Georgia Tech

CSE 6140/ CX 4140:

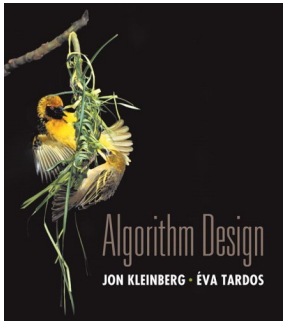
Computational Science and Engineering ALGORITHMS

Instructor: Anne Benoit

Visiting Associate Professor, CSE

Based on slides by Bistra Dilkina

Scheduling to Minimize Lateness [KT4]



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

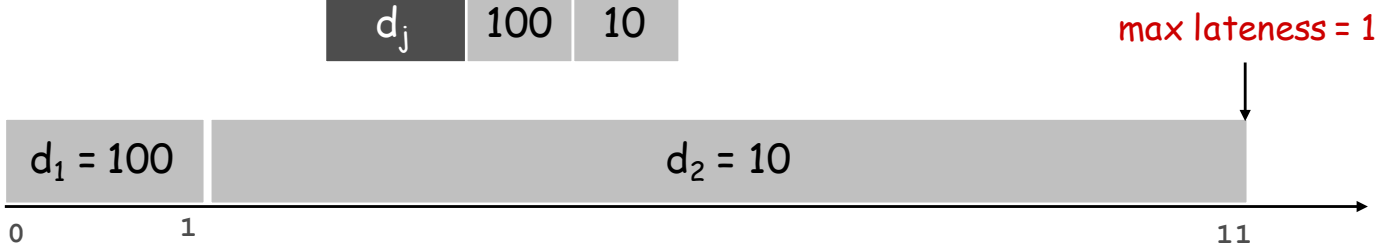
Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

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

	1	2
t_j	1	10
d_j	100	10

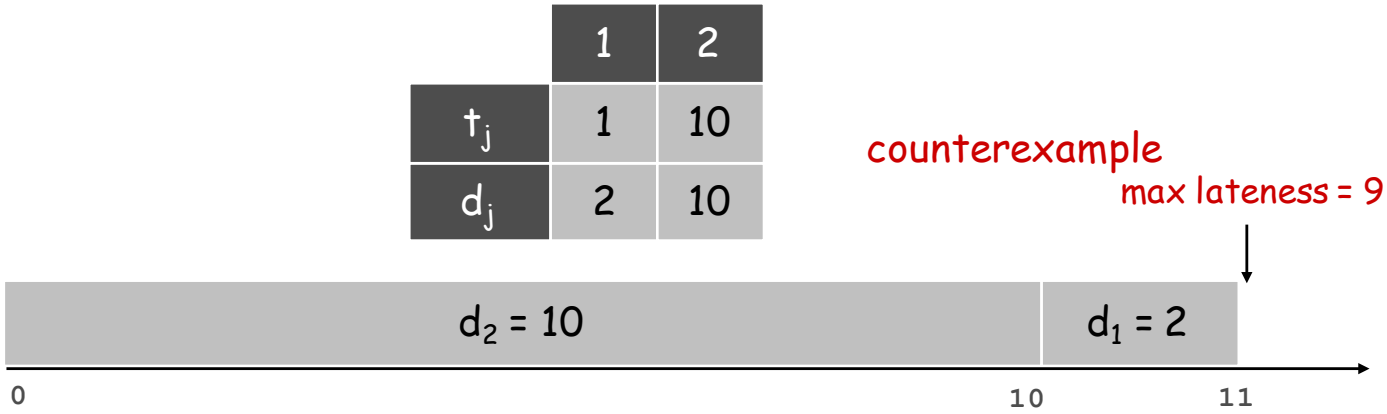
counterexample



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

	1	2
t_j	1	10
d_j	2	10

counterexample



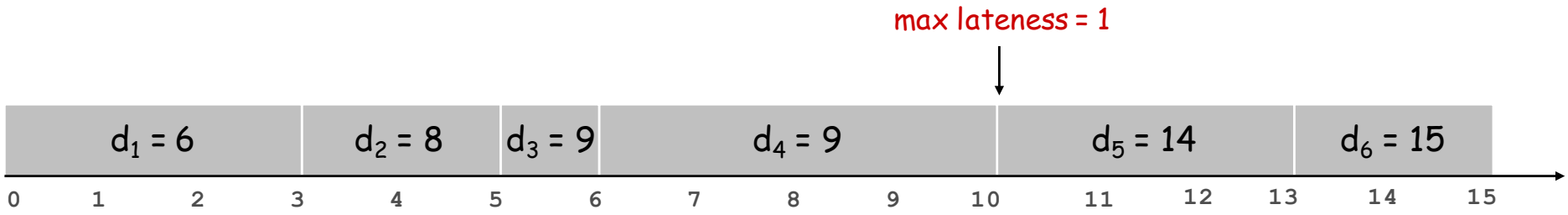
Minimizing Lateness: Greedy Algorithm

Greedy algorithm. Earliest deadline first.

Is this algorithm optimal or can you find a counter-example?

```
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]
```



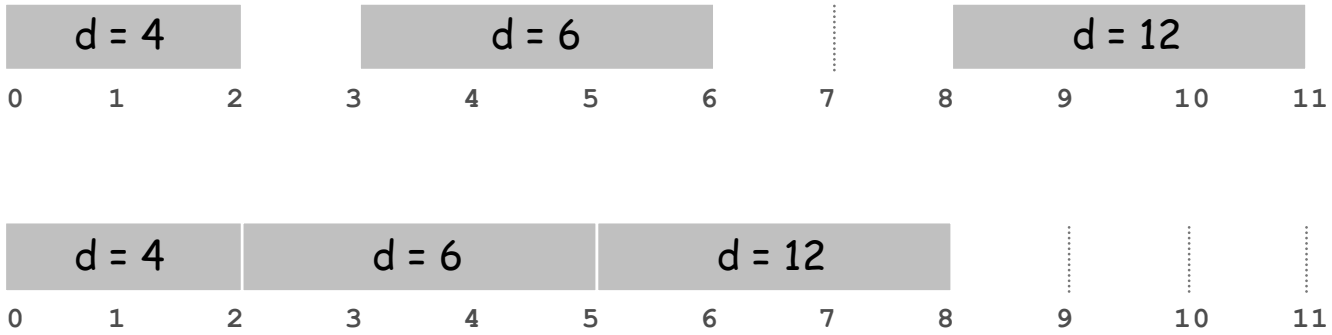
Proof of Greedy's optimality: Exchange Argument

- We will show that if there is another schedule O (think optimal schedule) then we can gradually change O so that
  - at each step the maximum lateness in O never gets worse
  - it eventually becomes the same cost as A

Minimizing Lateness: No Idle Time

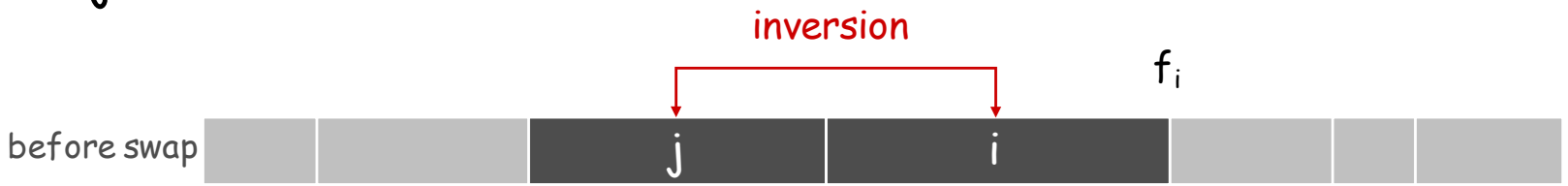
Observation. The greedy schedule has no idle time.

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



Minimizing Lateness: Inversions

Def. Given a schedule S, an inversion is a pair of jobs i and j such that: $d_i < d_j$, 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. (why?)

Minimizing Lateness: Inversions

Def. Given a schedule S, an inversion is a pair of jobs i and j such that: $d_i < d_j$, but j scheduled before i.

Lemma. All schedules with no inversions and no idle time have the same lateness

Pf.

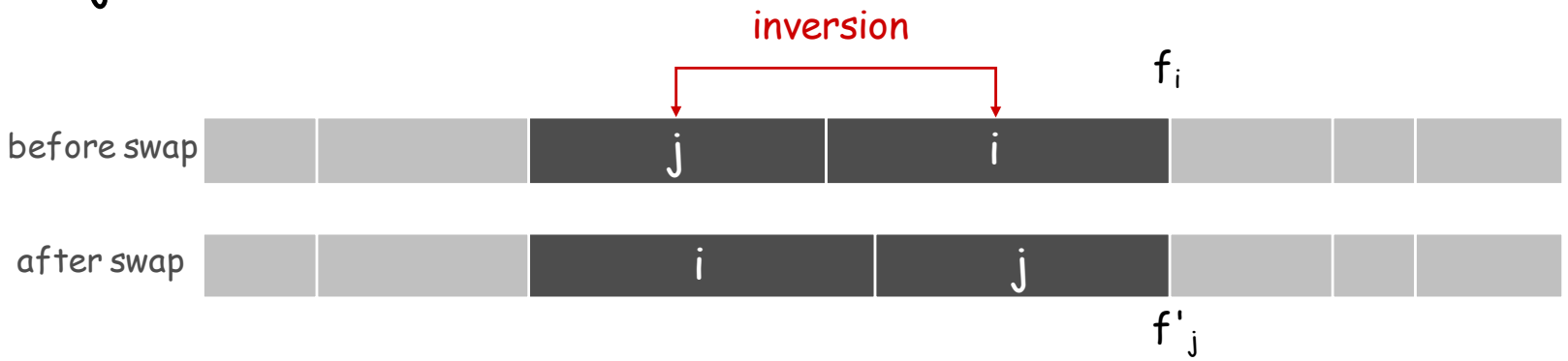
- All jobs with same deadline come in a block of consecutive jobs in any schedule with no inversions

- In any reordering of these jobs, the last job of the block has the same finish time and it is the worst finish time among the jobs in the block

- They all have same deadline, so latest job in the block is always last in the block and its lateness is always f-d (no matter reordering)

Minimizing Lateness: Inversions

Def. Given a schedule S, an inversion is a pair of jobs i and j such that: $d_i < d_j$, but j scheduled before i.



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

Pf. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards.

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

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

Minimizing Lateness: Analysis of Greedy Algorithm

Theorem. There is an optimal schedule with no inversions and no idle time.

Pf.

- By previous argument there is an optimal schedule O with no idle time

- Exchange argument:
- If O has an inversion, then it has a consecutive pair of requests in its schedule that are inverted and can be swapped without increasing lateness (our previous claim)

- Eventually, these swaps will produce an optimal schedule with no inversions

- Each swap decreases the number of inversions by 1
- There are a bounded number of (at most n(n-1)/2) inversions (we only care that this is finite.)

Minimizing Lateness: Analysis of Greedy Algorithm

Theorem. Greedy is optimal.

Pf.

- There is an optimal schedule with no idle time and no inversions
- Greedy has no idle time and no inversions
- All schedules with no idle time and no inversions have the same lateness
- Greedy has optimal lateness

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.

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.

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

Other greedy algorithms. Dijkstra, Kruskal, Prim, Huffman, …

Designing Greedy Algorithms

1. Decide on a greedy choice that allows us to optimize the problem locally;

2. Seach for a counterexample that shows that the algorithm is not optimal (and go back to step 1 if a counterexample is found), or prove its optimality through steps 3 & 4 (or other proof);

3. Show that there is always an optimal solution that performs the greedy choice of step 1;

4. Show that if we combine the greedy choice with an optimal solution of the subproblem that we still need to solve, then we obtain an optimal solution.

We make a local choice, and then we have a single subproblem to solve, given this choice. Top-down algorithm.

Graphs & Applications

Graph. $G = (V, E)$

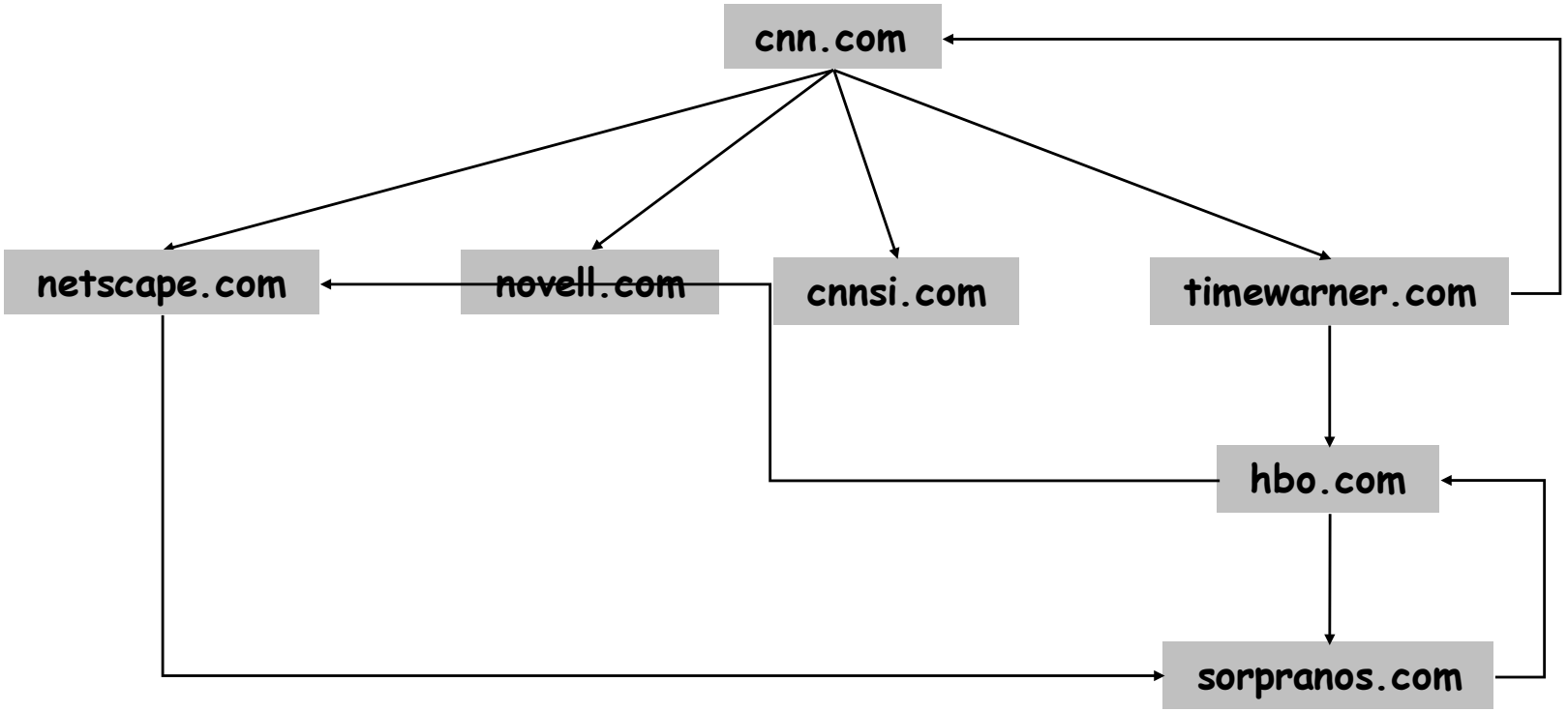
- V = nodes.
- E = edges between pairs of nodes (undirected, directed, weighted).
- Captures pairwise relationship between objects.
- Graph size parameters: $n = |V|$, $m = |E|$.

Graph	*Nodes*	*Edges*
transportation	street intersections	highways
communication	computers	fiber optic cables
World Wide Web	web pages	hyperlinks
social	people	relationships
food web	species	predator-prey
software systems	functions	function calls
scheduling	tasks	precedence constraints
circuits	gates	wires

World Wide Web

Web graph.

- Node: web page.
- Edge: hyperlink from one page to another.



9-11 Terrorist Network

Social network graph.

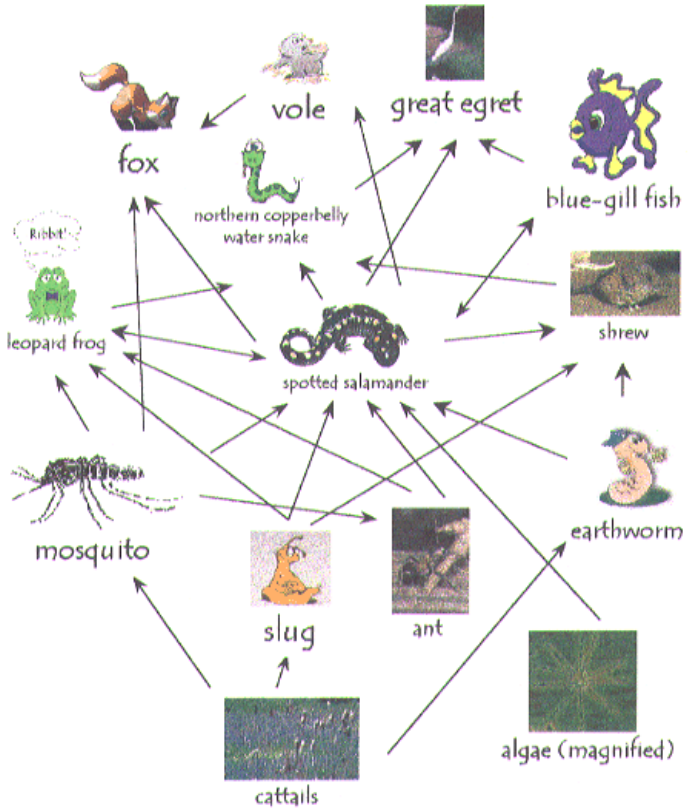
- Node: people.
- Edge: relationship between two people.

Reference: Valdis Krebs, http://www.firstmonday.org/issues/issue7_4/krebs

Ecological Food Web

Food web graph.

- Node = species.
- Edge = from prey to predator.

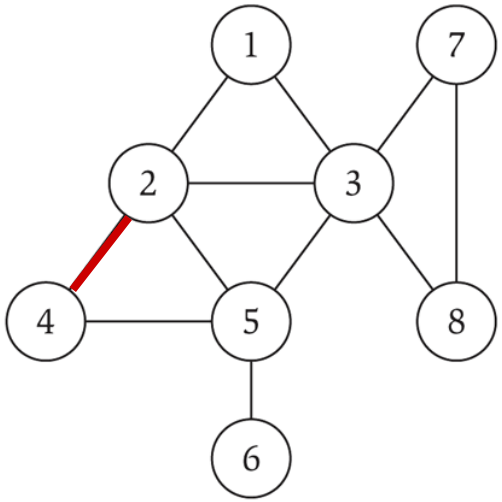


Reference: http://www.twingroves.district96.k12.il.us/Wetlands/Salamander/SalGraphics/salfoodweb.giff

Graph Representation: Adjacency Matrix

Adjacency matrix. n-by-n matrix with $A_{uv} = 1$ if (u, v) is an edge.

- Two representations of each edge.
- Space proportional to n^2.
- Checking if (u, v) is an edge takes $\Theta(1)$ time.
- Identifying all edges takes $\Theta(n^2)$ time.



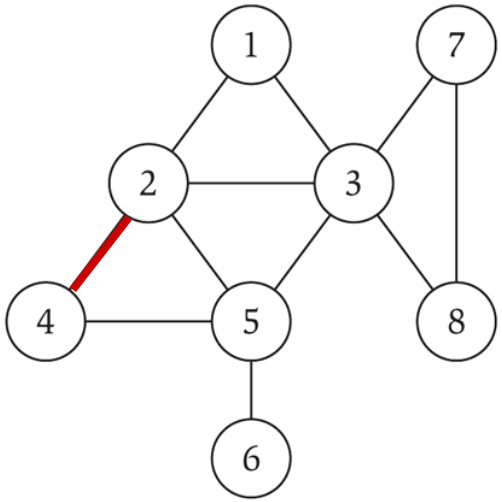
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	0	0	0
2	1	0	1	1	1	0	0	0
3	1	1	0	0	1	0	1	1
4	0	1	0	0	1	0	0	0
5	0	1	1	1	0	1	0	0
6	0	0	0	0	1	0	0	0
7	0	0	1	0	0	0	0	1
8	0	0	1	0	0	0	1	0

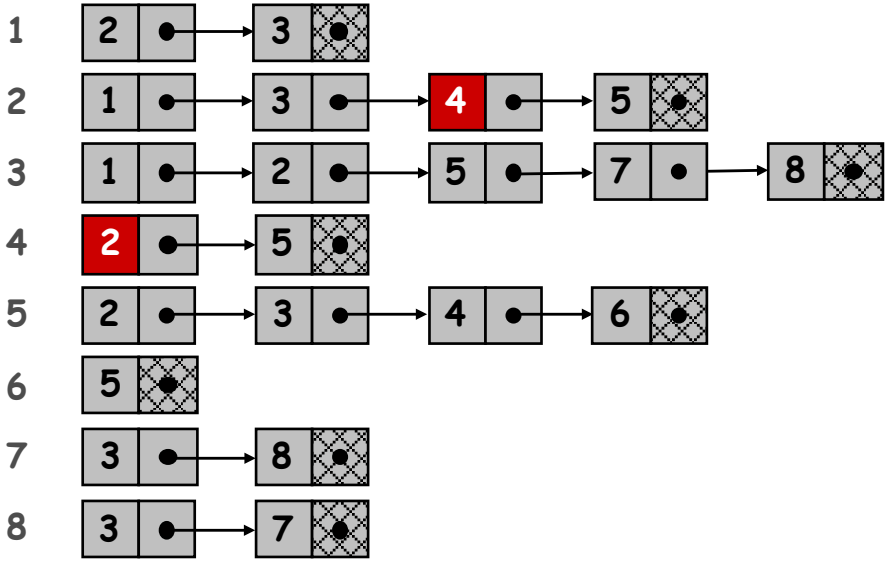
Graph Representation: Adjacency List

Adjacency list. Node indexed array of lists.

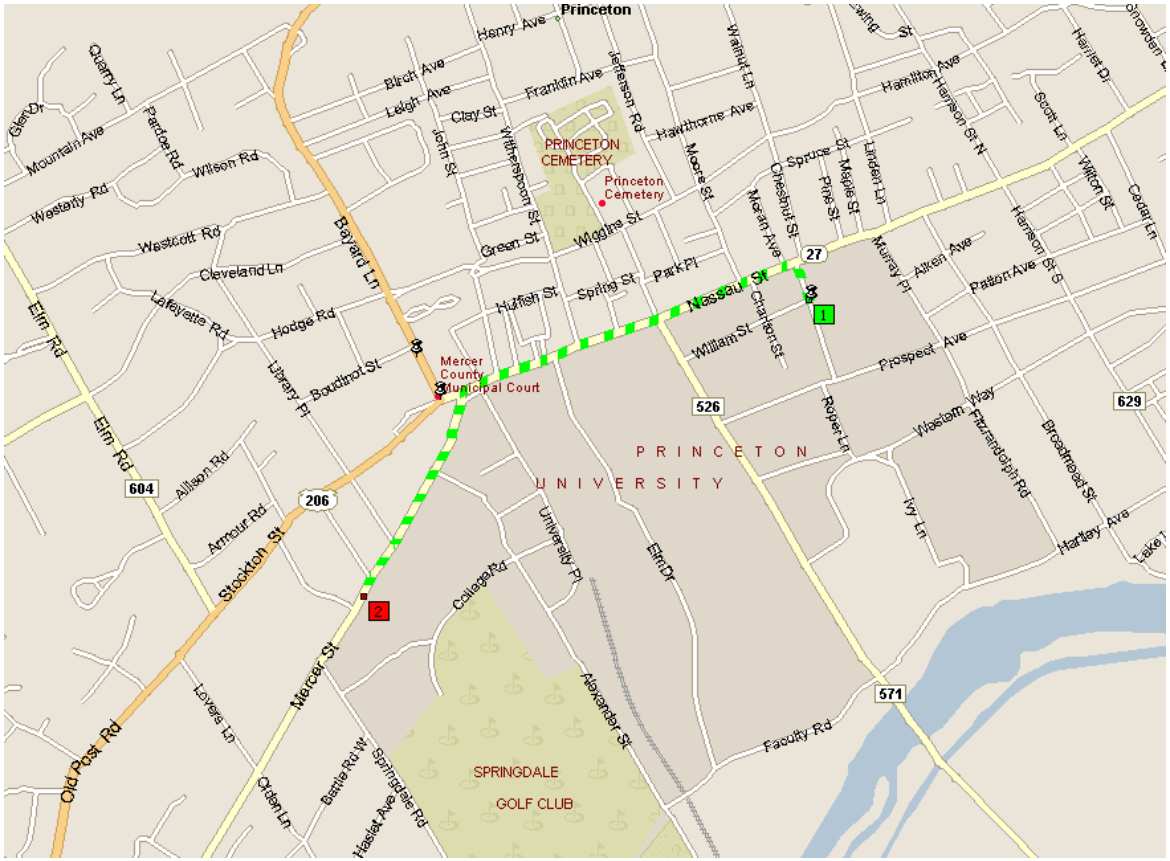
- Two representations of each edge.
- Space proportional to m + n.
- Checking if (u, v) is an edge takes O(deg(u)) time.
- Identifying all edges takes $\Theta(m + n)$ time.

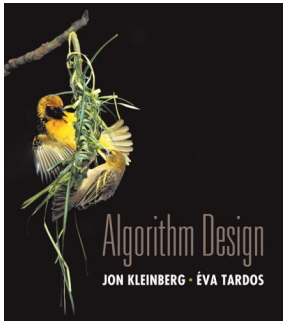
degree = number of neighbors of u





Shortest Paths in a Graph





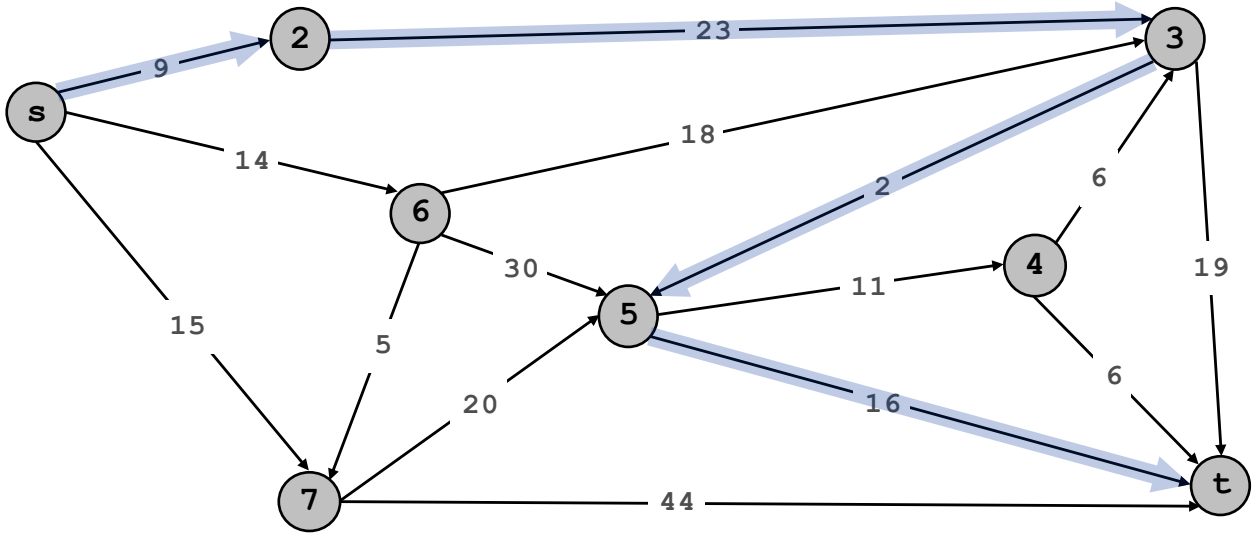
Shortest Path Problem

Shortest path network.

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

Shortest path problem: find shortest directed path from s to other nodes.

cost of path = sum of edge costs in path



Cost of path s-2-3-5-t
= 9 + 23 + 2 + 16
= 50.

Applications

- Map routing.
- Robot navigation.
- Urban traffic planning.
- Optimal truck routing through given traffic congestion pattern.
- Routing of telecommunications messages.
- Network routing protocols (OSPF, BGP, RIP).
- Circuit design – critical path analysis.

Dijkstra's Algorithm

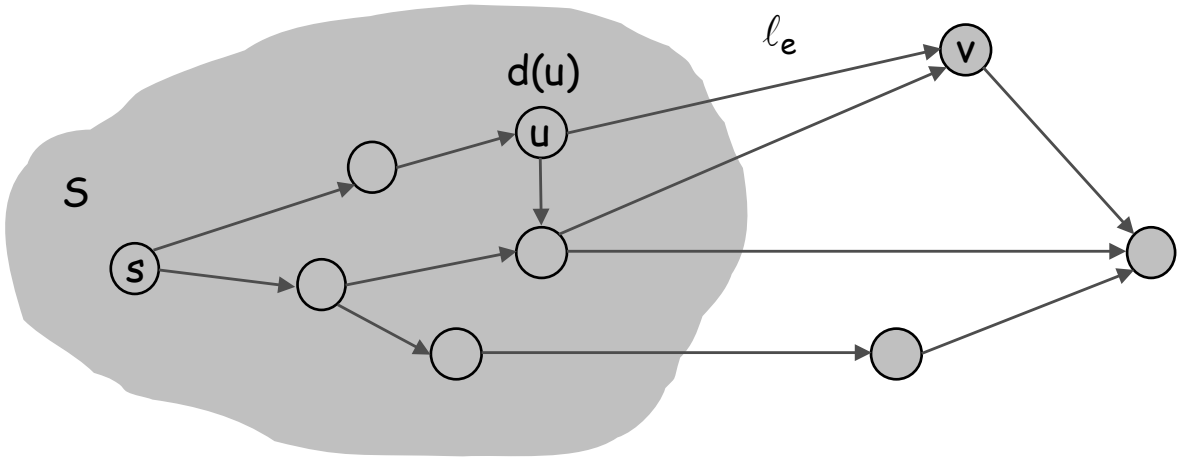
Dijkstra's algorithm.

- 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 greedily 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

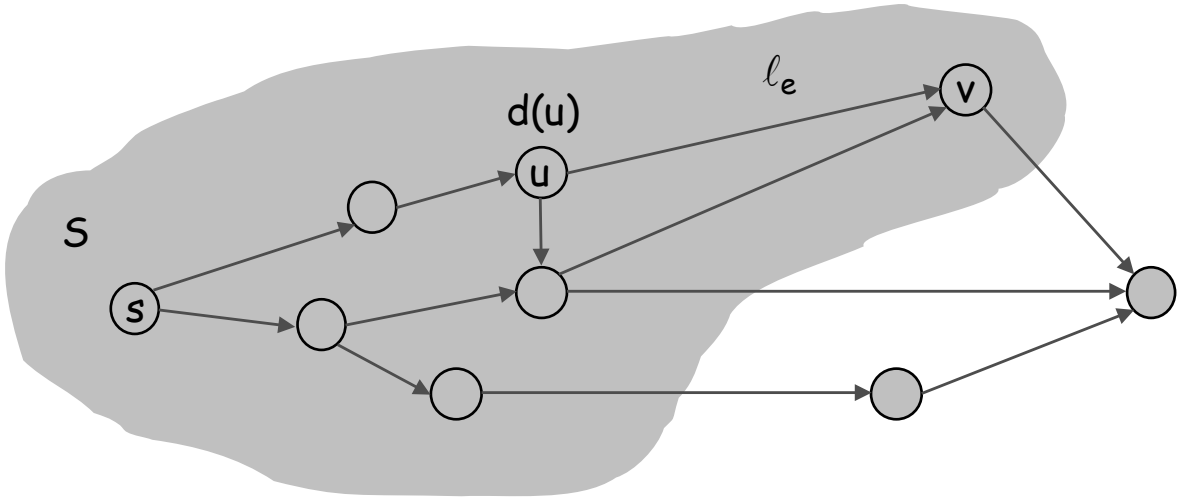
Dijkstra's algorithm.

- 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 greedily 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

Dijkstra's algorithm.

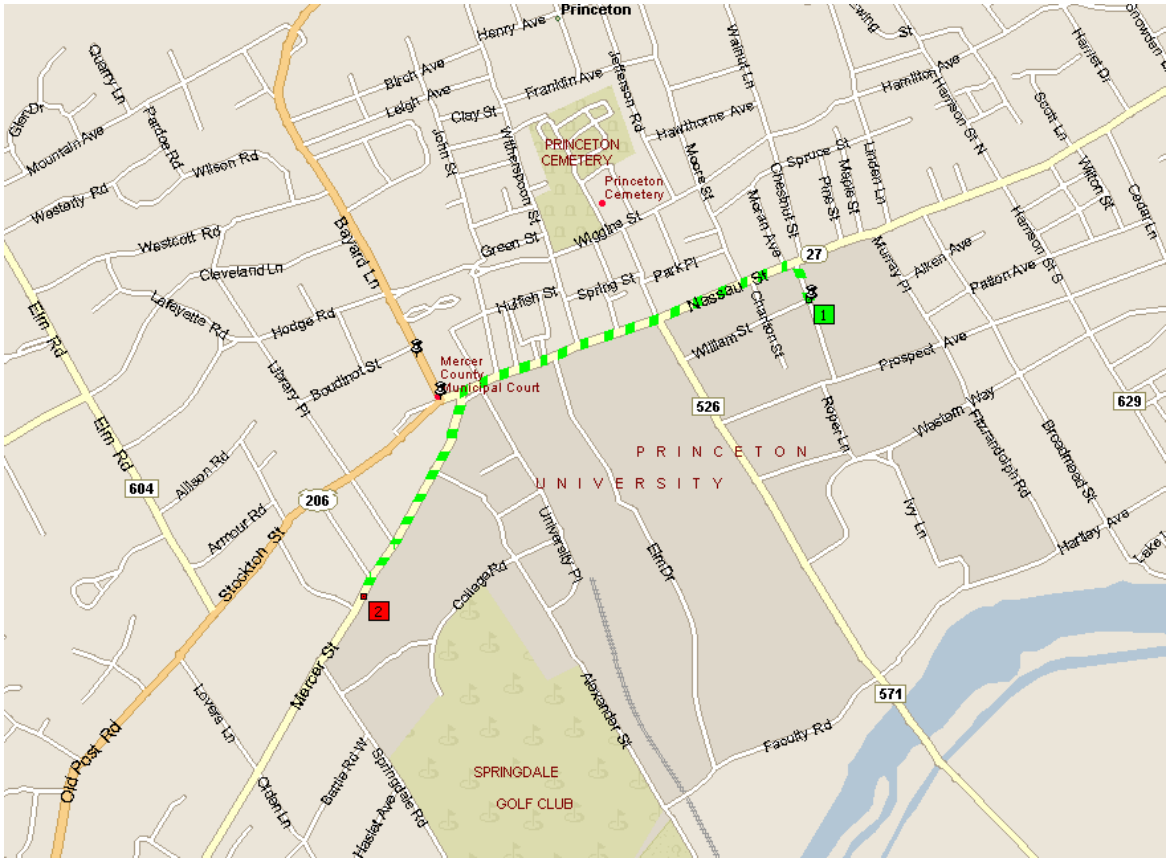
- Set of explored nodes S = ∅
- Initialize source $\pi(s)=0$, and other nodes $\pi(v)=\infty$
- Insert all nodes to priority queue Q with keys $\pi(v)$
- While Q is not empty
  - u = Q.ExtractMin()
  - Add u to S, and set d(u) = $\pi(u)$
  - For each edge (u,v) where $v \notin S$ and d(u) + l(u,v) < $\pi(v)$
    - Q.ChangeKey(v, $\pi(v)$ =d(u) + l(u,v))

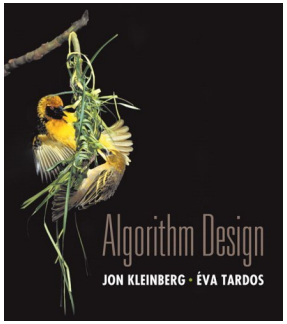
Also can keep track of responsible edge, parent(v)=(u,v)

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

Shortest Paths- Dijkstra's demo



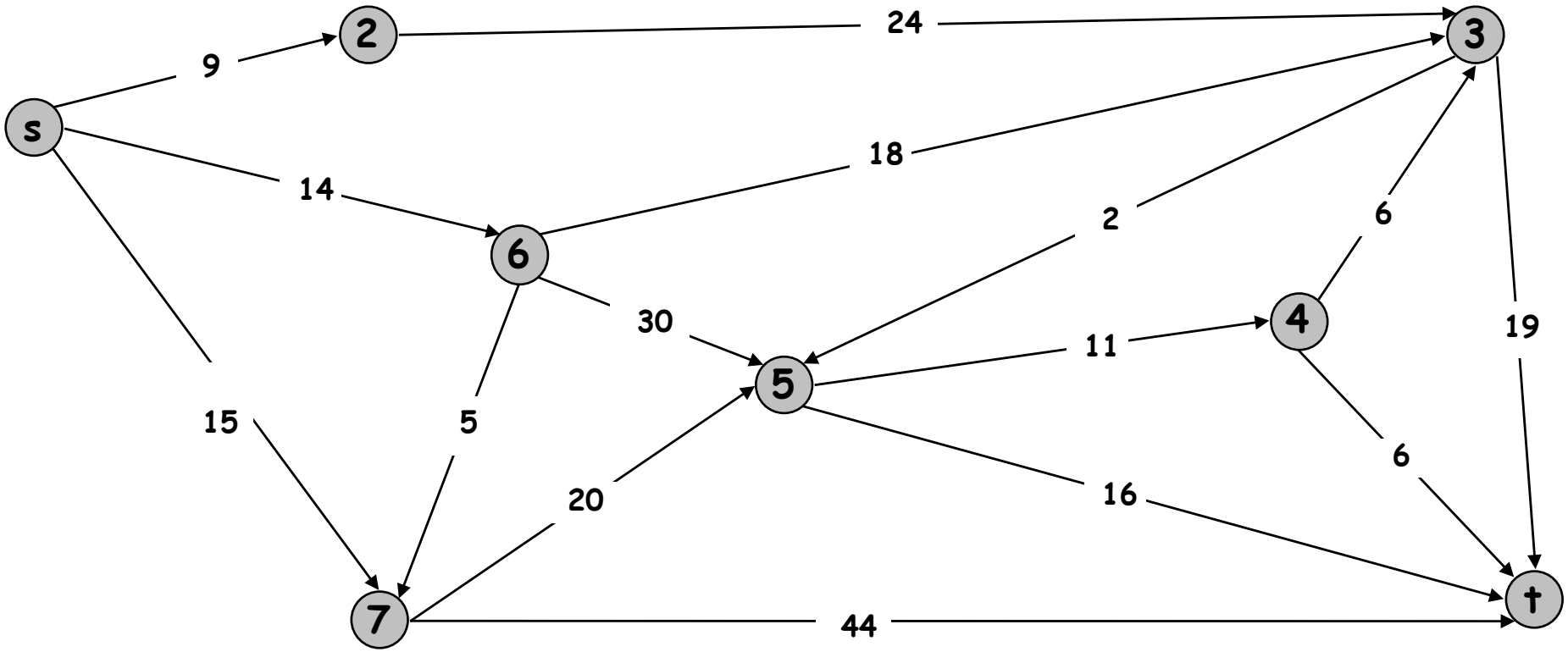


PEARSON Addison Wesley

Slides by Kevin Wayne.

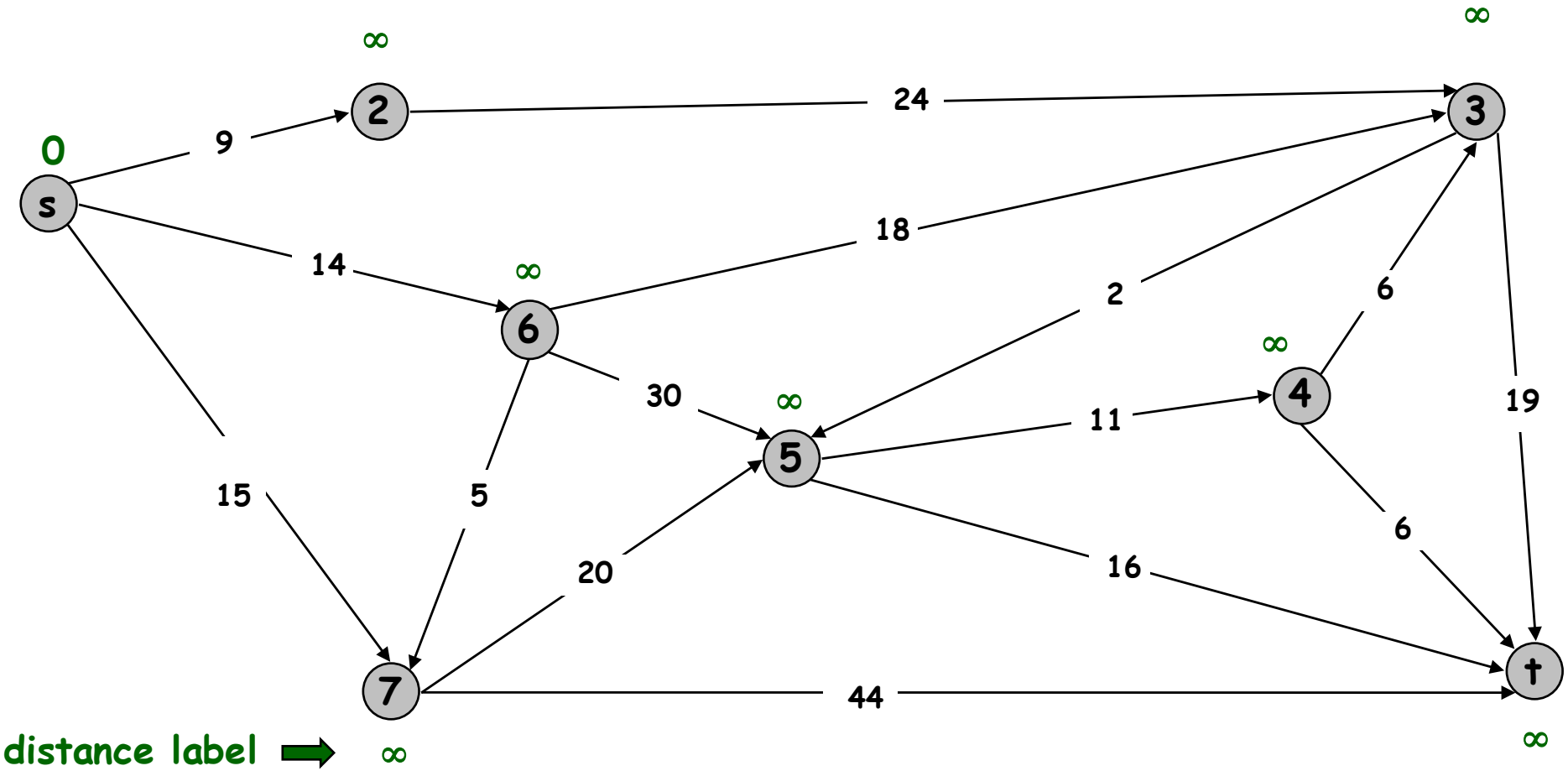
Dijkstra's Shortest Path Algorithm

Find shortest path from s to t.



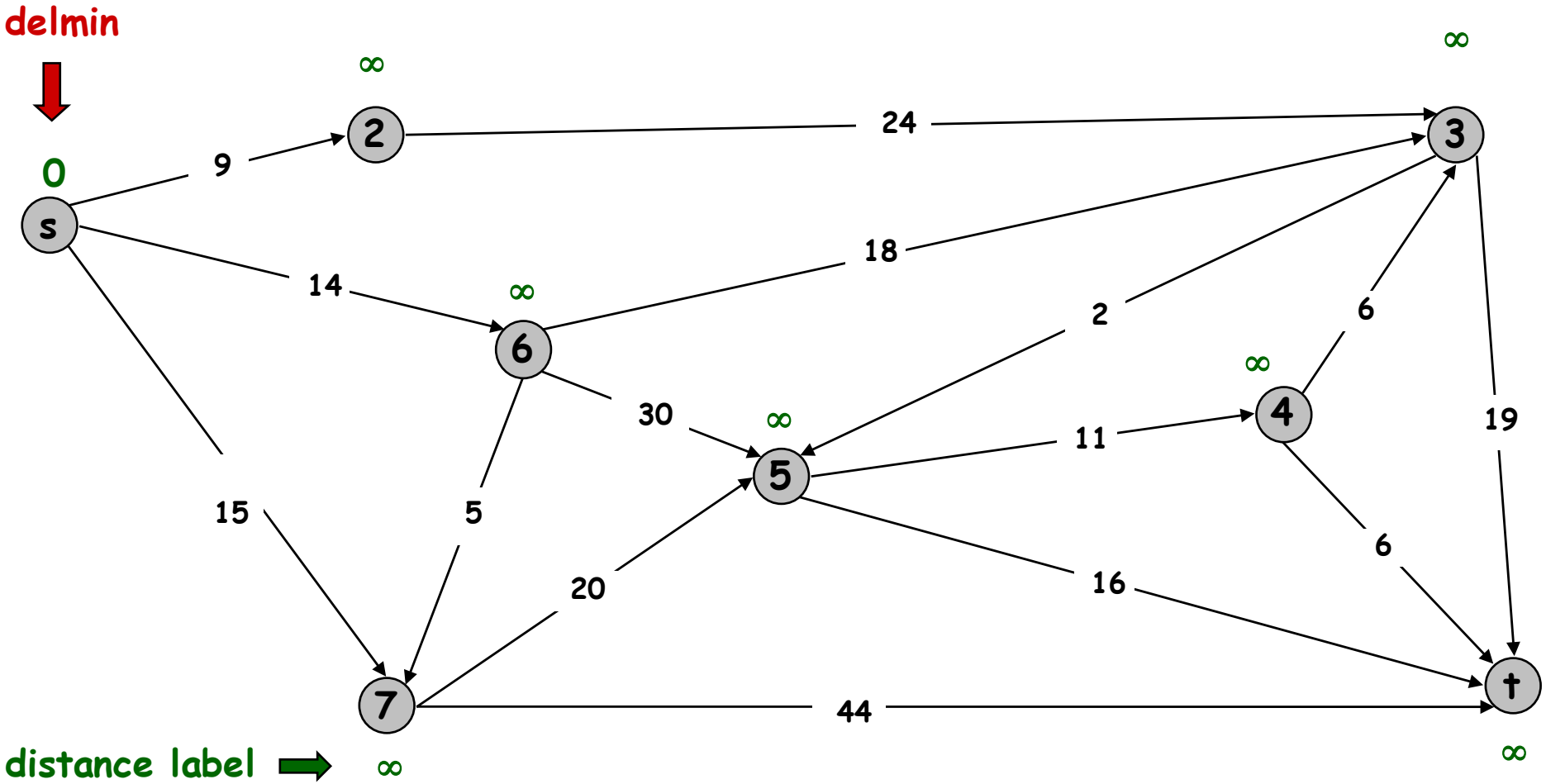
Dijkstra's Shortest Path Algorithm

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



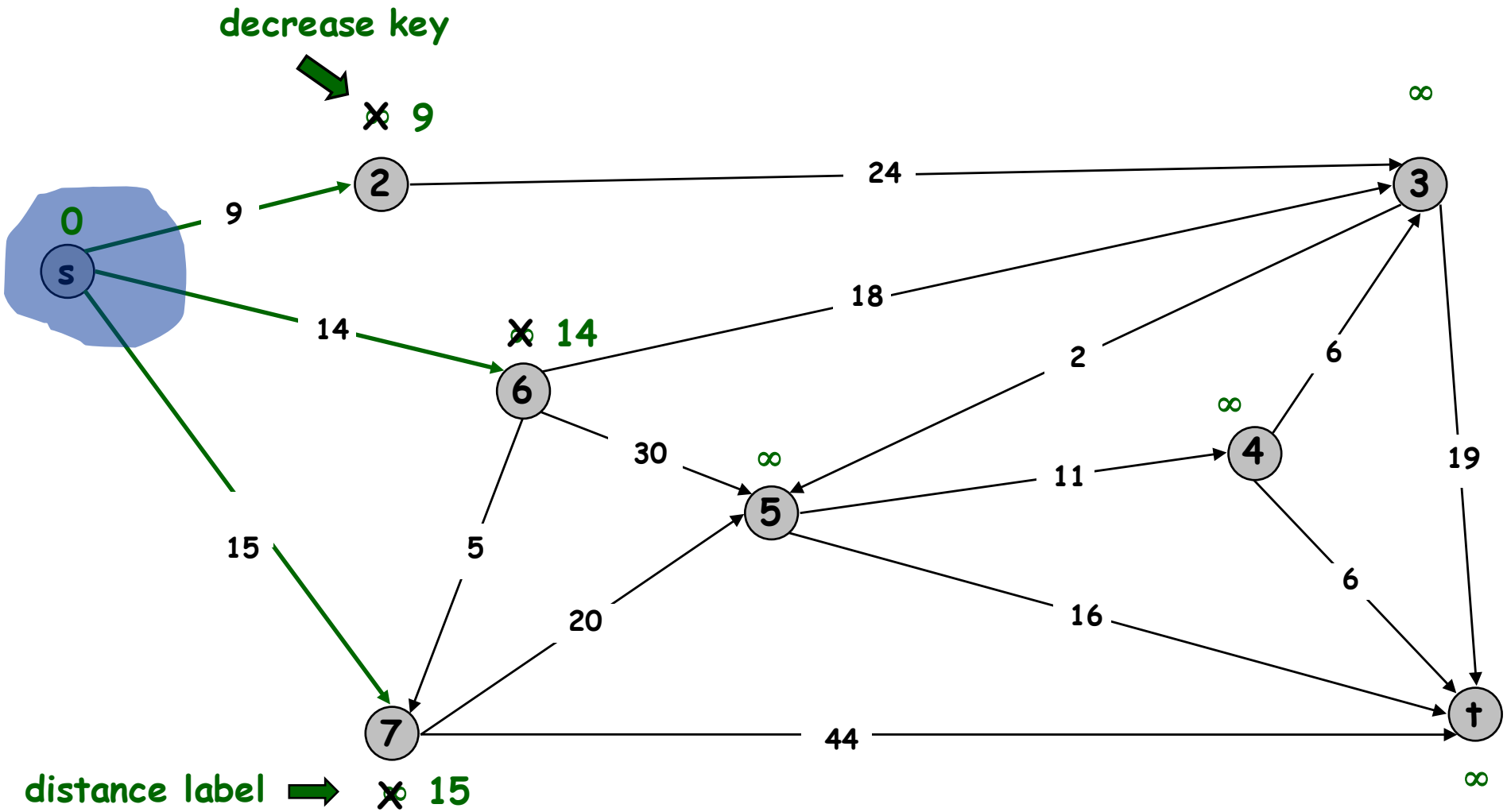
Dijkstra's Shortest Path Algorithm

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



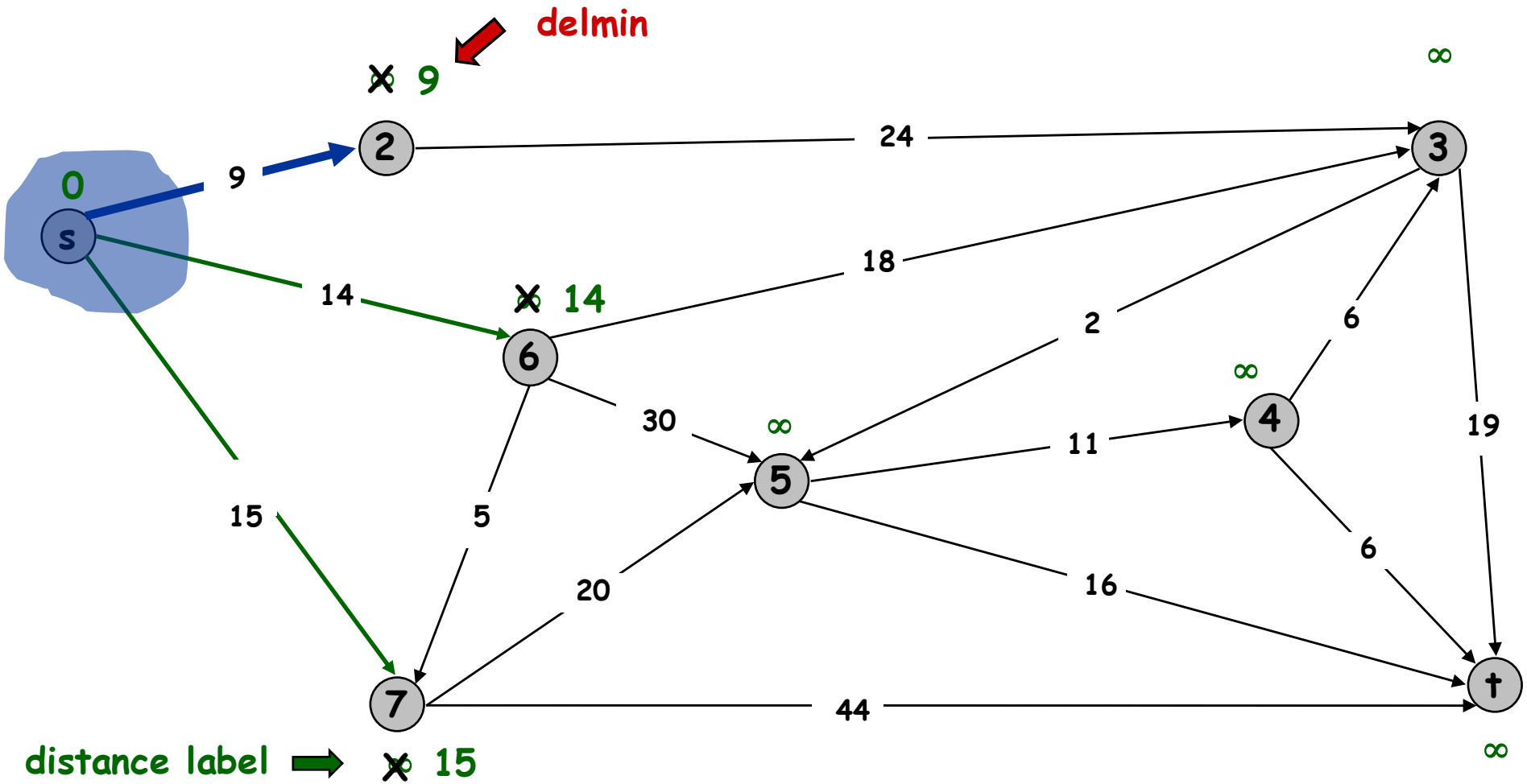
Dijkstra's Shortest Path Algorithm

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



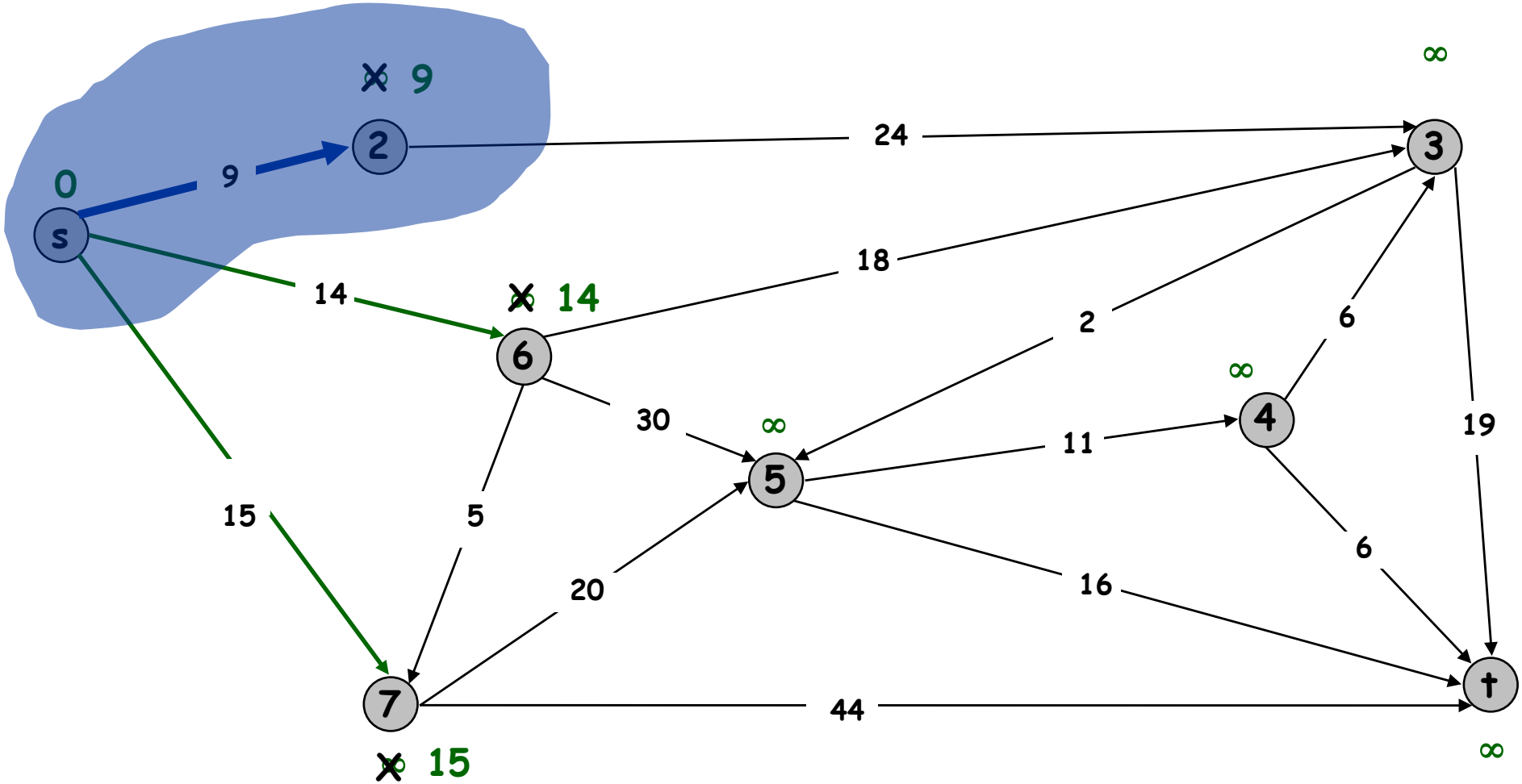
Dijkstra's Shortest Path Algorithm

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



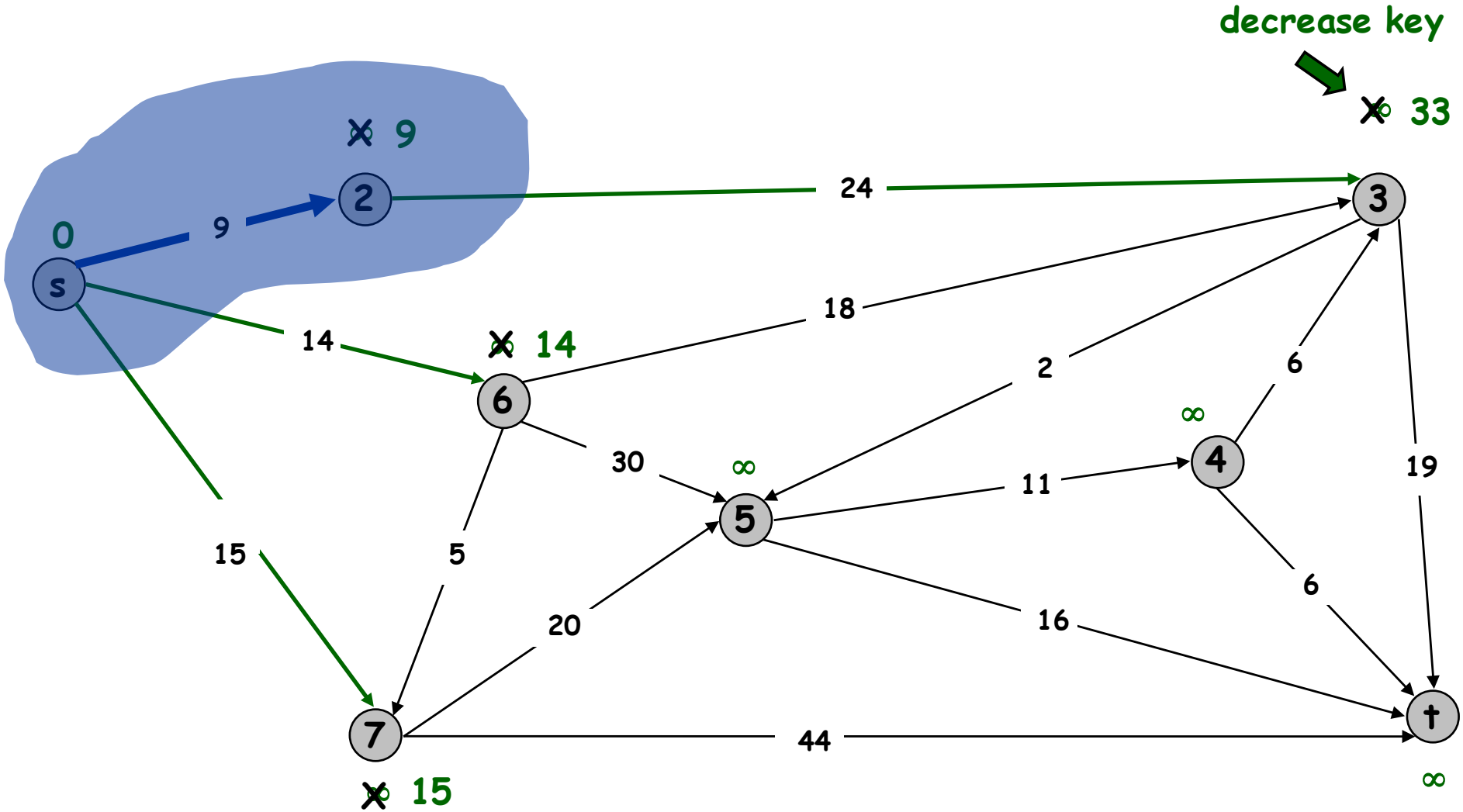
Dijkstra's Shortest Path Algorithm

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



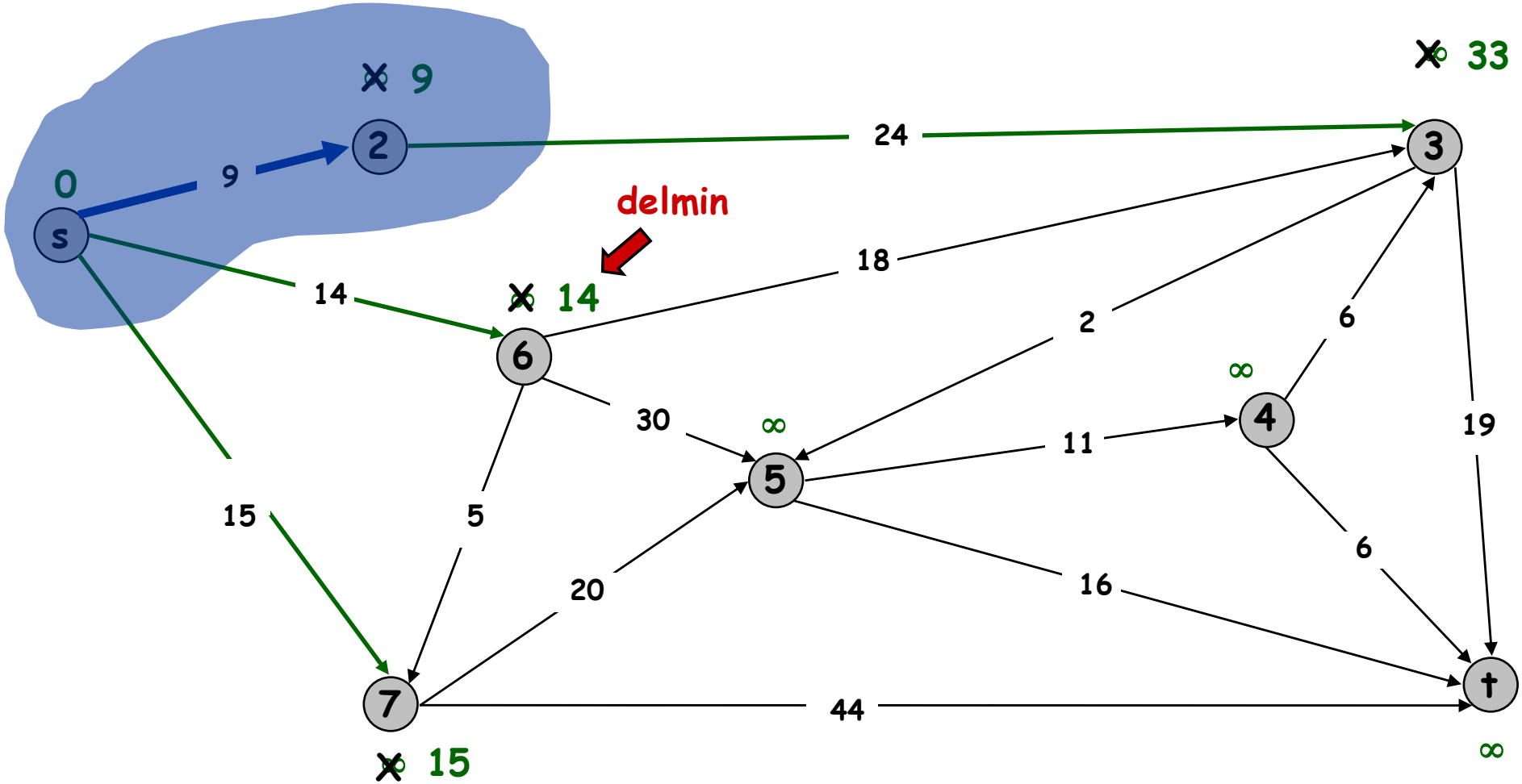
Dijkstra's Shortest Path Algorithm

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



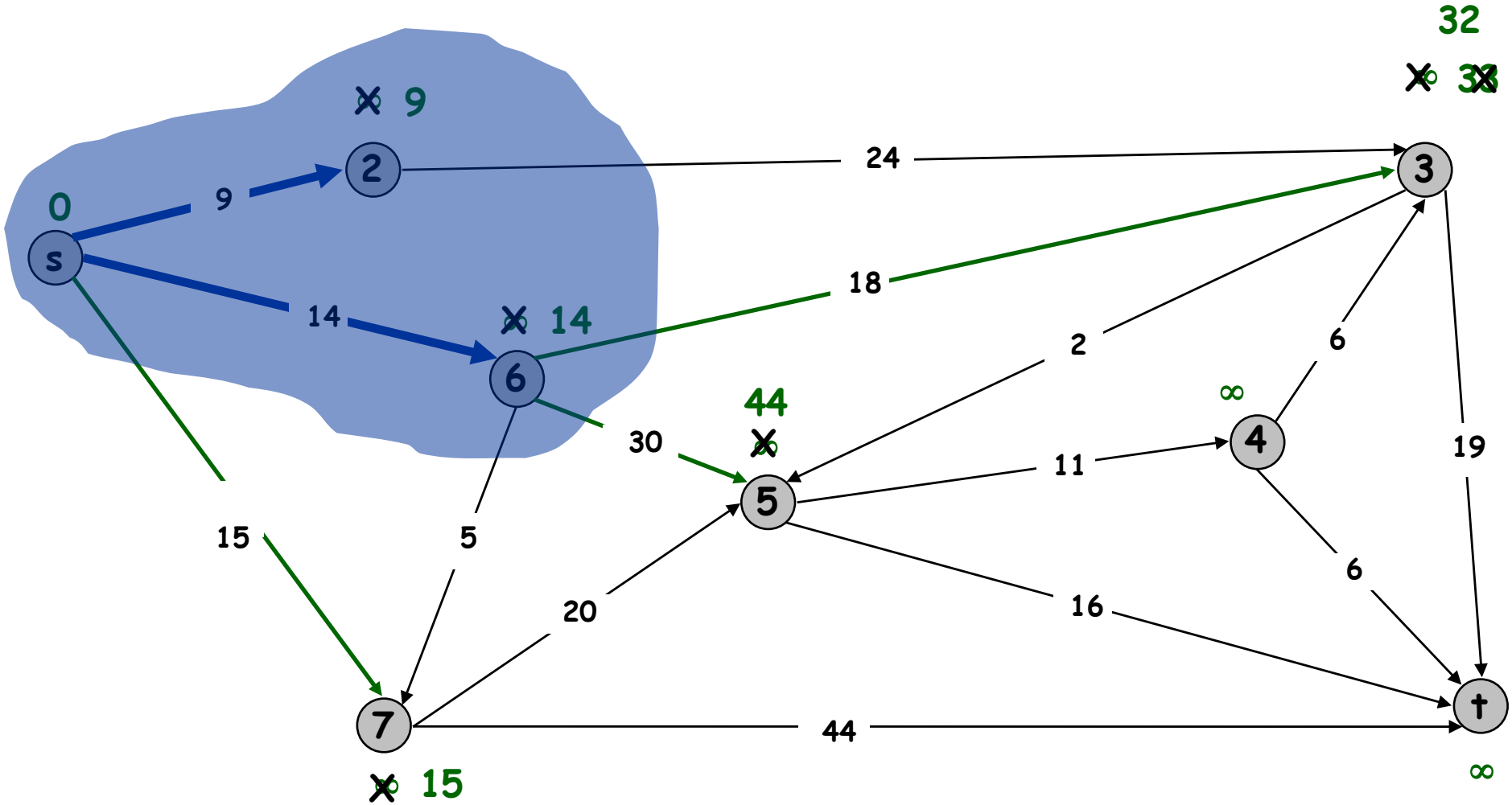
Dijkstra's Shortest Path Algorithm

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



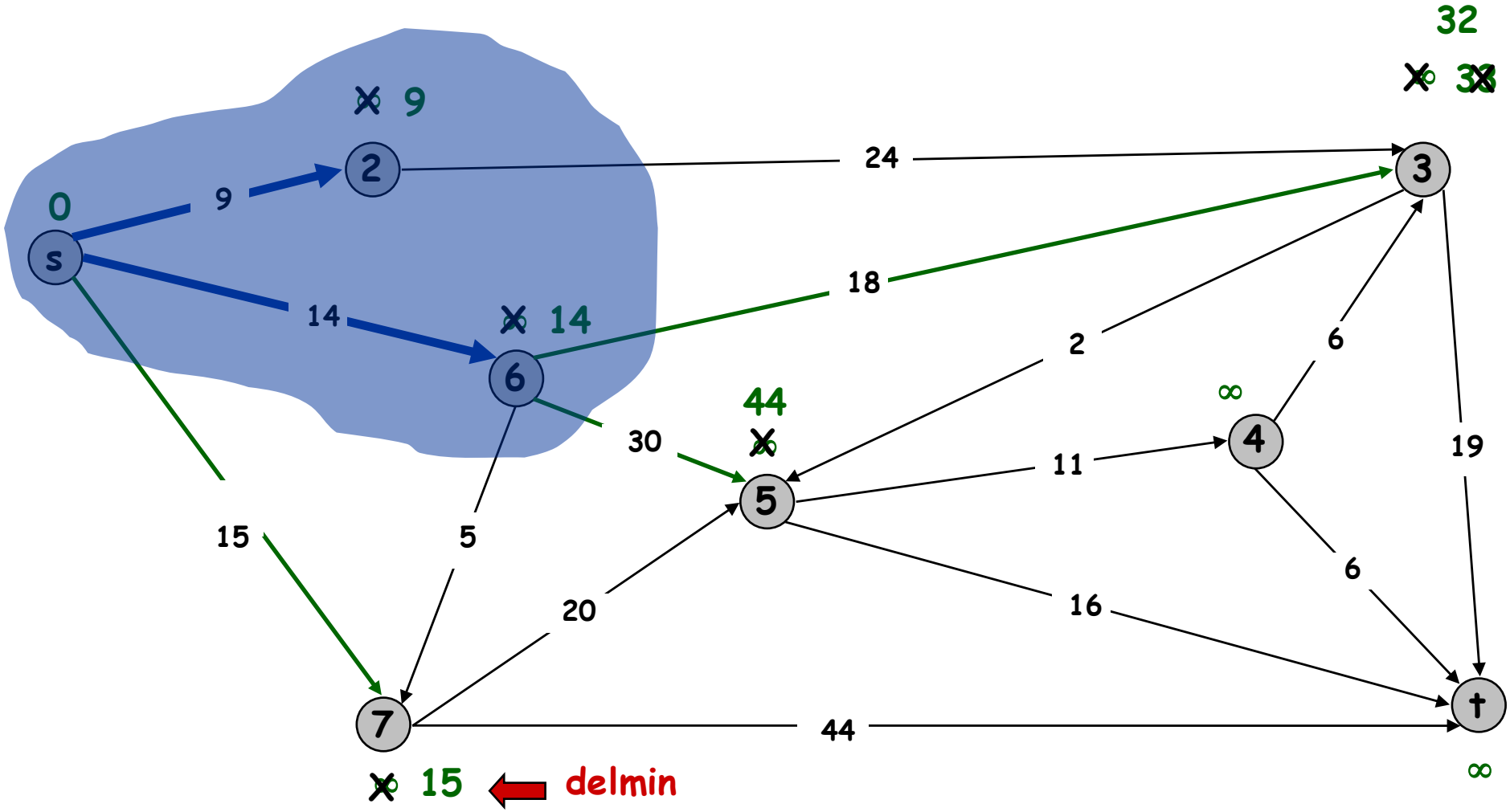
Dijkstra's Shortest Path Algorithm

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



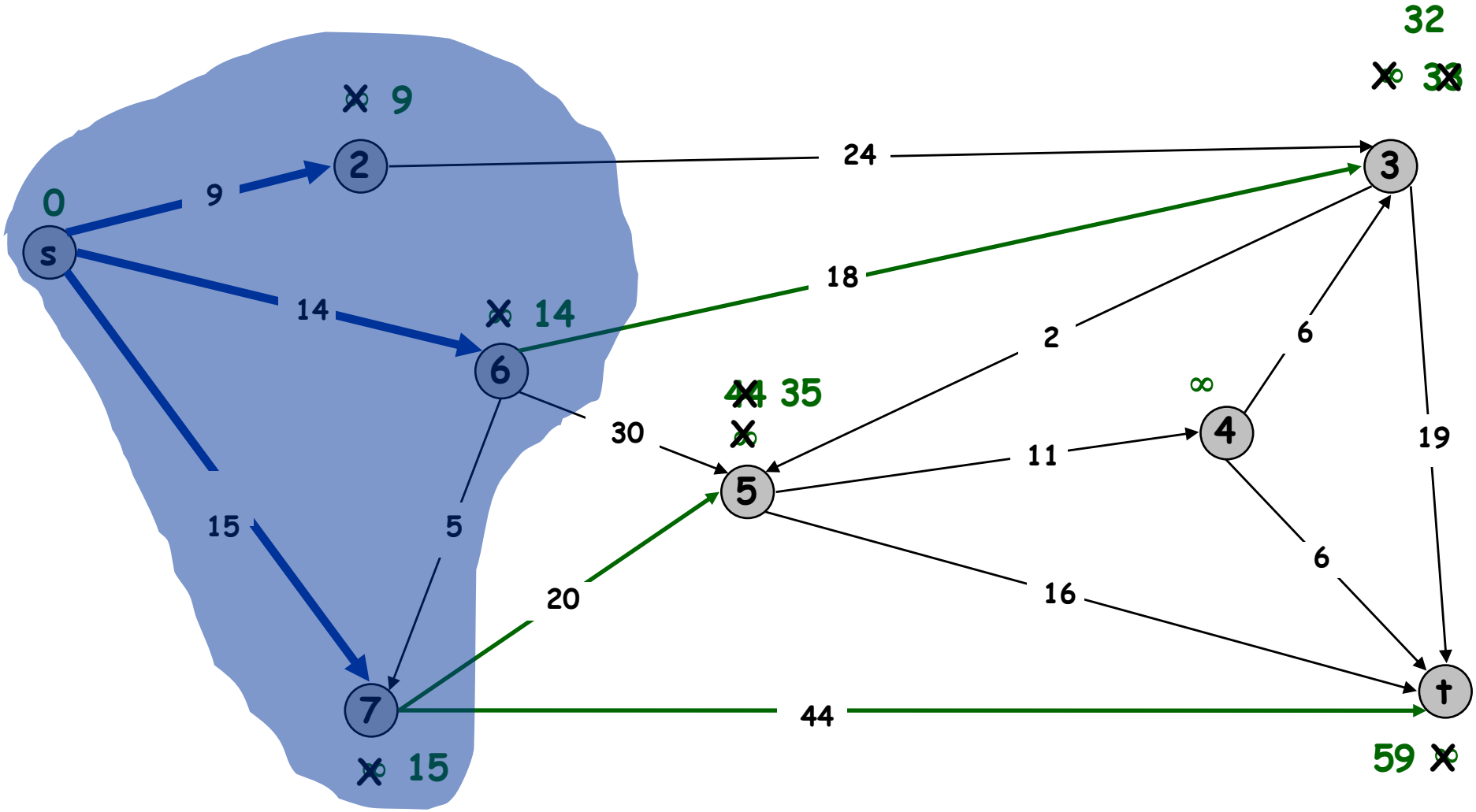
Dijkstra's Shortest Path Algorithm

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



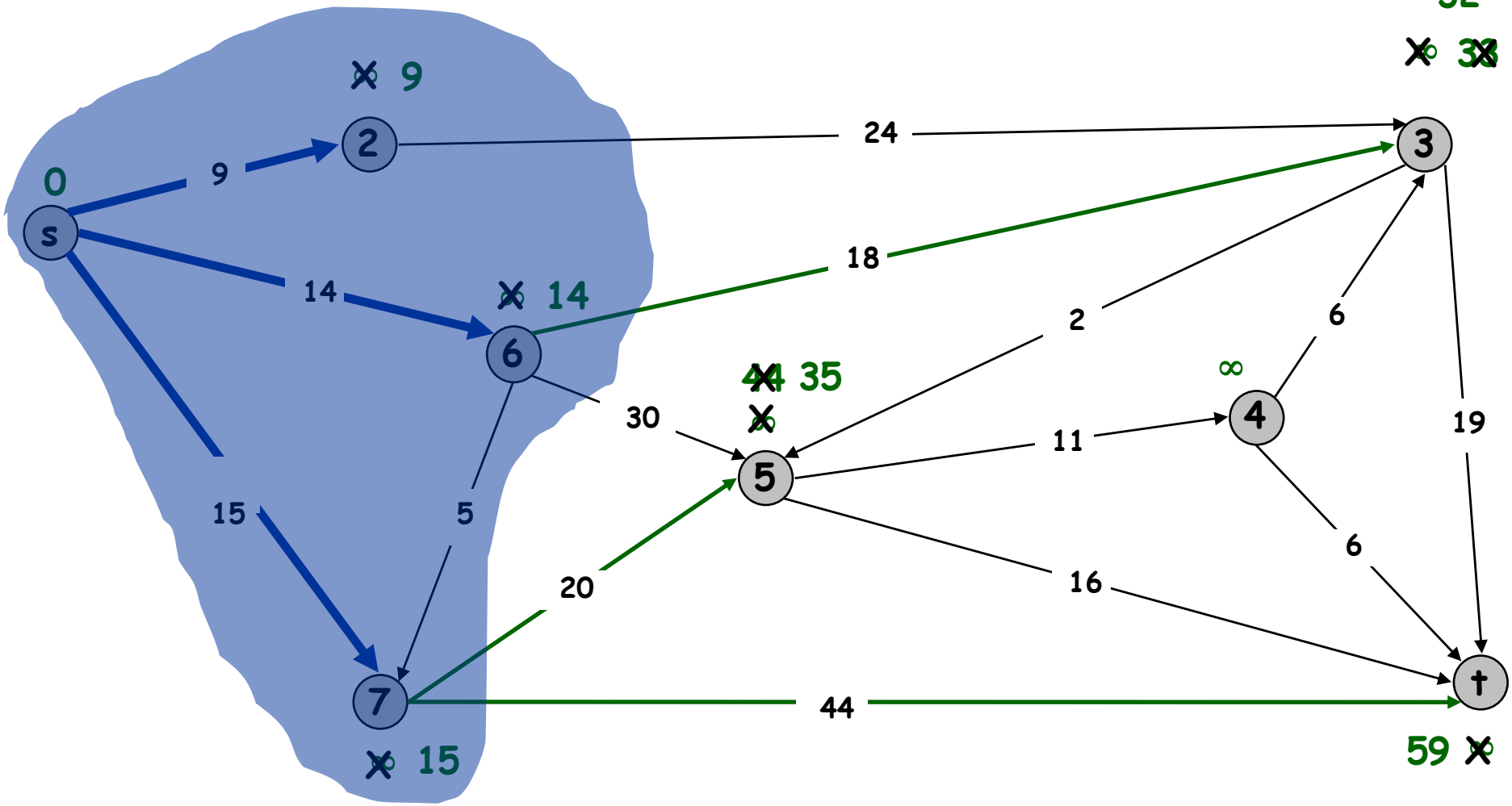
Dijkstra's Shortest Path Algorithm

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



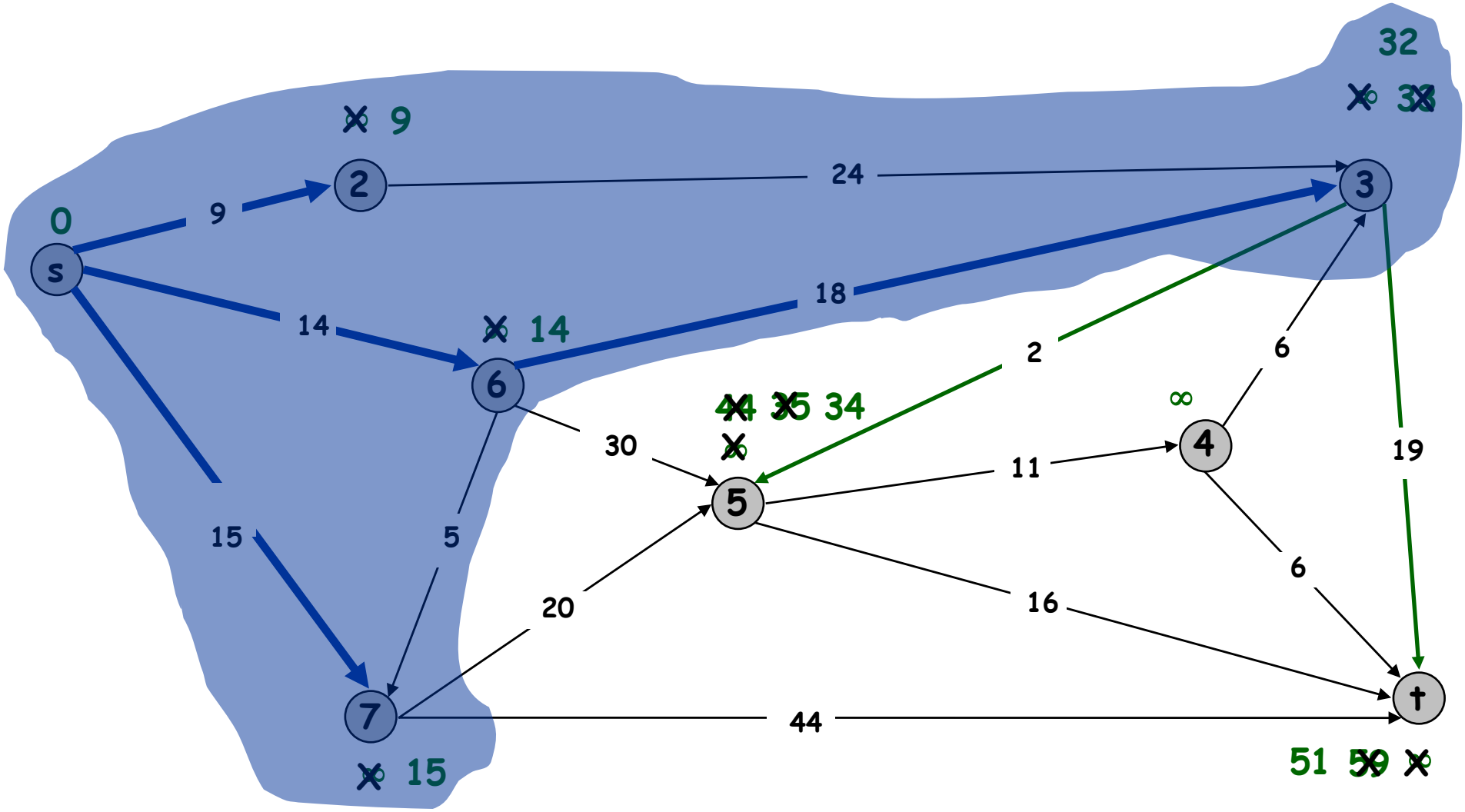
Dijkstra's Shortest Path Algorithm

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



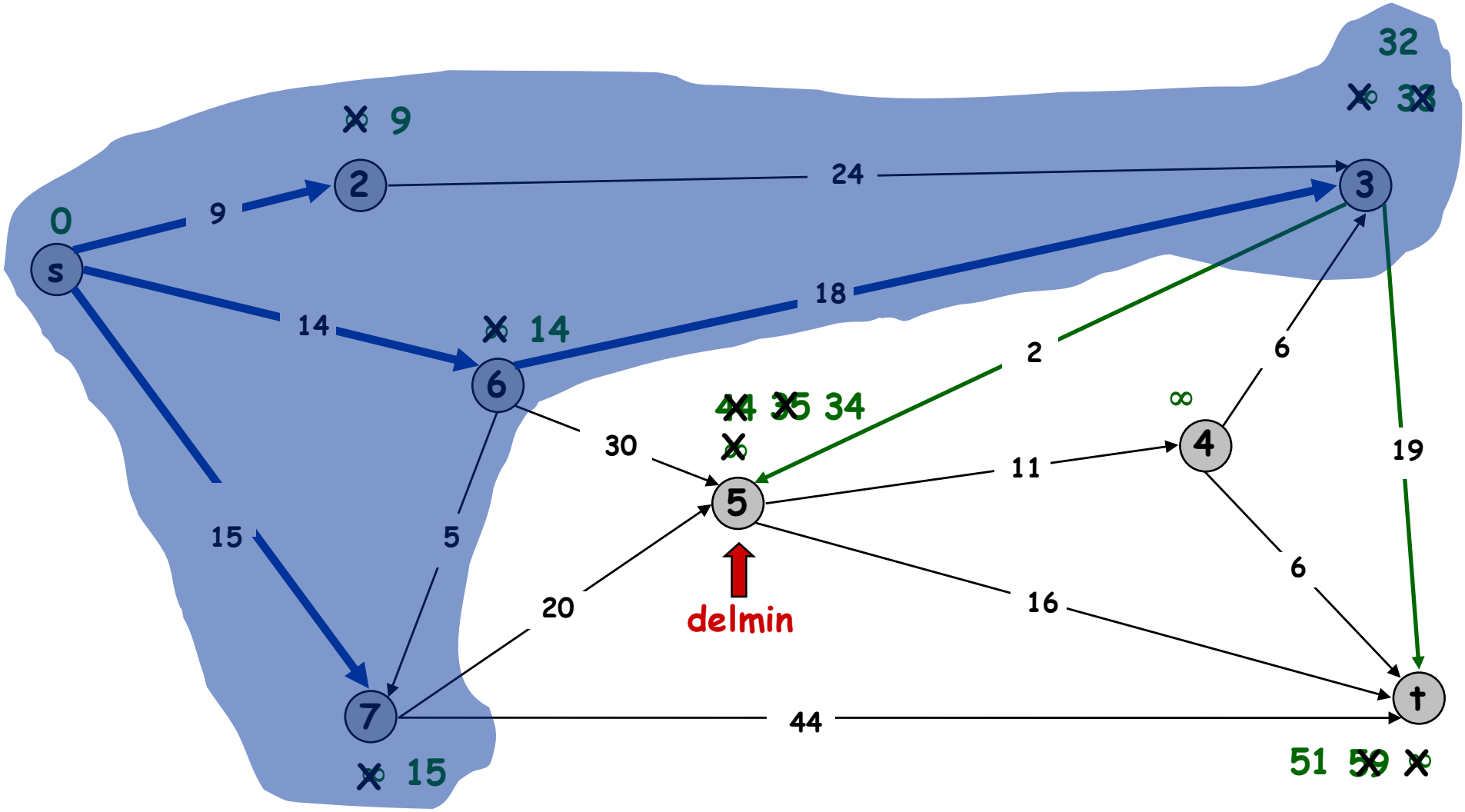
Dijkstra's Shortest Path Algorithm

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



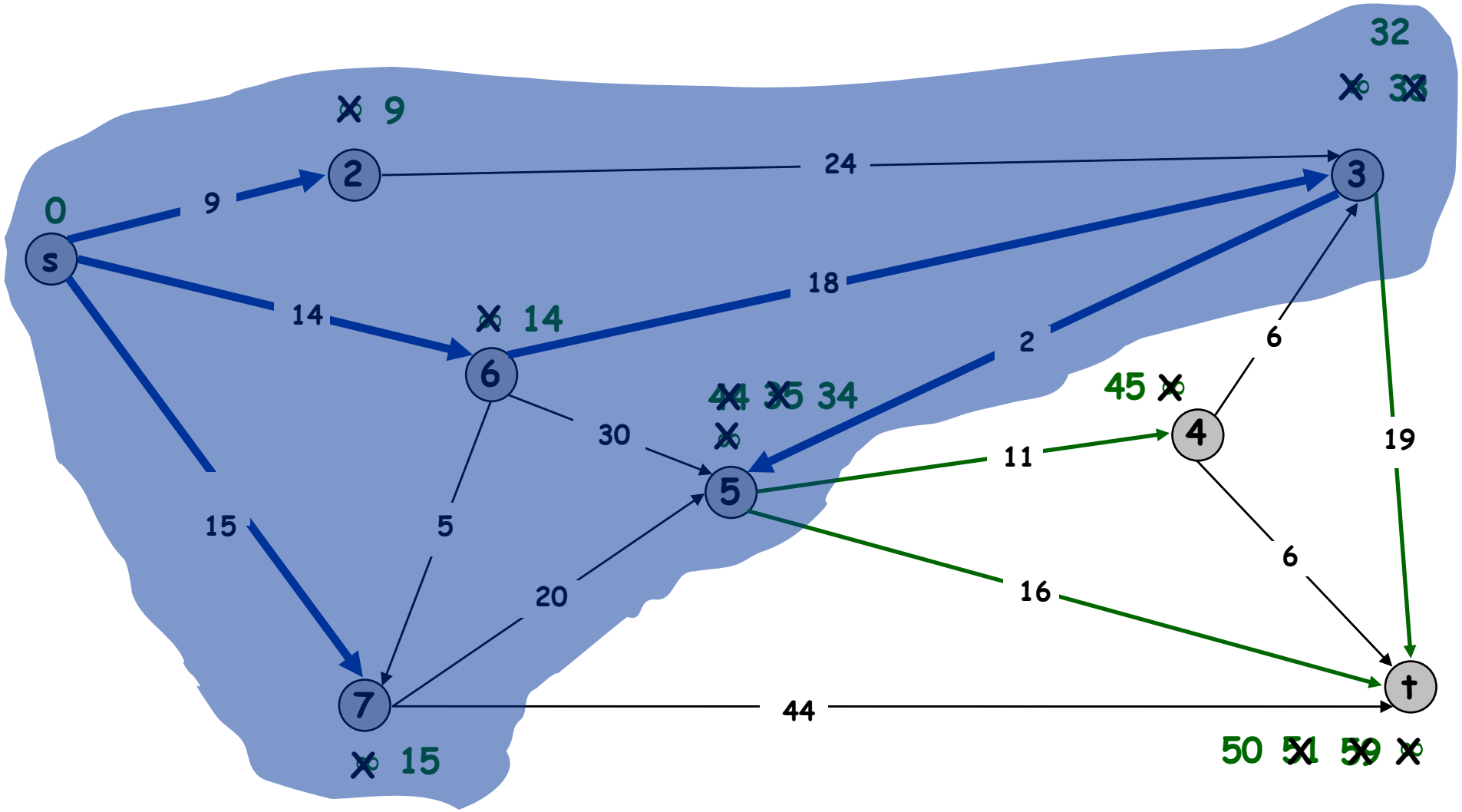
Dijkstra's Shortest Path Algorithm

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



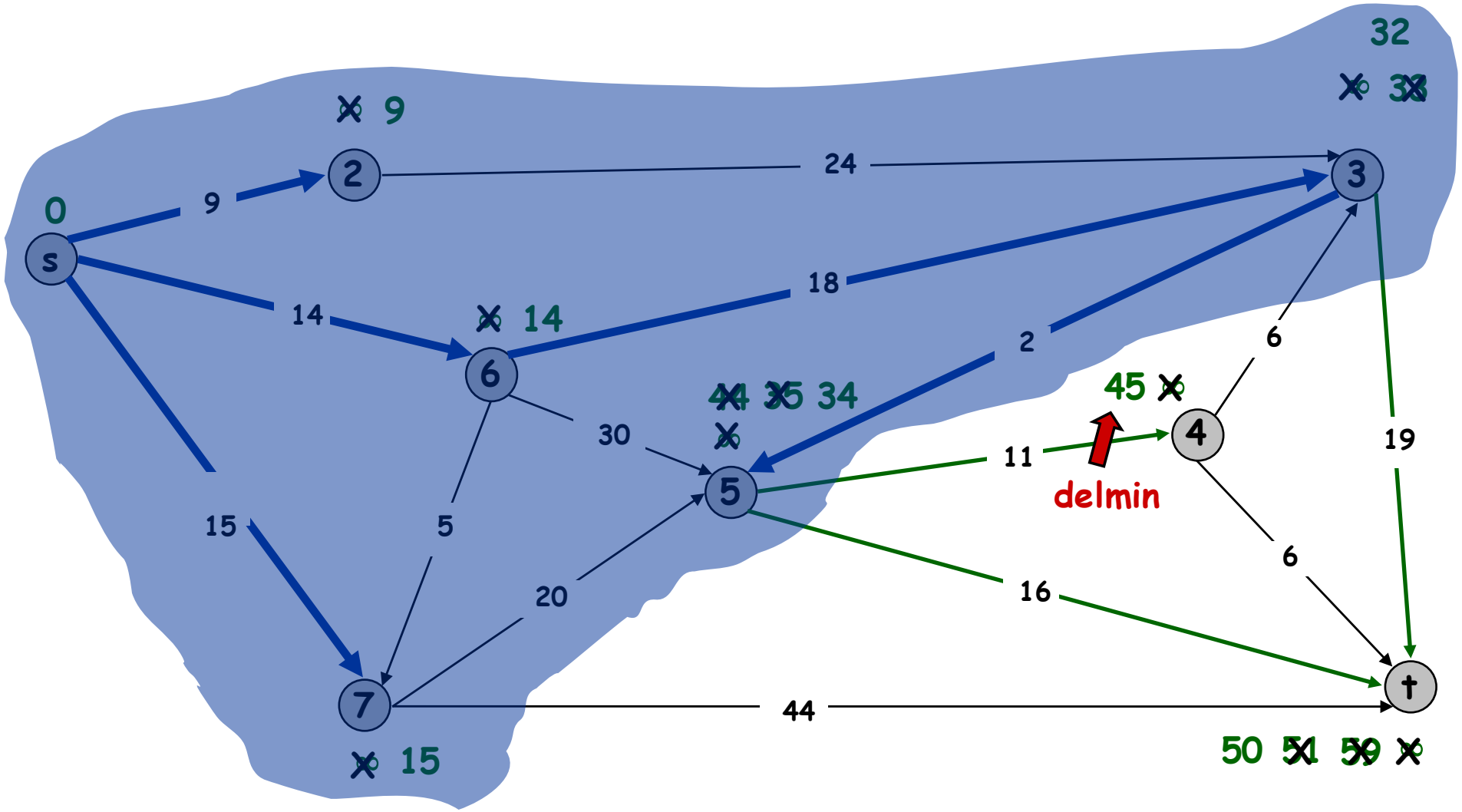
Dijkstra's Shortest Path Algorithm

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



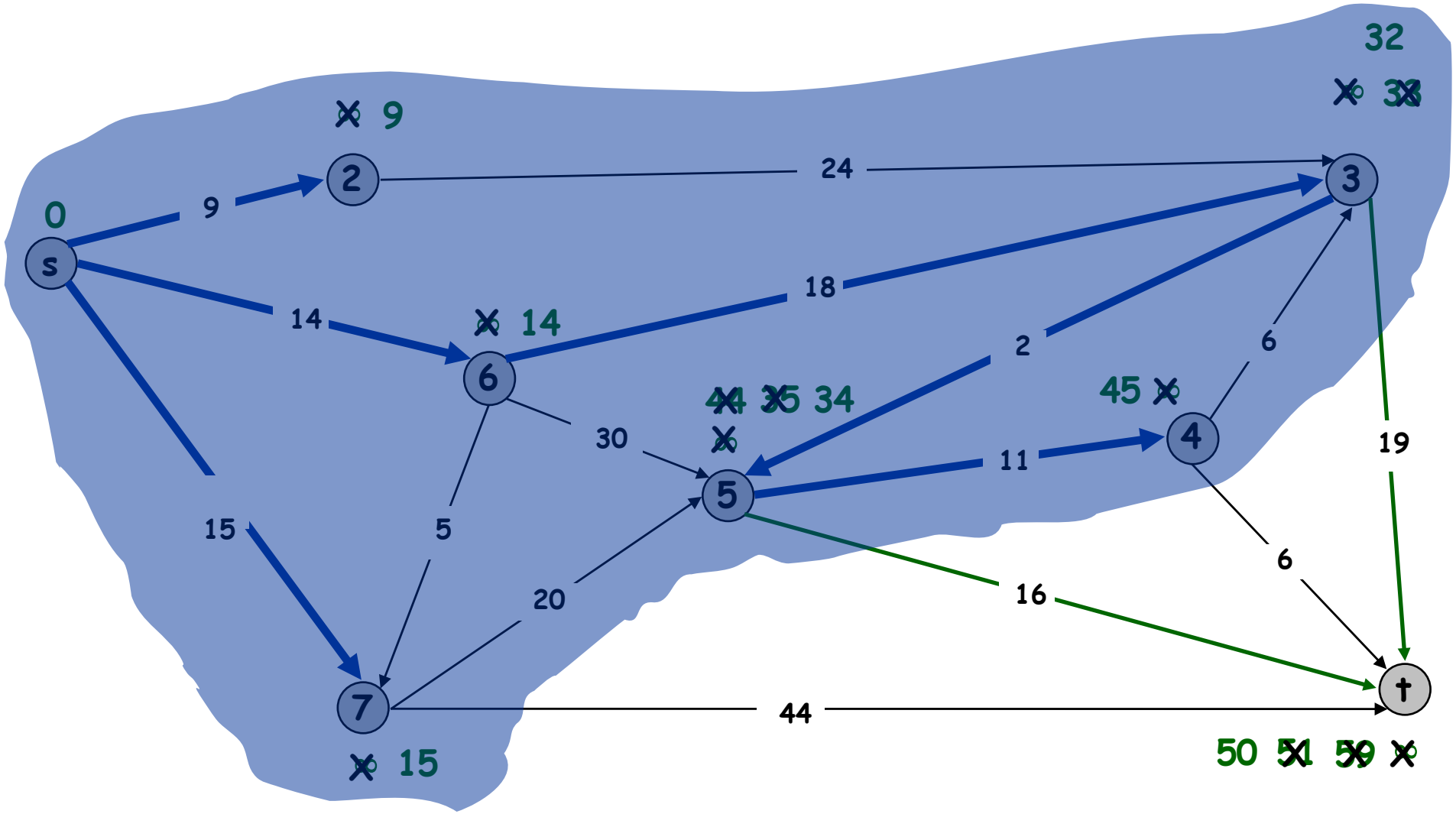
Dijkstra's Shortest Path Algorithm

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



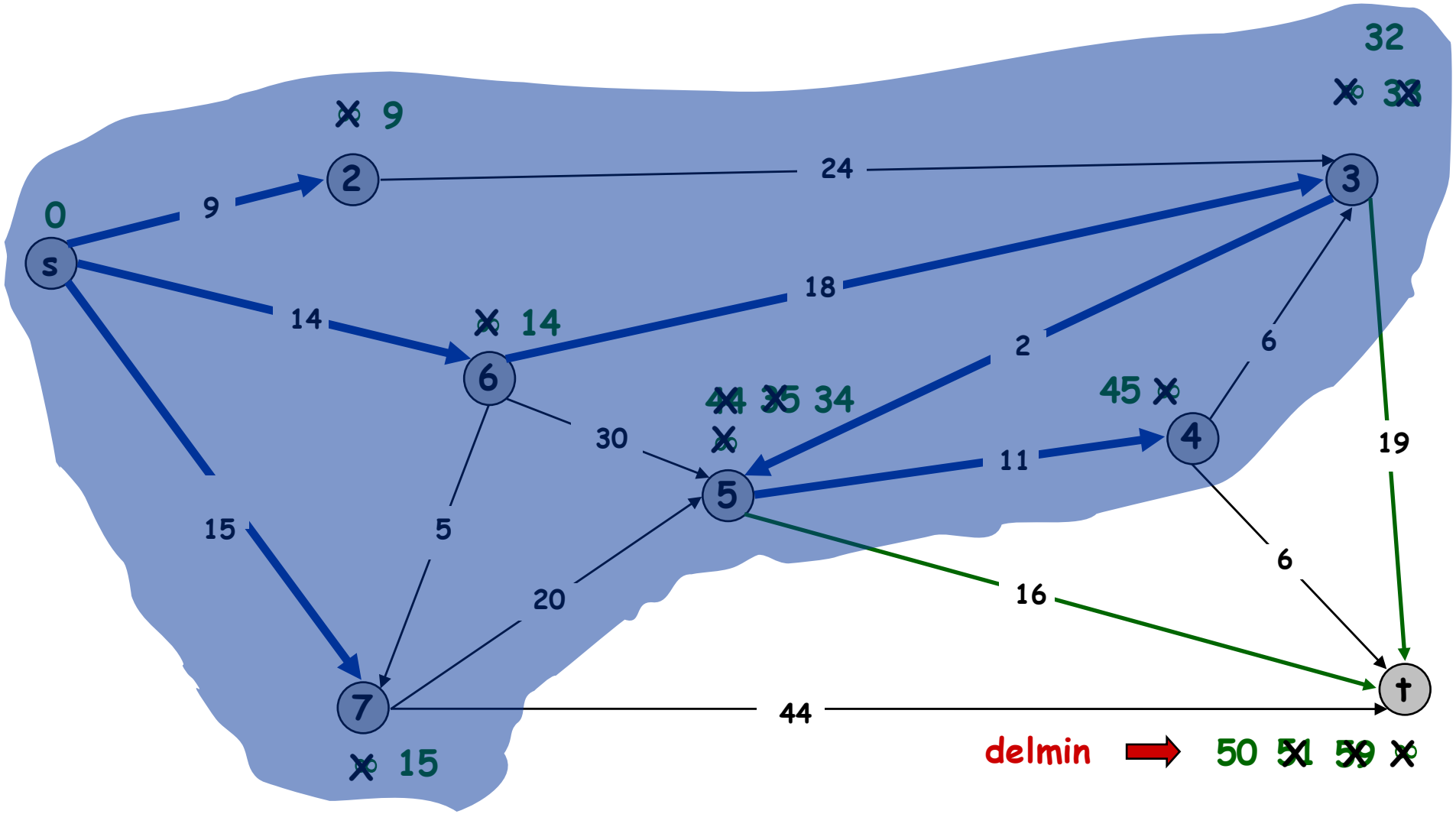
Dijkstra's Shortest Path Algorithm

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



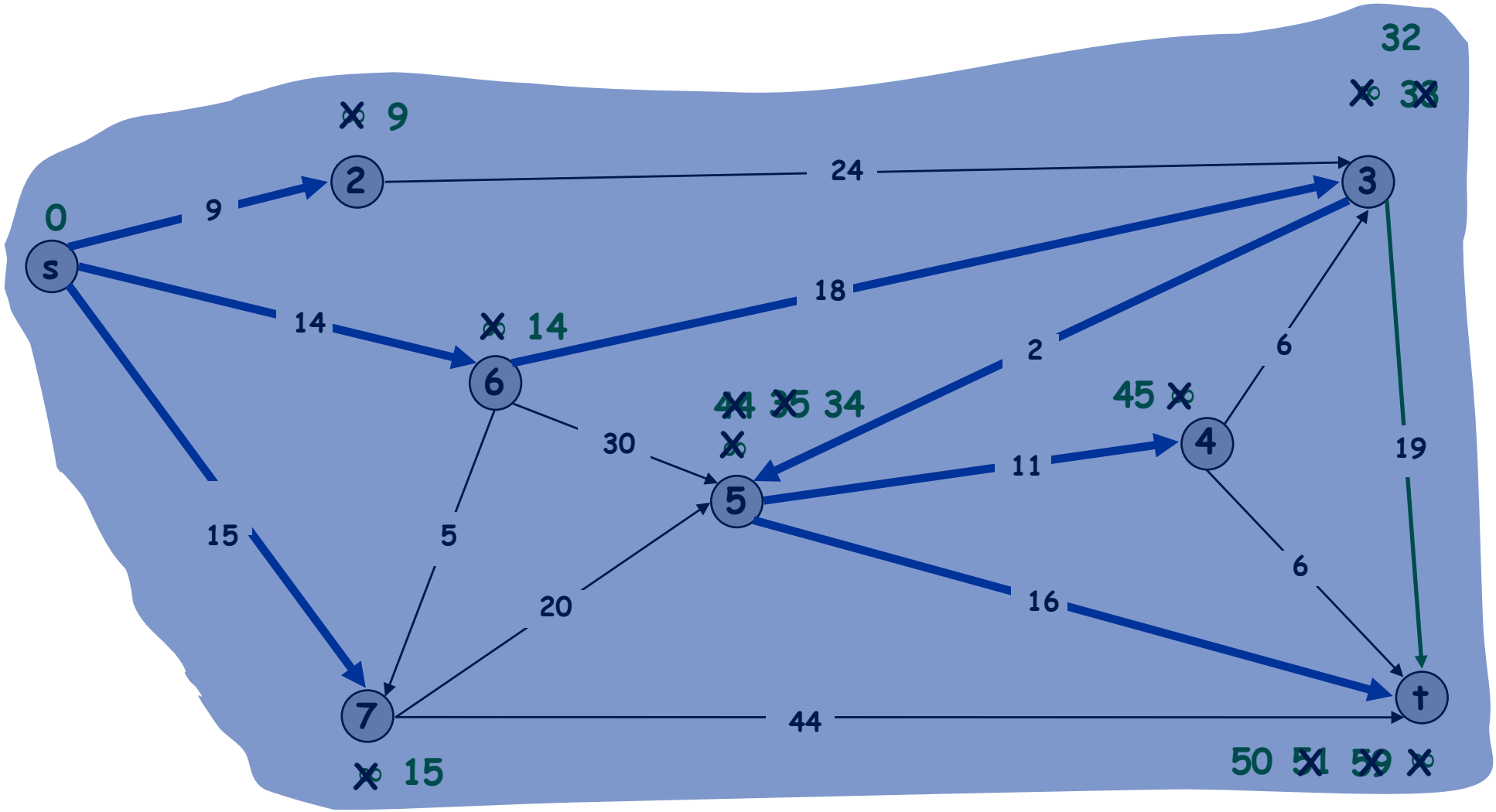
Dijkstra's Shortest Path Algorithm

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



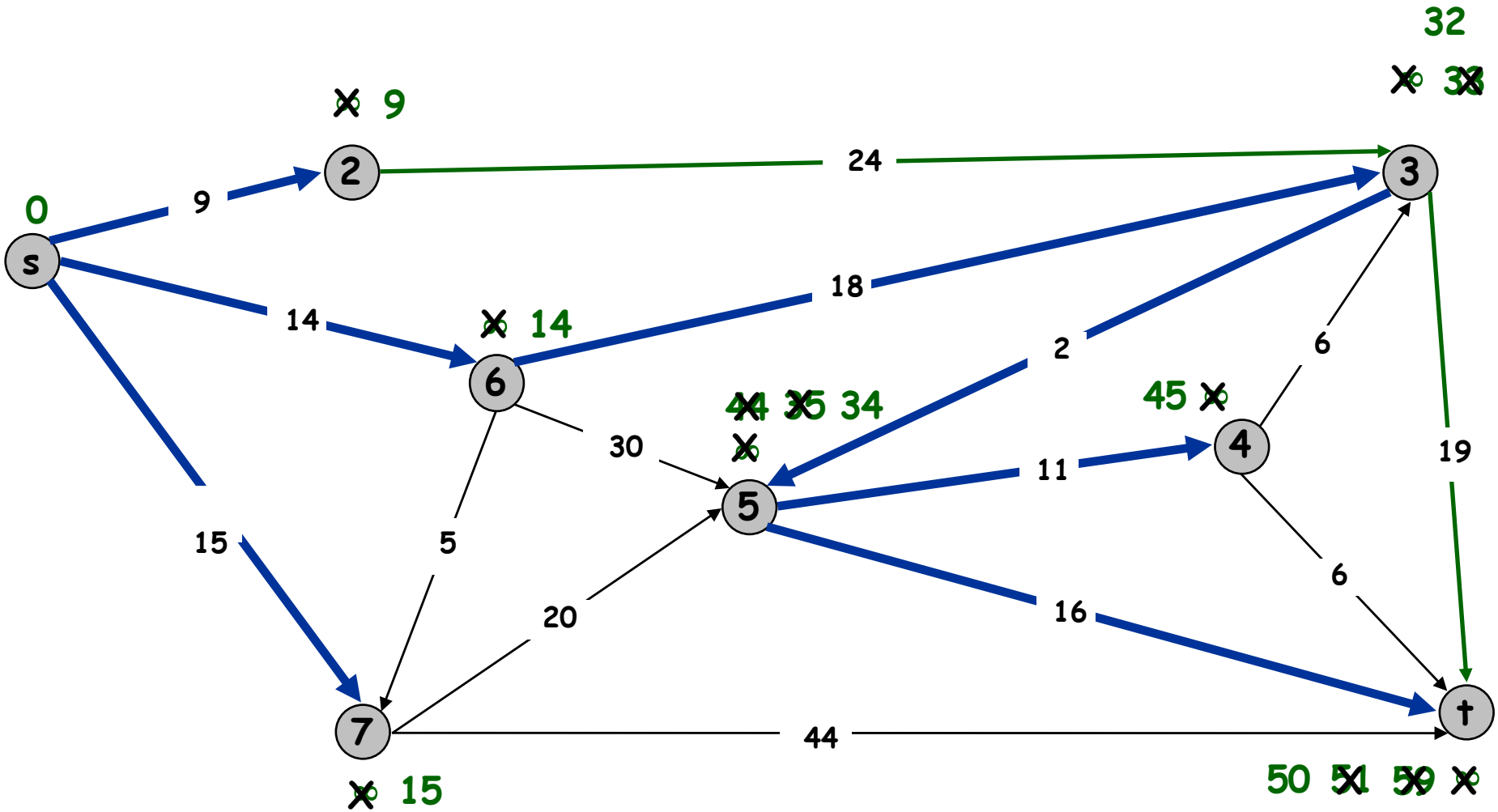
Dijkstra's Shortest Path Algorithm

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

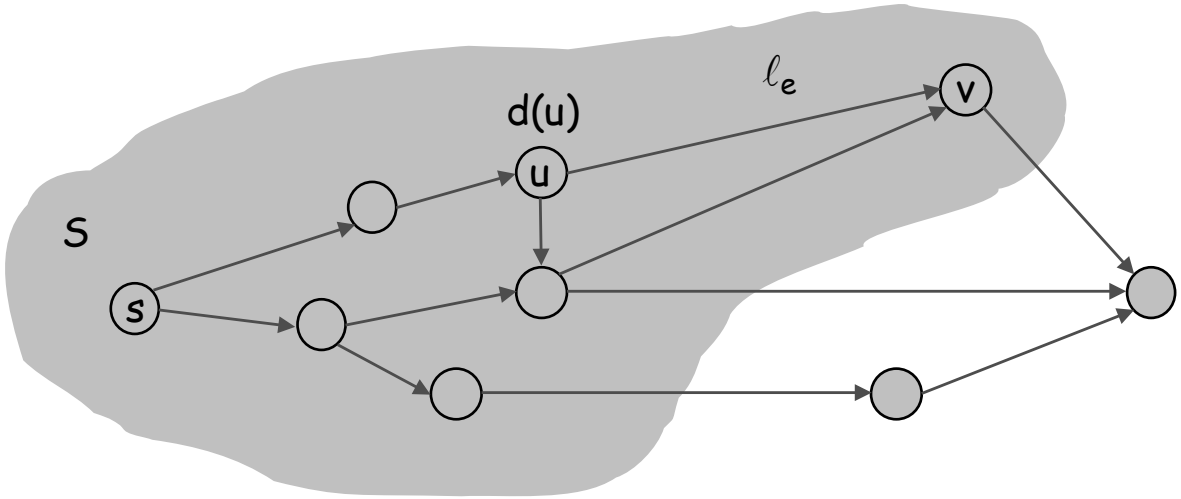
Dijkstra's algorithm.

- 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 greedily 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: Proof of Correctness (greedy stays ahead)

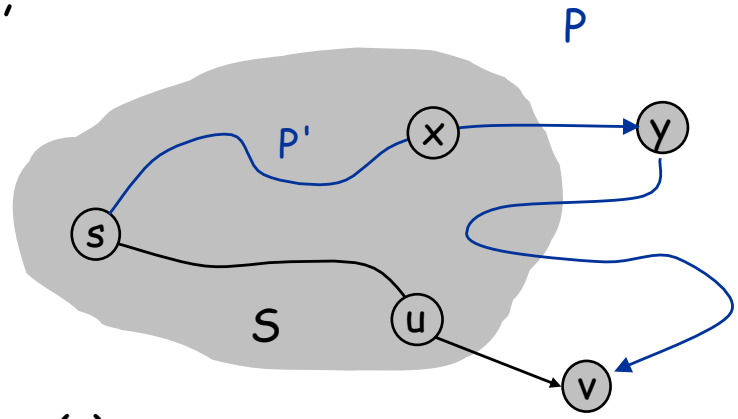
Invariant. For each node $u \in S$, $d(u)$ is the length of the shortest s-u path.

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.



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

- $\geq$: nonnegative weights
- $\geq$: inductive hypothesis
- $\geq$: defn of $\pi(y)$
- $\geq$: Dijkstra chose v instead of y