CS5112: Algorithms and Data Structures for Applications

Dijkstra’s algorithm

Ramin Zabih

Some slides from: K. Wayne

Cornell University

CORNELL TECH

Administrivia

- Web site is: https://github.com/cornelltech/CS5112-F19
  - As usual, this is pretty much all you need to know
  - Will be updated with lectures and other announcements
  - Also contains CMS link for student grades
- Quiz #1 will be out Thursday, due in 24 hours
  - Multiple choice, on the honor system
  - We drop your lowest quiz, these are mostly to help you keep up

Lecture Outline

- The shortest path problem
- Dijkstra’s algorithm
- Applications: image editing and pirate grammar
- Bonus application: modeling a CS5112 student

Two very common approaches in CS

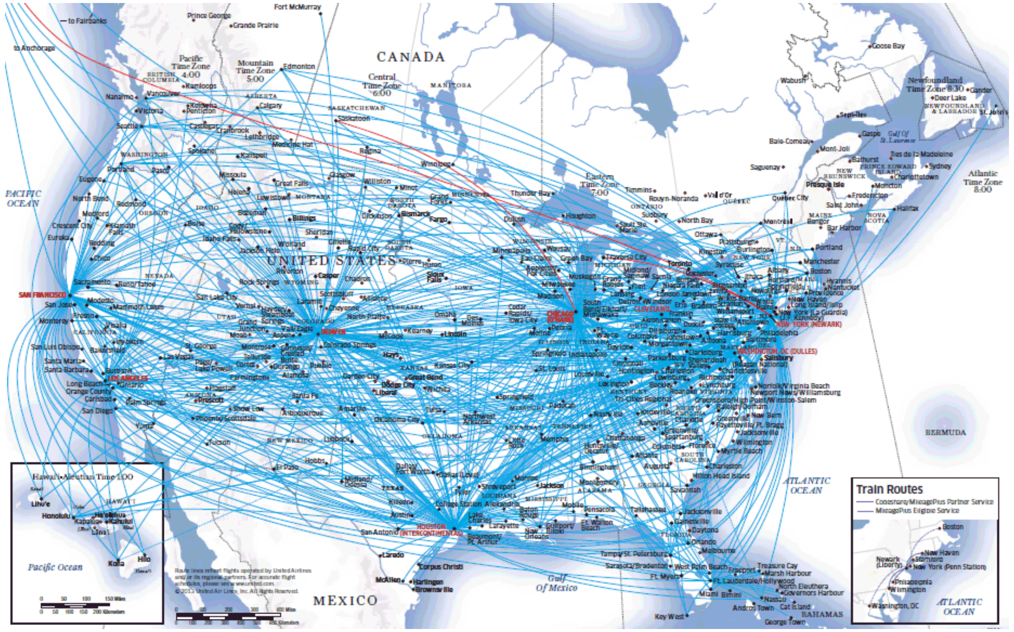
- Given a problem where you are searching for a solution:
  - Try everything (exhaustive search)
  - Do what seems best at the moment, repeatedly (greedy algorithms)
- Exhaustive search (almost) never works on serious problems
- Greedy algorithms are widely used
  - Currently famous example: SGD for neural networks
- Note: there are other approaches
  - Such as smart exhaustive search, e.g. dynamic programming

The shortest path problem

- 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 to solve!

Obvious application of shortest paths: airfare

- Nodes are cities, edges are direct flights, weights are airfare
- What is the **cheapest** way to get from LGA to Ithaca?
  - Presumably you can charter a plane



Fixing the problem definition

- Suppose that there is a flight from Boise to El Paso, and back again, that the airline pays you $1 to fly around
- Further, suppose that you can get to Boise (or El Paso)
- You can make an arbitrary amount of money by just flying back and forth!
- This is a cycle in the graph whose sum of weights is negative
- Easy solution: require positive edge weights
  – Or maybe detect negative cycles?

Not so obvious applications

- Making fake photographs
- Speech recognition/predicting stock prices by DTW
- Pirate grammar!
- Modeling a student (at end of class)

Making fake photographs

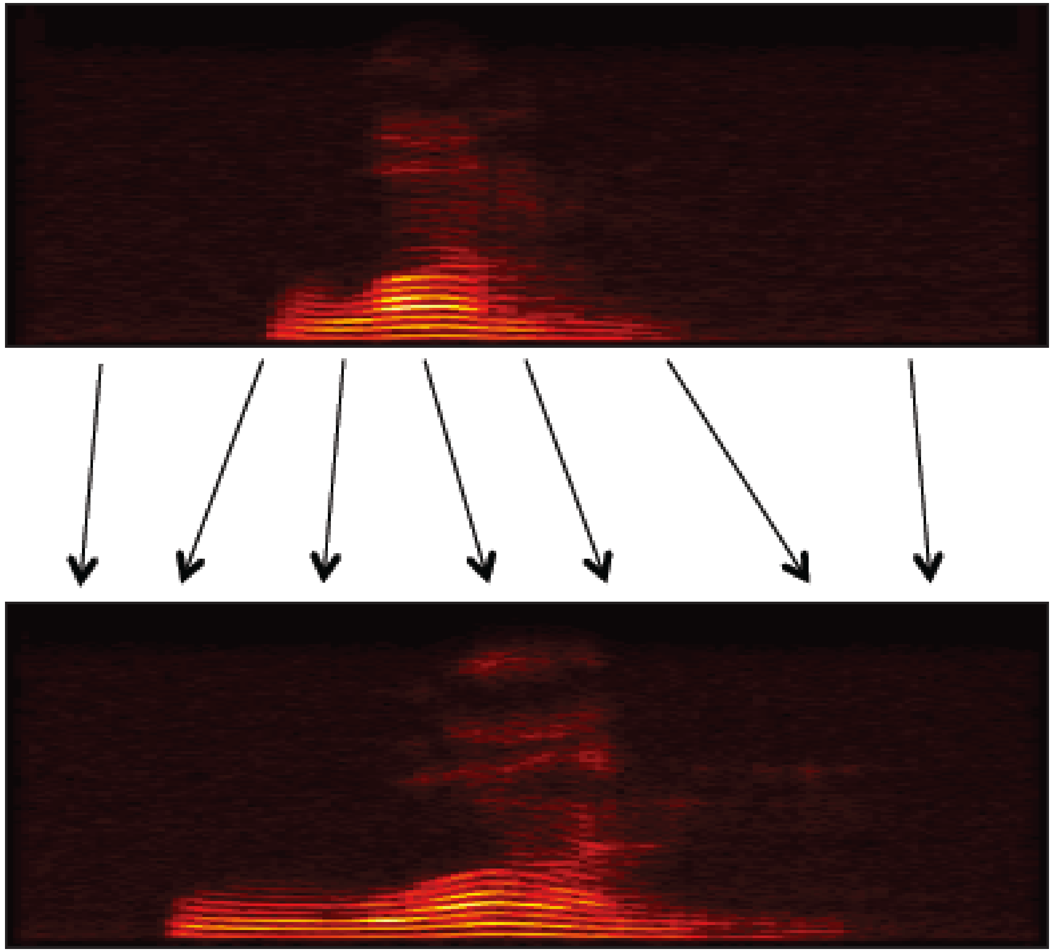
- How do we create images like this:

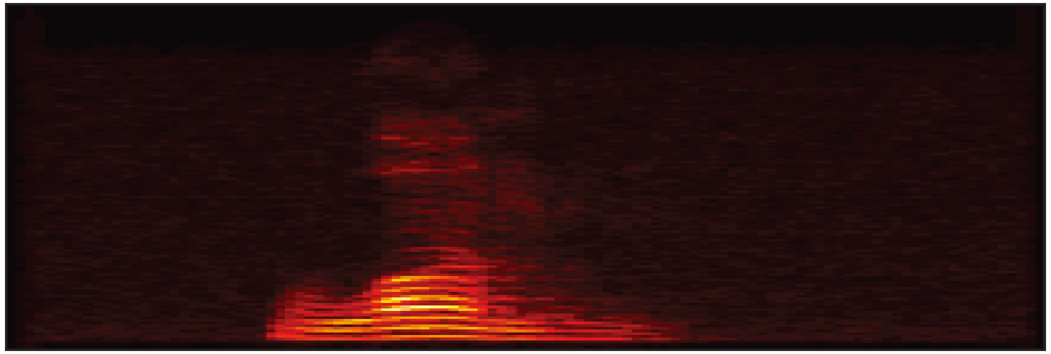
- Given an image, how do you cut out an object from it?
- You don't want to manually select the pixels

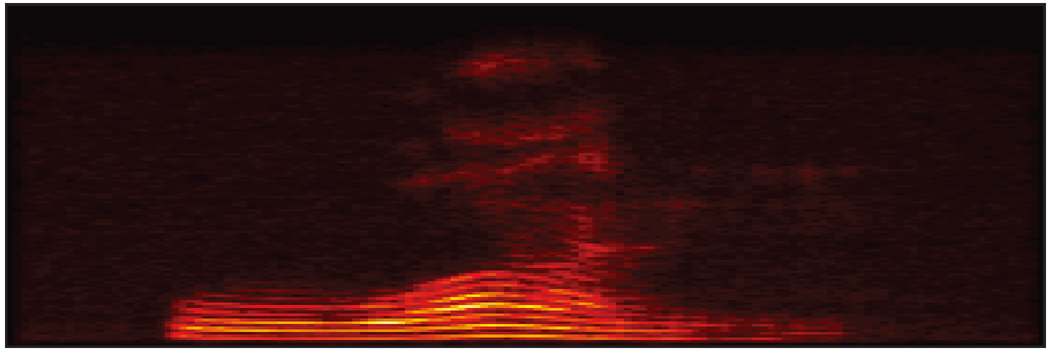
Intelligent scissors

- Idea: shortest paths
  - E.N. Mortensen and W.A. Barrett, Interactive Segmentation with Intelligent Scissors, SIGGRAPH 1995
- Adobe calls this the “Magnetic Lasso”
- Video here

Dynamic Time Warping (DTW)



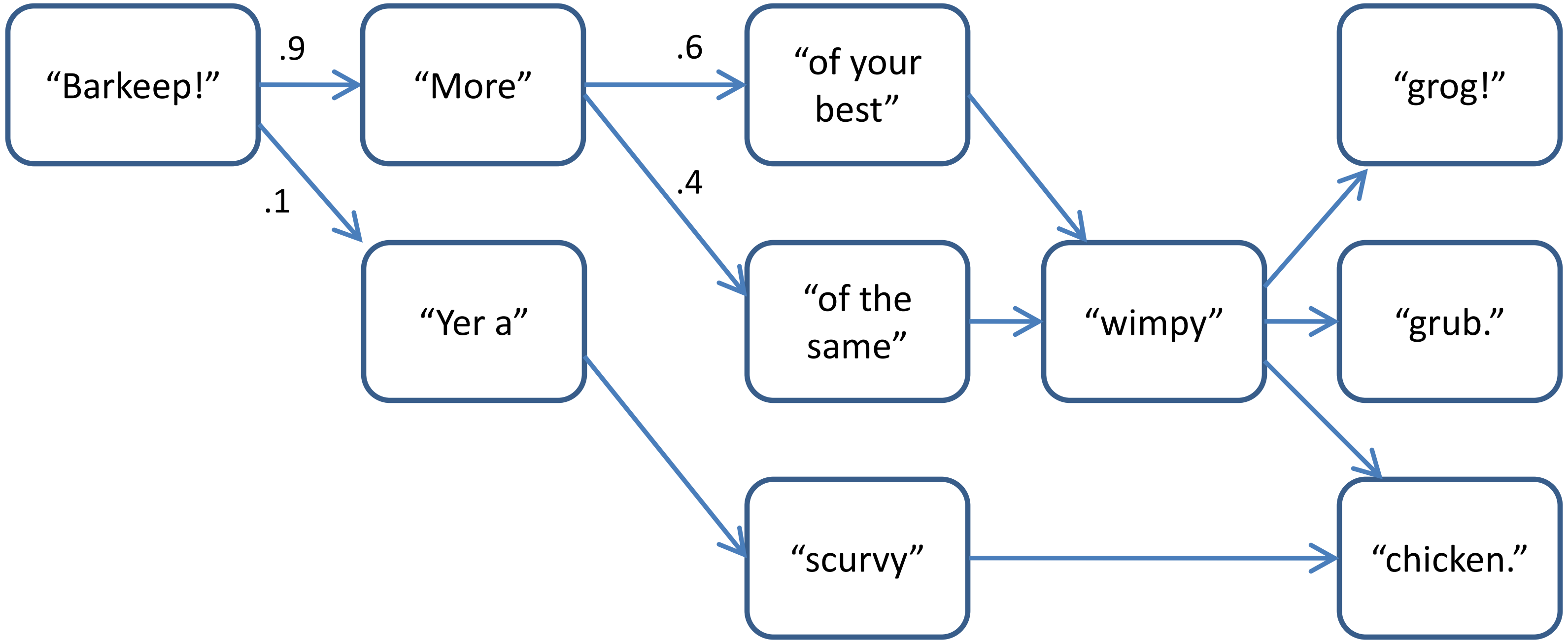




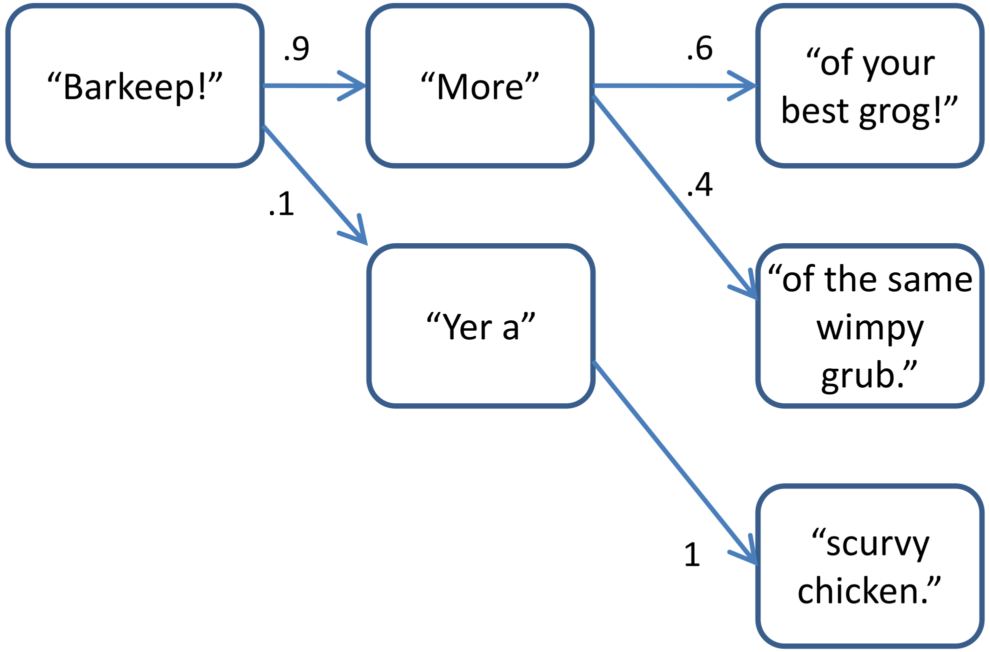
Rules of Pirate grammar

- Pirates always start their sentences with “Barkeep!”
  - 90% of the time they next say “More” (i.e., they order)
  - 10% of the time they next say “Yer a” (i.e., they insult)
  - If they say “More”, they next say:
    - 60% “Of your best”
    - 40% “Of the same”
- Lots more rules, discovered by experts in pirate linguistics
- Question: what sentence is a pirate most likely to say?

Pirate grammar as a graph



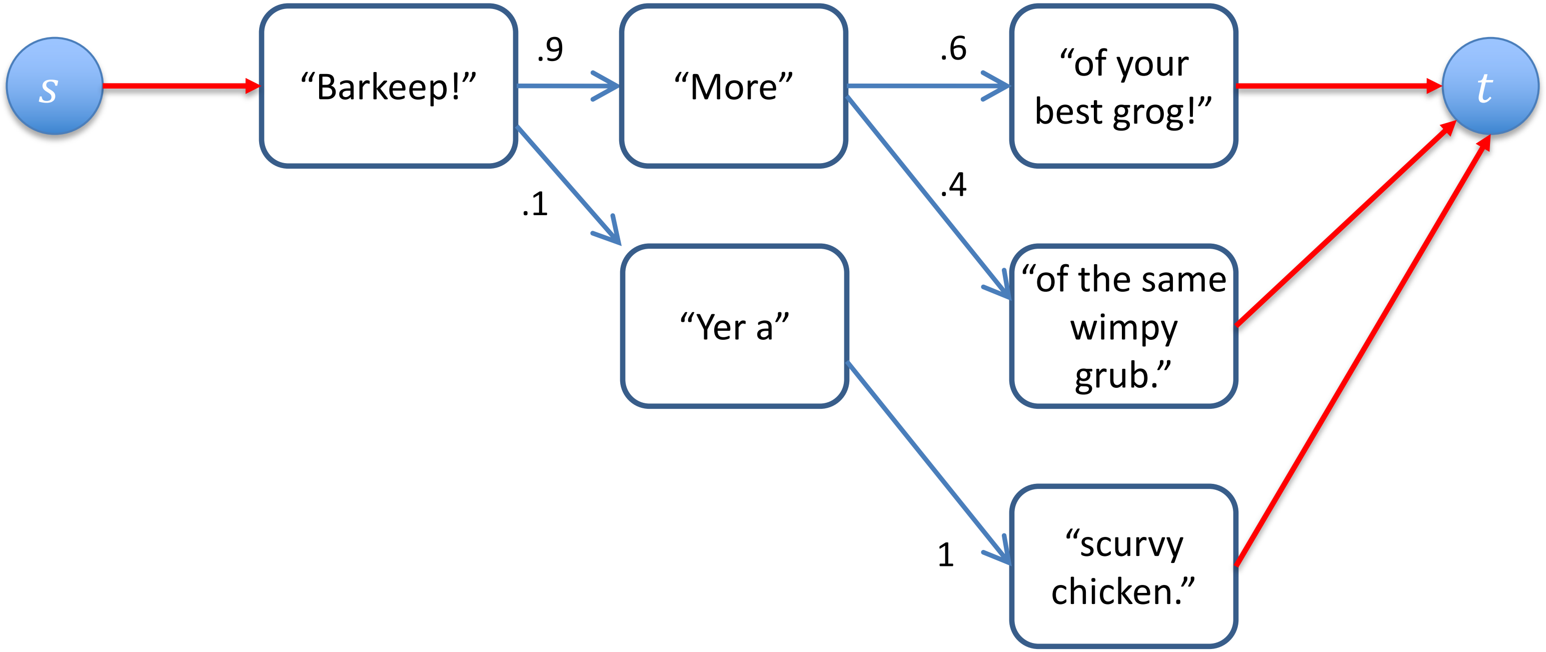
Simplified pirate grammar



How to make this into shortest paths?

- On the surface this is not at all obvious
  - Which is why this is worth thinking about carefully
- What we actually need to determine is the probability of any individual sentence
  - Example: “Barkeep! More of your best grog!” = .9 * .6 = .45
- So we look at all paths from the root to a leaf node
  - Each edge has a probability
  - Multiply these together and find the max
- This looks like “find the path where the product of the edges is maximized”, not “find the shortest path from s to t”

Easy part: Add a fake source and sink

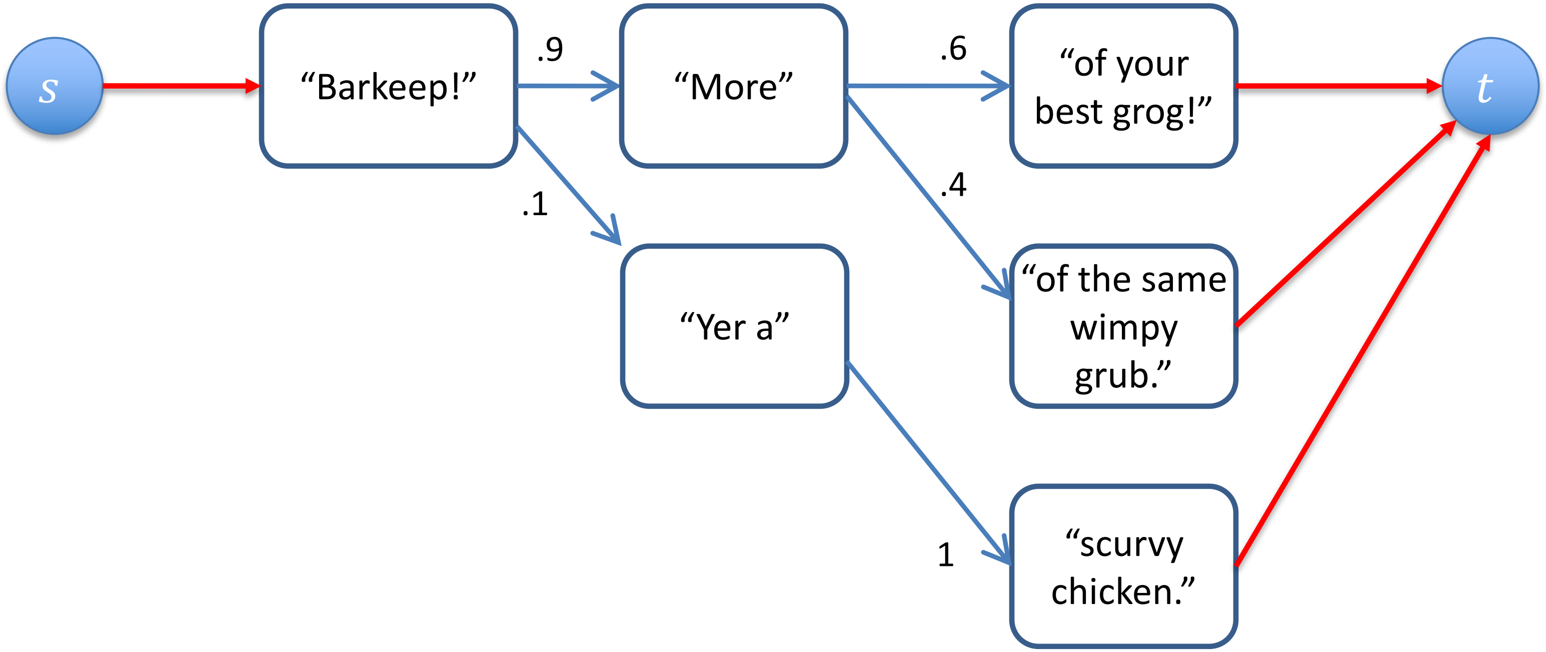


- Red links have probability 1
- Now we need to find the "highest product path" from s to t

Algebra to the rescue

- We want to maximize the product of edge probabilities
  - Which are numbers between 0 and 1
- Instead we need to minimize the sum of edge weights
- We know that log is monotonic, and $\log \prod_i p_i = \sum_i \log p_i$
- Maximize the product of edge probabilities = maximize the sum of log probabilities
  - Which are negative: $0 < p_i \leq 1 \Rightarrow \log p_i \leq 0$
- Maximizing anything is the same as minimizing its negative

Algebra in action



$\log_{10}(.9) \approx -0.046$
$\log_{10}(.1) = -1$

Key property of shortest paths

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

    shortest s-v path

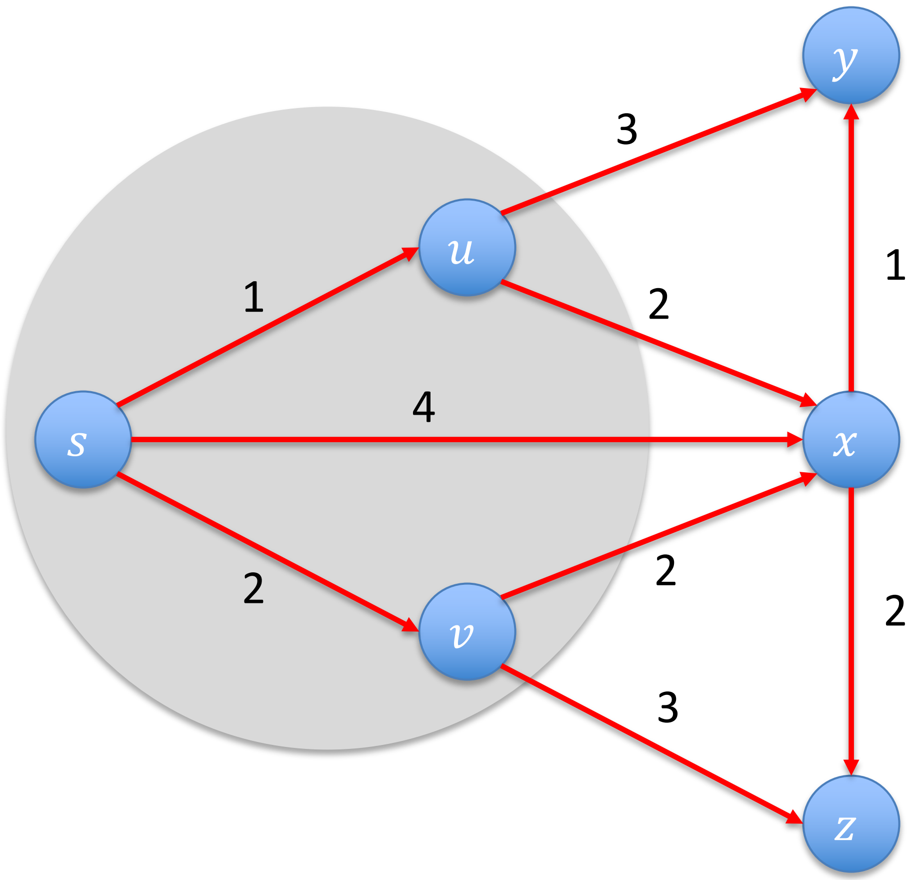
  - Otherwise, we would take that "shortcut" instead, and create an even shorter path
- Considering $s - v - t$ paths, only need shortest $s - v$ path
  - Don't need to try everything!
- This is basically the optimal substructure of shortest paths

Shortest paths by increasing budgets

- Here is the basic idea, which we will simply speed up
- Where can you fly from LGA on a $1 budget?
  - Does that get you to Ithaca?
- If it does, you are done
- If not, add $1 to your budget and do it again
- You can think of this as expanding a ball around s until you eventually get to t
  - Though we are doing this on a graph

Example

- For \$1 can get to u
- For \$2 can also get to v
- Gray area shows budget at \$2
- At \$3 we can also get to x via u
- Key concepts:
  - Explored nodes: $\{s, u, v\}$
  - Fringe: $\{x, y, z\}$

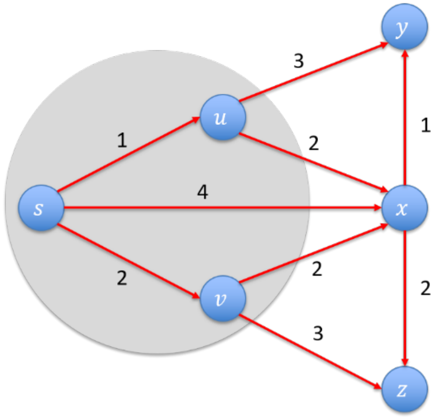


Key concepts

- Explored nodes: we know the cheapest way to get there
  – Shown as inside the gray zone
- Fringe nodes: unexplored and adjacent to an explored node
- When we increase the budget we add a fringe node into the set of explored nodes
  – This is pretty inefficient, hold that thought
- Keep on doing this until t (i.e. Ithaca) is in the explored nodes

Budget approach is crazy

- Suppose the cheapest flight from LGA is \$500
- In our example, imagine increasing by \$.01
  - So we consider \$2.01, \$2.02, …
- But we know that nothing will happen until we increase our budget to \$3
  - Why not just do this directly?

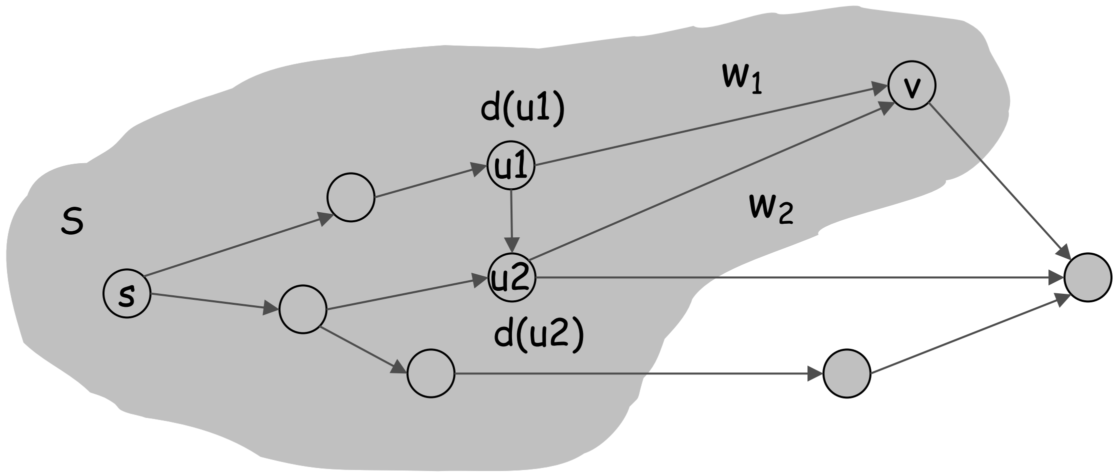


Dijkstra’s algorithm

- We maintain an explored set S with an invariant:
  - For each $u \in S$ 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 $d(u)$
  - Add an unexplored node v to S
    - But, which one to choose?
    - On the fringe of S, so we add just one edge

Choice of edge for a fringe node

- The fringe node v can be adjacent to several nodes in S
  - If we choose to add v, pick the right node in S to connect it to

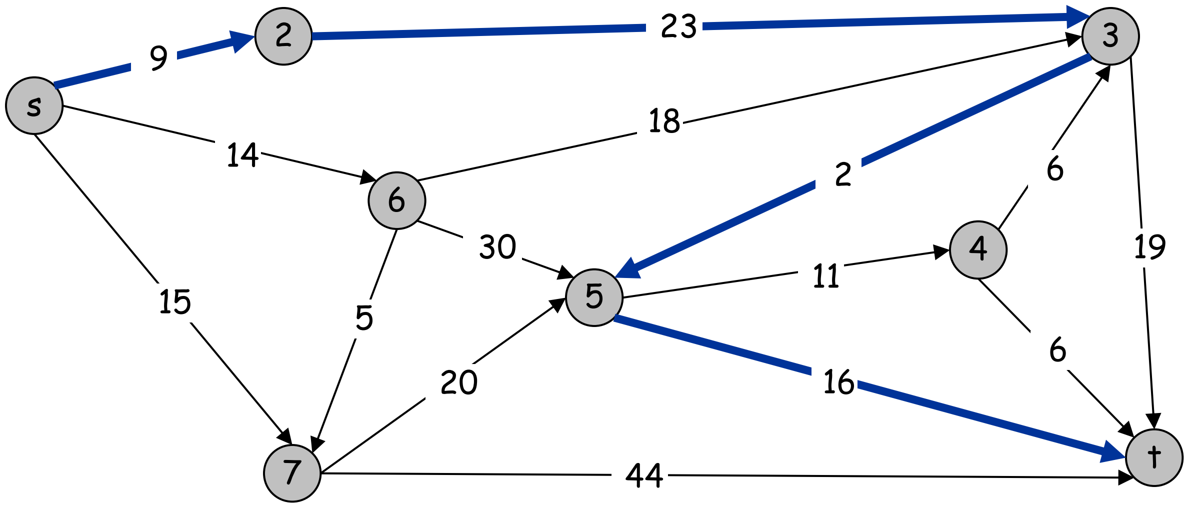


$d(u1) + w_1$
versus
$d(u2) + w_2$

Choice of fringe 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 π(v)
  - Think of this as "cheapest price to add v to S"
  - But can we pick an arbitrary v to add?
- Can prove that this would break our invariant about S!
- Pick v with smallest π(v), then add it to S with $d(v)$ = π(v)

Shortest path example



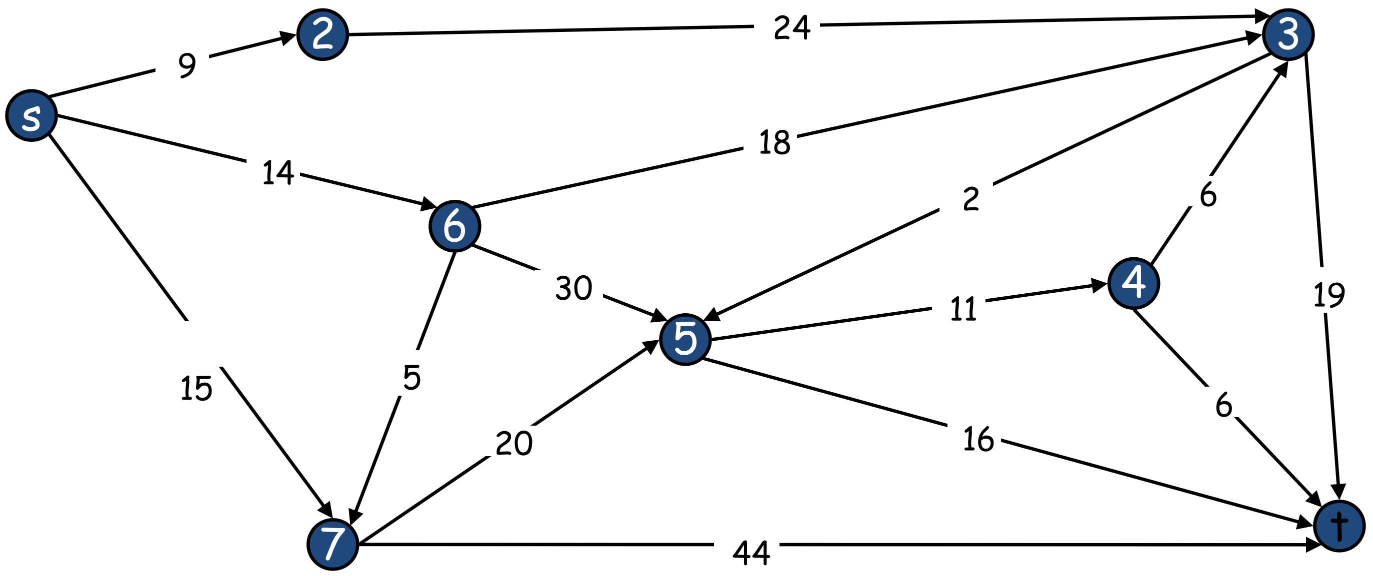
Cost of path s-2-3-5-t
= 9 + 23 + 2 + 16
= 48.

Dijkstra's algorithm

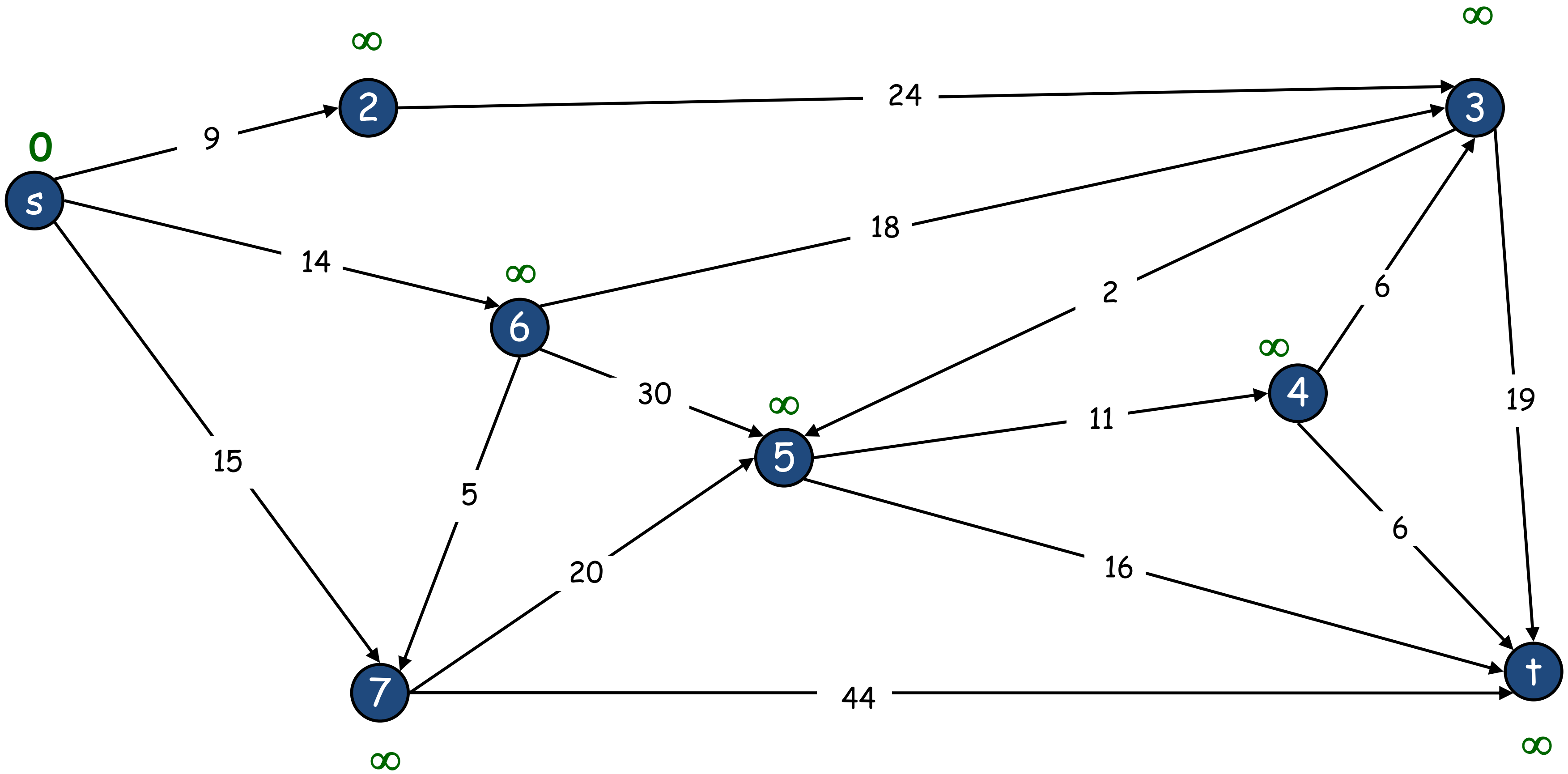
- Start with $S = \{s\}$, all other nodes in Q
  - $d(s) = 0$, else $d(v) = \infty$ (i.e. upper bound)
- Pick v on fringe of S that minimizes $\pi(v)$
  - I.e., the $v \in Q$ with a neighbor in S that is cheapest to add to S
- On recursive call, we will have
  - $d(v) = \pi(v)$
  - v is now in S, and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

Dijkstra's Shortest Path Algorithm

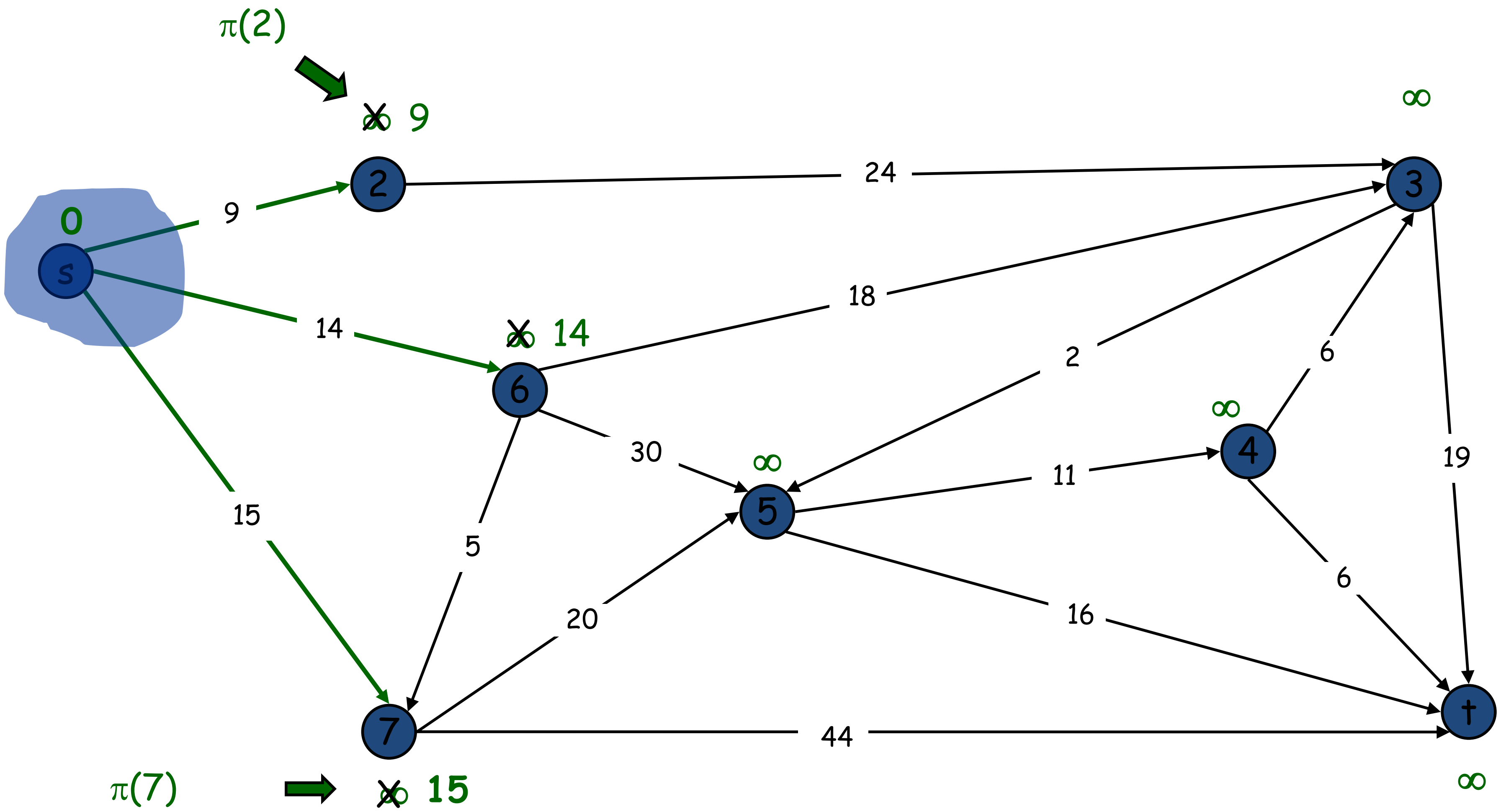
- Find shortest path from s to t.
- Blue edges: shortest path to a node within S.
- Green edges: what we would add for each fringe vertex.



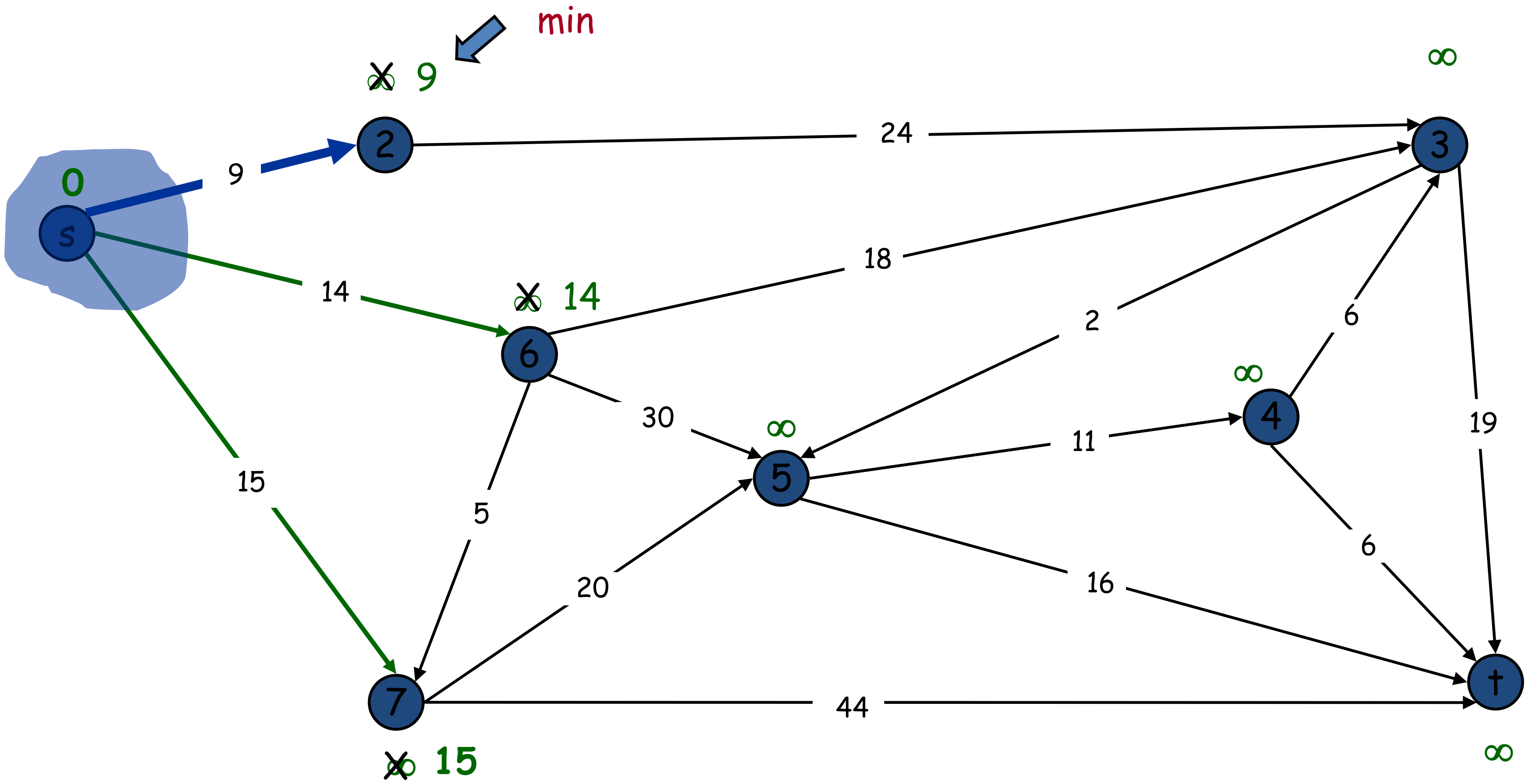
$S = \{ s \}$
$Q = \{ 2, 3, 4, 5, 6, 7, t \}$



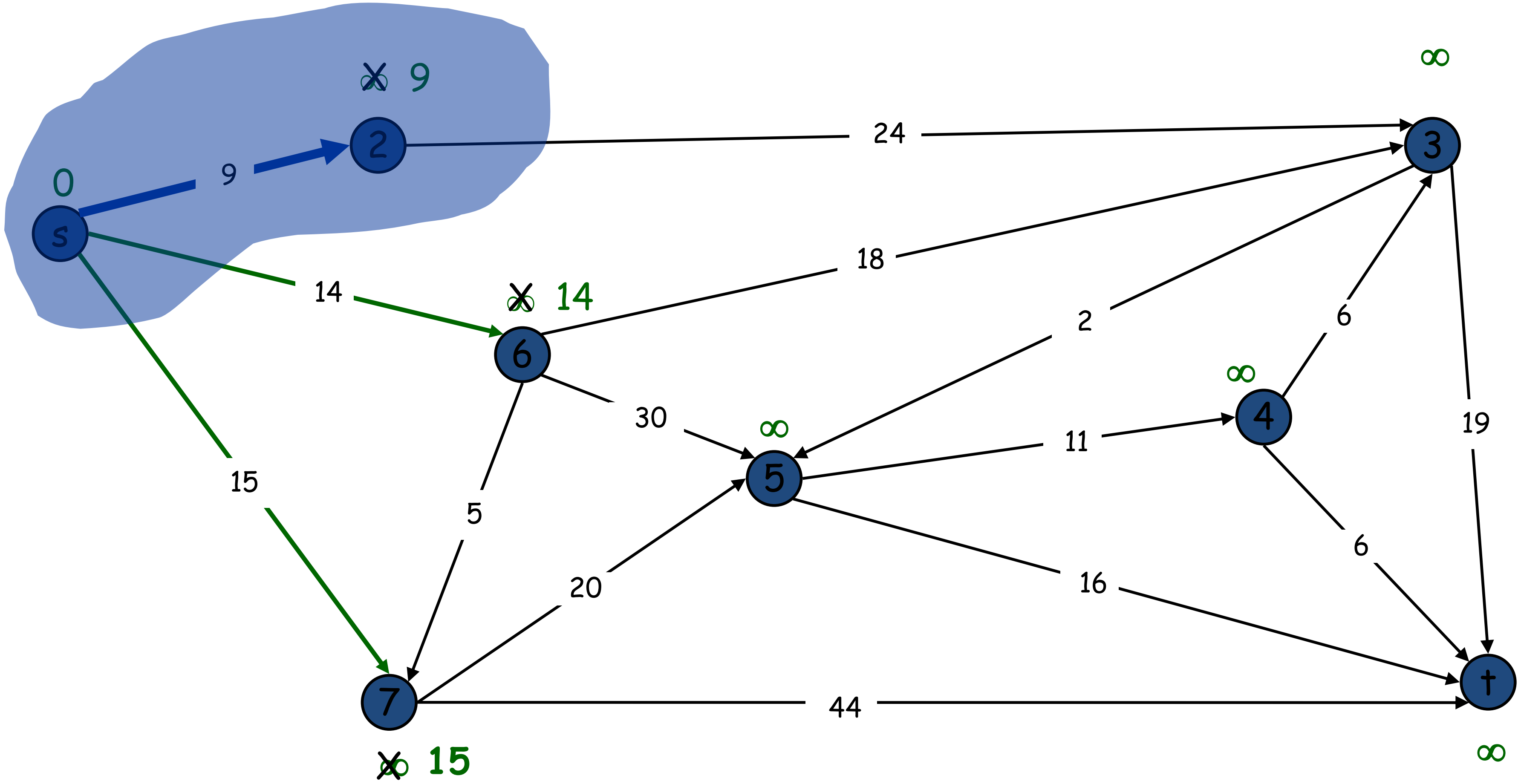
S = { s }
Q = { 2, 3, 4, 5, 6, 7, t }



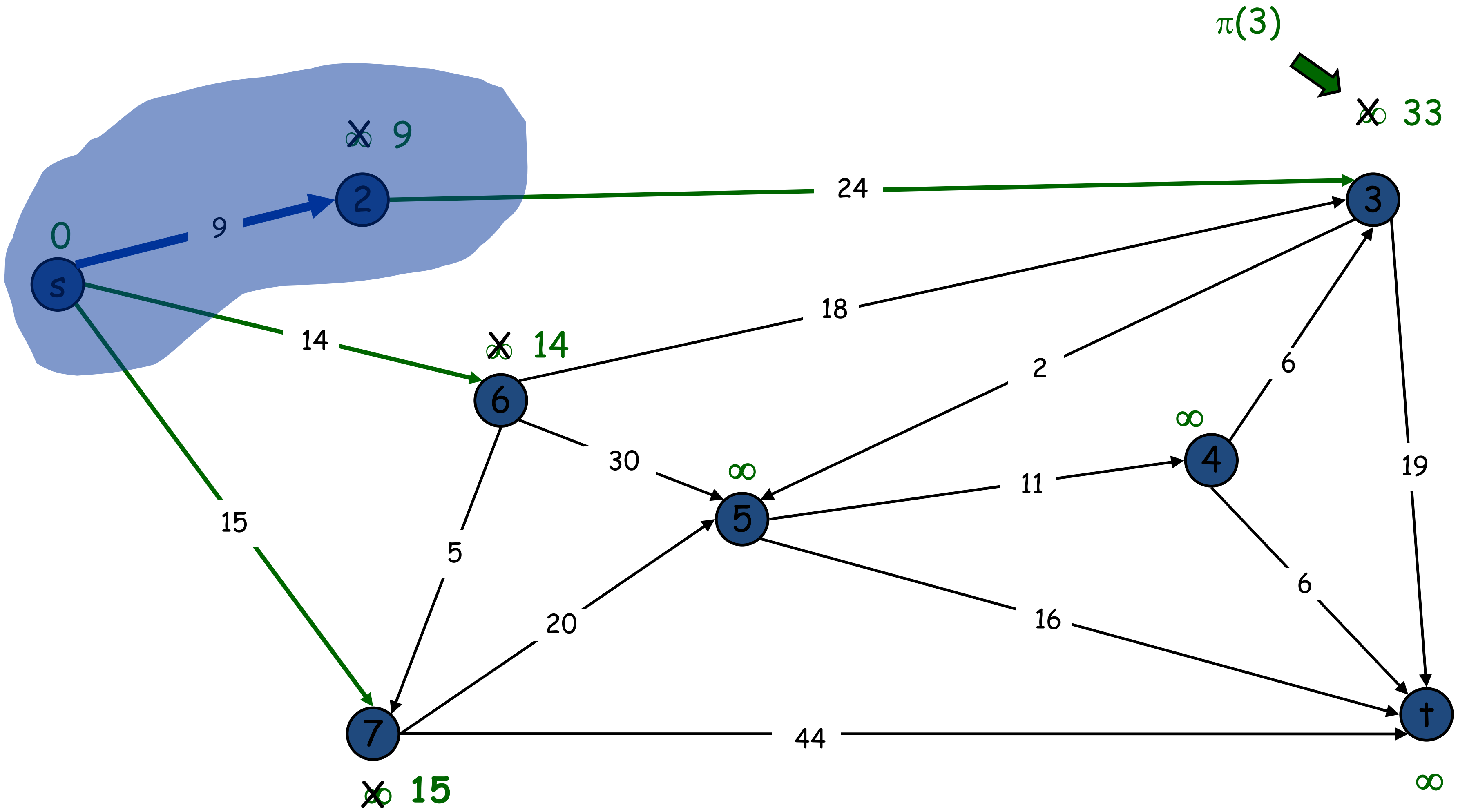
S = { s }
Q = { 2, 3, 4, 5, 6, 7, t }



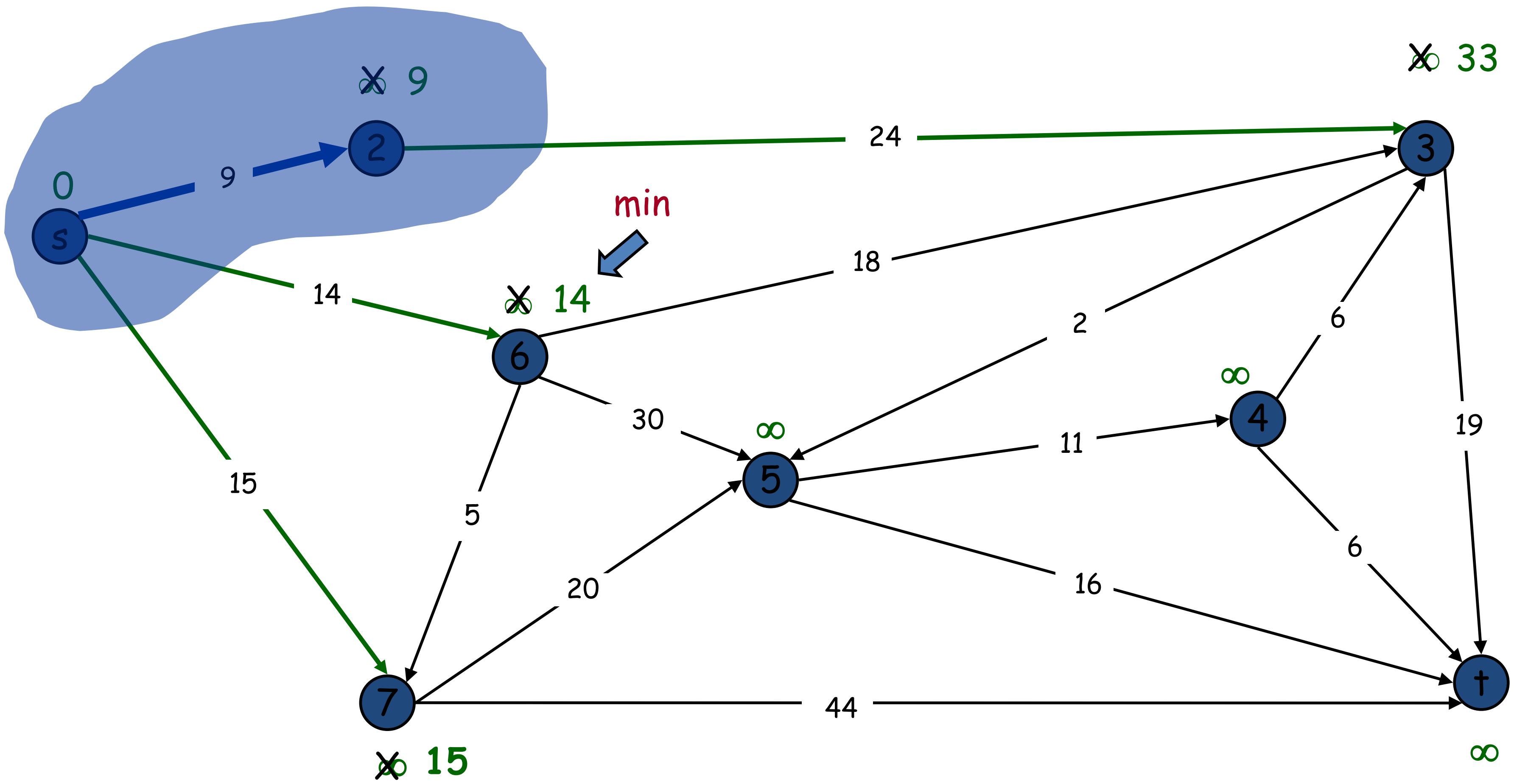
S = { s, 2 }
Q = { 3, 4, 5, 6, 7, t }



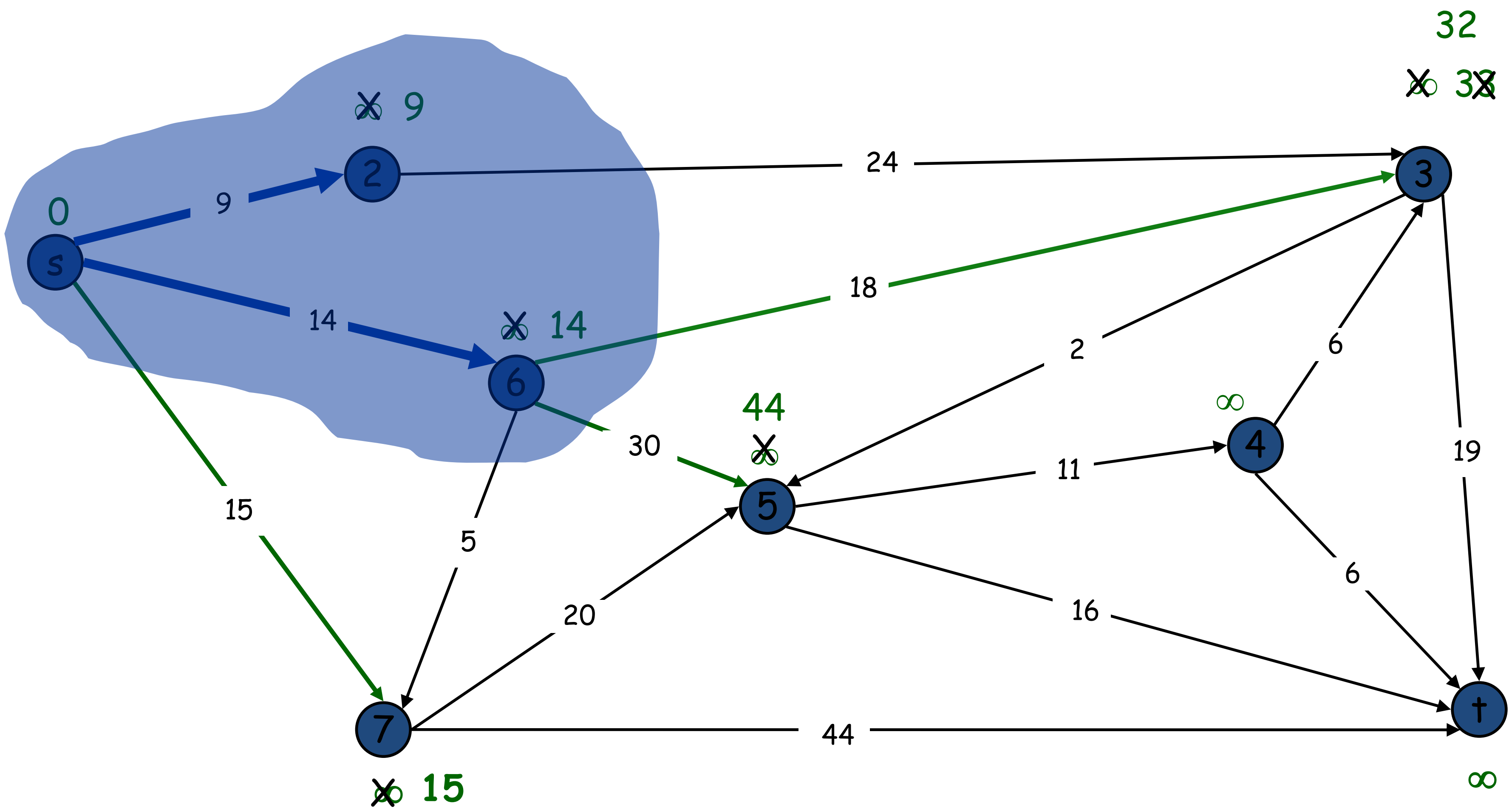
S = { s, 2 }
Q = { 3, 4, 5, 6, 7, t }



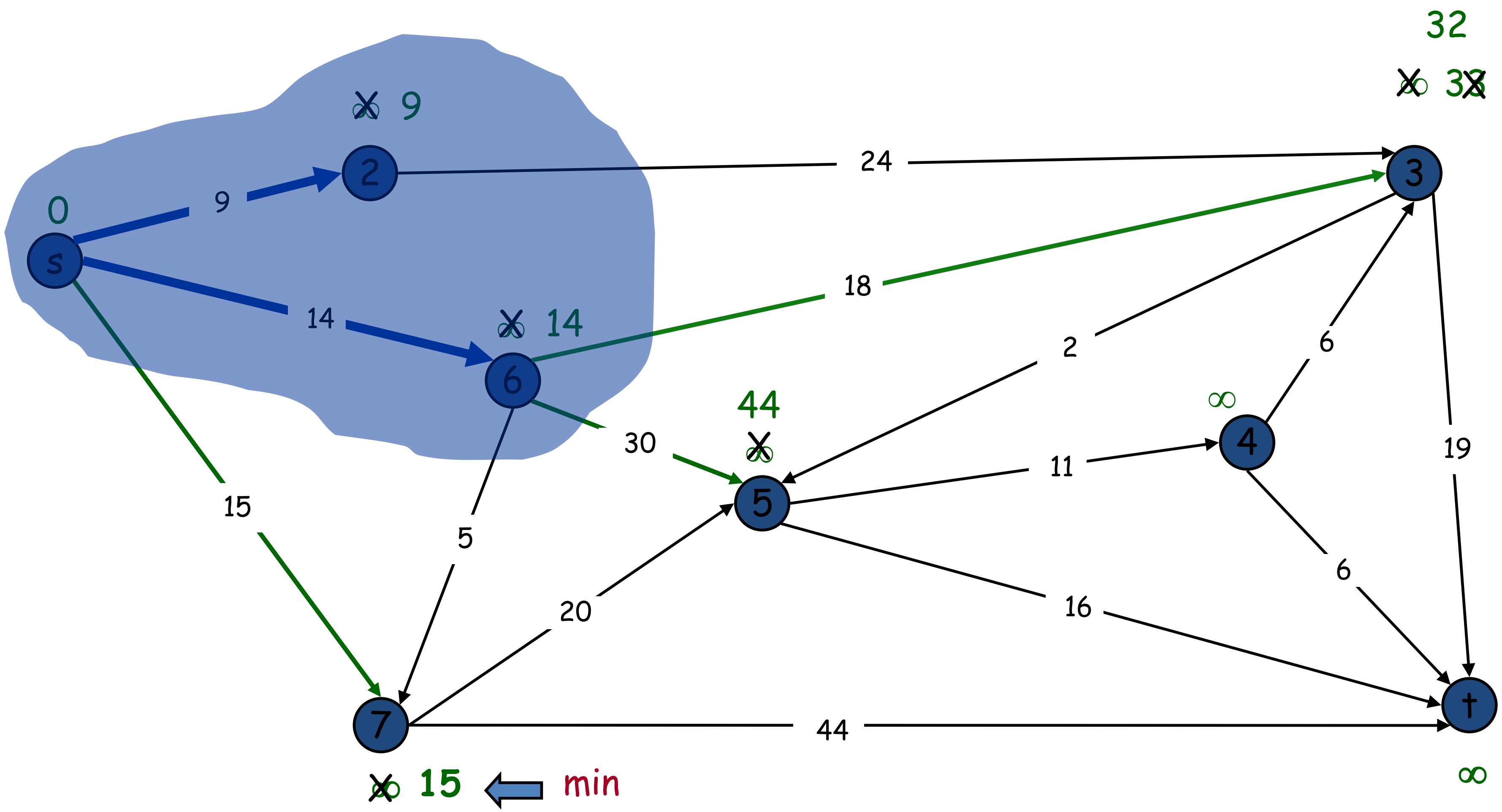
S = { s, 2 }
Q = { 3, 4, 5, 6, 7, t }



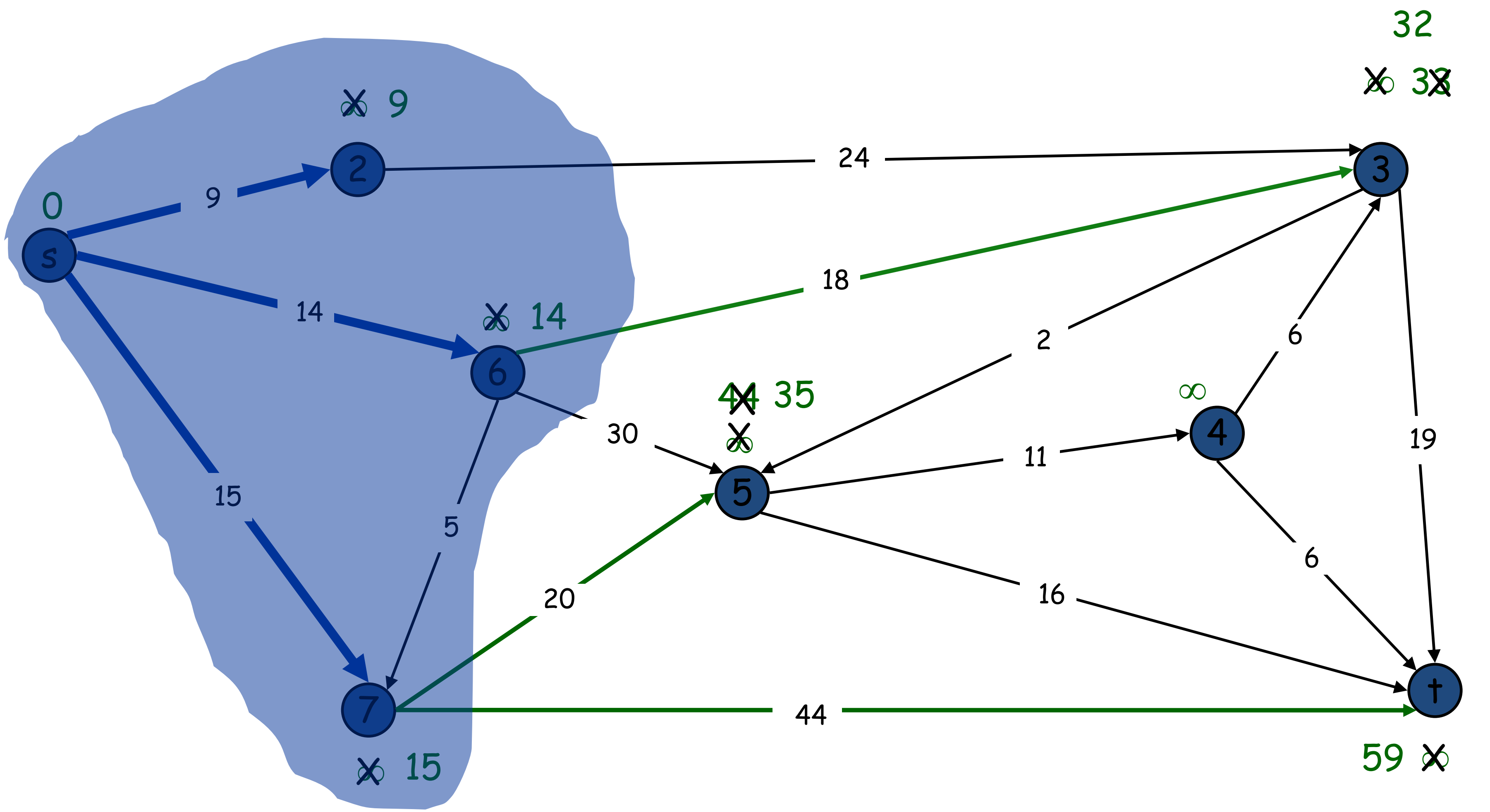
S = { s, 2, 6 }
Q = { 3, 4, 5, 7, t }



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

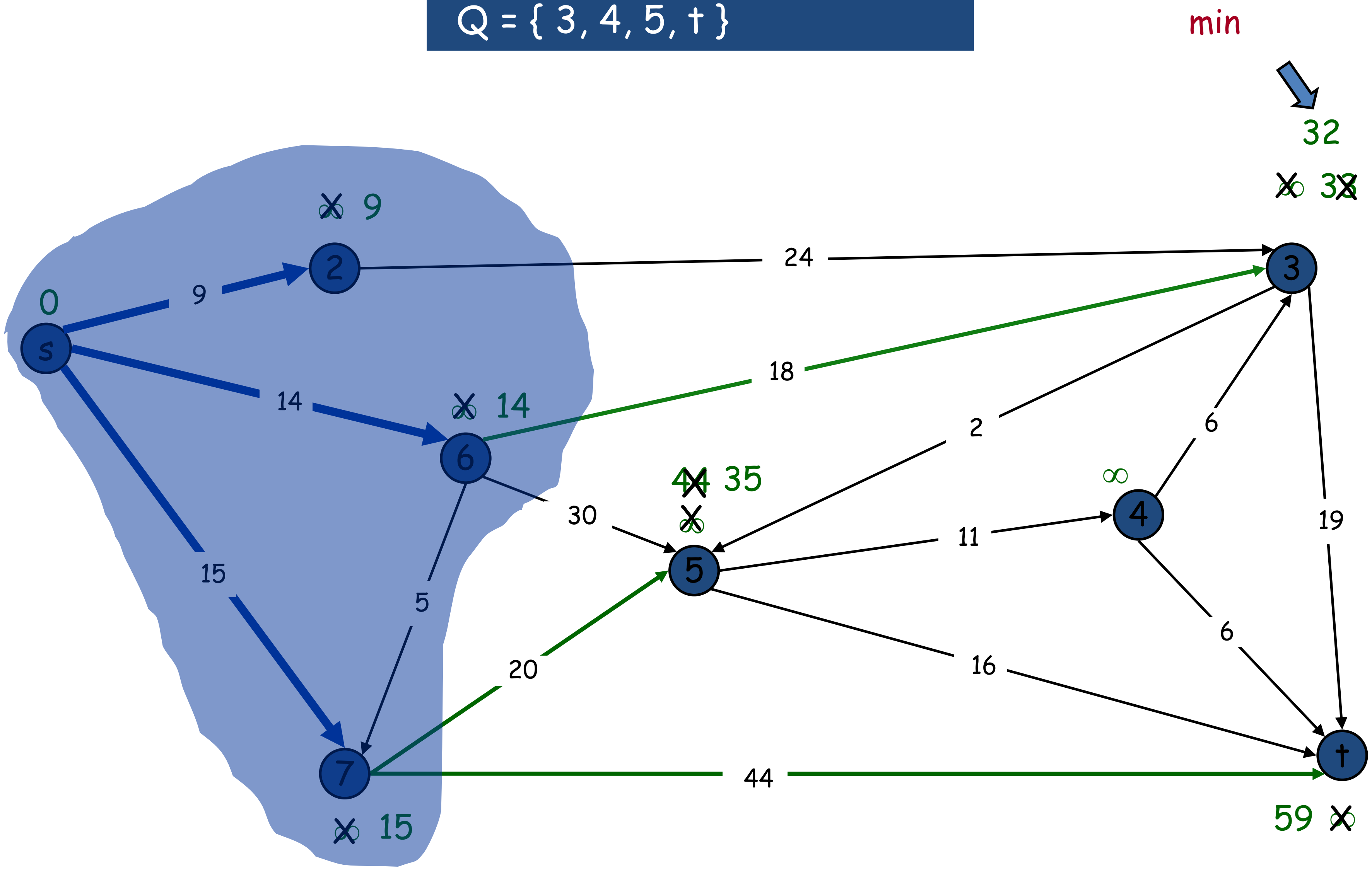


S = { s, 2, 6, 7 }
Q = { 3, 4, 5, t }



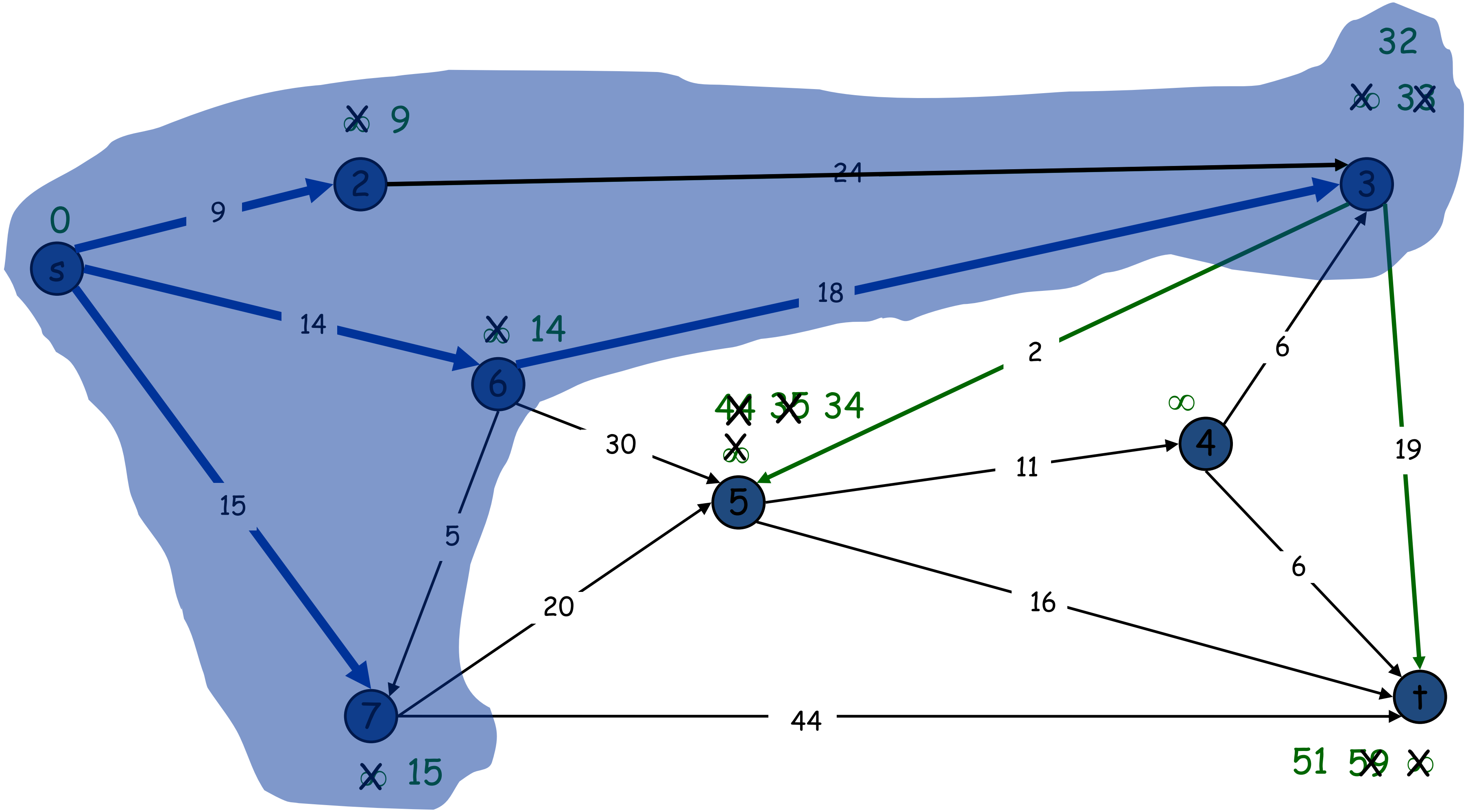
$S = \{ s, 2, 6, 7 \}$

$Q = \{ 3, 4, 5, t \}$

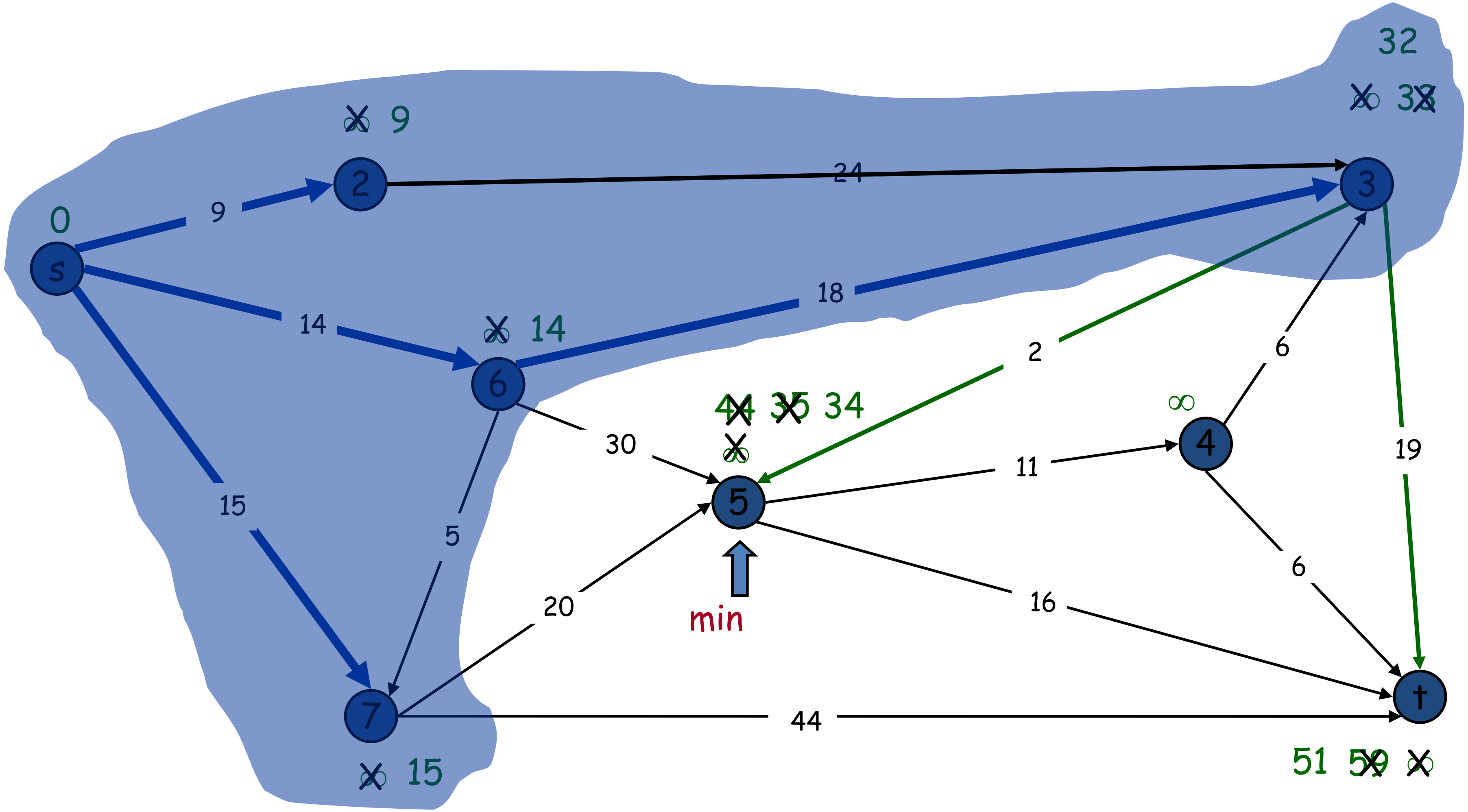


min
32
∞ 33
∞ 9
0
s
2
3
24
9
18
14
∞ 14
6
2
6
44 35
∞
∞
4
30
11
19
5
15
5
6
20
16
7
t
44
∞ 15
59 ∞

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

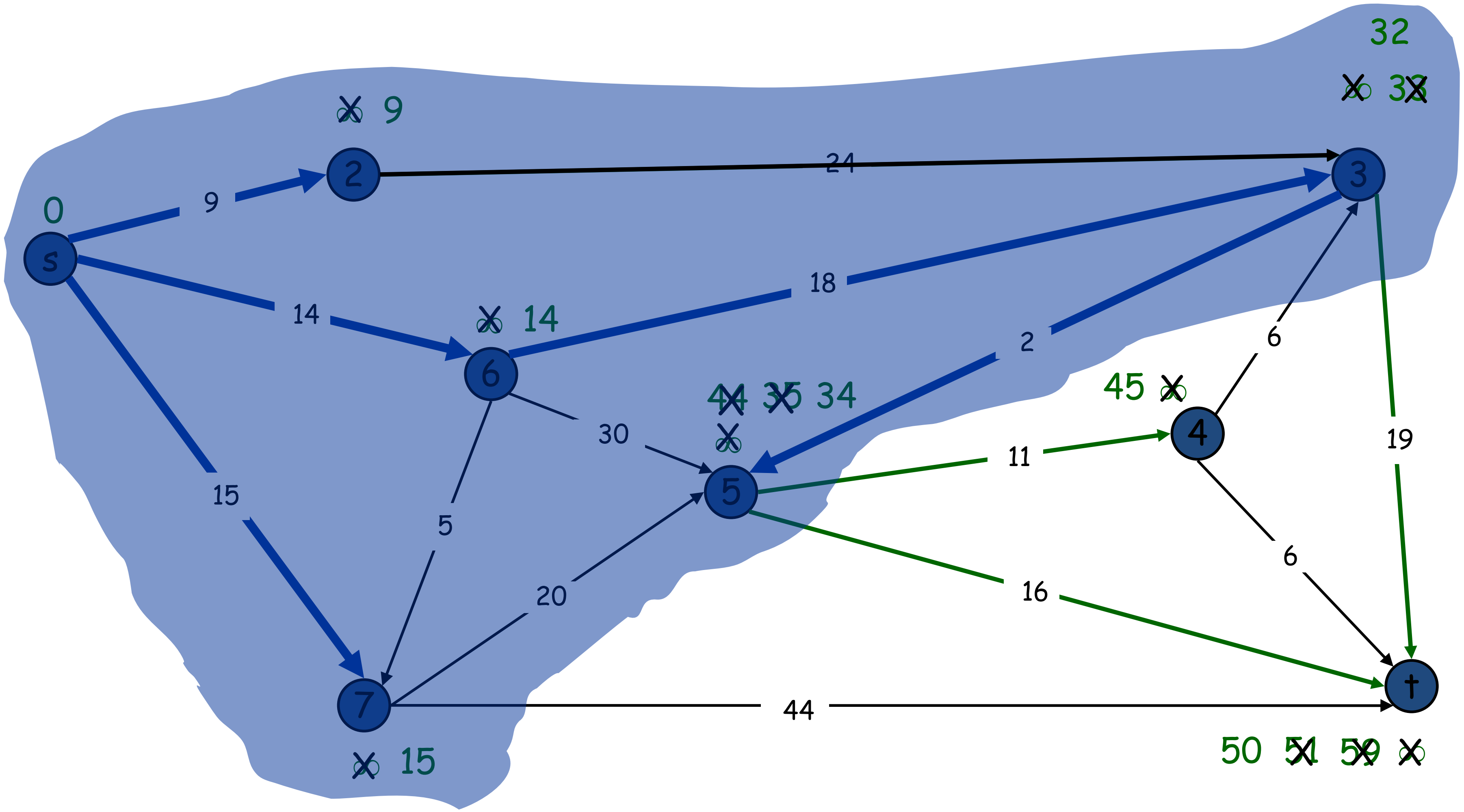

32
∞ 33 32
∞ 9
0
s
2
3
9
24
14
18
∞ 14
6
2
6
44 35 34
∞
∞
4
5
30
11
19
15
5
20
16
6
7
t
44
∞ 15
51 59 ∞

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



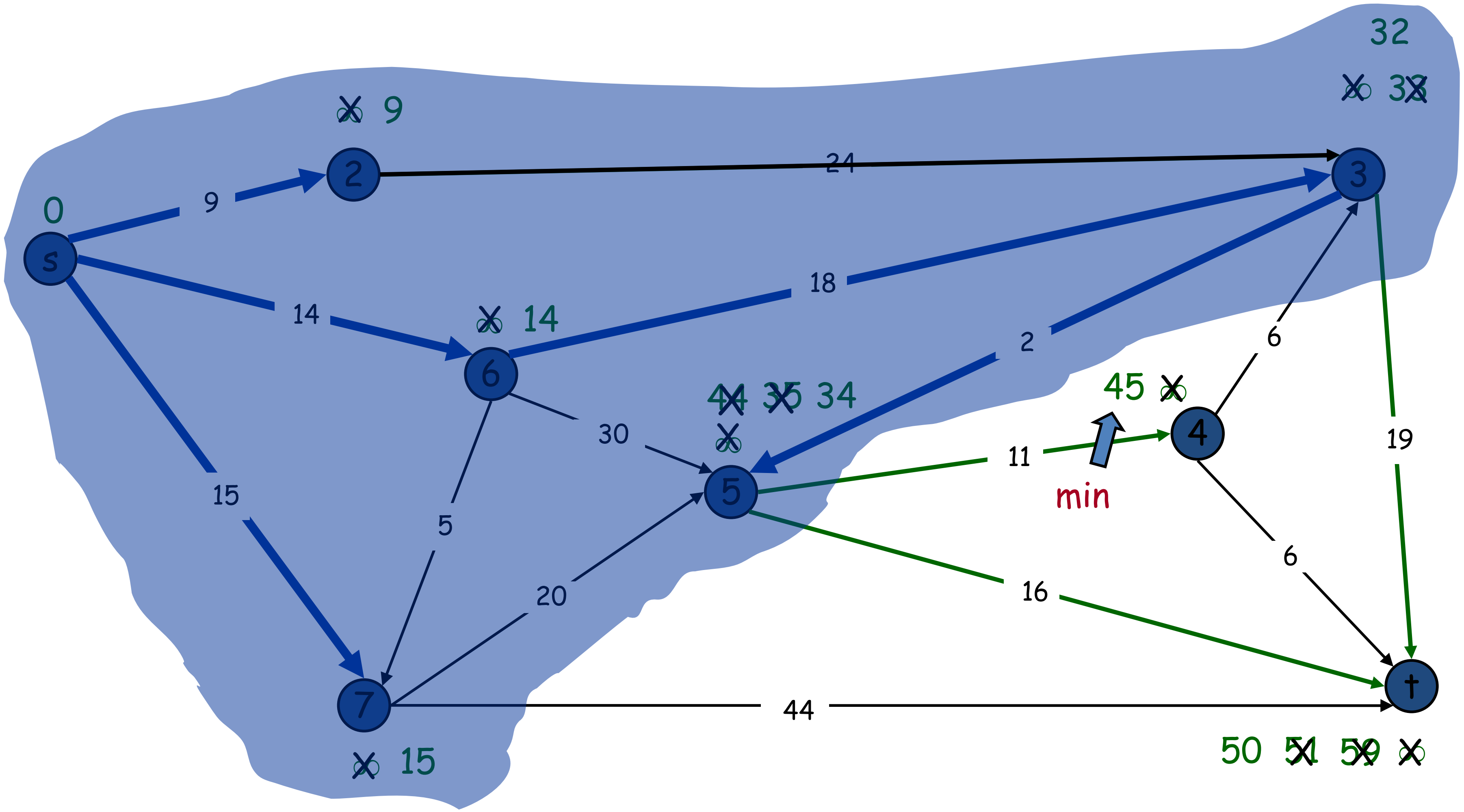
0
9
14
15
32
33
44 35 34
∞
51 59 ∞
s
2
3
4
5
6
7
t
24
18
2
6
30
11
19
5
20
16
6
44
min

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



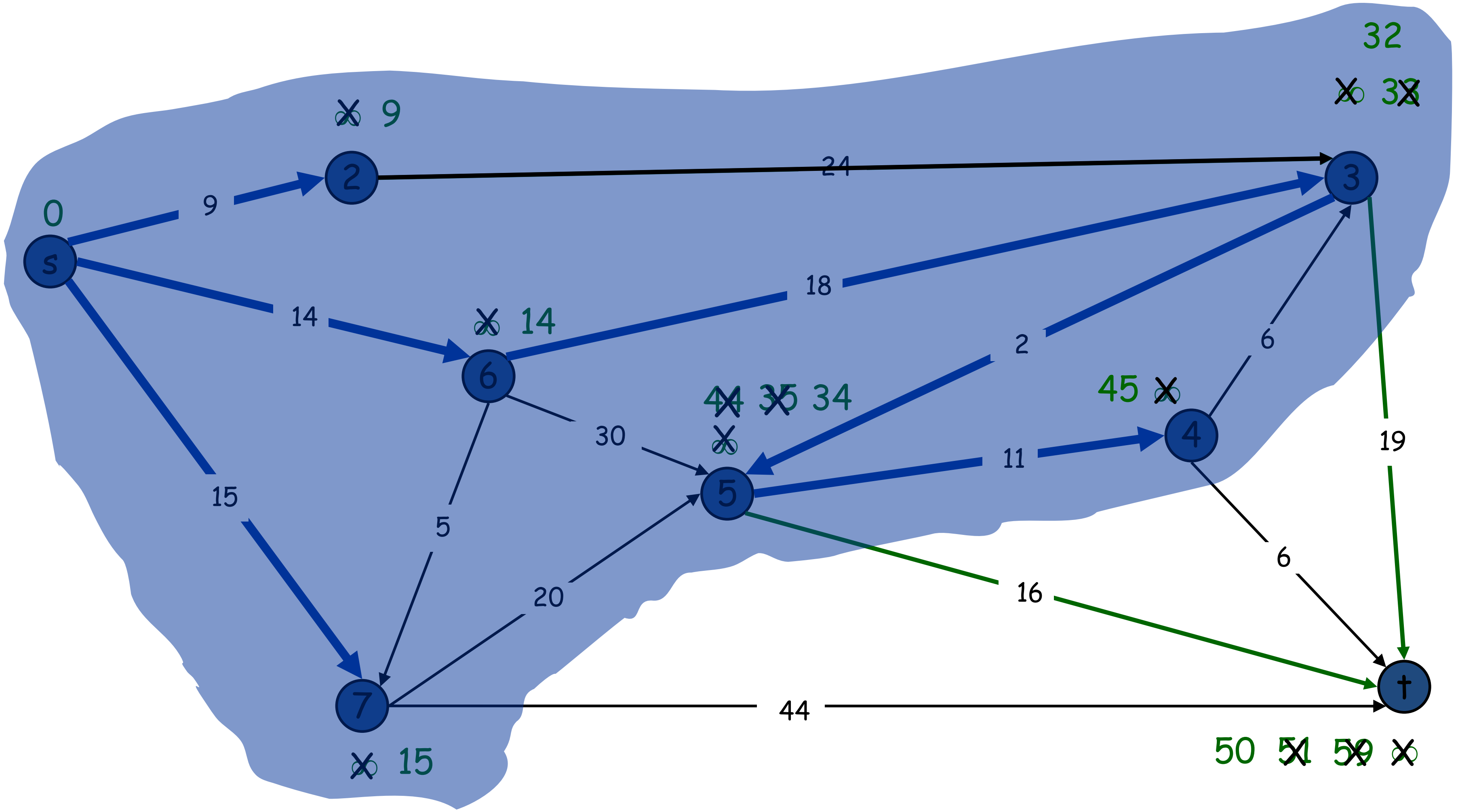
s
2
3
4
5
6
7
t
0
9
14
15
32
33
34
35
44
45
50
51
59
24
18
2
30
11
16
19
6
6
5
20
44

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

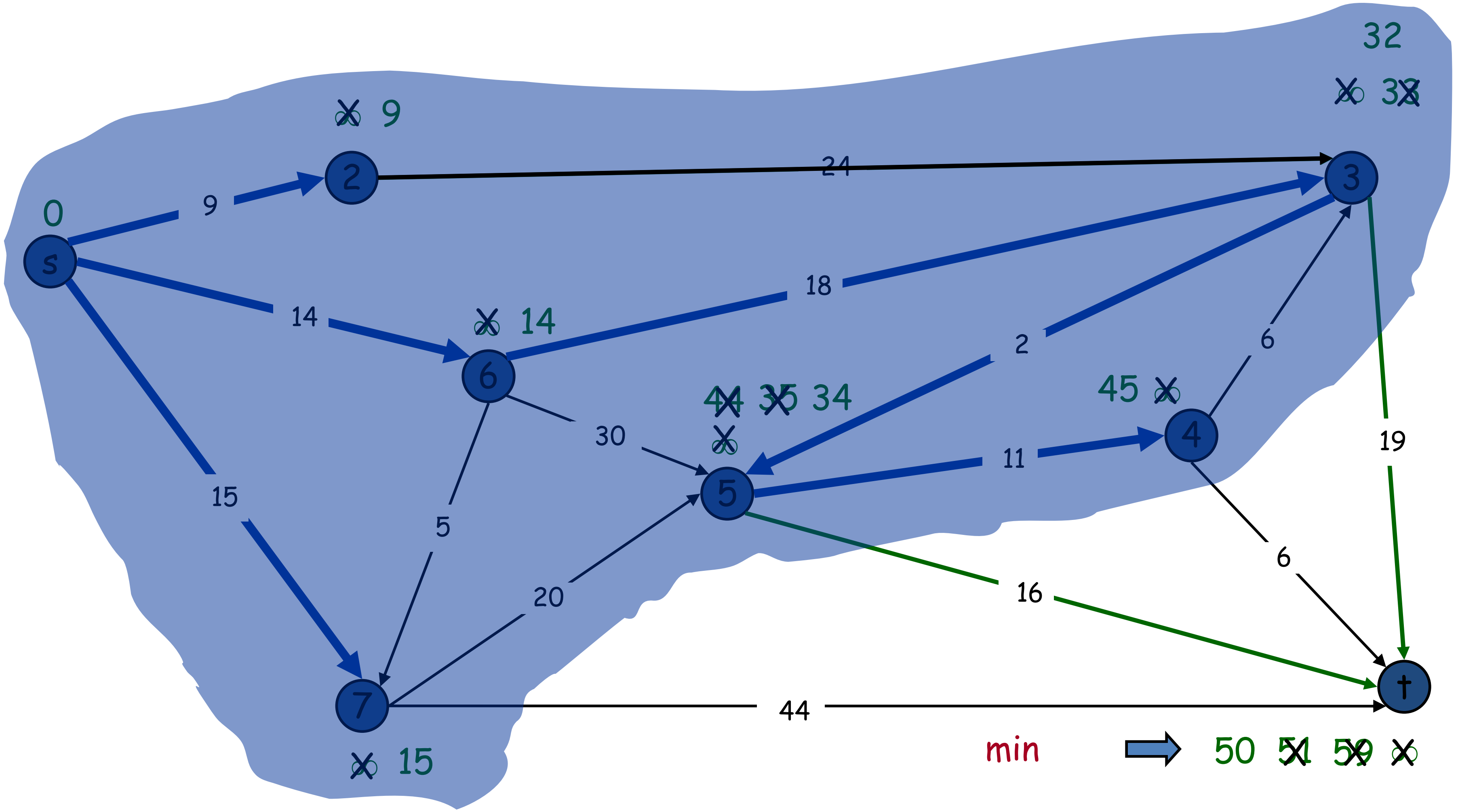


s
2
3
4
5
6
7
t
0
9
14
15
32
33
44 35 34
45
50 51 59
min
9
14
15
24
18
2
30
5
20
44
11
16
6
6
19

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

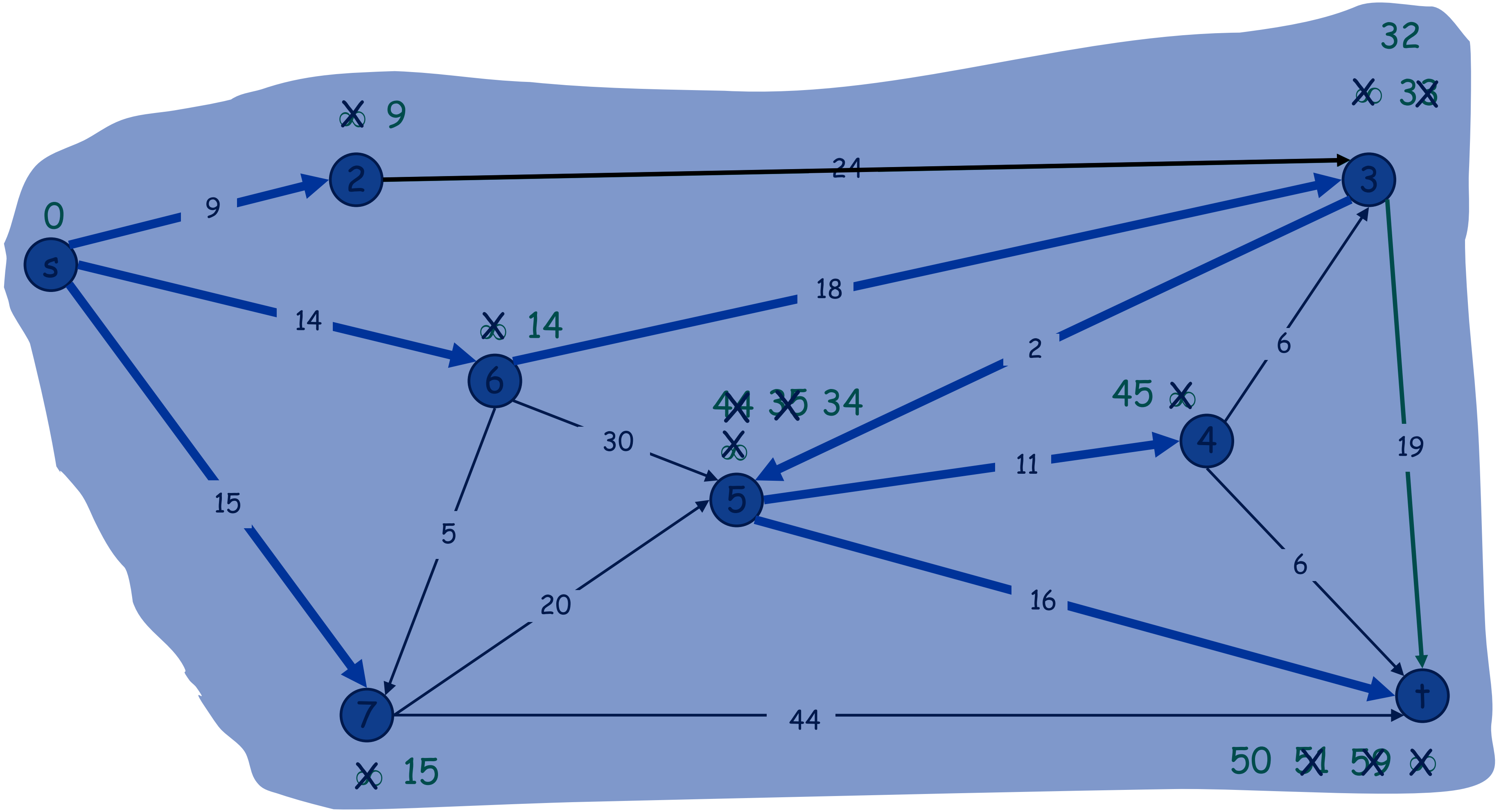


S = { s, 2, 3, 4, 5, 6, 7 }
Q = { t }



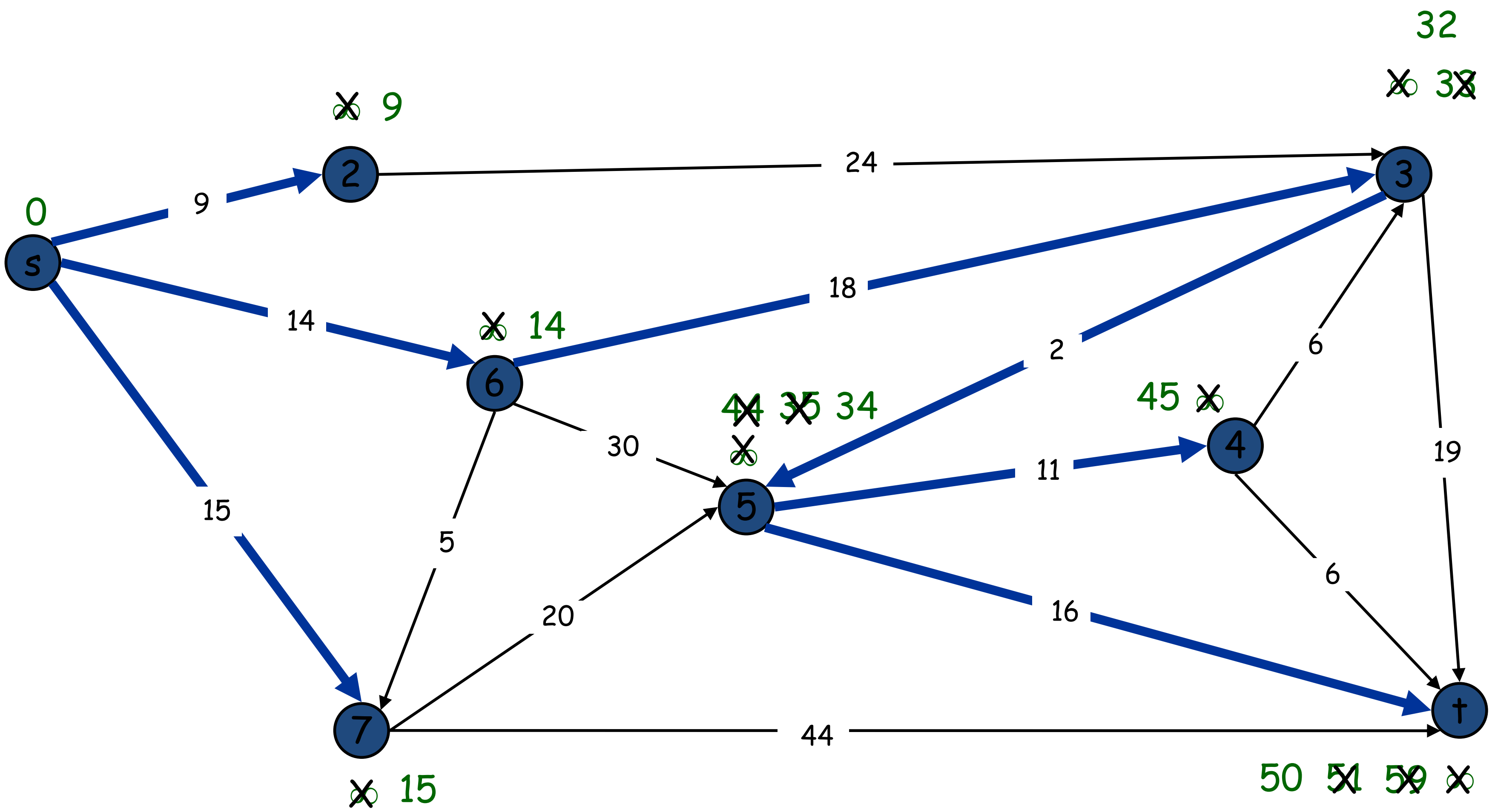
s
2
3
4
5
6
7
t
0
9
14
15
32
34
45
9
24
18
2
6
14
30
11
15
5
20
16
6
19
44
min
50

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



32
∞ 33
∞ 9
0
s
2
3
9
24
18
2
6
14
∞ 14
6
45 ∞
4
44 35 34
∞
5
30
11
19
15
5
20
16
6
7
44
t
∞ 15
50 51 59 ∞

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


s
2
3
4
5
6
7
t
0
9
14
15
32
33
34
35
44
45
50
51
59
24
18
2
30
11
6
19
16
5
20
44

Implementation notes

- There are many ways to speed this up in practice
- Graph representations
- Naïve Dijkstra with n nodes and m edges is $O(mn)$
- We need to remove from Q the node v with smallest $\pi(v)$
  - Priority queue implements remove-min in $O(\log n)$
  - This makes Dijkstra run in $O(m \log n)$ time

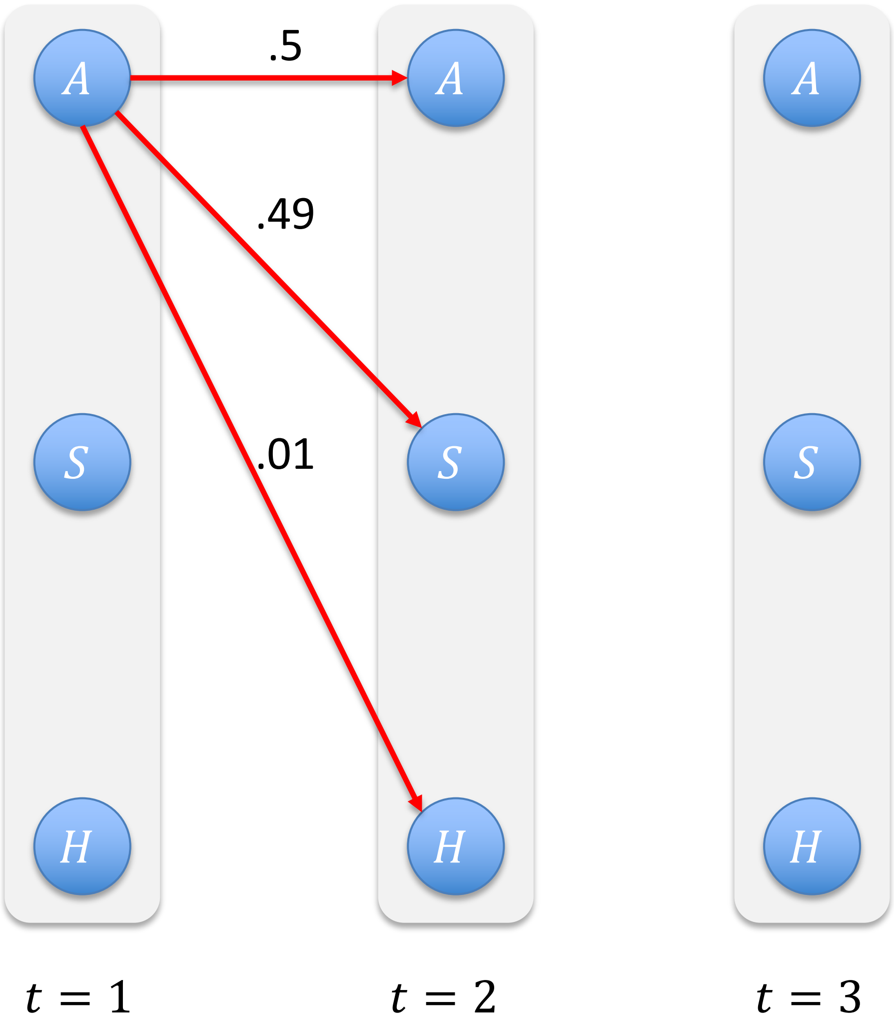
Another class of examples

- Let's model student behavior over time (hourly basis)
- Students have 3 possible states:
  - Awake (A)
  - Sleeping (S)
  - Doing CS5112 homework (H)
- If you know their state at time t you know the probability of their other states at time $t + 1$
  - Example: A goes to A (.5), S (.49), H(.01)

Trellis graph

- We want to find the most likely 12 hour day for a student
- At every time t there are 3 nodes, for A/S/H
- There are edges with transition probabilities
  - – Just like pirate grammar!
- So a day is a 12-node path through the graph
- This is closely related to a "Hidden Markov Model"
  - – Widely used! Famous examples include speech, handwriting, computer vision, bioinformatics, etc.

Example



- Important note: with S states and time T there are $O(ST)$ nodes in the graph and $O(S^2\, T)$ edges
- So running time of naïve Dijkstra is $O(S^3 T^2)$
- Can reduce this to $O(S^2\, T)$ with dynamic programming (Viterbi)