BLG 336E - Analysis of Algorithms II

Recitation 4

March 22th, 2022

Greedy Algorithms

Muhammed Raşit Erol

OUTLINE

1. Dijkstra's Algorithm
2. Interval Scheduling
3. Questions

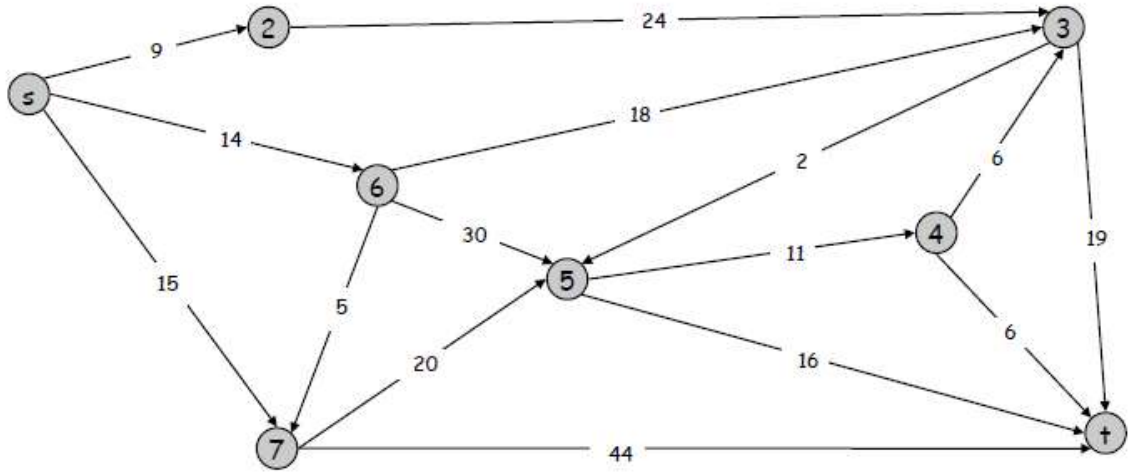
Dijkstra's algorithm.

- 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)

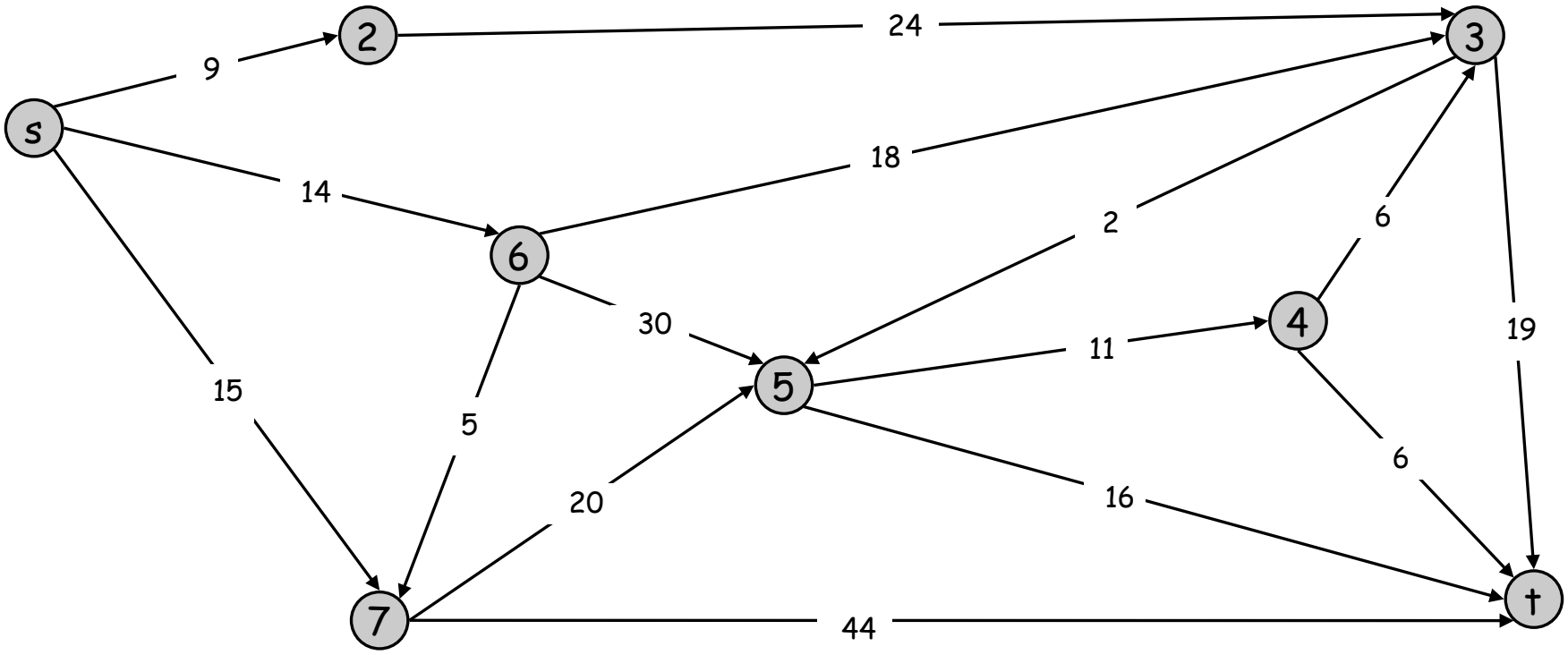
add v to S, and set d(v) = π(v).



Dijkstra’s Algorithm Demo

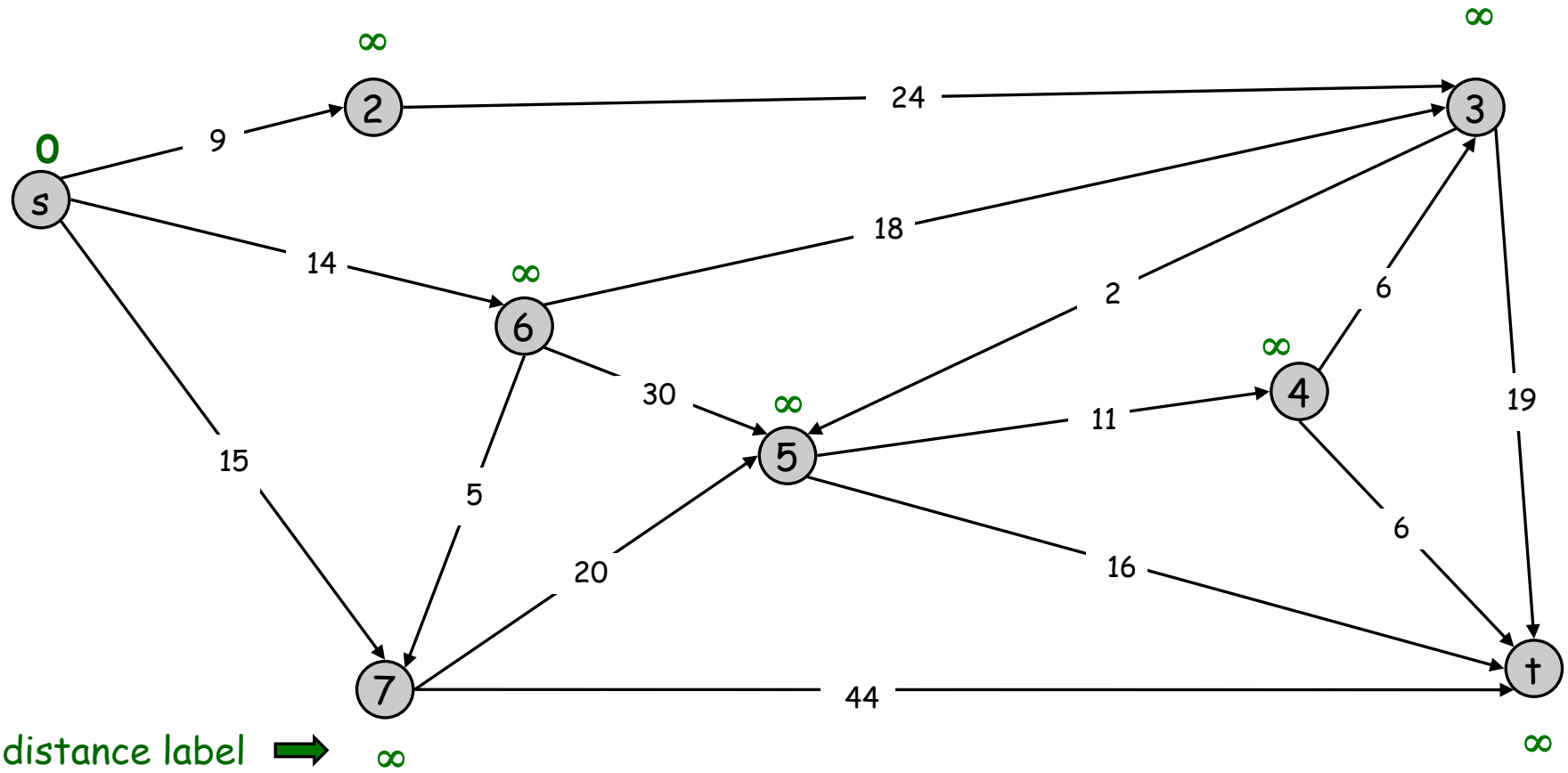
Dijkstra's Shortest Path Algorithm

Find shortest path from s to t.



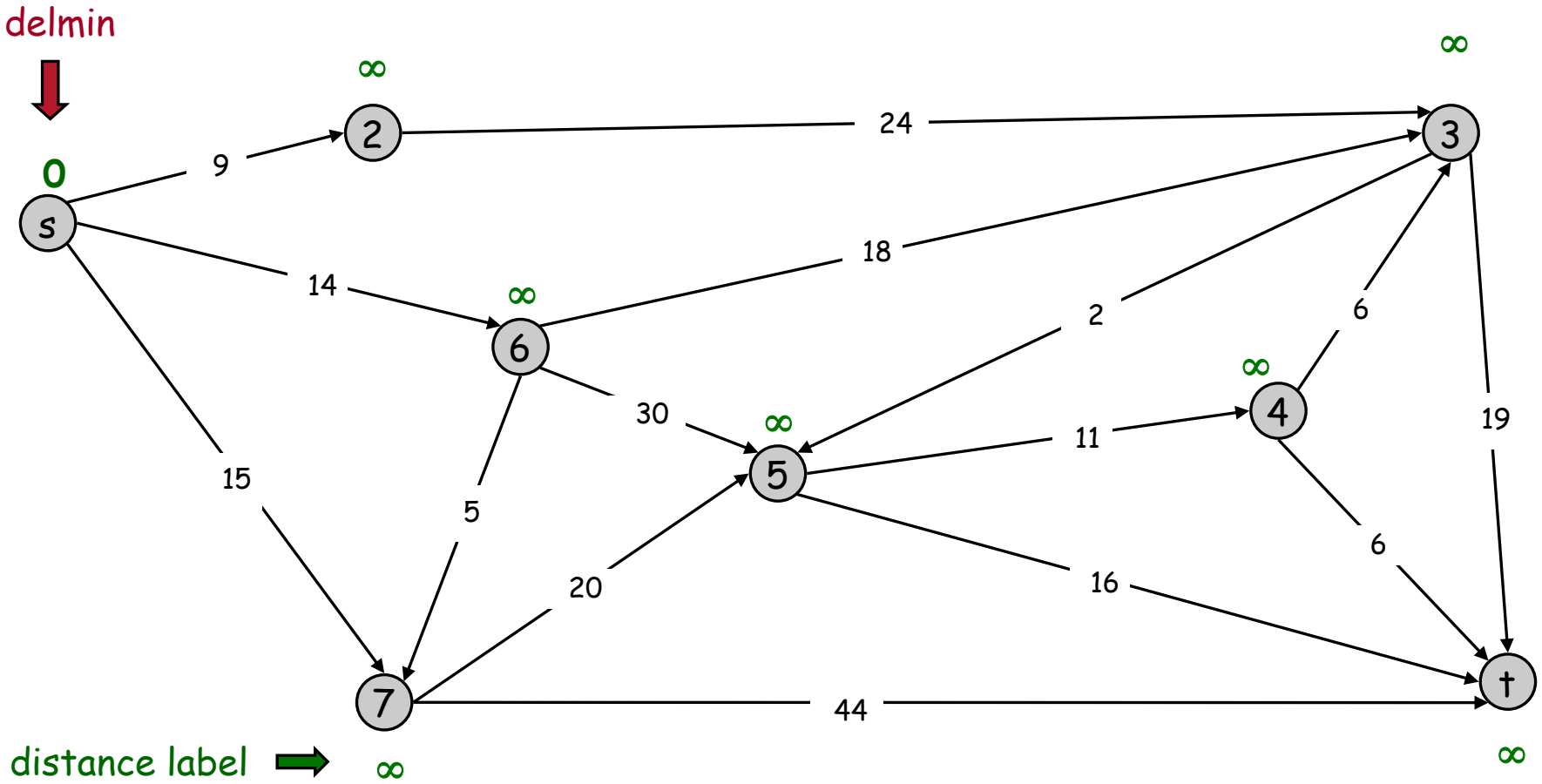
Dijkstra's Shortest Path Algorithm

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



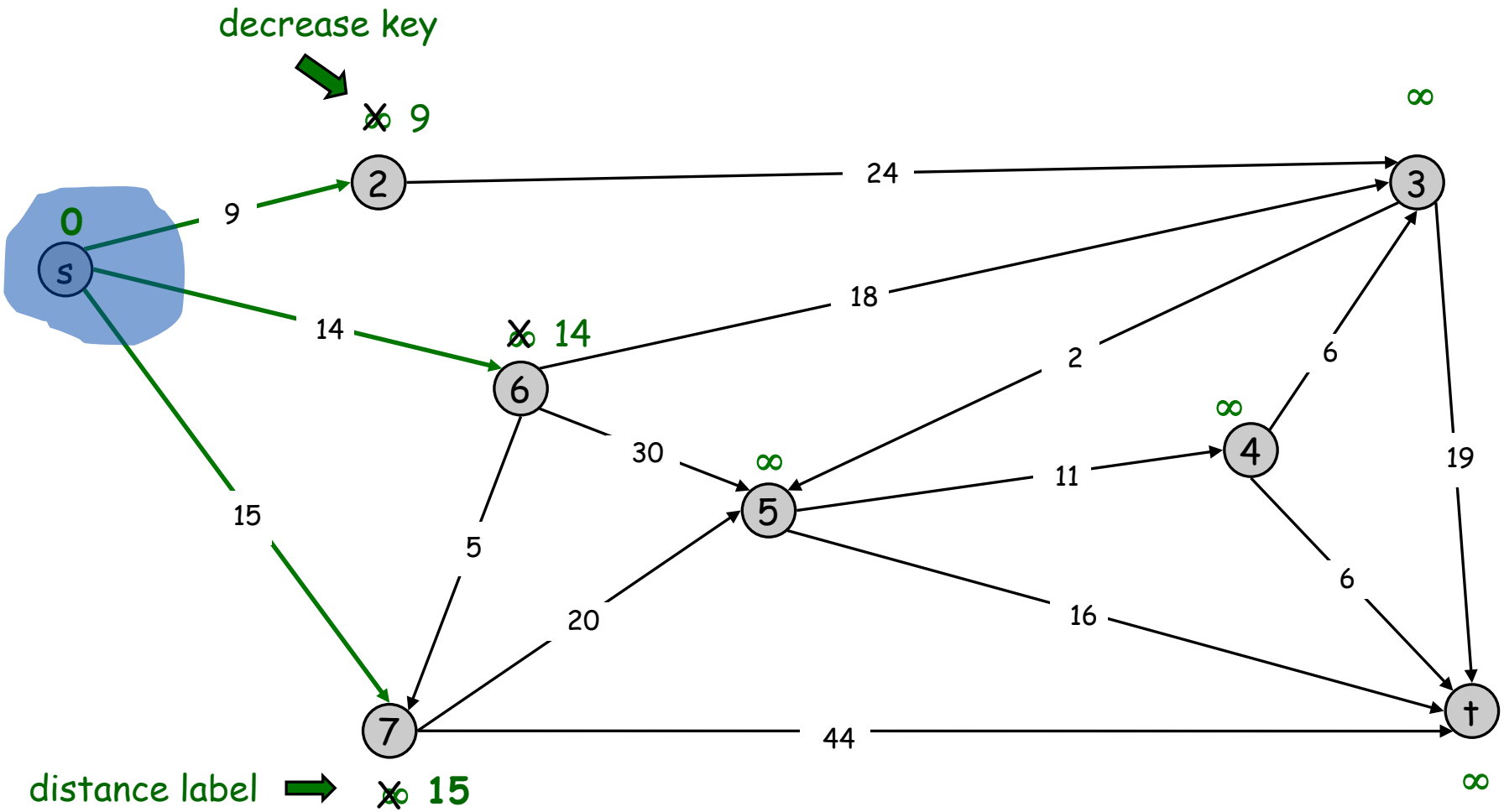
Dijkstra's Shortest Path Algorithm

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



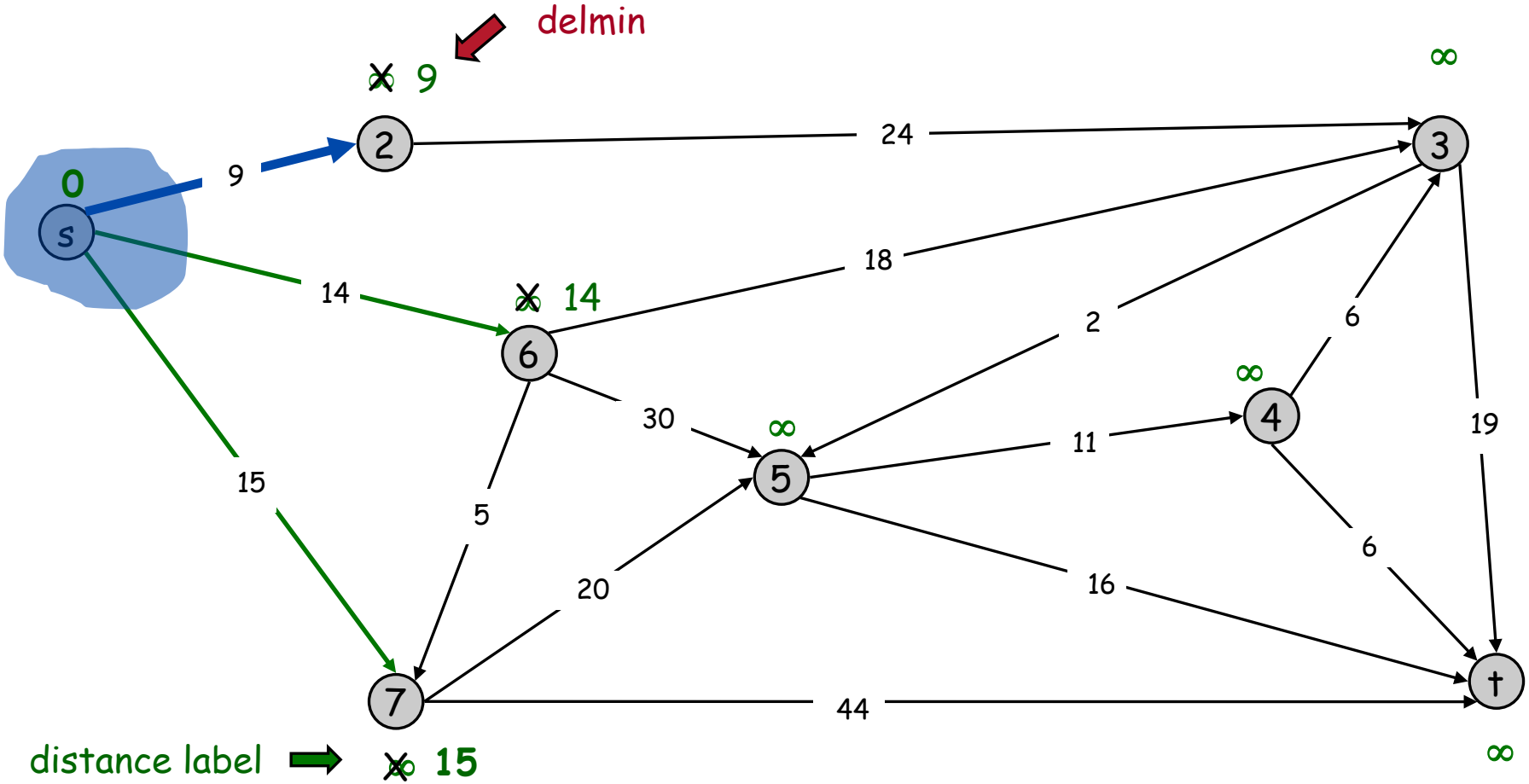
Dijkstra's Shortest Path Algorithm

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



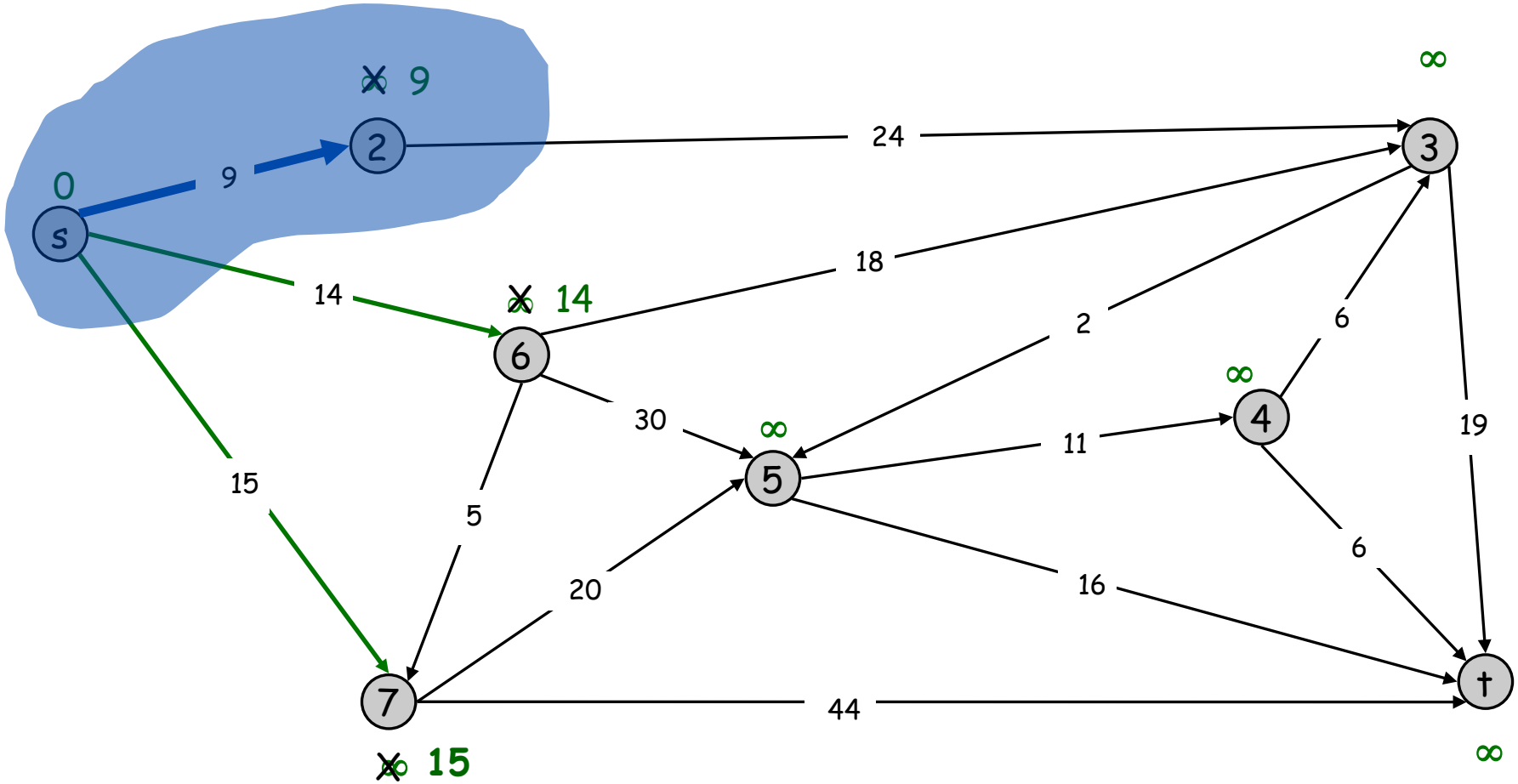
Dijkstra's Shortest Path Algorithm

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



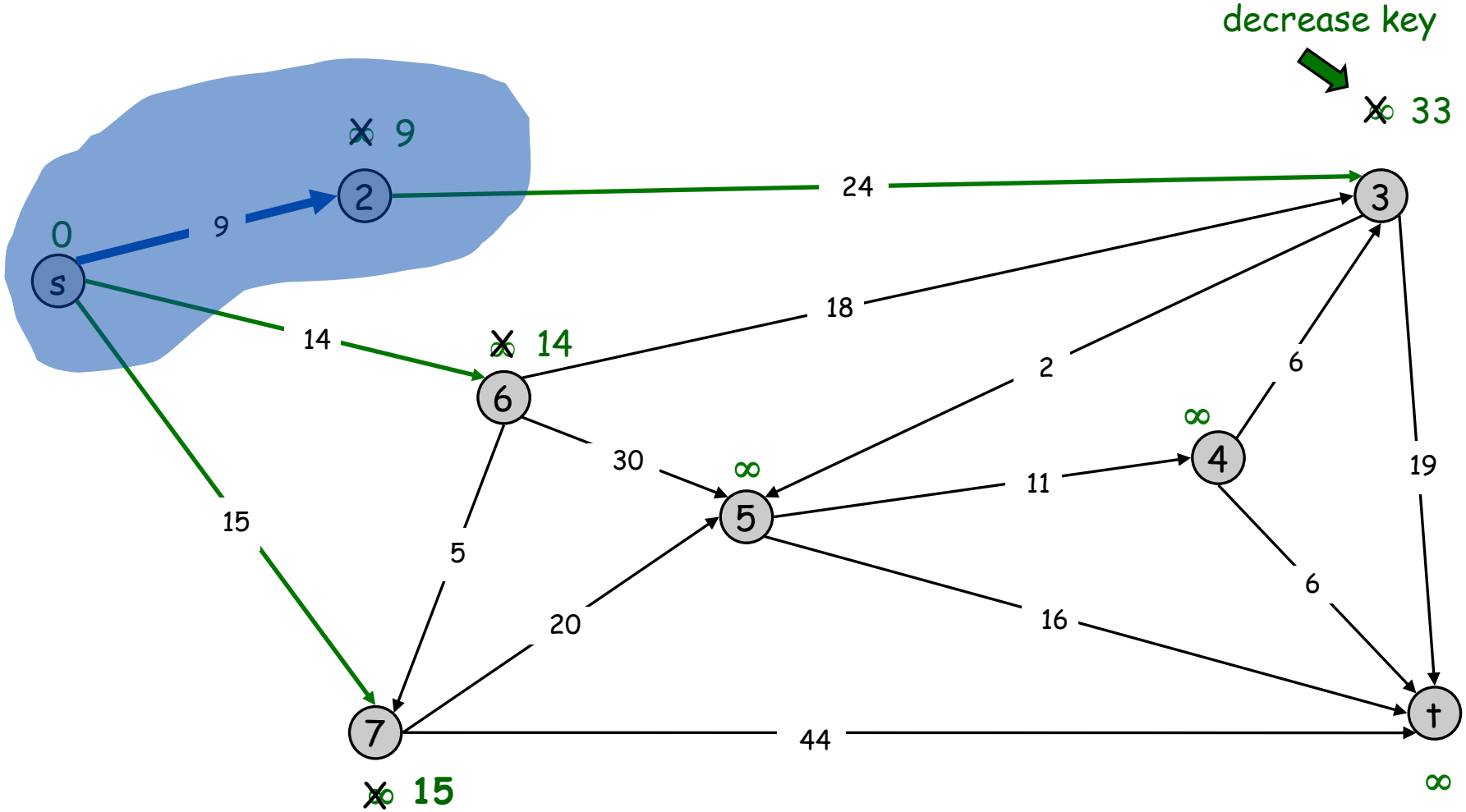
Dijkstra's Shortest Path Algorithm

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



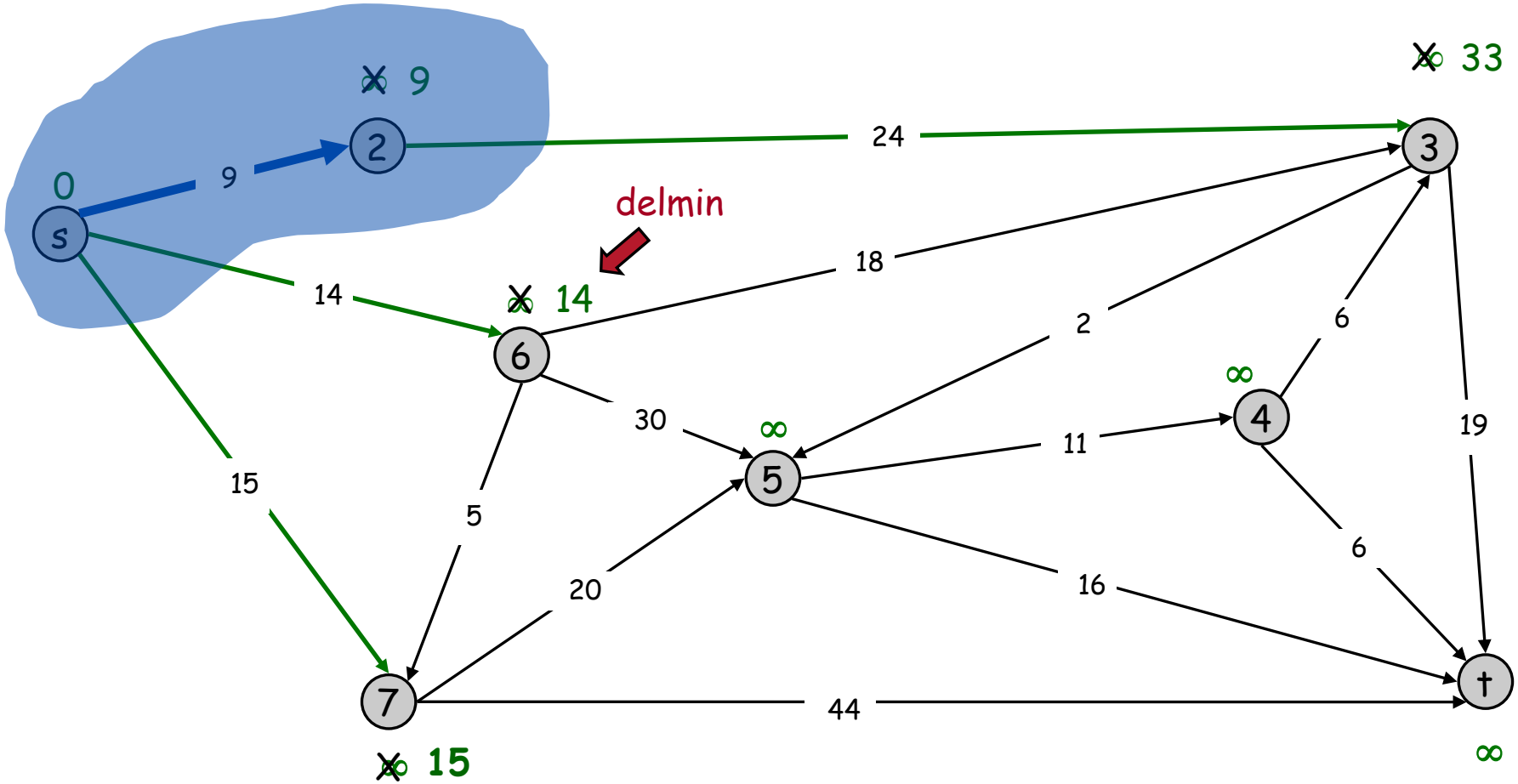
Dijkstra's Shortest Path Algorithm

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



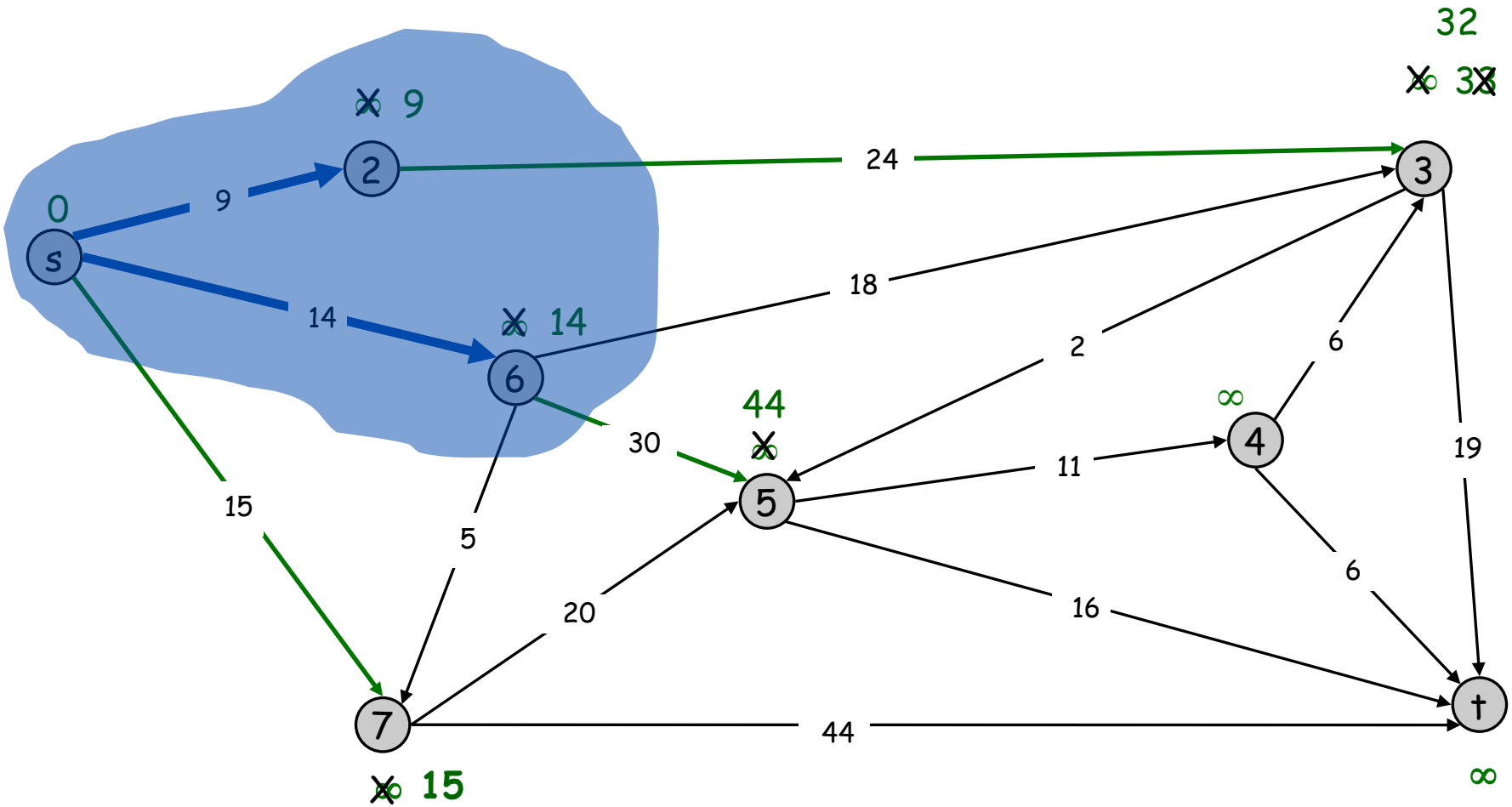
Dijkstra's Shortest Path Algorithm

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



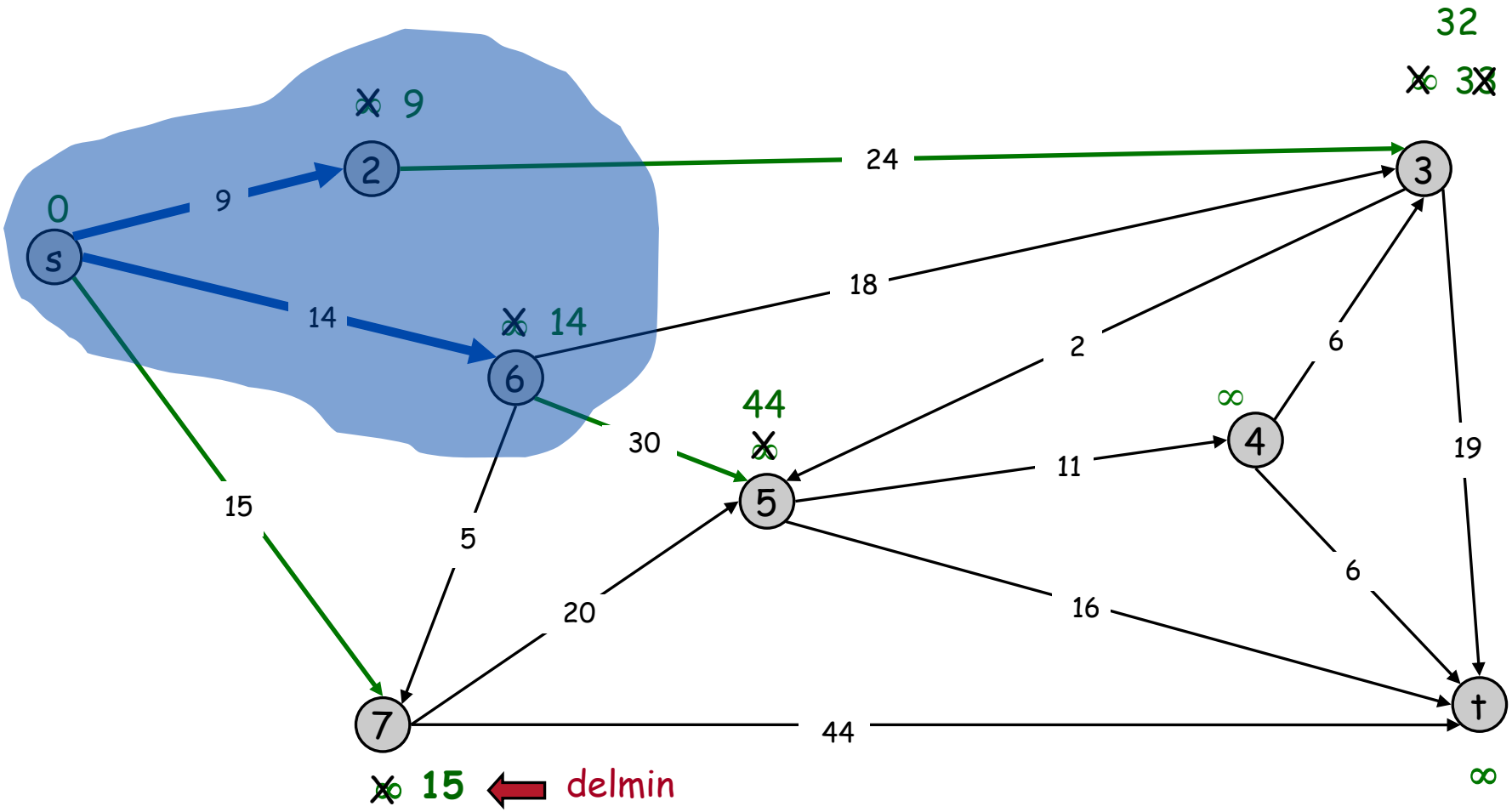
Dijkstra's Shortest Path Algorithm

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



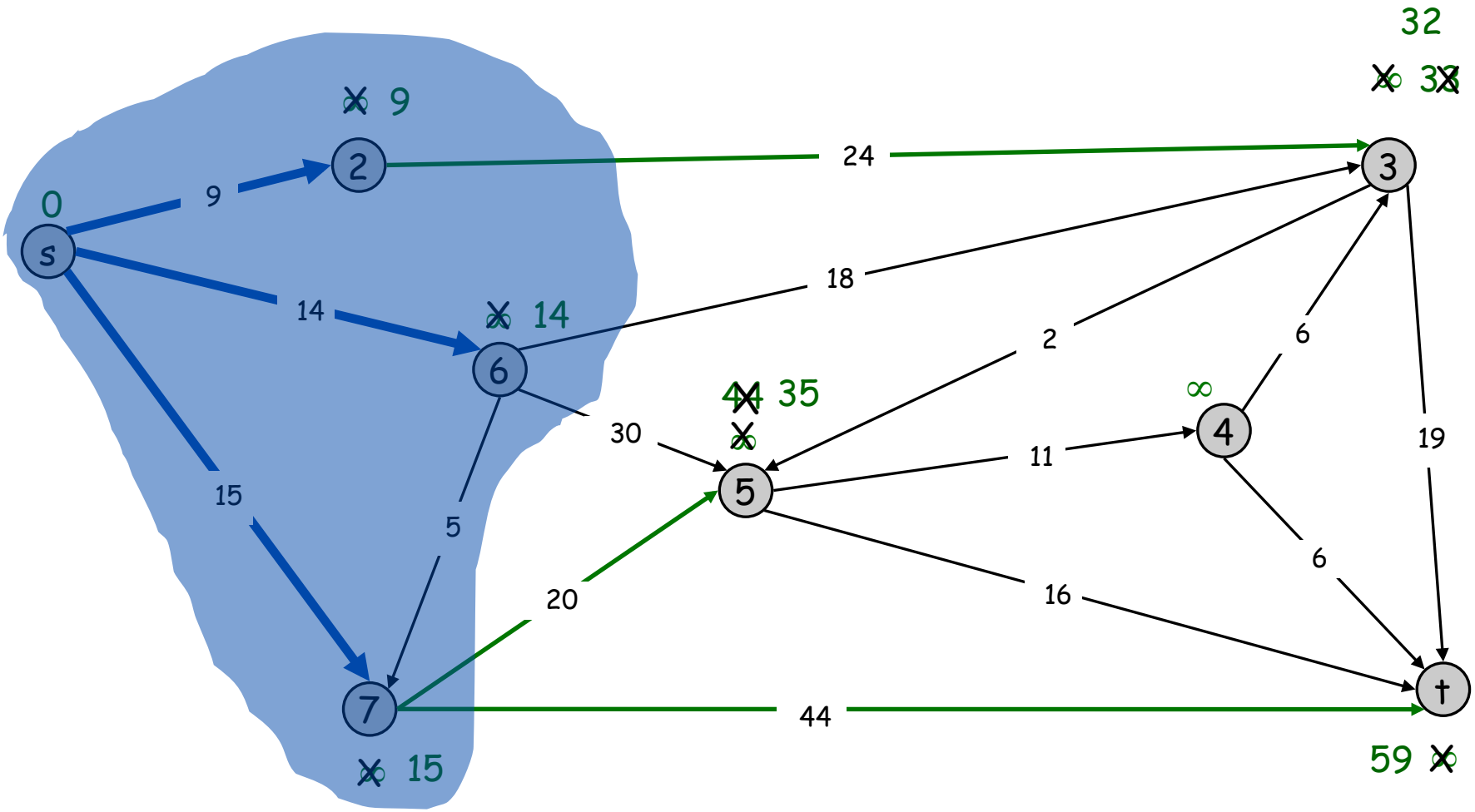
Dijkstra's Shortest Path Algorithm

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



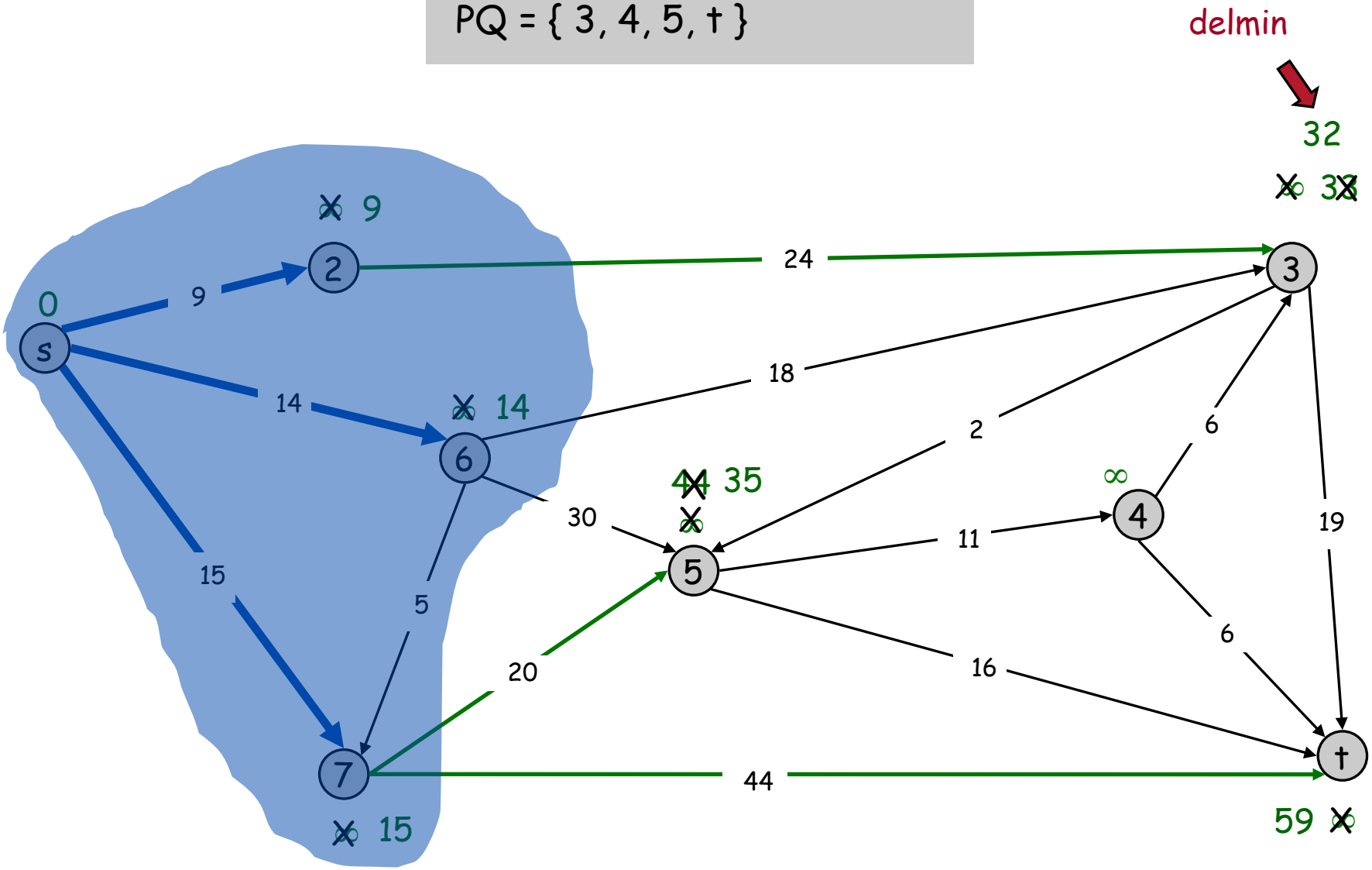
Dijkstra's Shortest Path Algorithm

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



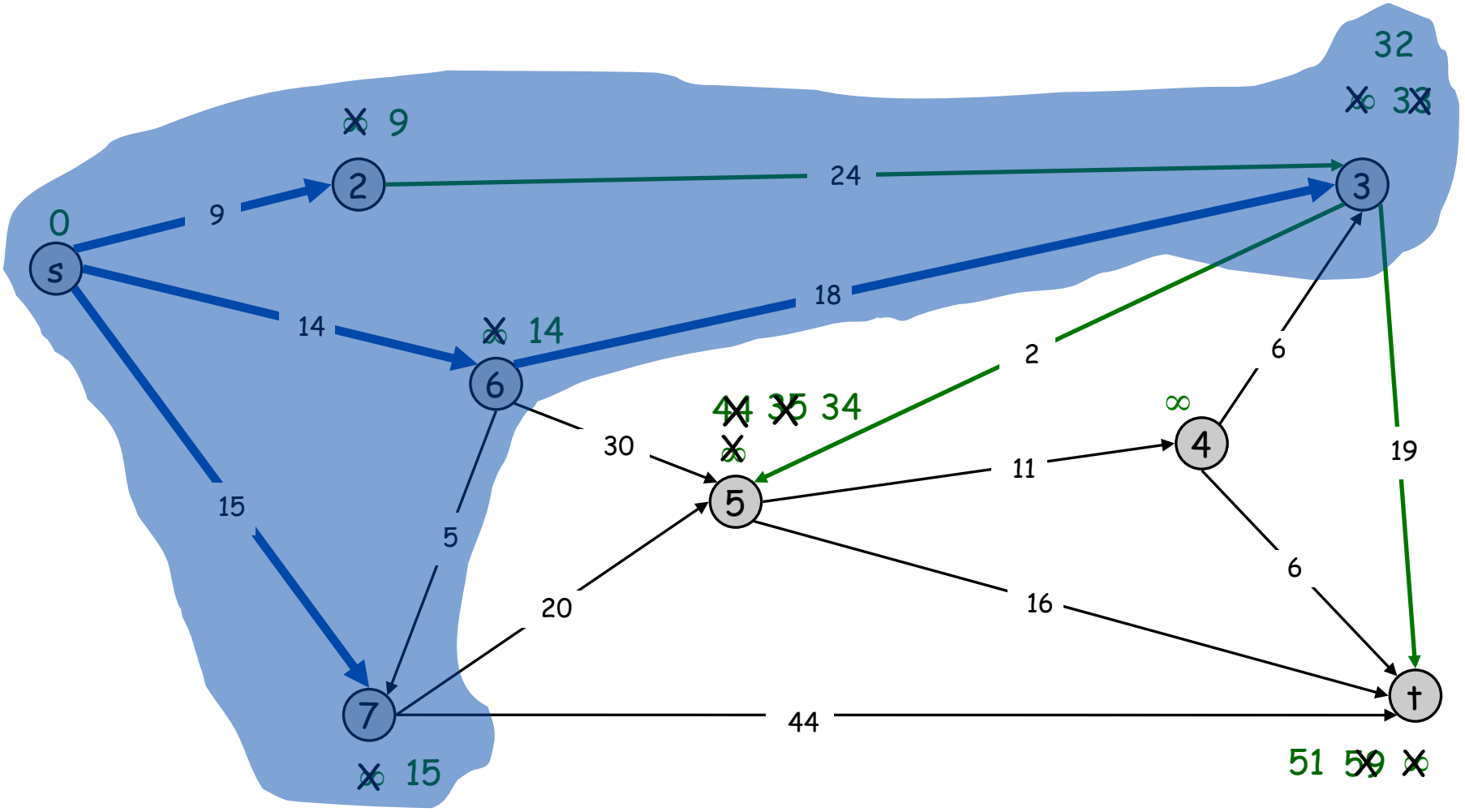
Dijkstra's Shortest Path Algorithm

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



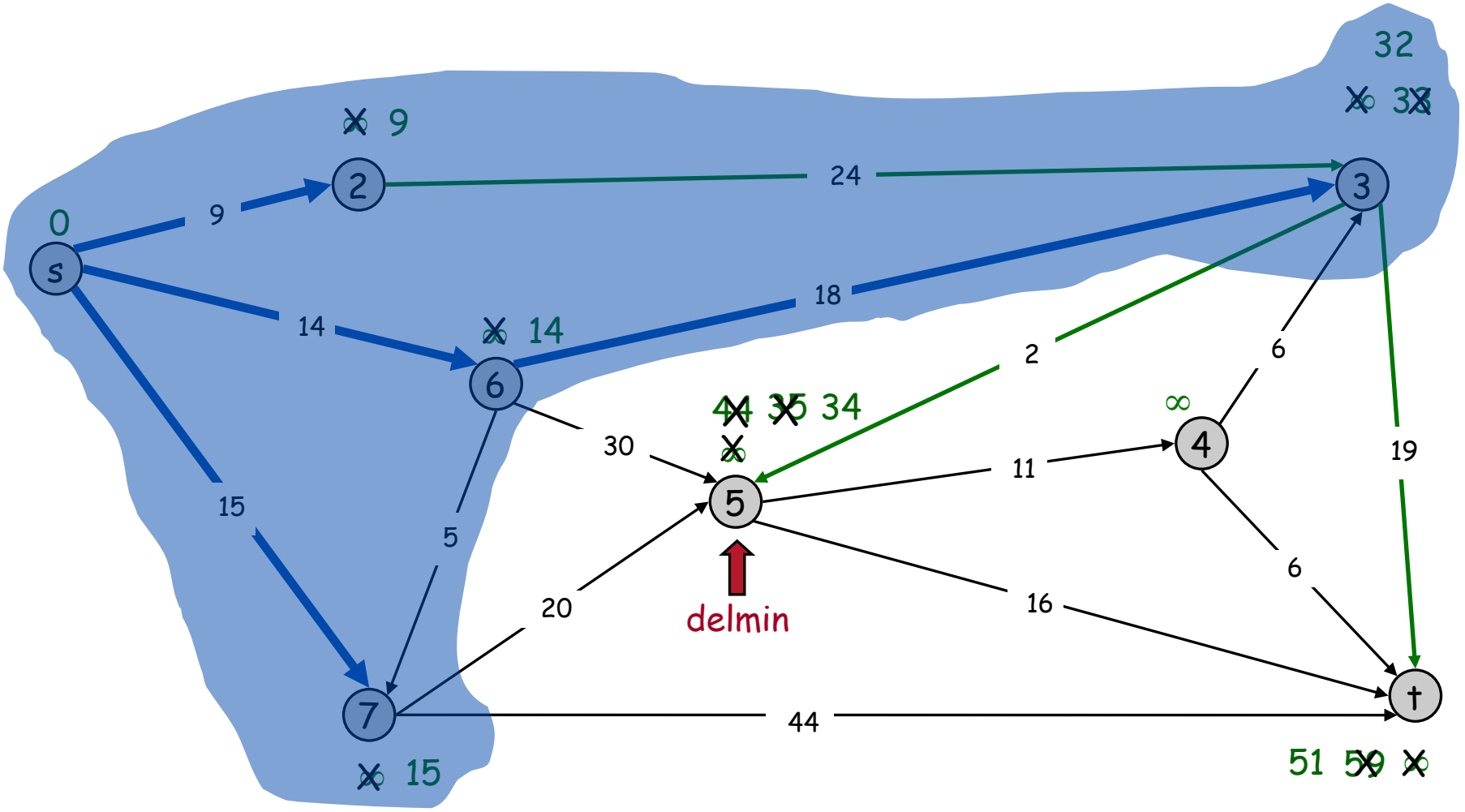
Dijkstra's Shortest Path Algorithm

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



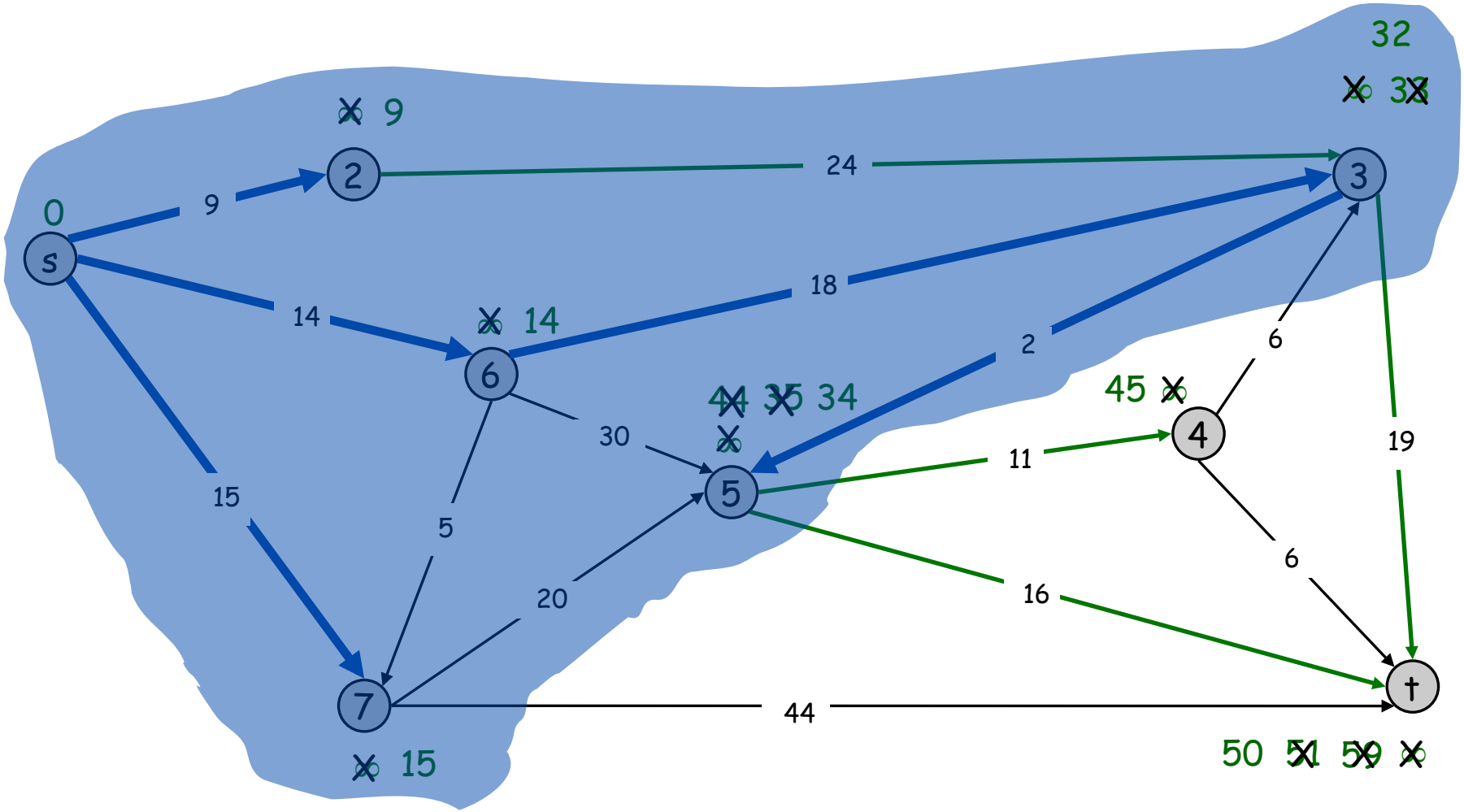
Dijkstra's Shortest Path Algorithm

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



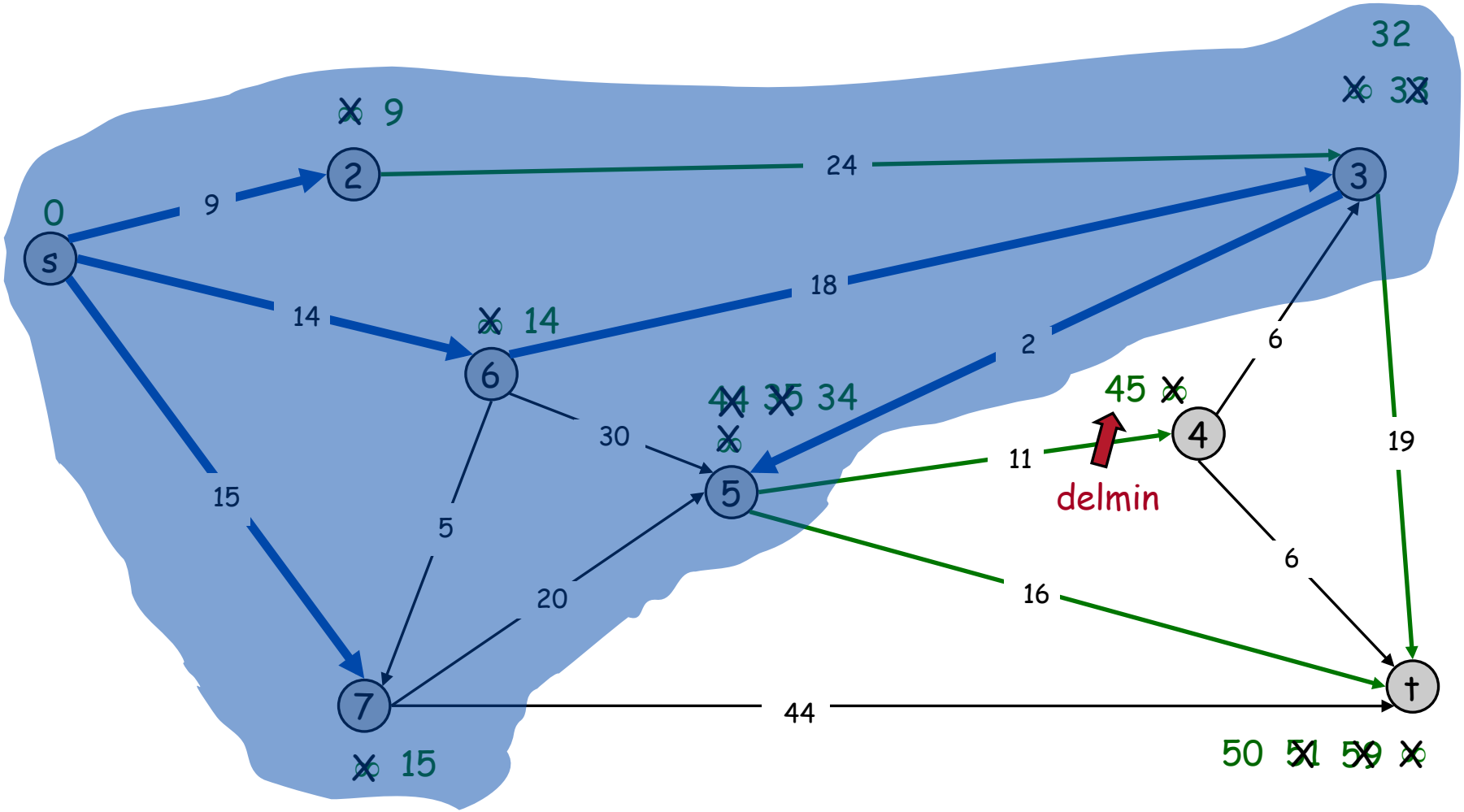
Dijkstra's Shortest Path Algorithm

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



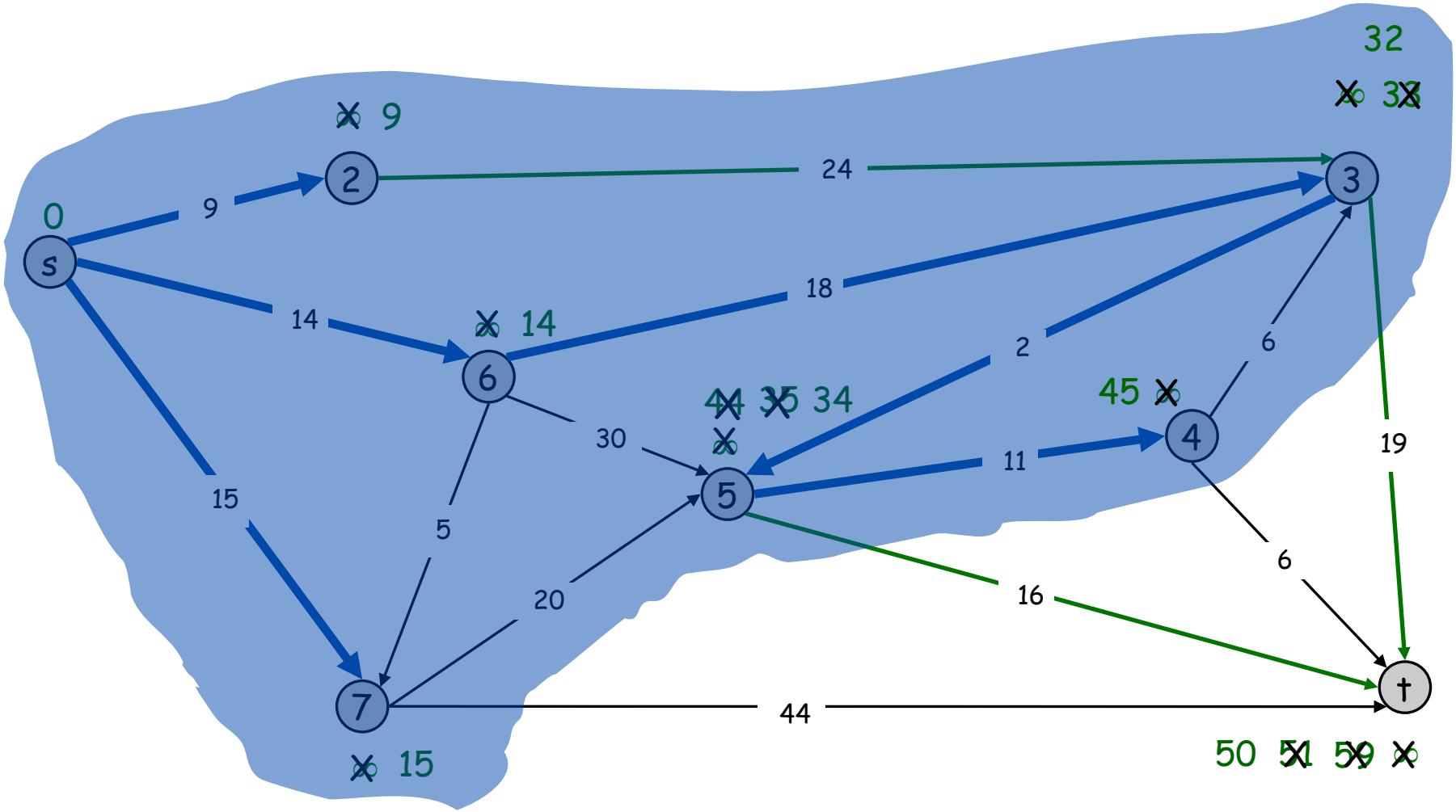
Dijkstra's Shortest Path Algorithm

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



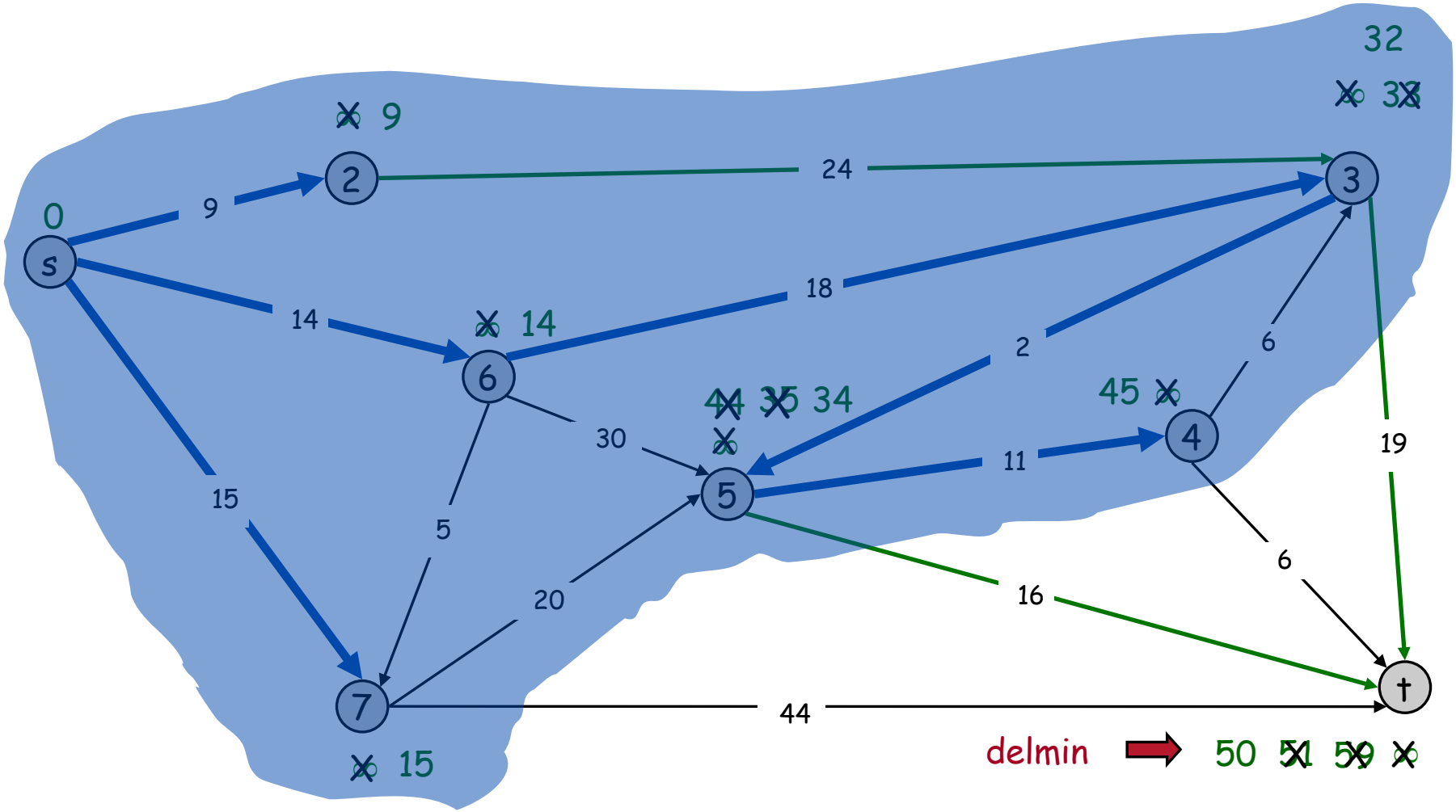
Dijkstra's Shortest Path Algorithm

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



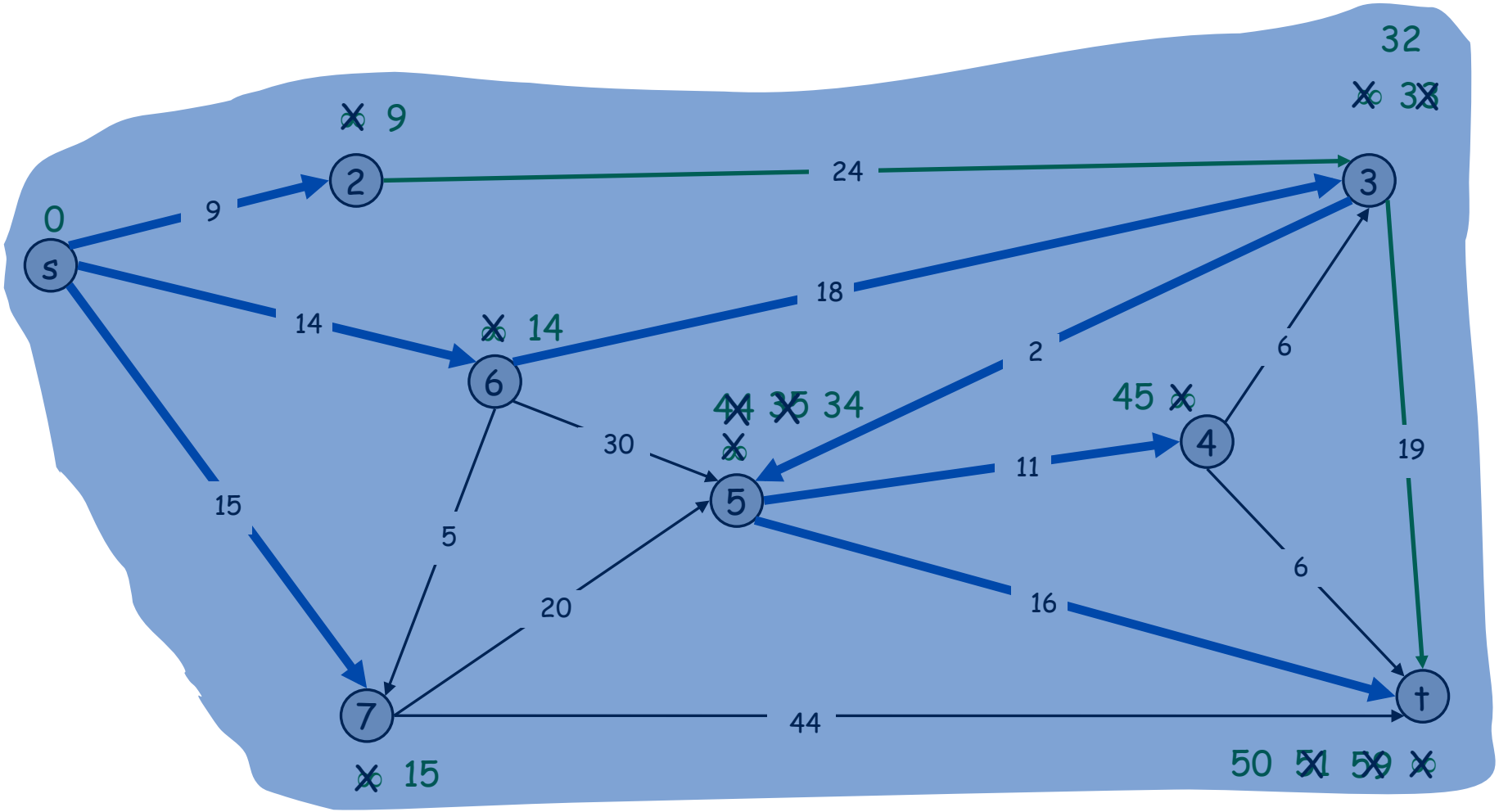
Dijkstra's Shortest Path Algorithm

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



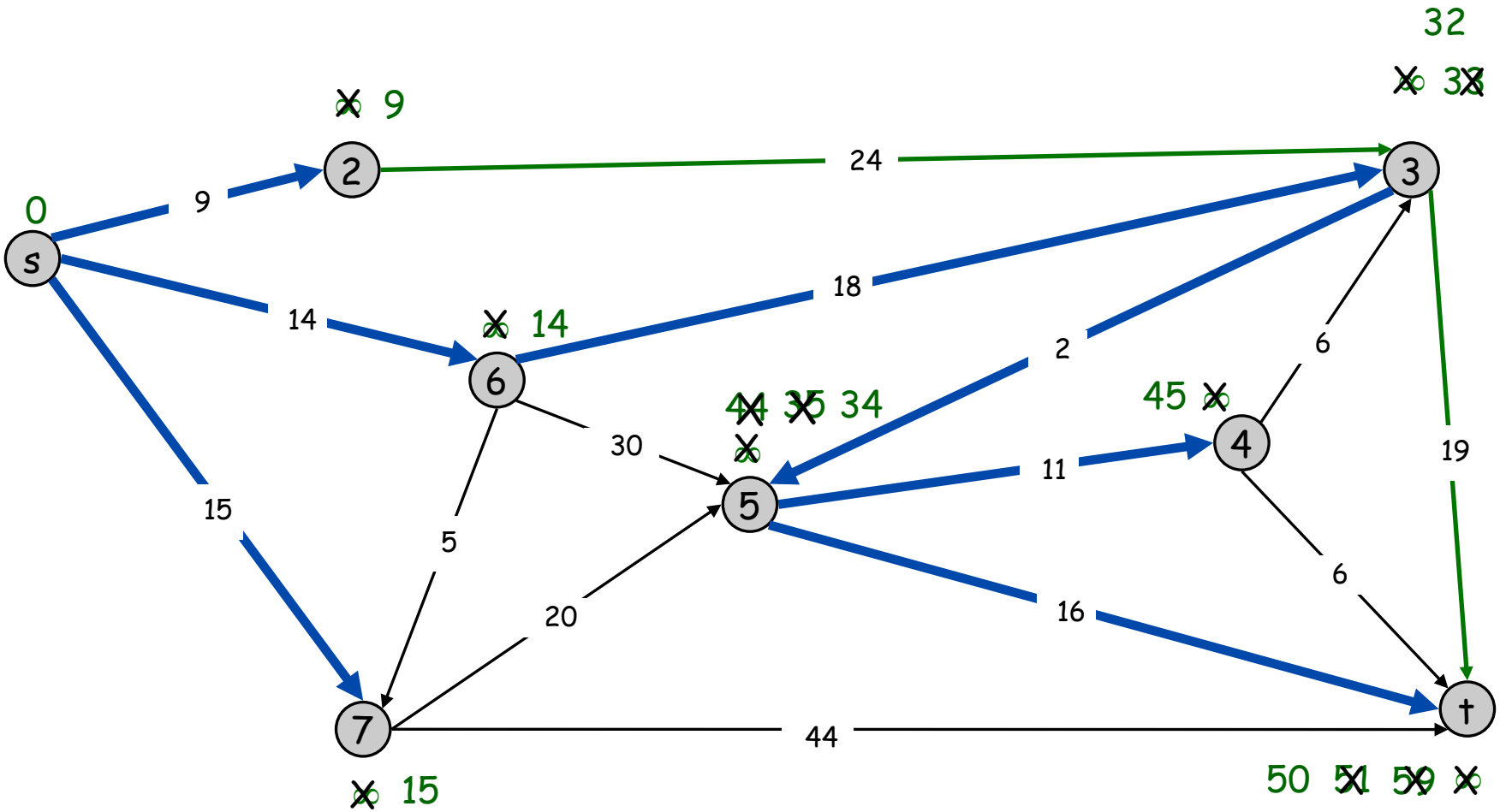
Dijkstra's Shortest Path Algorithm

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



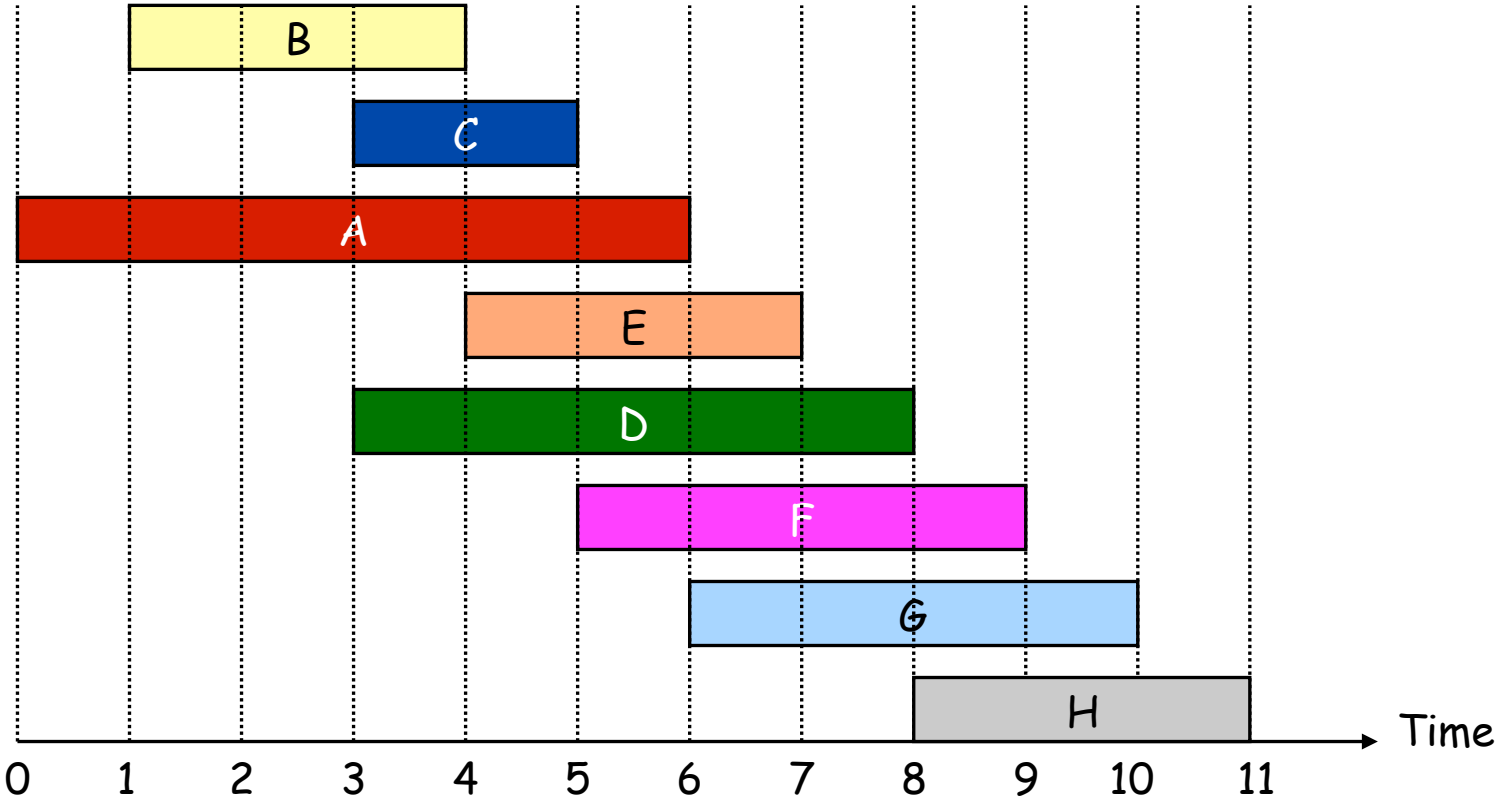
Dijkstra's Shortest Path Algorithm

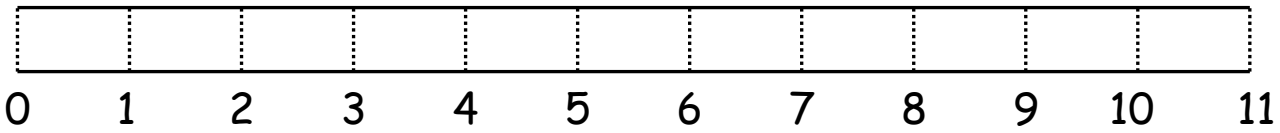
S = { s, 2, 3, 4, 5, 6, 7, t }
PQ = { }



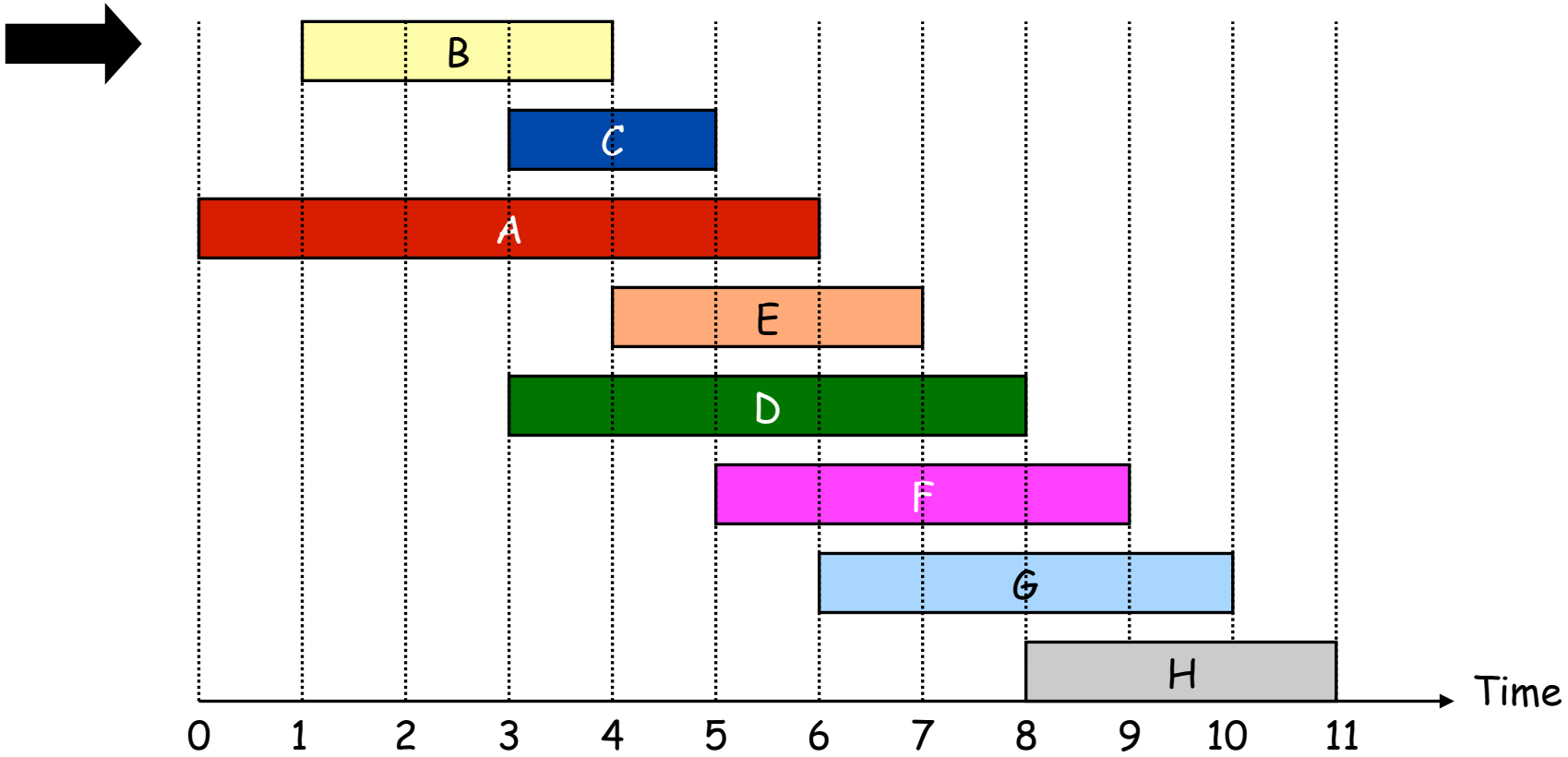
Interval Scheduling Demo

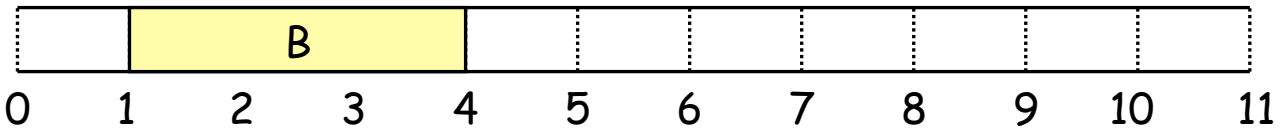
Interval Scheduling



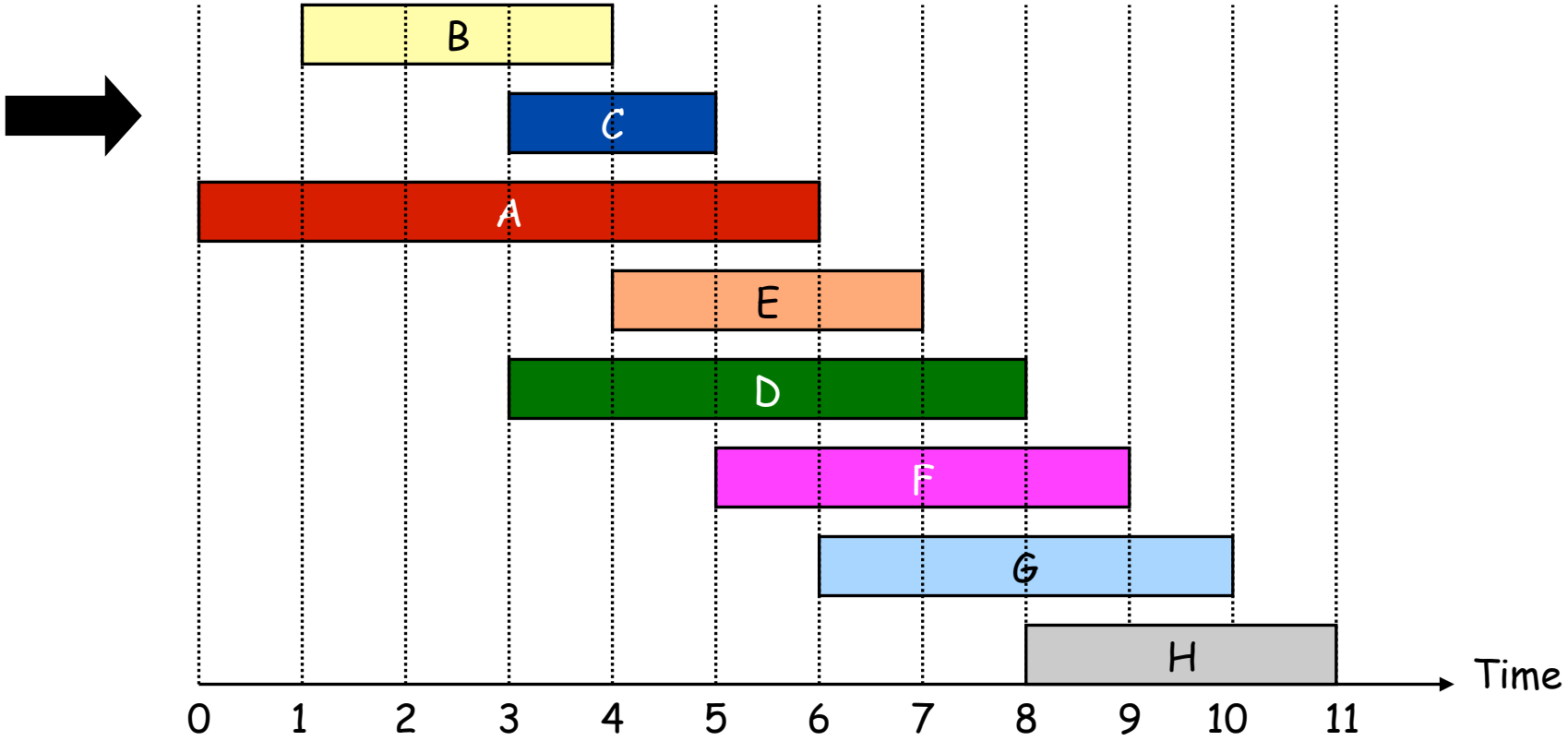


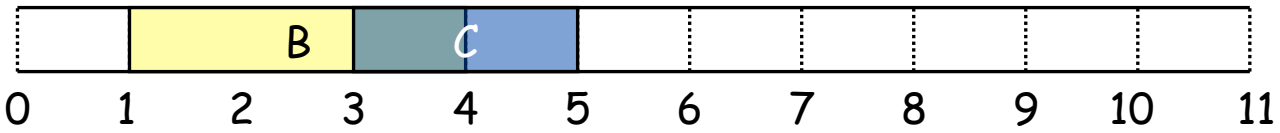
Interval Scheduling



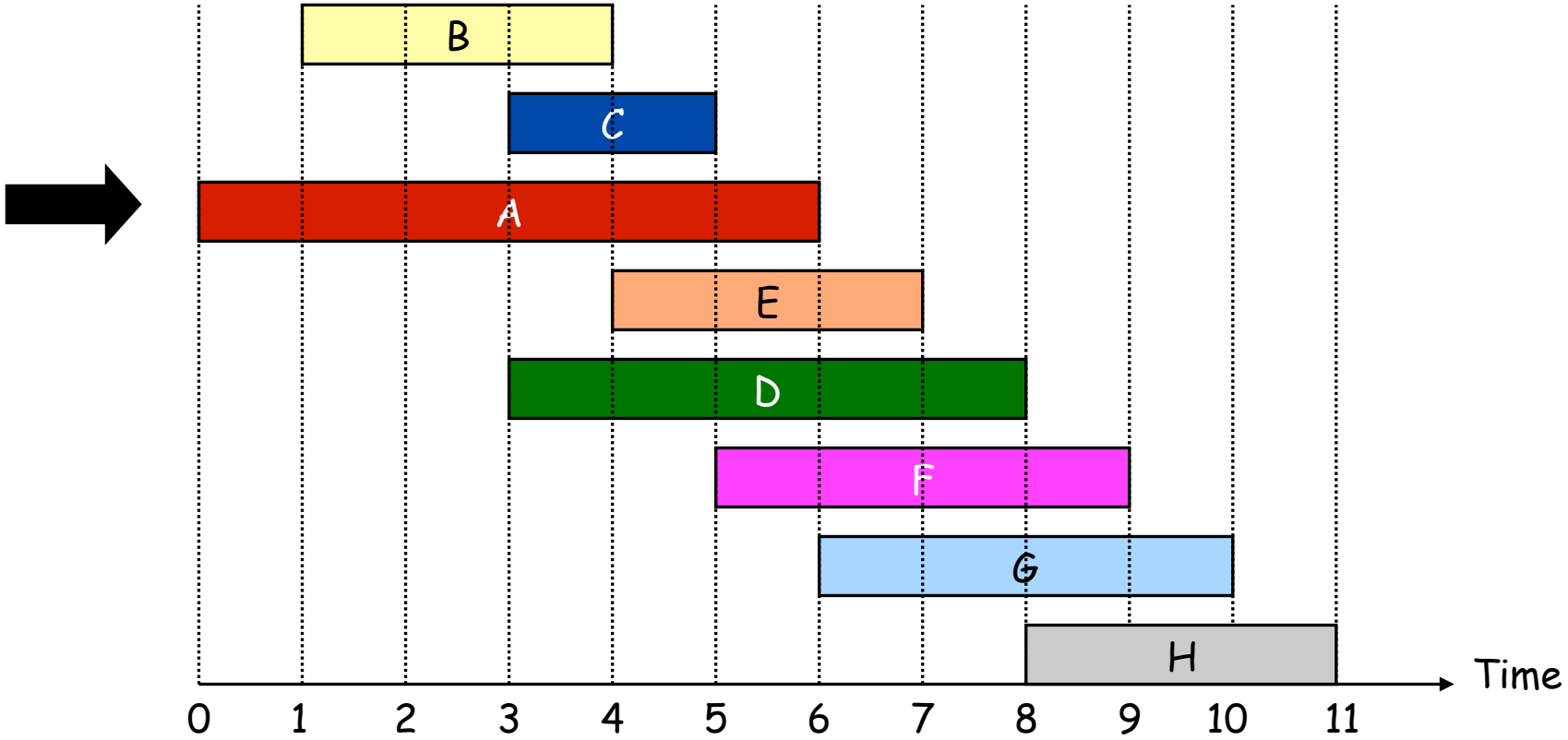


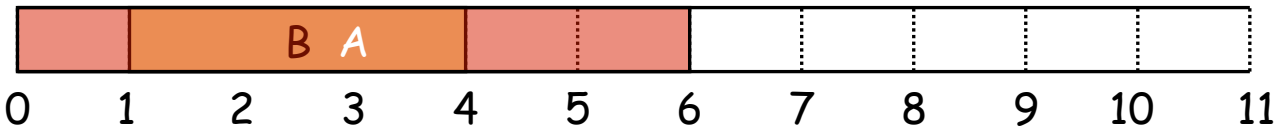
Interval Scheduling



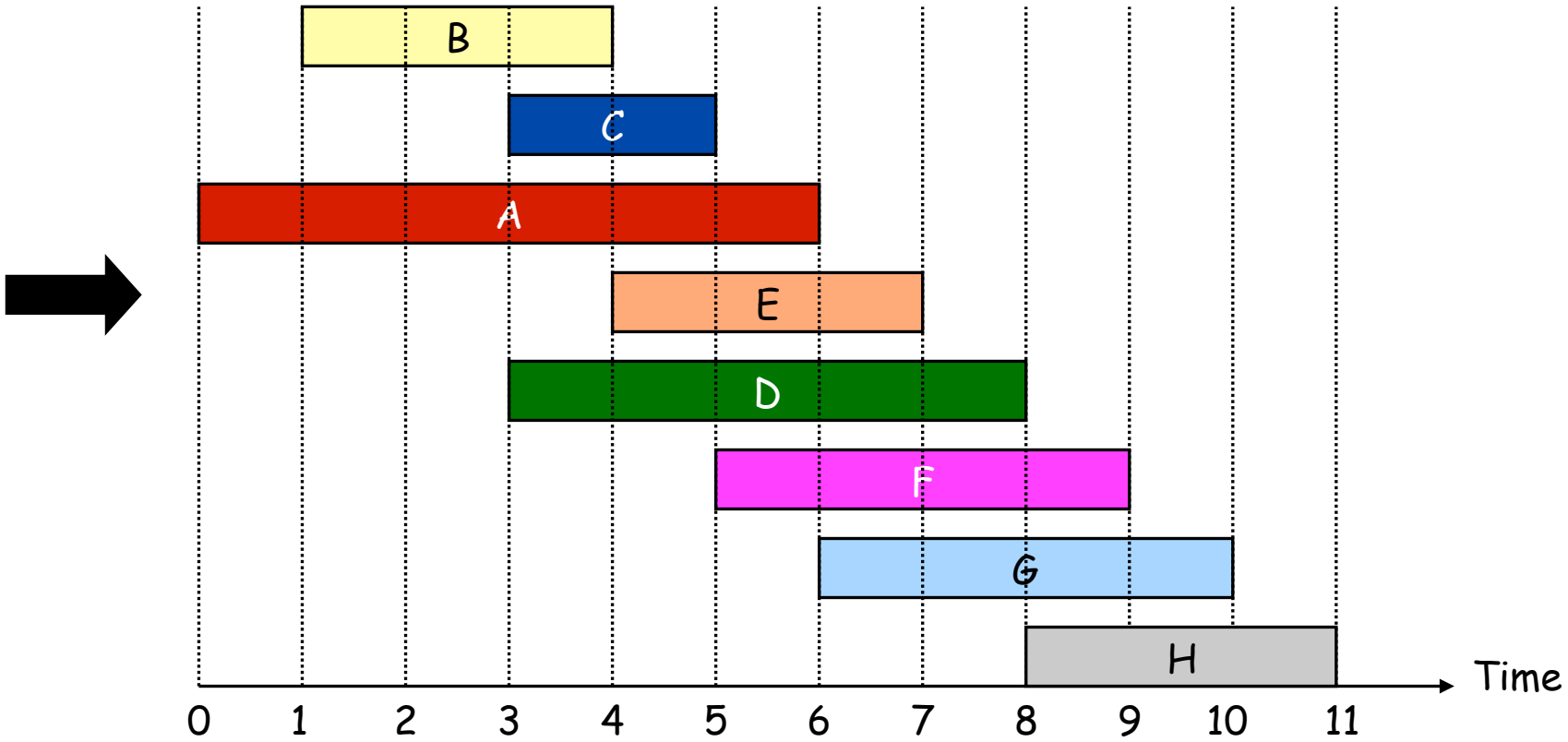


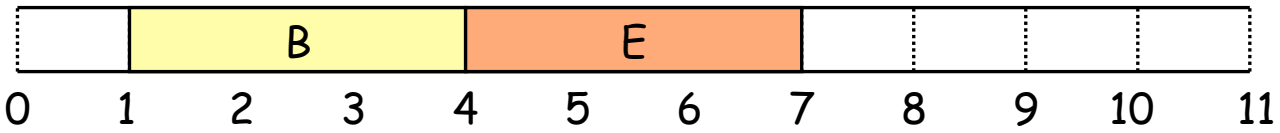
Interval Scheduling



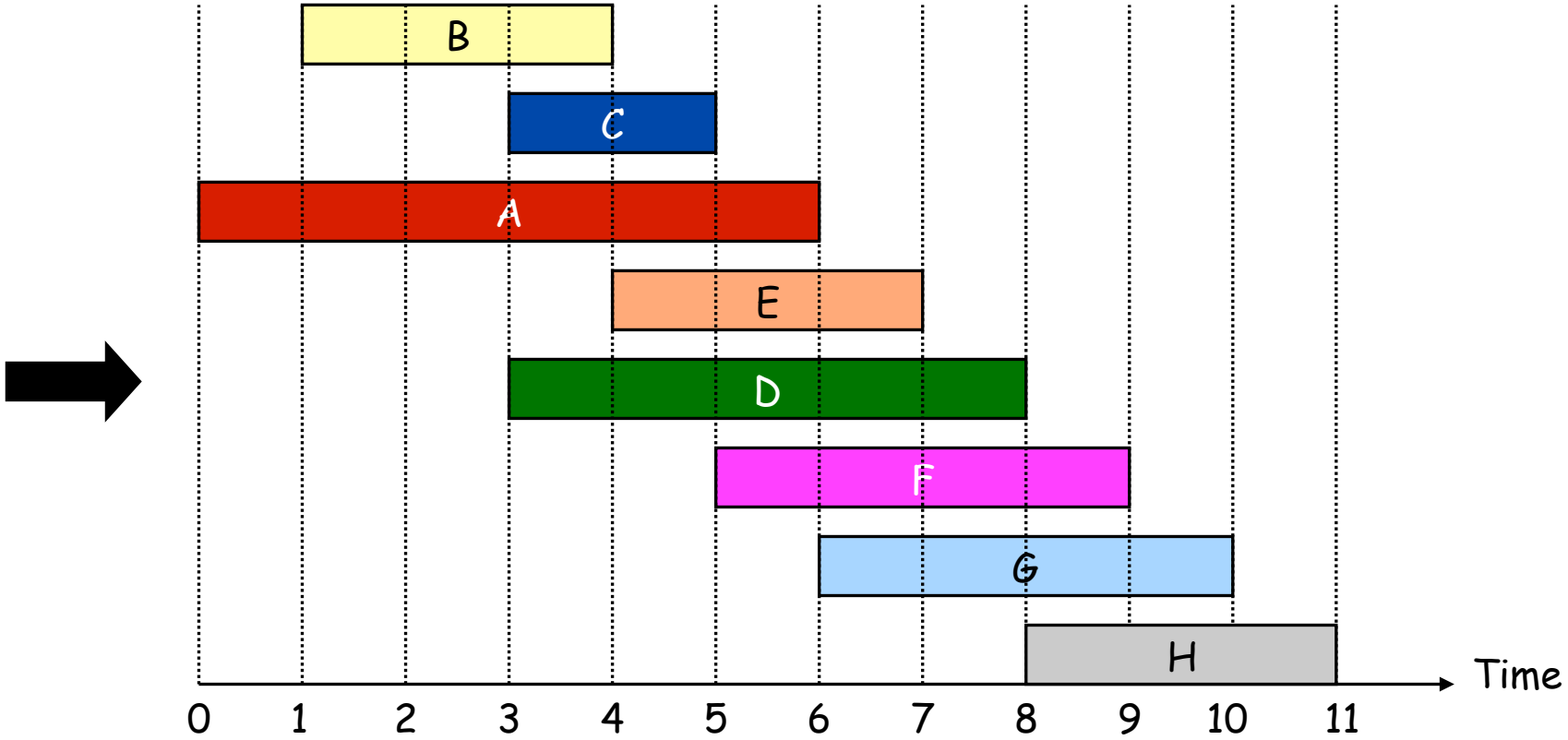


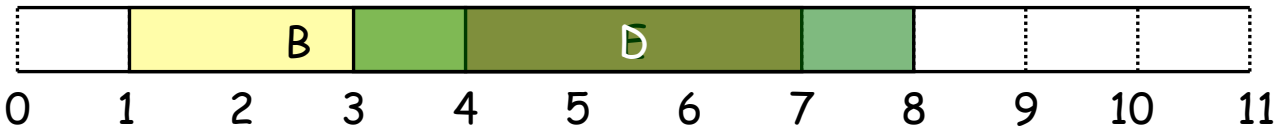
Interval Scheduling



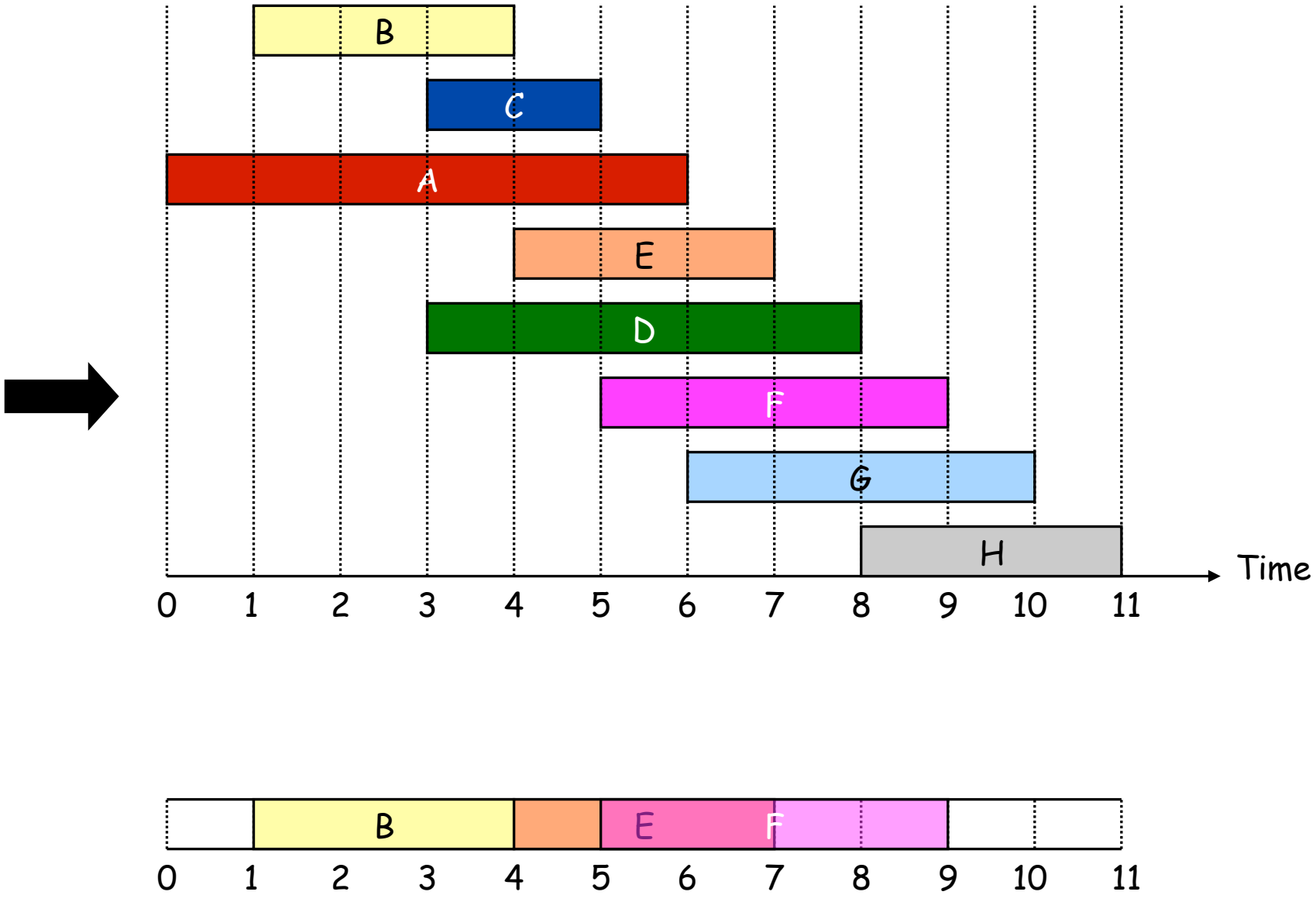


Interval Scheduling

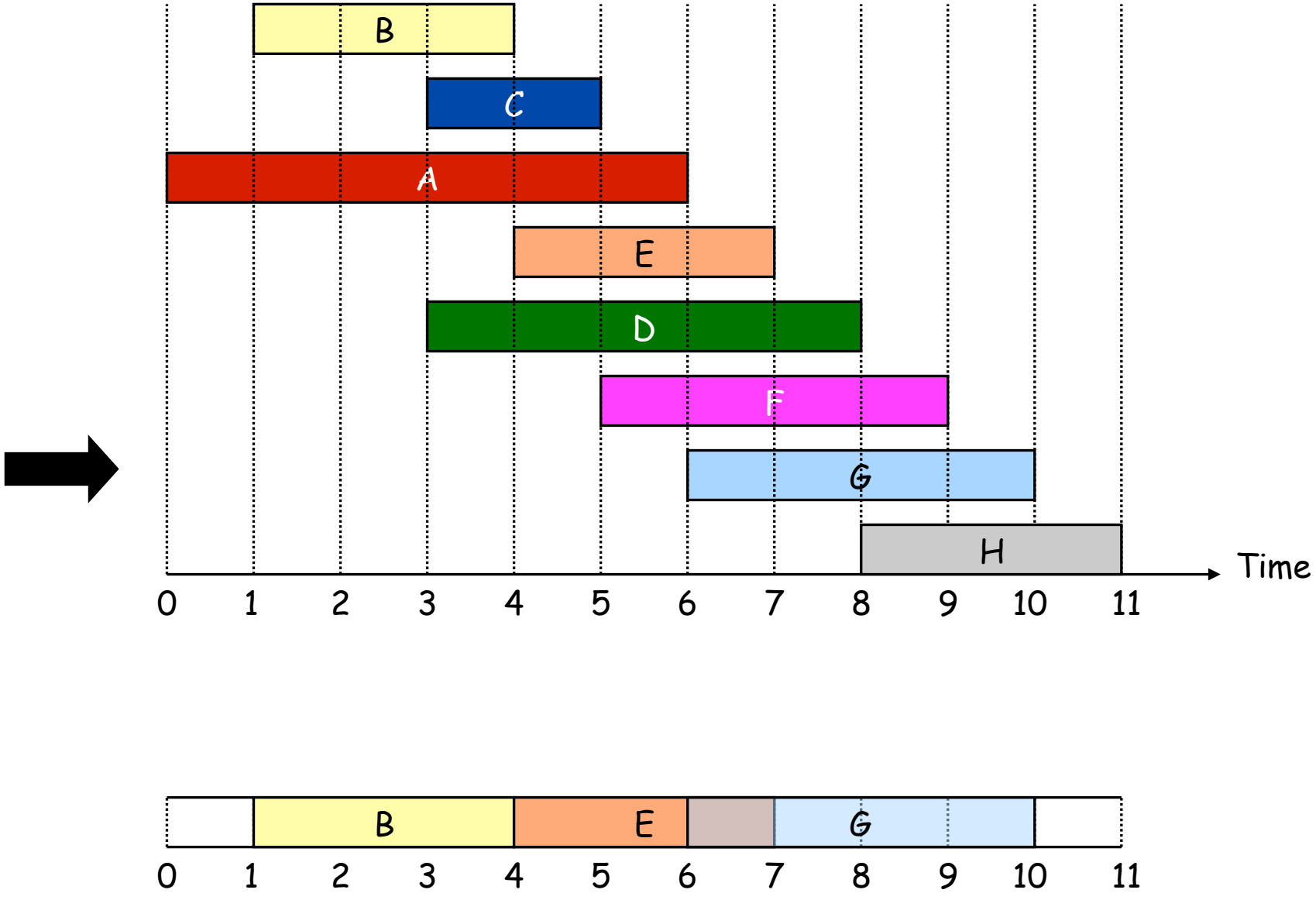




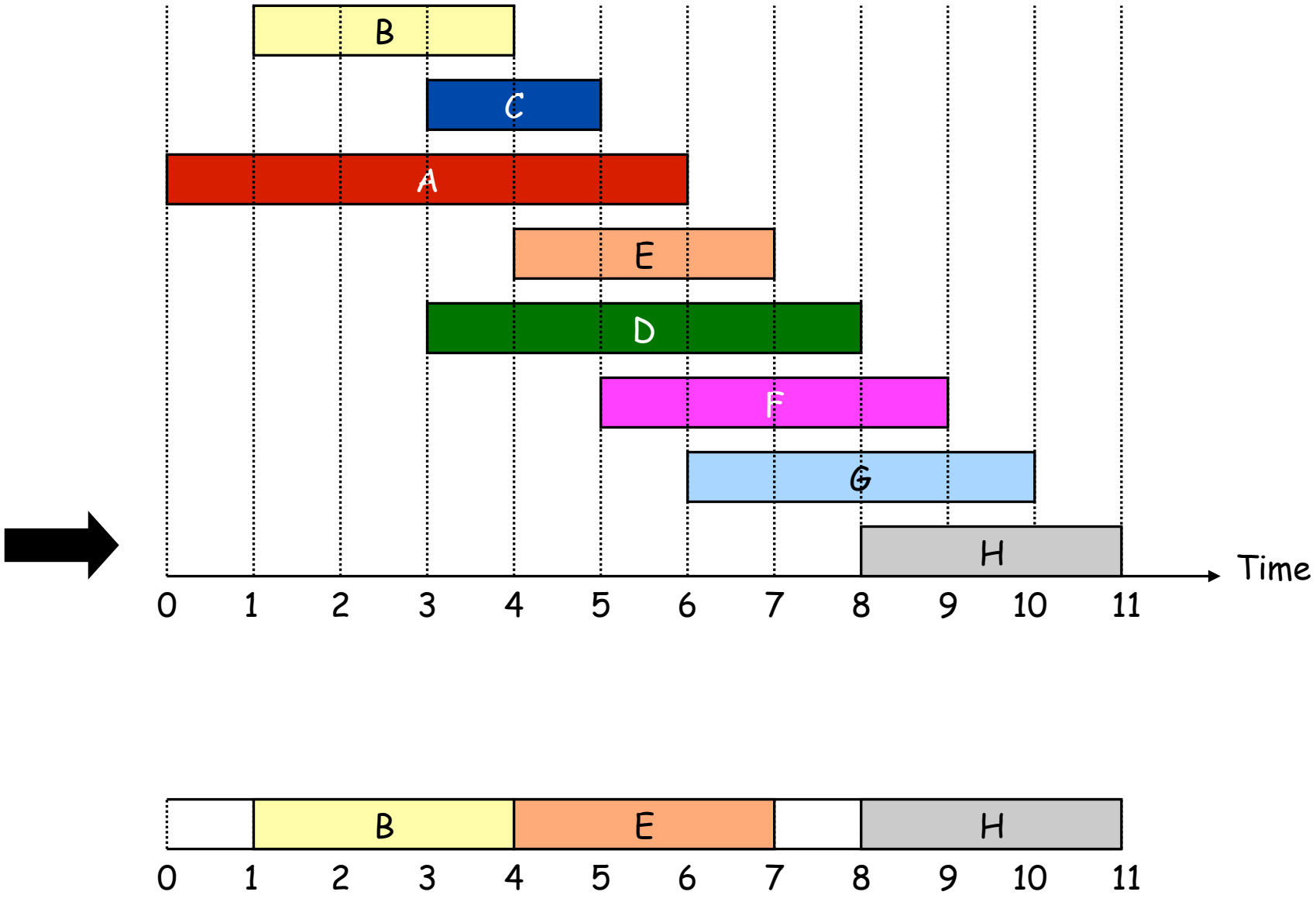
Interval Scheduling



Interval Scheduling



Interval Scheduling



Question 1:

Can you use Dijkstra on graph that has negative edge weights?

Question 1:

Can you use Dijkstra on graph that has negative edge weights?

NO!

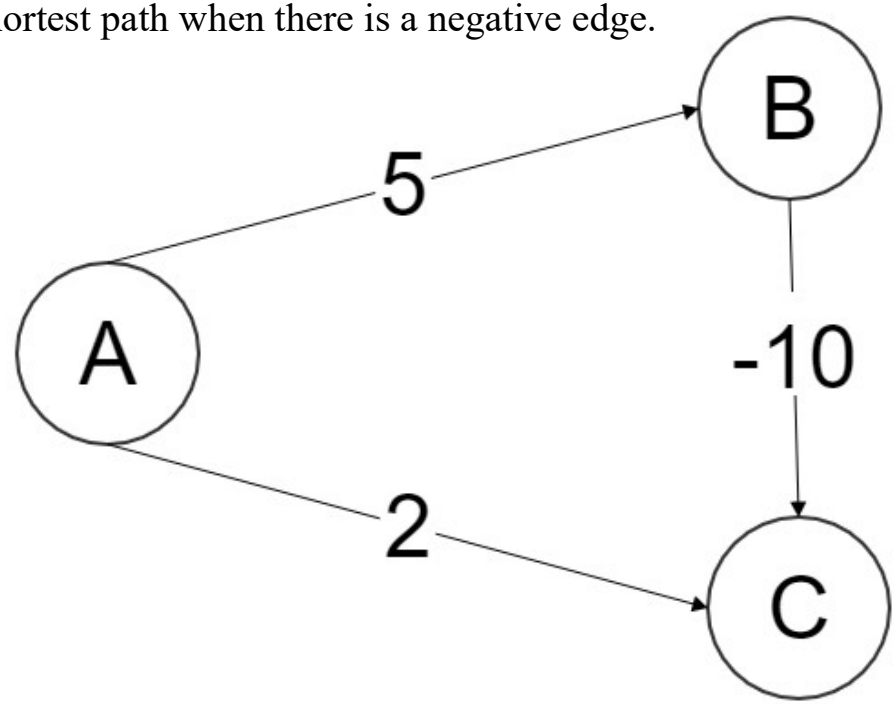
Because, if all weights are non-negative*, adding an edge can never make a path shorter*. That's why picking the shortest candidate edge (local optimality) always ends up being correct (global optimality).

Question 1:

Can you use Dijkstra on graph that has negative edge weights?

Example: Assume you want to find the path from A to C.
The algorithm will prefer “2” over “5”.
However, The path “5,-10” is shorter.
Therefore, Dijkstra may not find the shortest path when there is a negative edge.

There is another algorithm for that.



Question 2:

Can you use Dijkstra on undirected graphs?

Question 2:

Can you use Dijkstra on undirected graphs?

An undirected graph is basically the same as a directed graph with bidirectional connections (= two connections in opposite directions) between the connected nodes. So you don't really have to do anything to make it work for an undirected graph. You only need to know all of the nodes that can be reached from every given node through e.g. an adjacency list.

Question 3:

Given a set of N intervals, how many possible subsets are there, ignoring whether the intervals are compatible or not ?

Question 3:

Given a set of N intervals, how many possible subsets are there, ignoring whether the intervals are compatible or not ?

2^N. Why? Any subset of intervals defines a binary labelling of the subsets e.g. 1 if an interval is in the subset, or 0 if an interval is not in the subset. Since there are two possible labels per interval and the labellings are independent (if we ignore compatibility), then there are 2^N = 2*2*….*2 labellings in total.