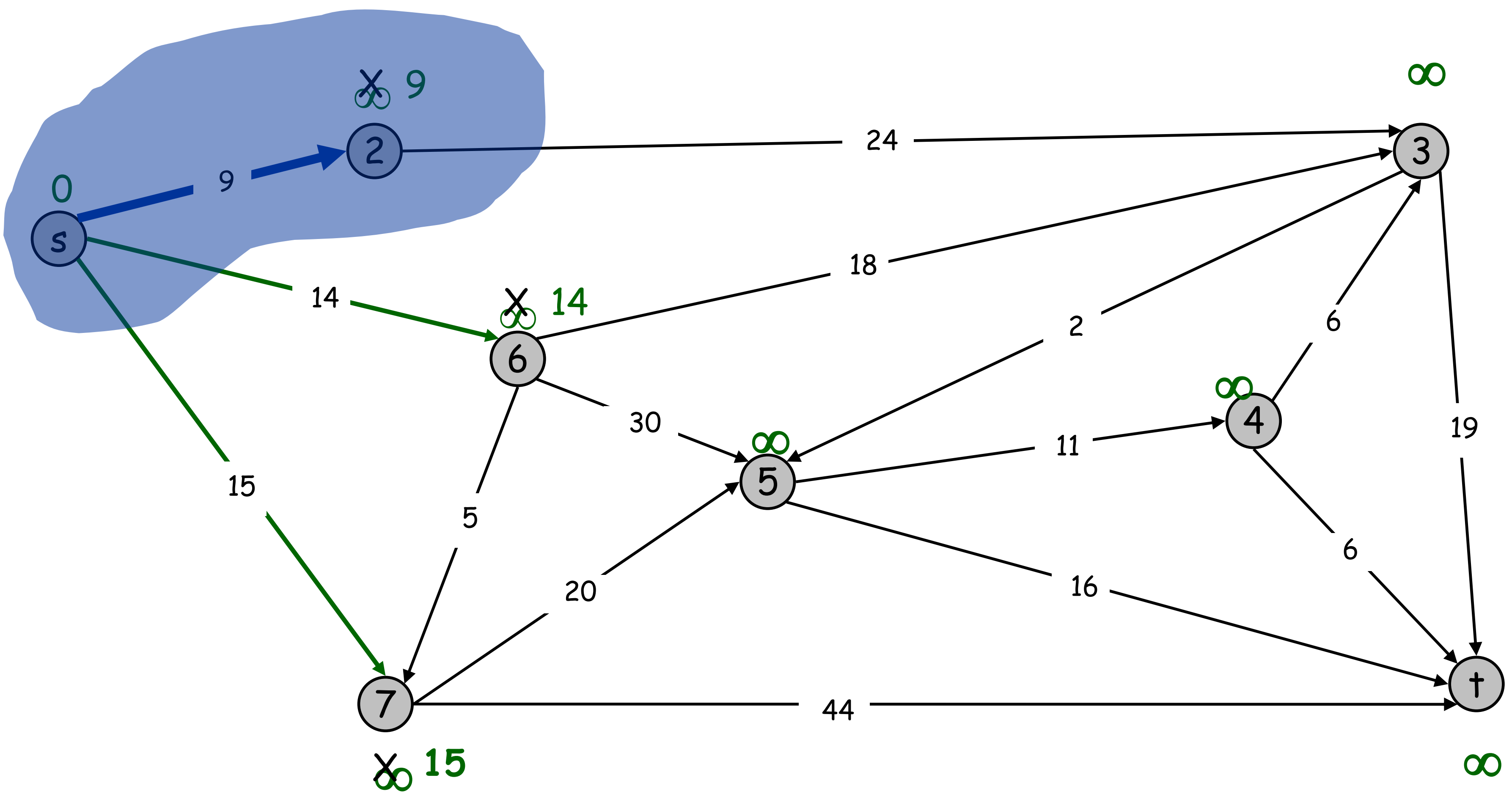
Q = { 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2 }

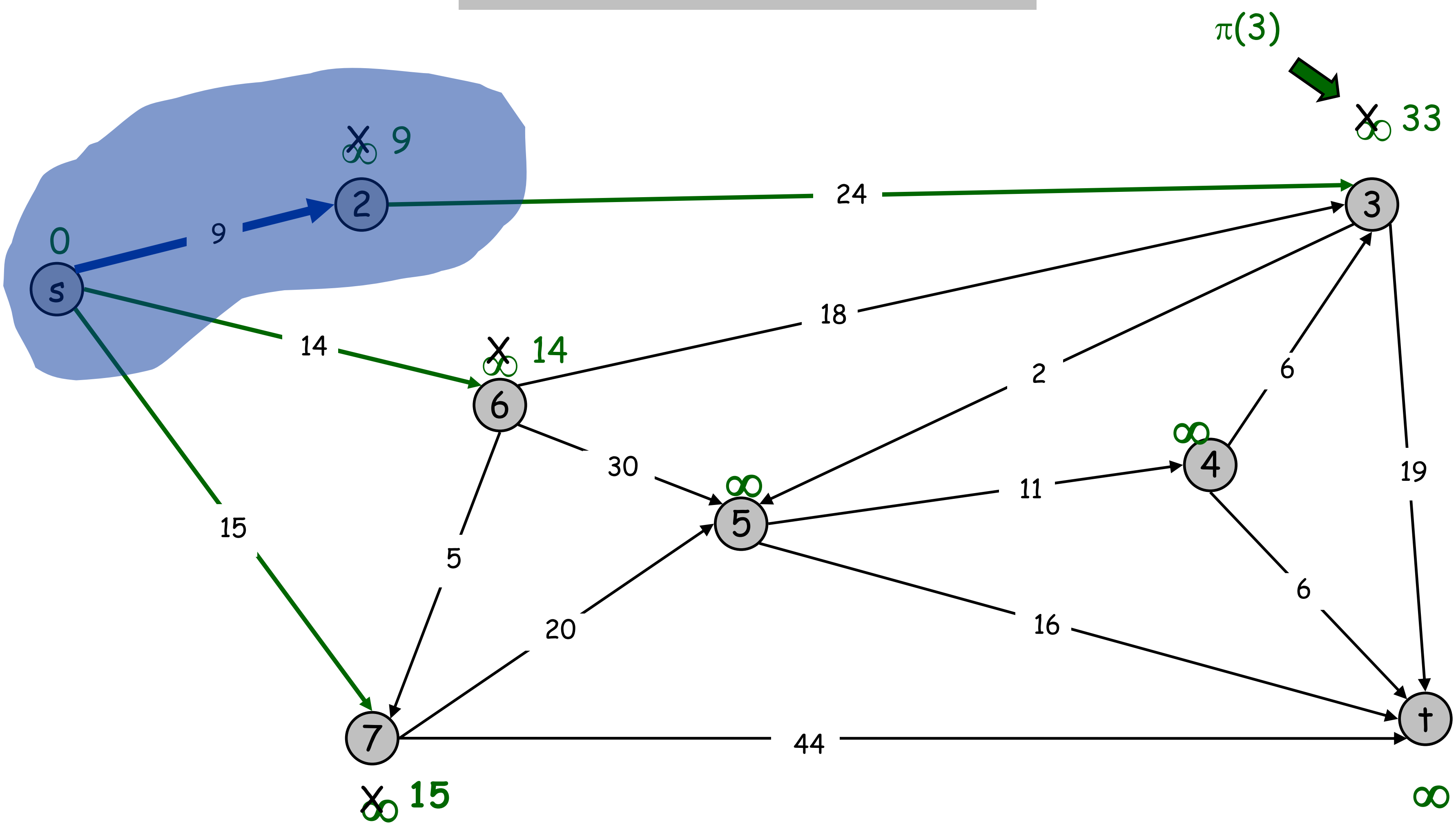
Q = { 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2 }

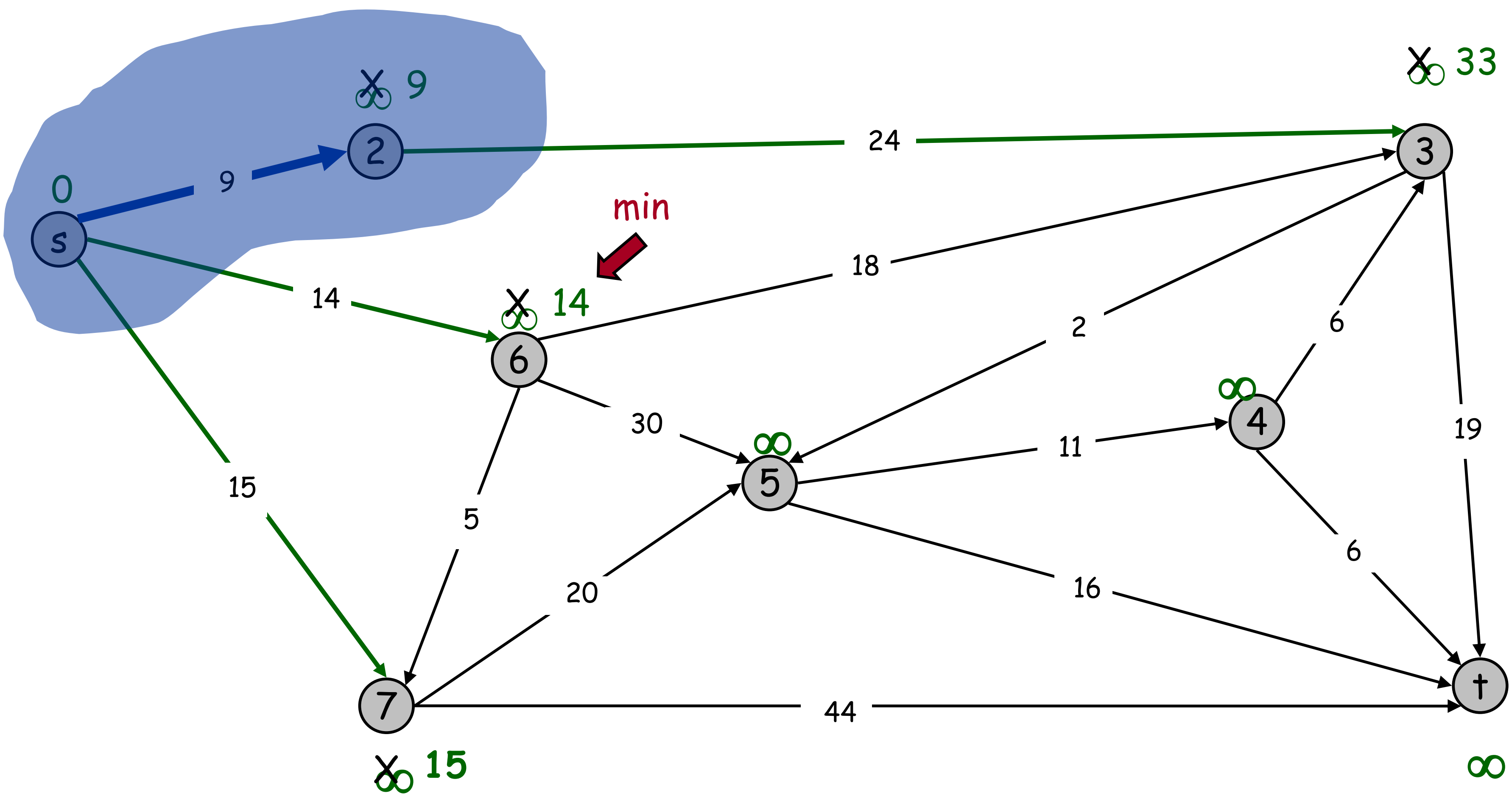
Q = { 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2 }

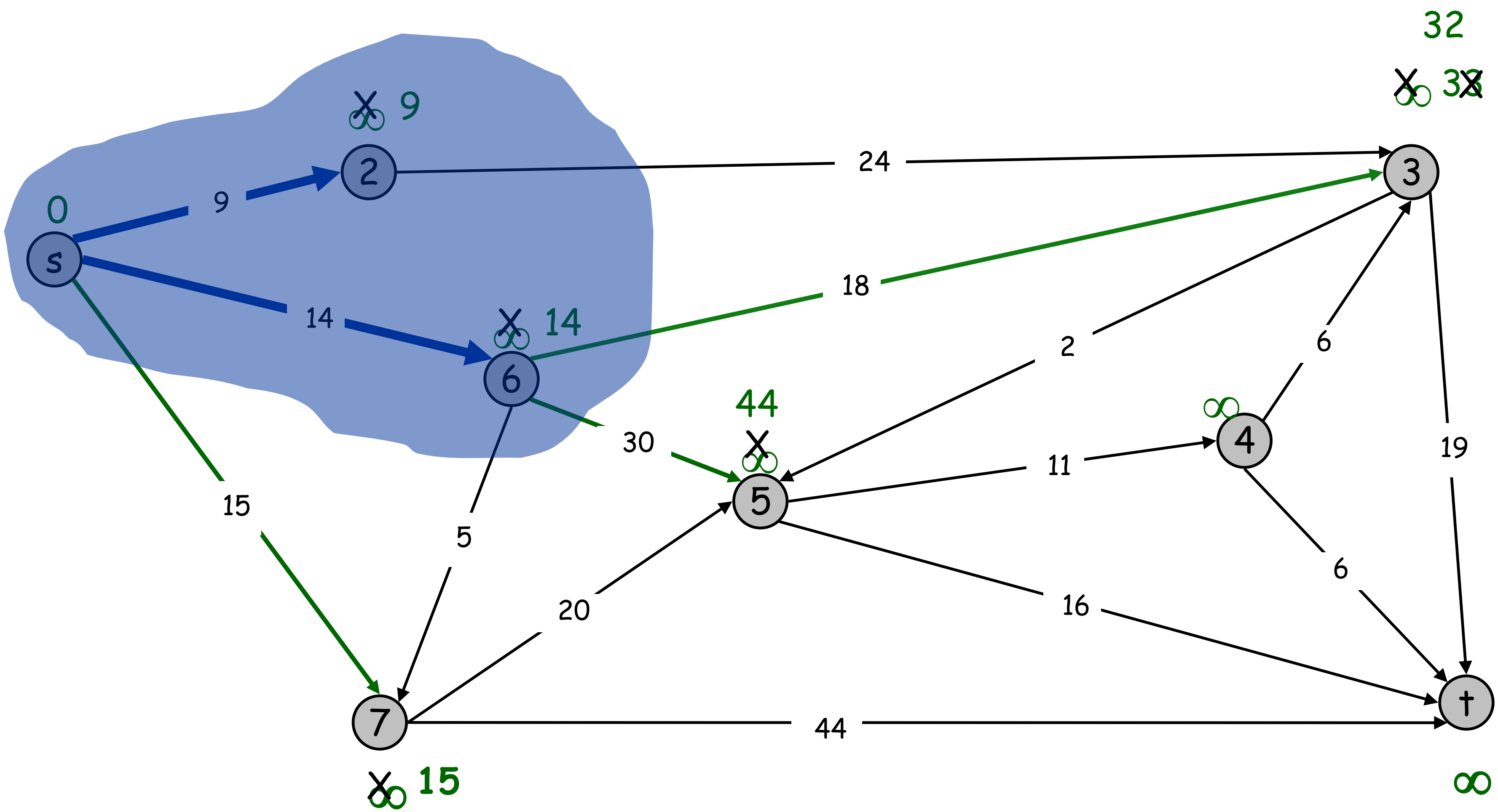
Q = { 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

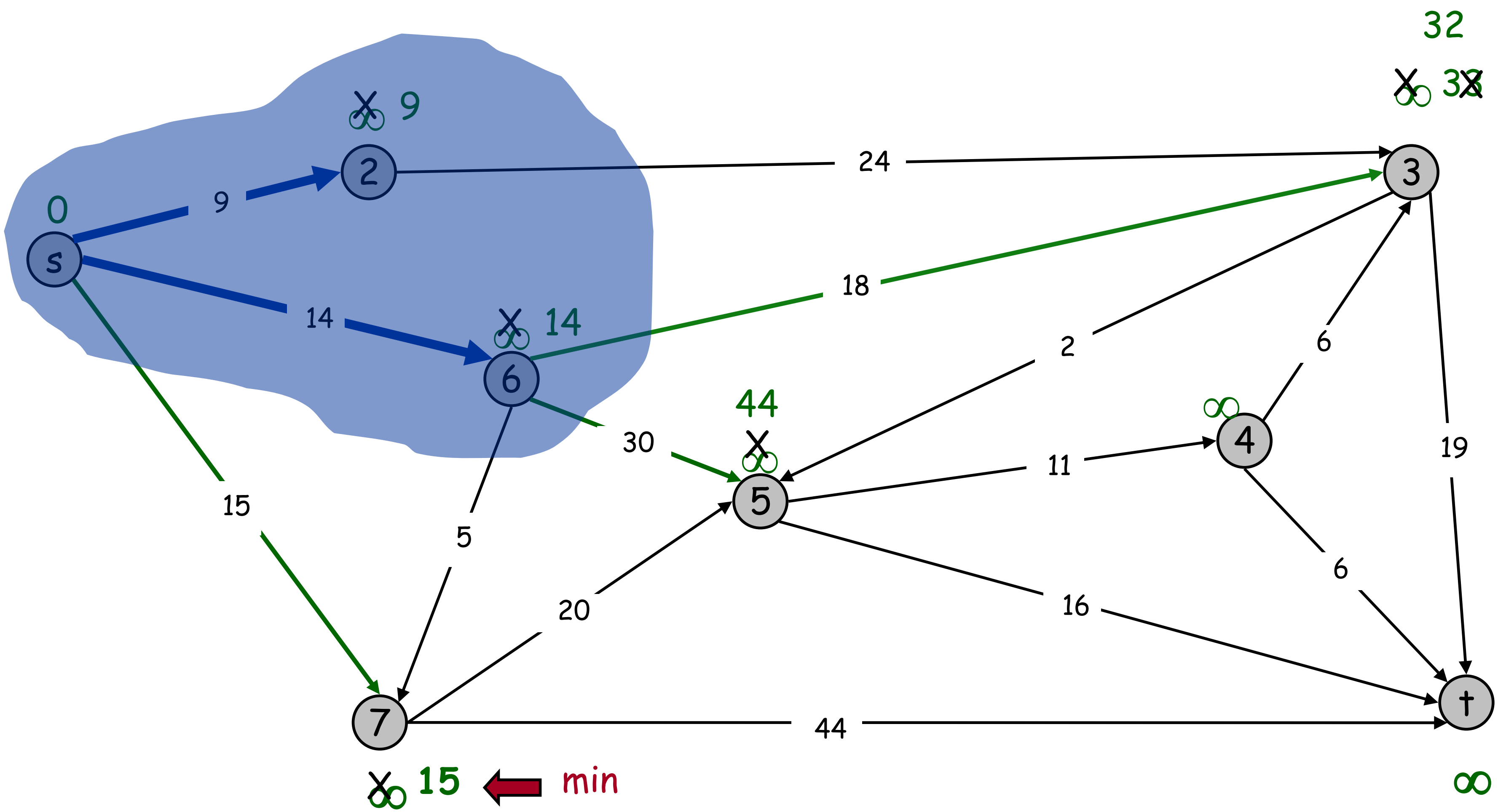
S = { s, 2, 6 }

Q = { 3, 4, 5, 7, t }



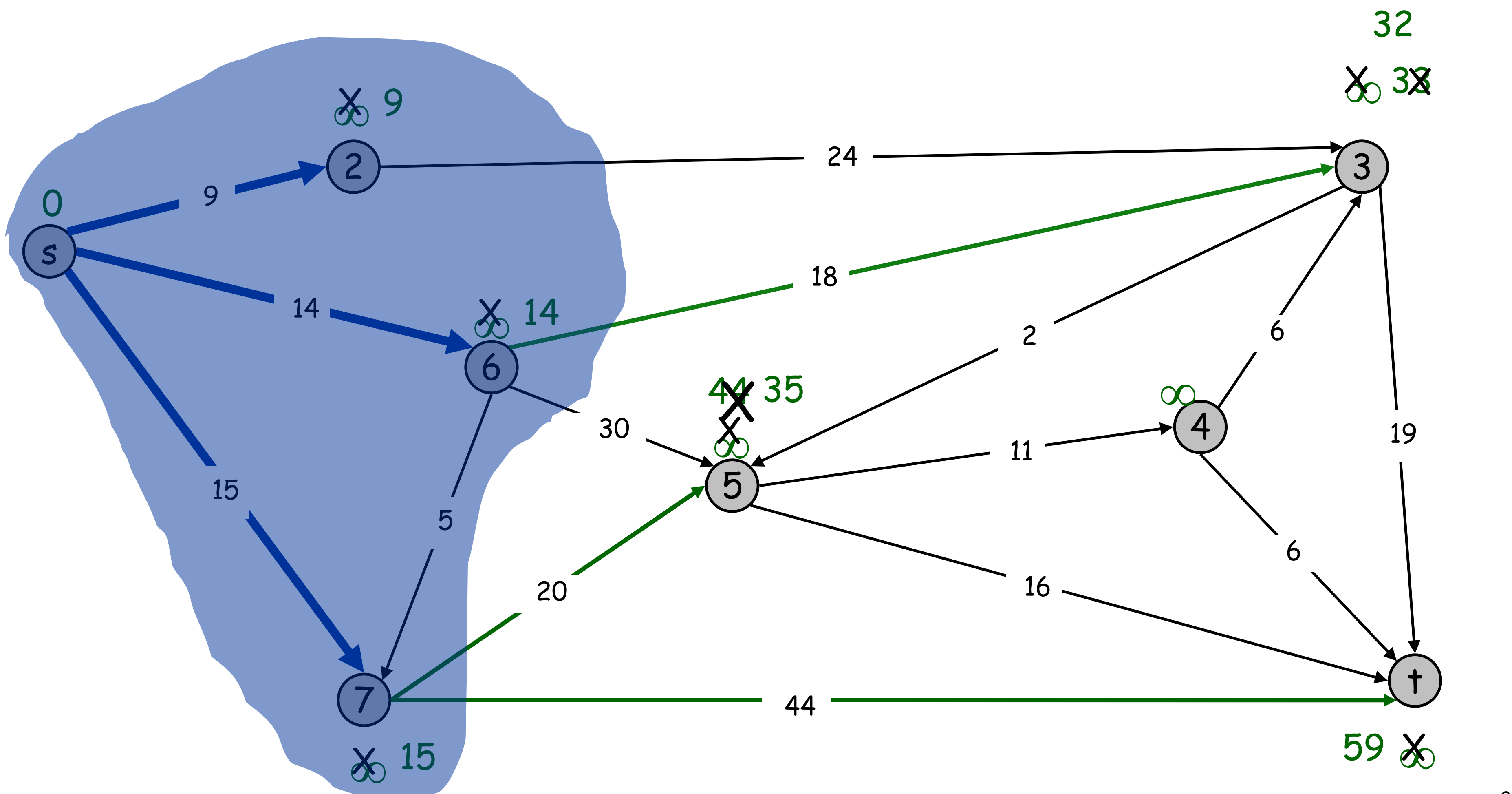
Dijkstra's Shortest Path Algorithm

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



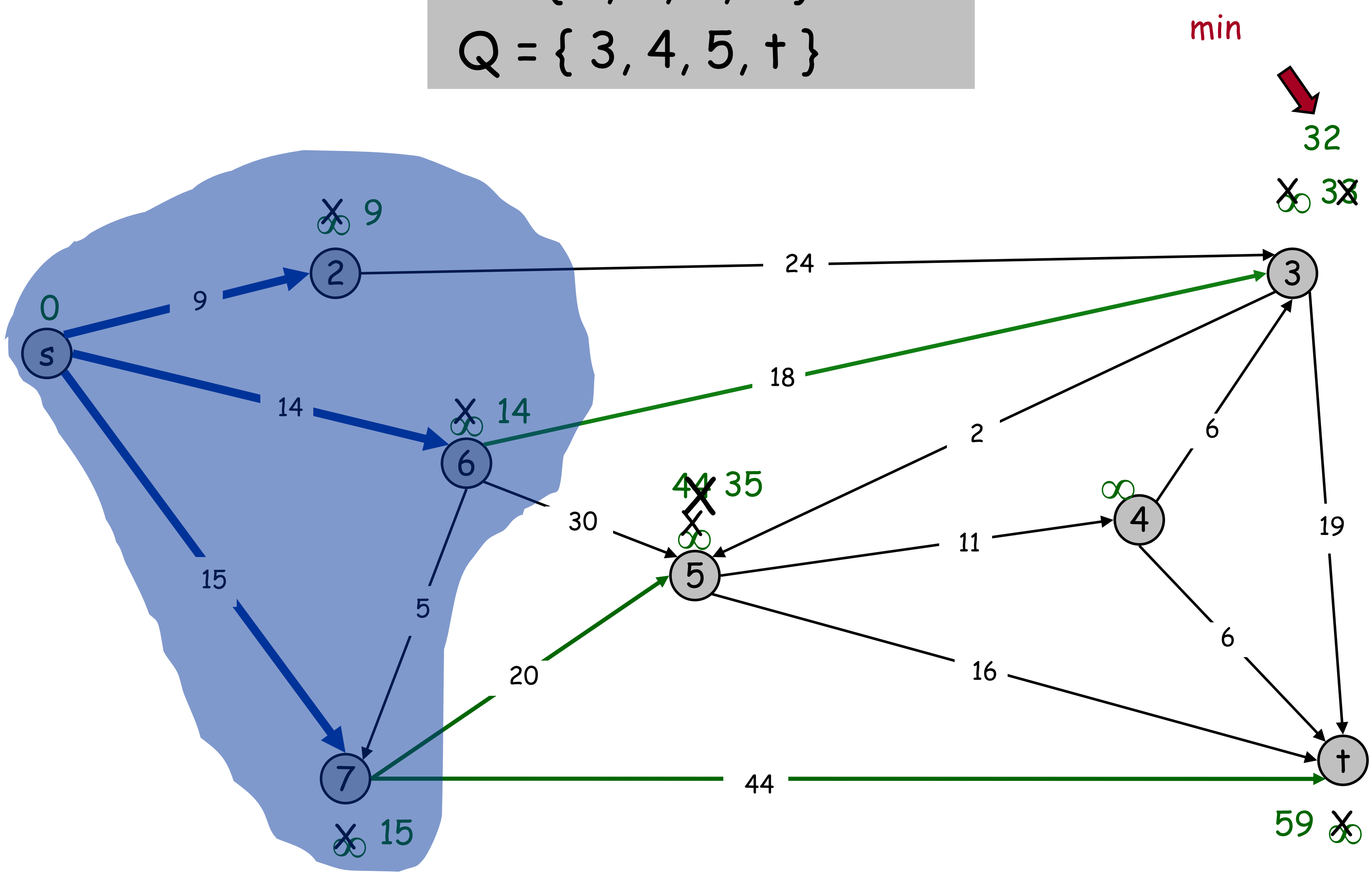
Dijkstra's Shortest Path Algorithm

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



Dijkstra's Shortest Path Algorithm

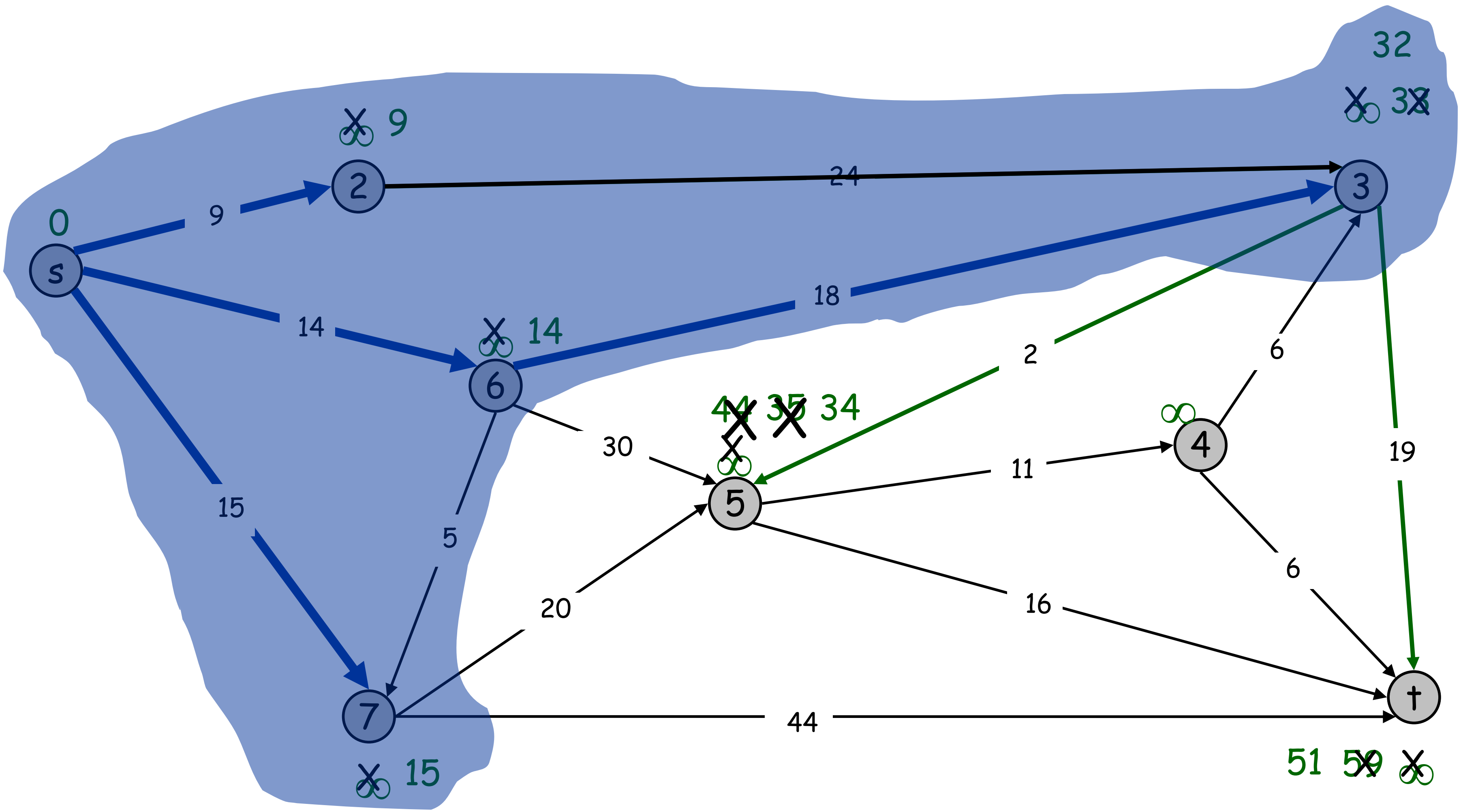
S = { s, 2, 6, 7 }
Q = { 3, 4, 5, t }



Dijkstra's Shortest Path Algorithm

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

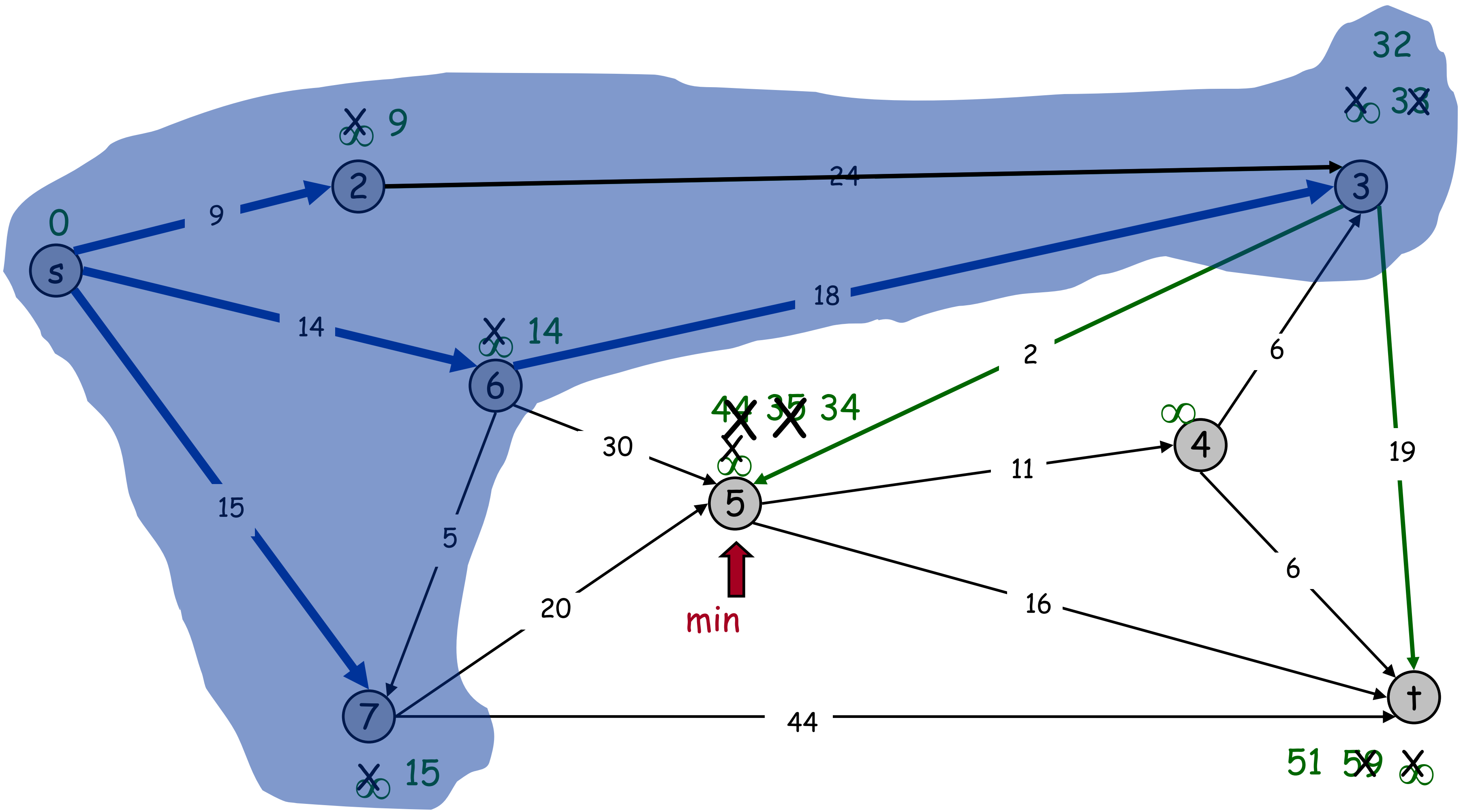
Q = { 4, 5, t }



Dijkstra's Shortest Path Algorithm

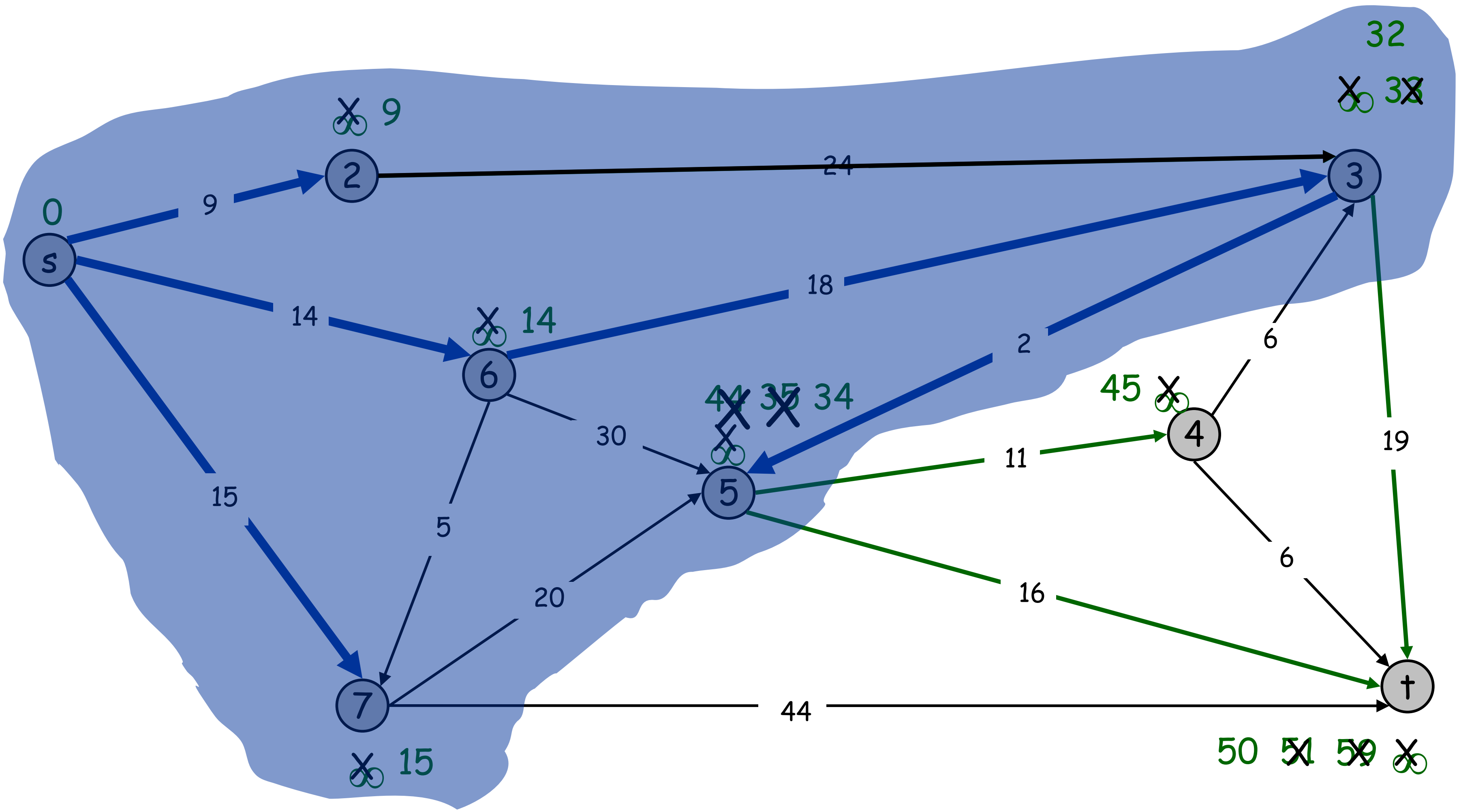
S = { s, 2, 3, 6, 7 }

Q = { 4, 5, t }



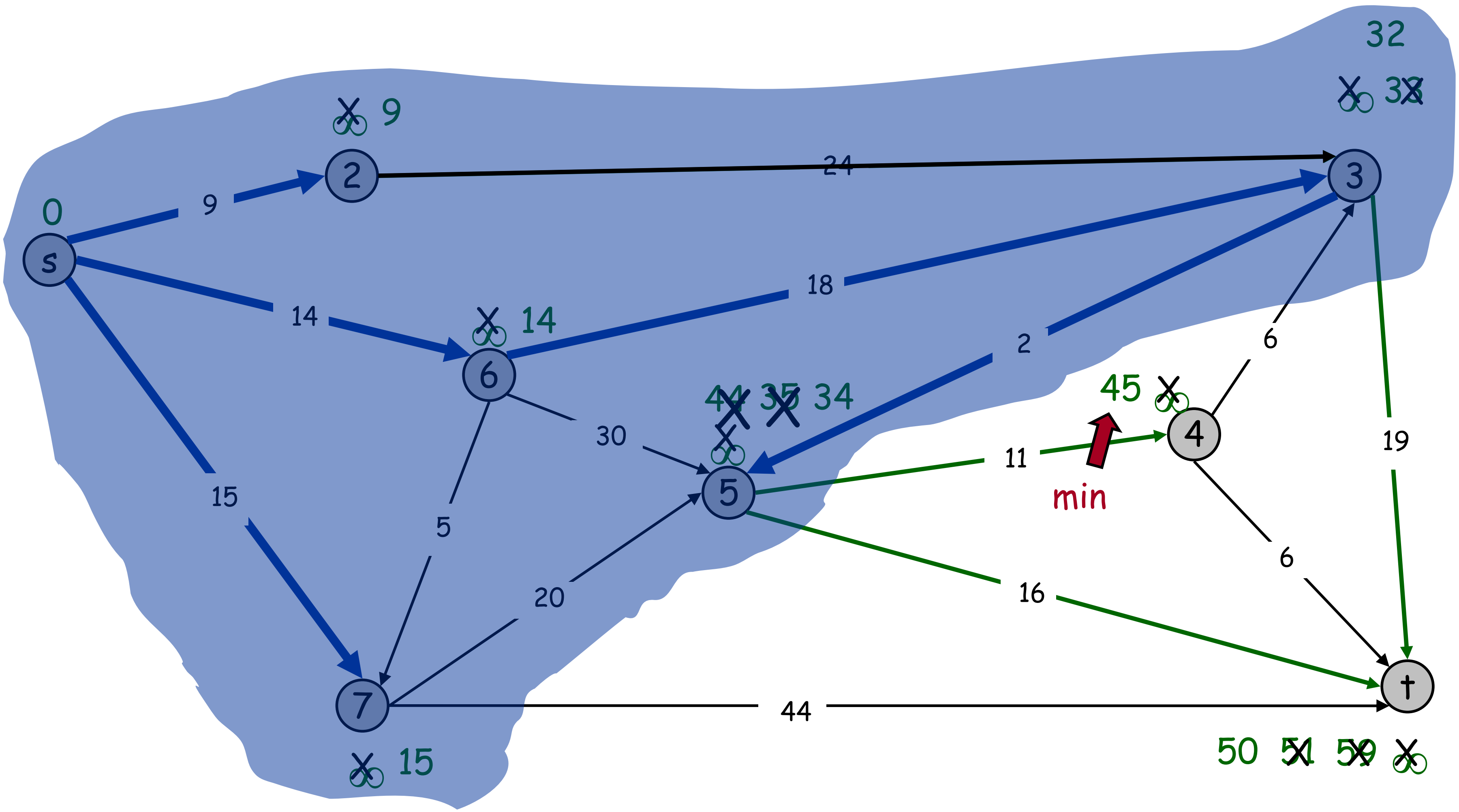
Dijkstra's Shortest Path Algorithm

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



Dijkstra's Shortest Path Algorithm

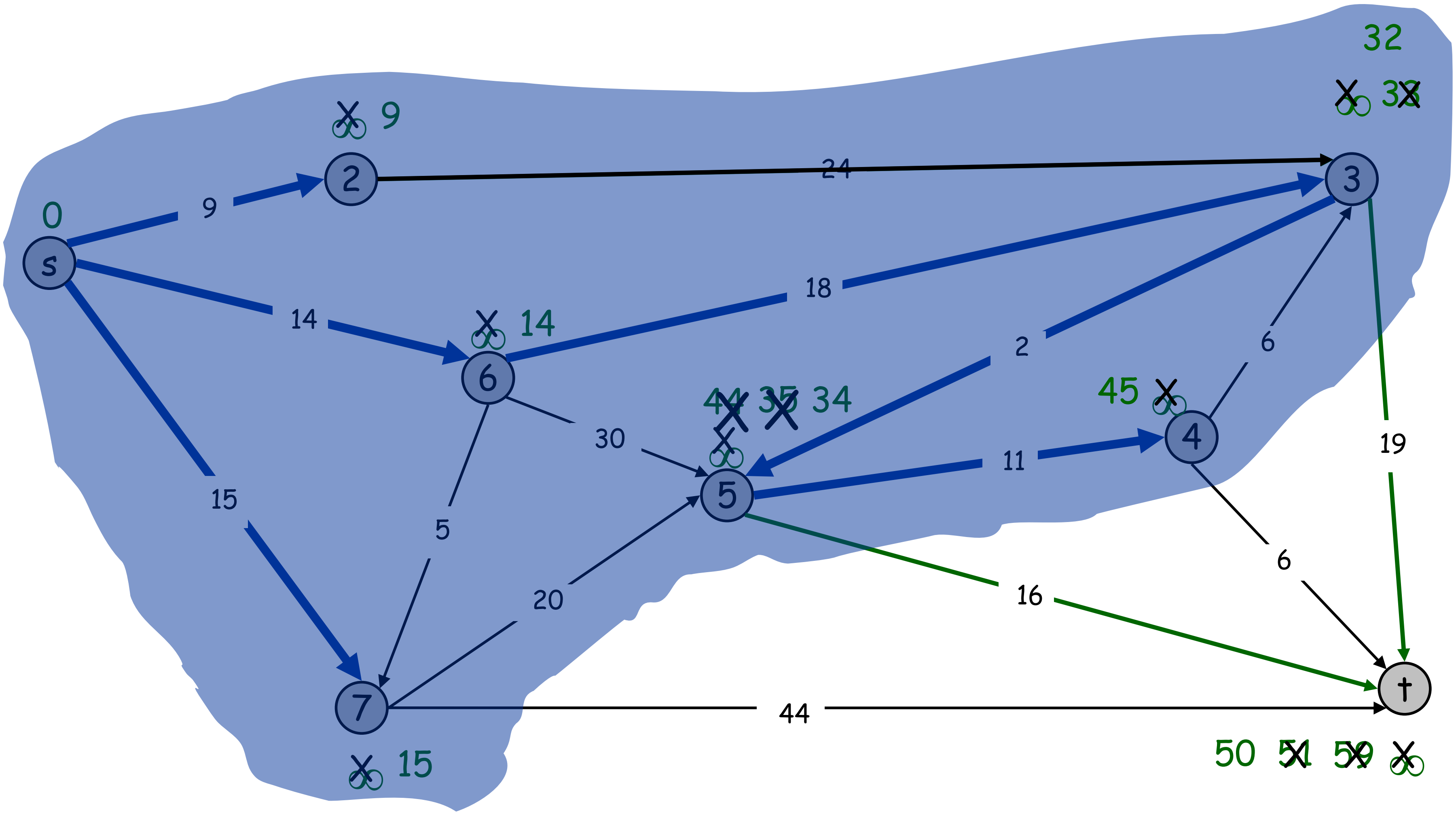
S = { s, 2, 3, 5, 6, 7 }
Q = { 4, t }



Dijkstra's Shortest Path Algorithm

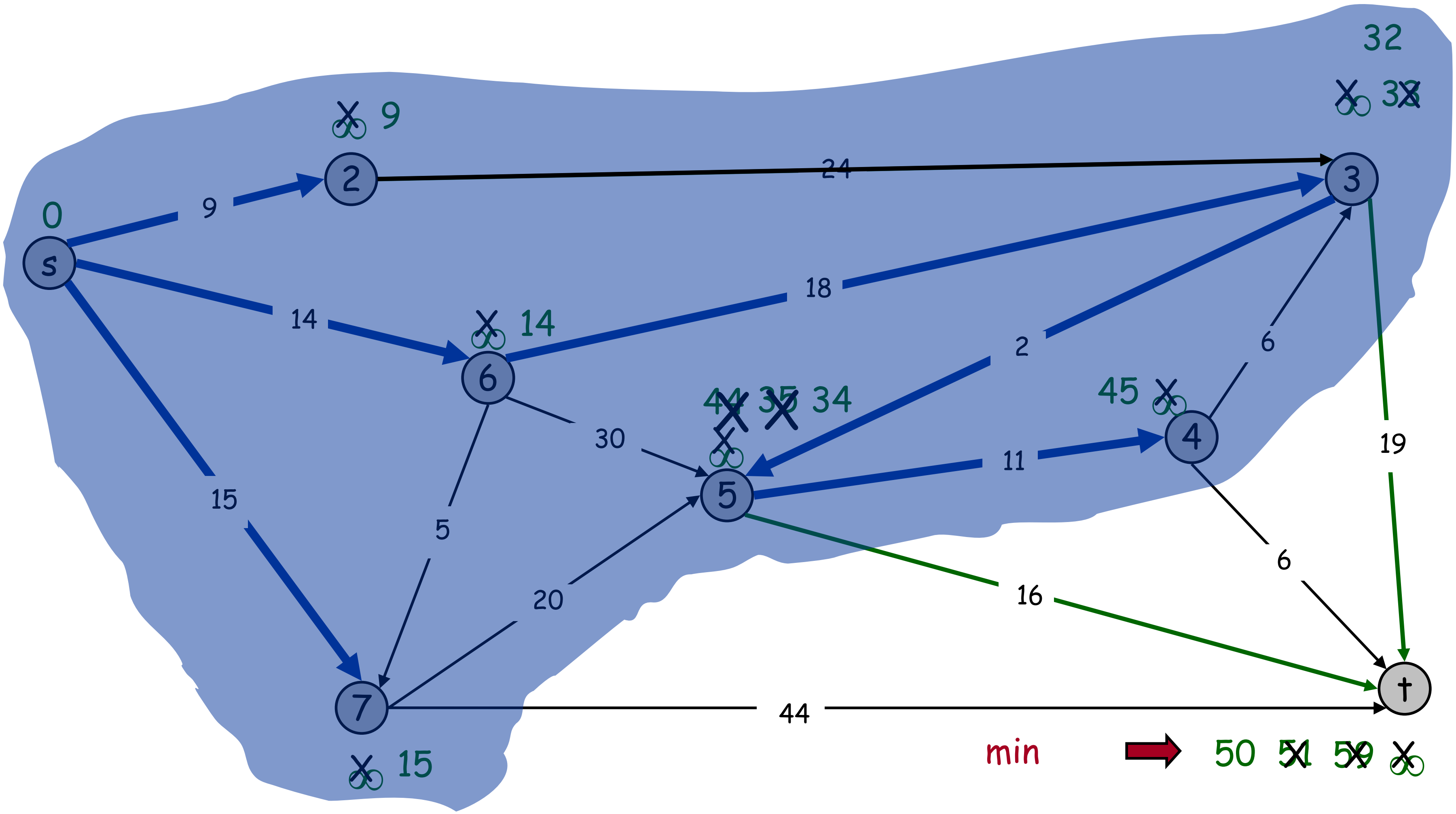
S = { s, 2, 3, 4, 5, 6, 7 }

Q = { t }



Dijkstra's Shortest Path Algorithm

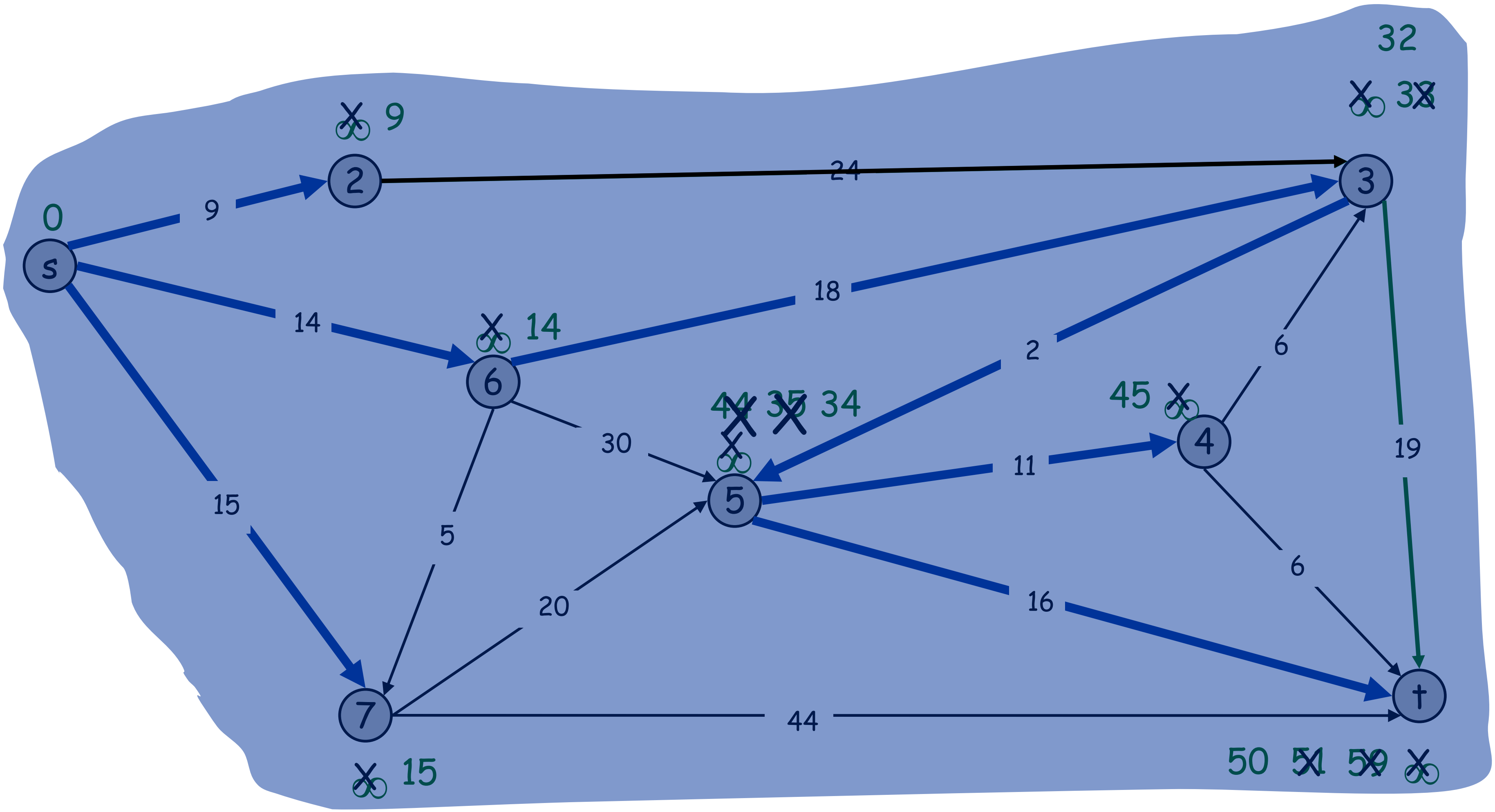
S = { s, 2, 3, 4, 5, 6, 7 }
Q = { t }



Dijkstra's Shortest Path Algorithm

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

Q = { }

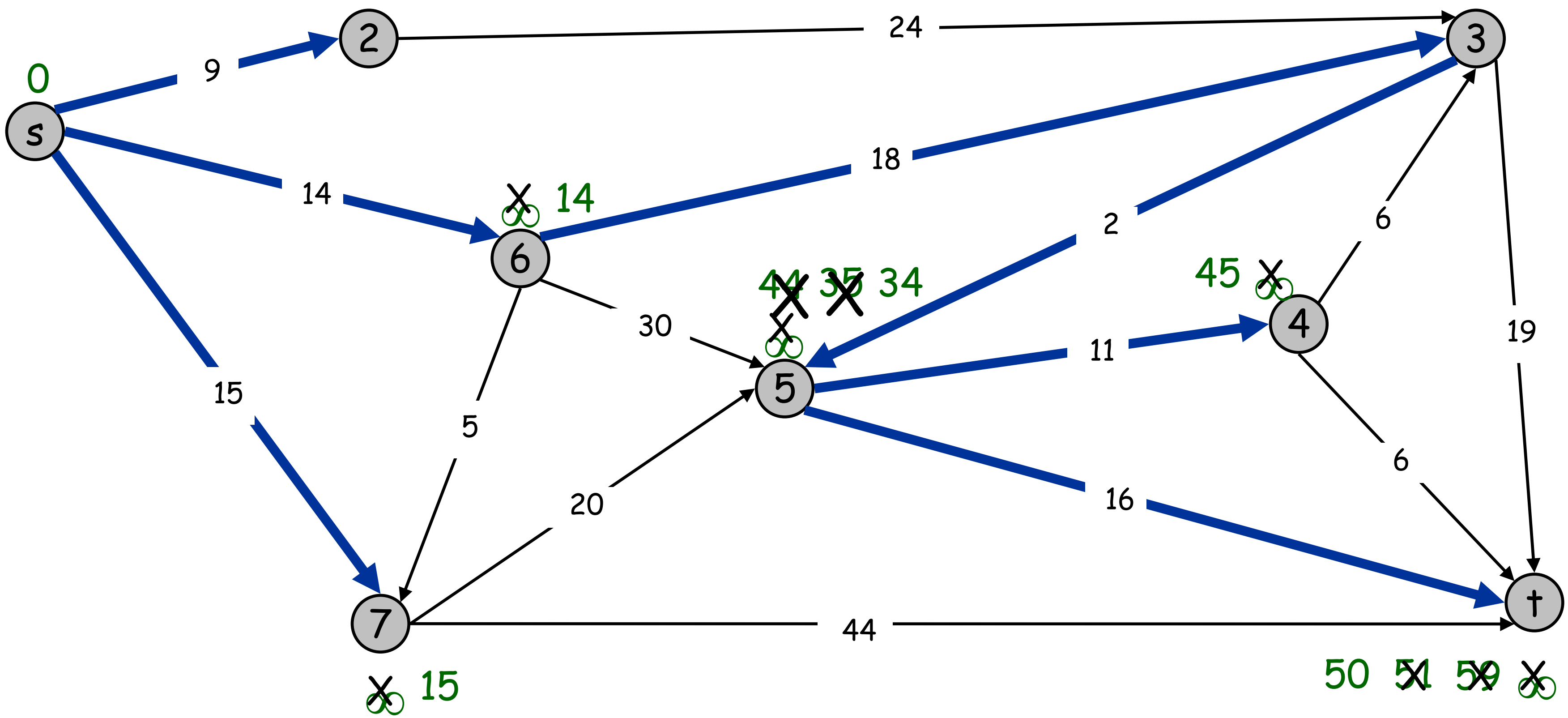


Dijkstra's Shortest Path Algorithm

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

Q = { }

Note: we've built a tree that "spans" the graph!



Correctness proof (sketch)

- Induction on the size of the graph
- P[n] = “algorithm works for all graphs with n nodes”

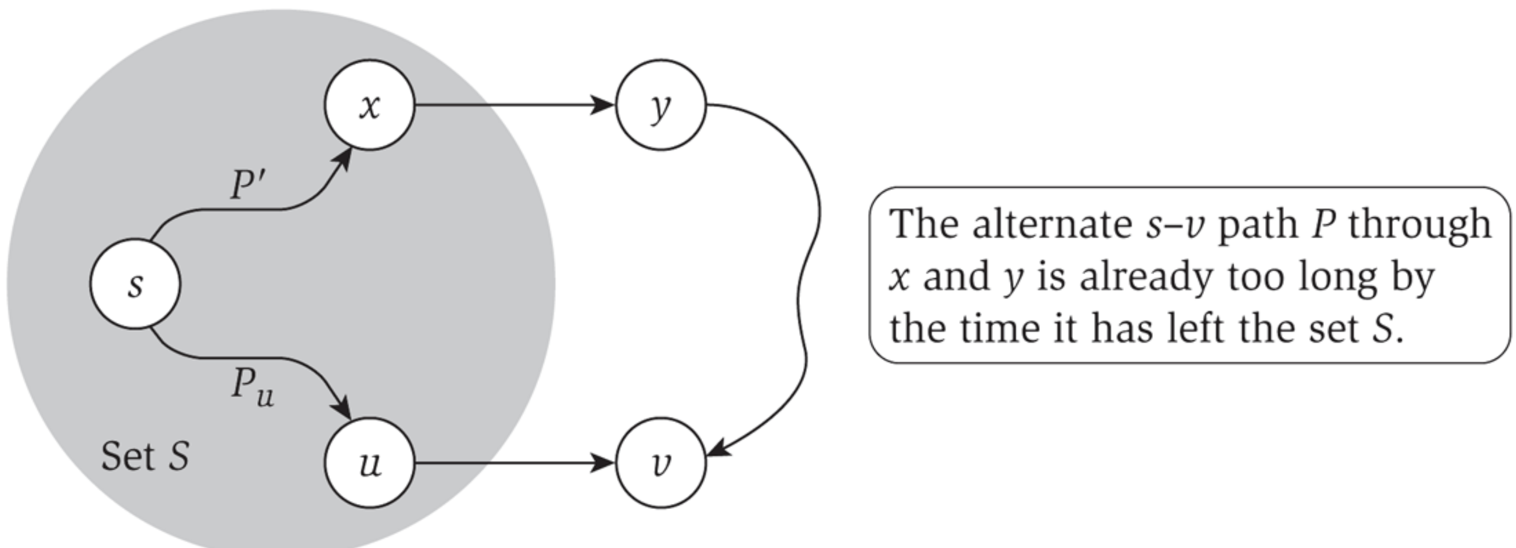


Figure 4.8 The shortest path P_v and an alternate s-v path P through the node y.

Applications and extensions

- Pirate's favorite sentence?
  - Is there a challenge in just using the probabilities as edge lengths?
  - How do we solve it, legitimately?
- All-pairs shortest paths
  - Easy solution: run from each source!
  - In practice, this is often best
    - But there are better asymptotic solutionsi