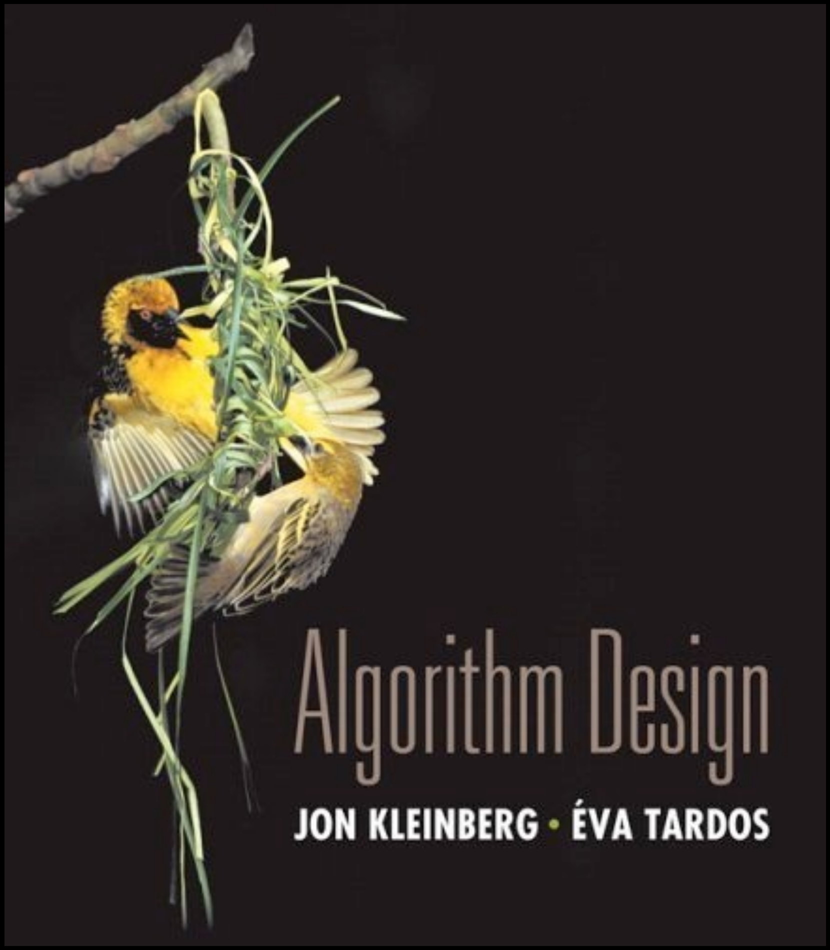


Chapter 4

Greedy Algorithms

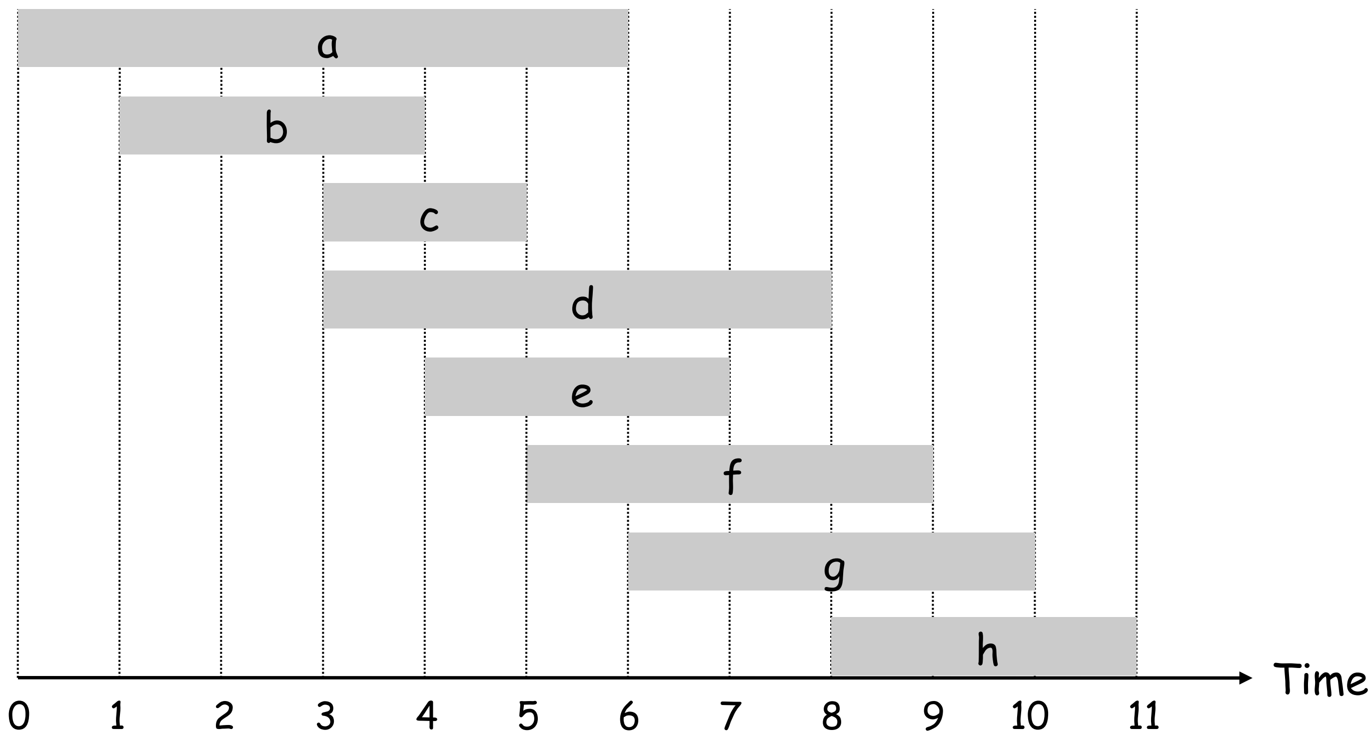
Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.

4.1 Interval Scheduling

Interval Scheduling

Interval scheduling.

- Job j starts at s_j and finishes at f_j.
- Two jobs compatible if they don't overlap.
- Goal: find maximum subset of mutually compatible jobs.



Interval Scheduling: Greedy Algorithms

Greedy template. Consider jobs in some order. Take each job provided it's compatible with the ones already taken.

- [Earliest start time] Consider jobs in ascending order of start time s_j.
- [Earliest finish time] Consider jobs in ascending order of finish time f_j.
- [Shortest interval] Consider jobs in ascending order of interval length $f_j - s_j$.
- [Fewest conflicts] For each job, count the number of conflicting jobs c_j. Schedule in ascending order of conflicts c_j.

Interval Scheduling: Greedy Algorithms

Greedy template. Consider jobs in some order. Take each job provided it's compatible with the ones already taken.

breaks earliest start time

breaks shortest interval

breaks fewest conflicts

Interval Scheduling: Greedy Algorithm

Greedy algorithm. Consider jobs in increasing order of finish time. Take each job provided it's compatible with the ones already taken.

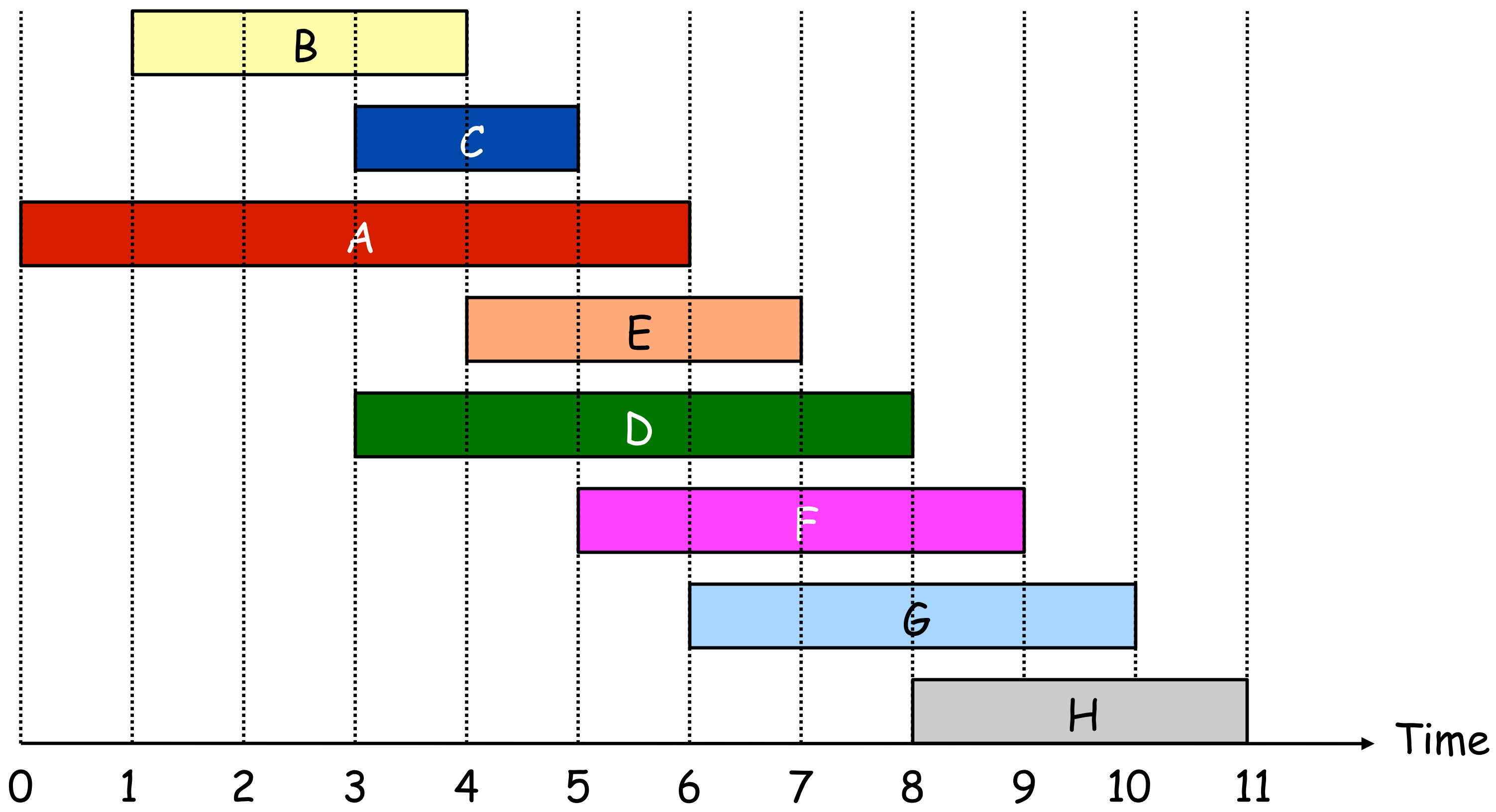
jobs selected

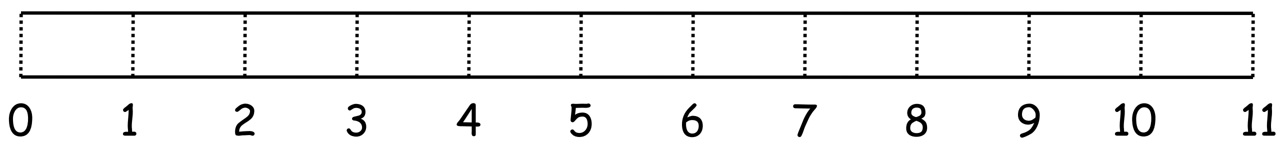
```
Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn.

A ← ∅
for j = 1 to n {
   if (job j compatible with A)
      A ← A ∪ {j}
}
return A
```

Implementation. O(n log n).

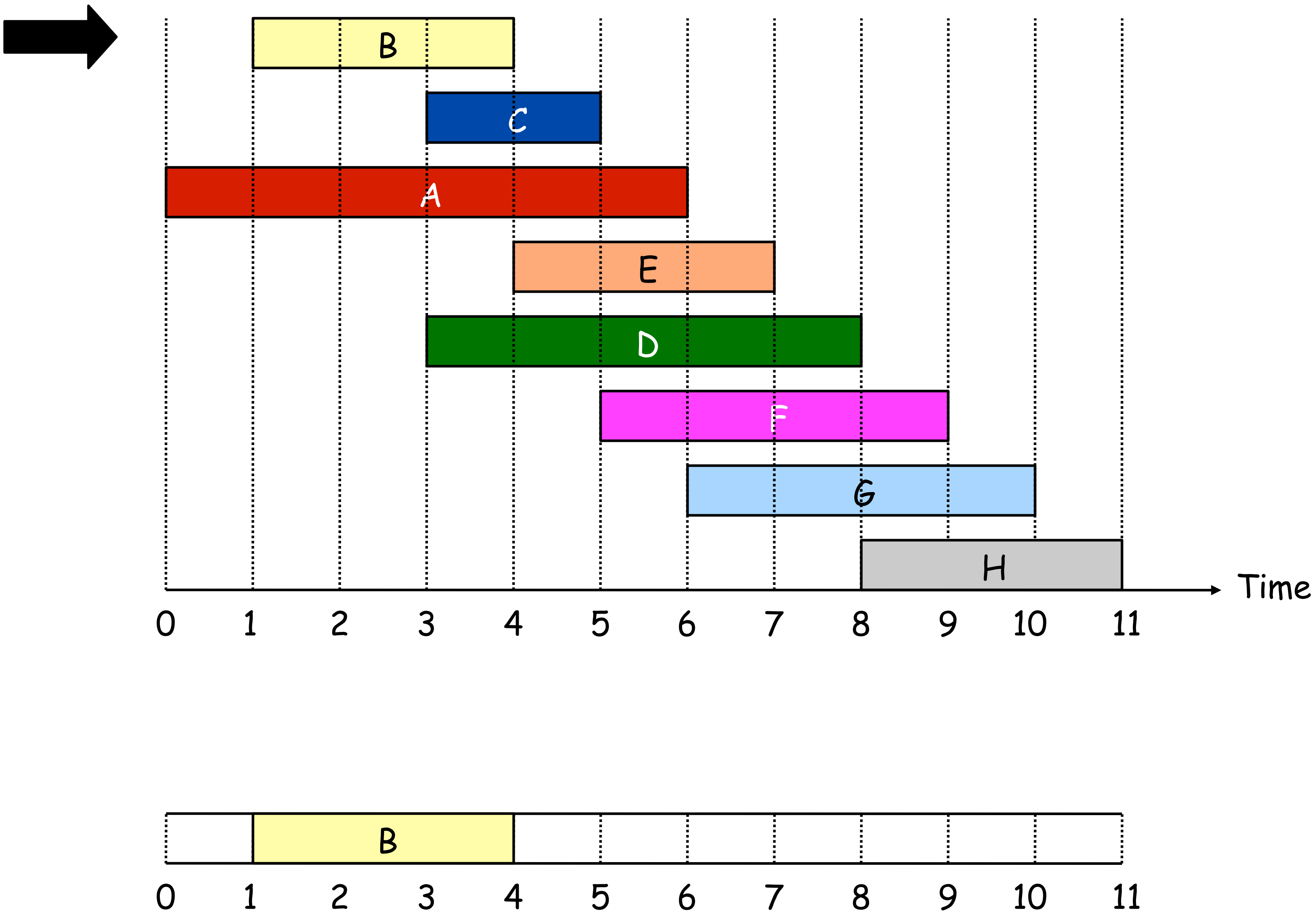
- Remember job j* that was added last to A.
- Job j is compatible with A if $s_j \geq f_{j^*}$.

Interval Scheduling

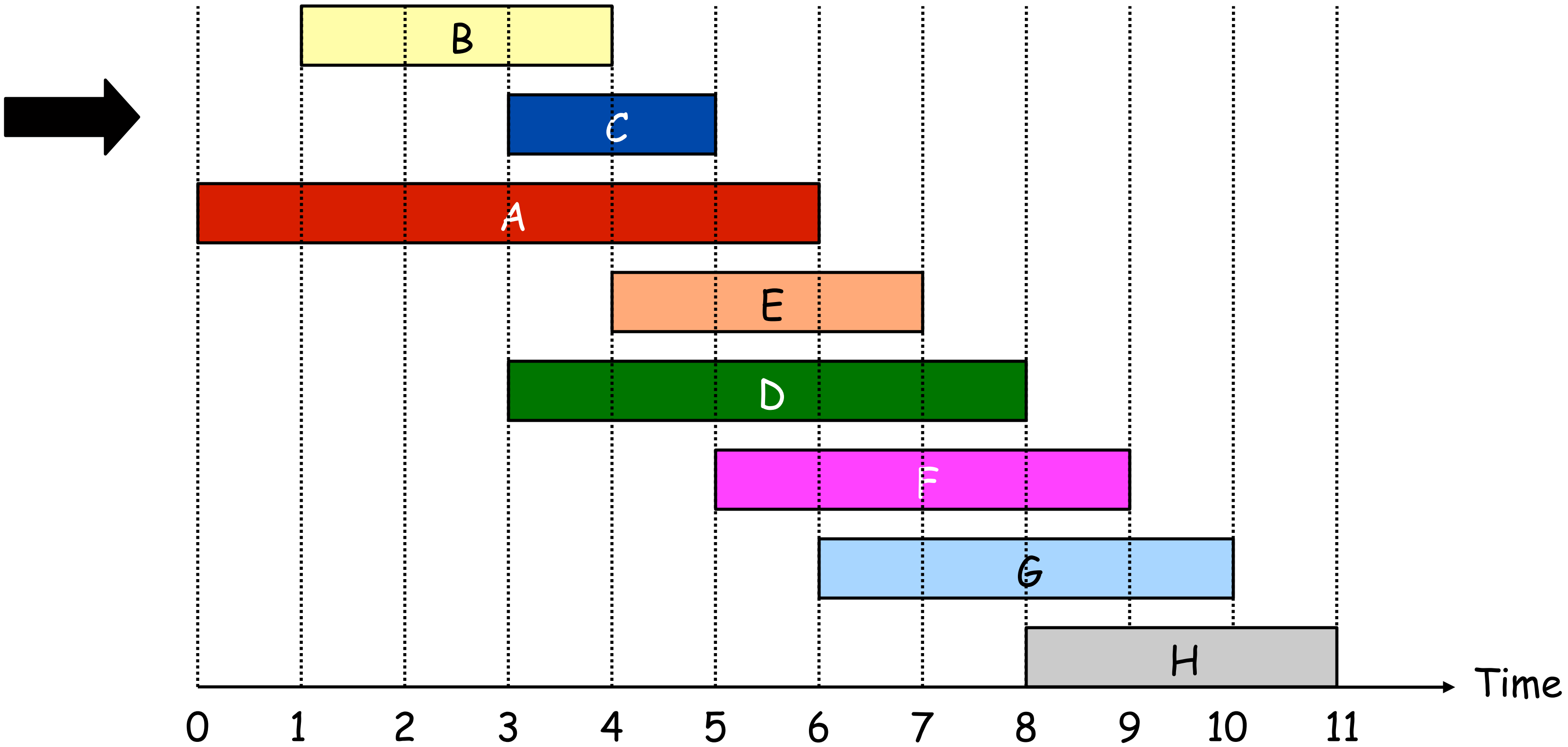


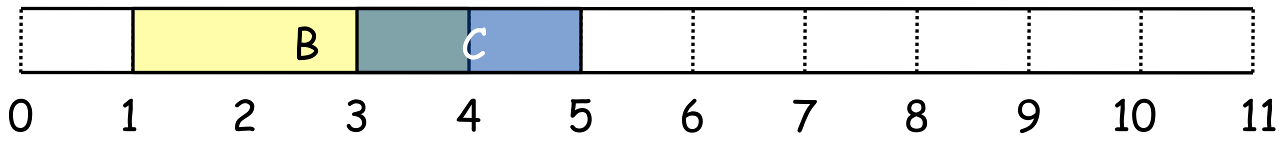


Interval Scheduling

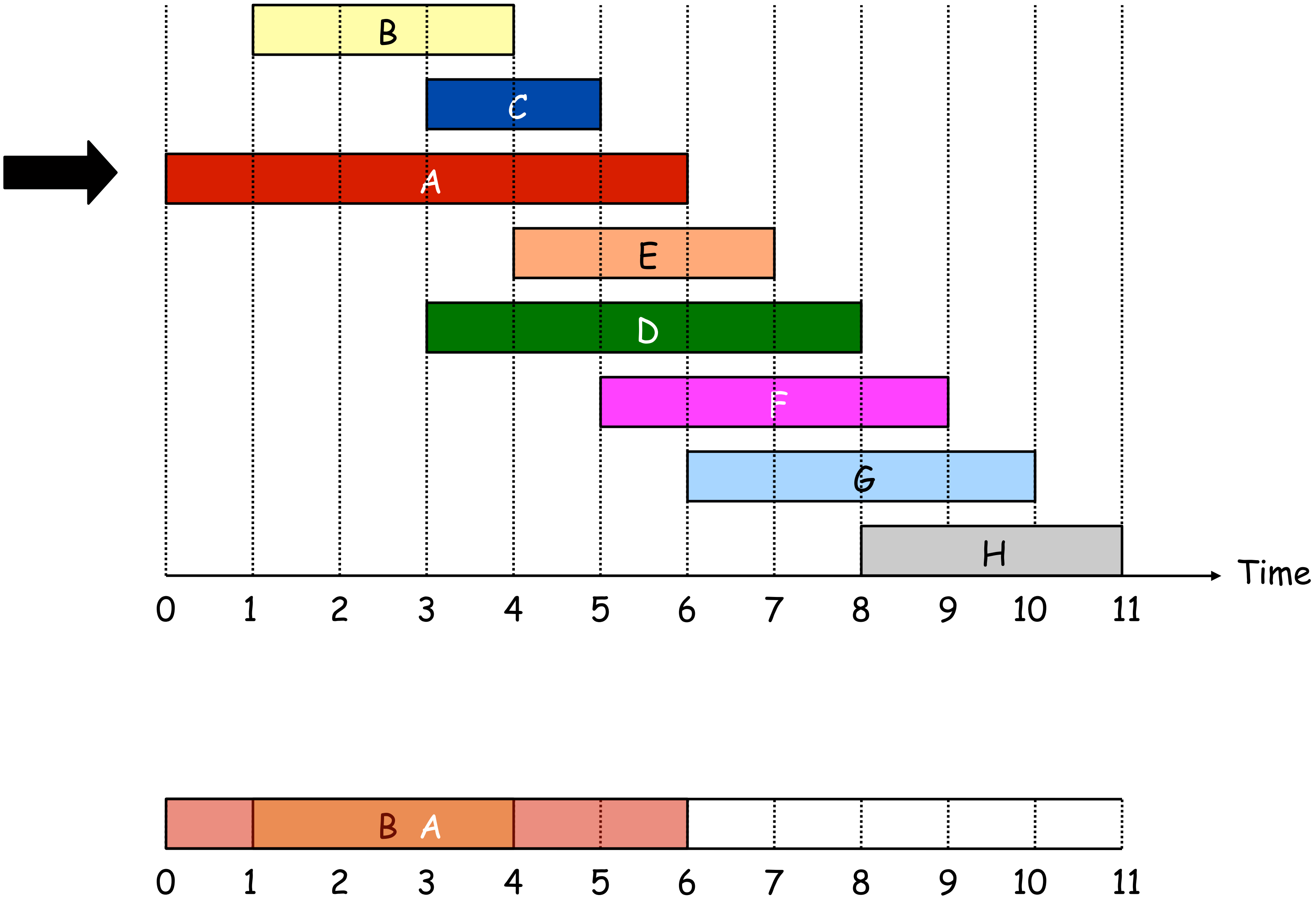


Interval Scheduling

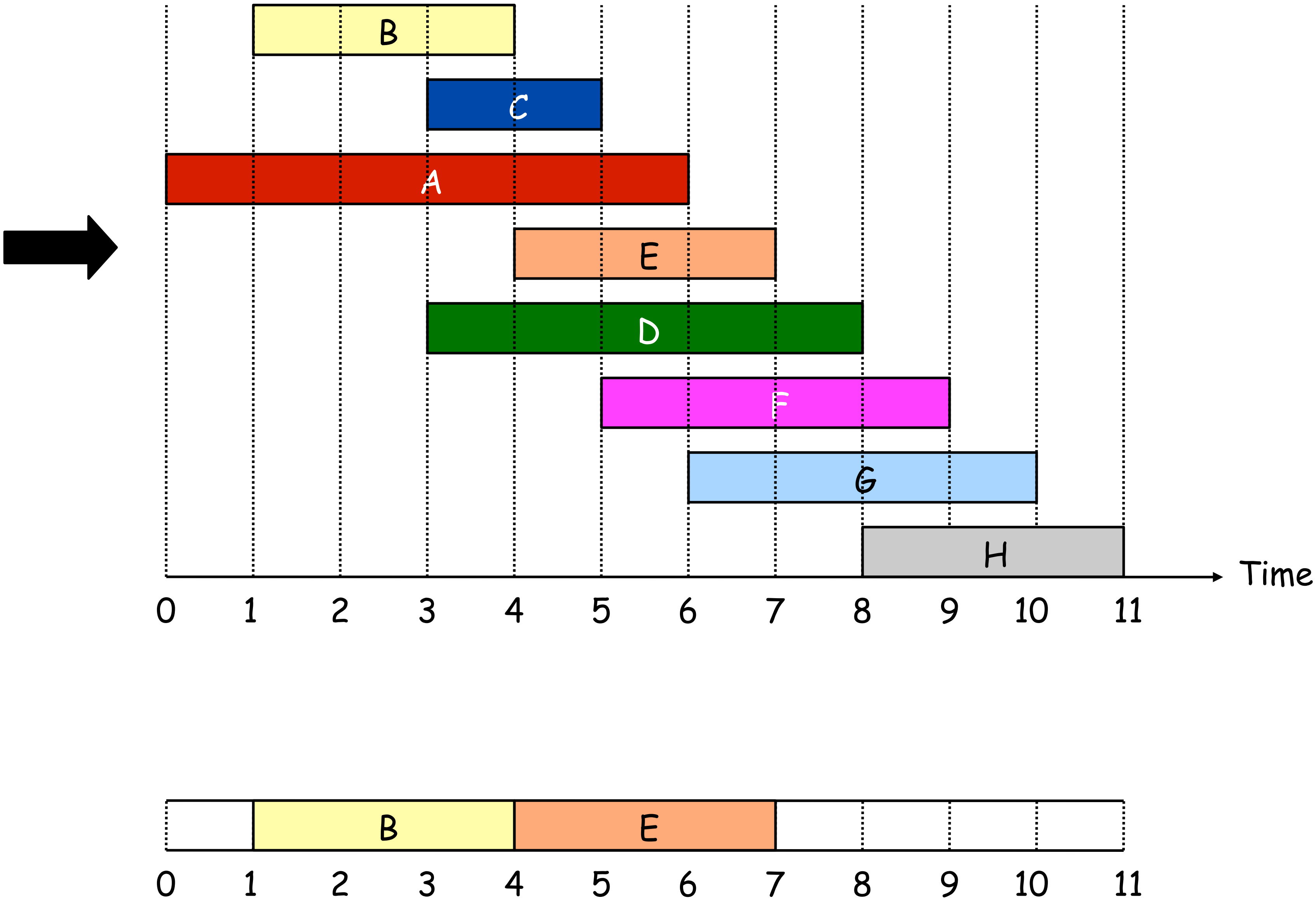




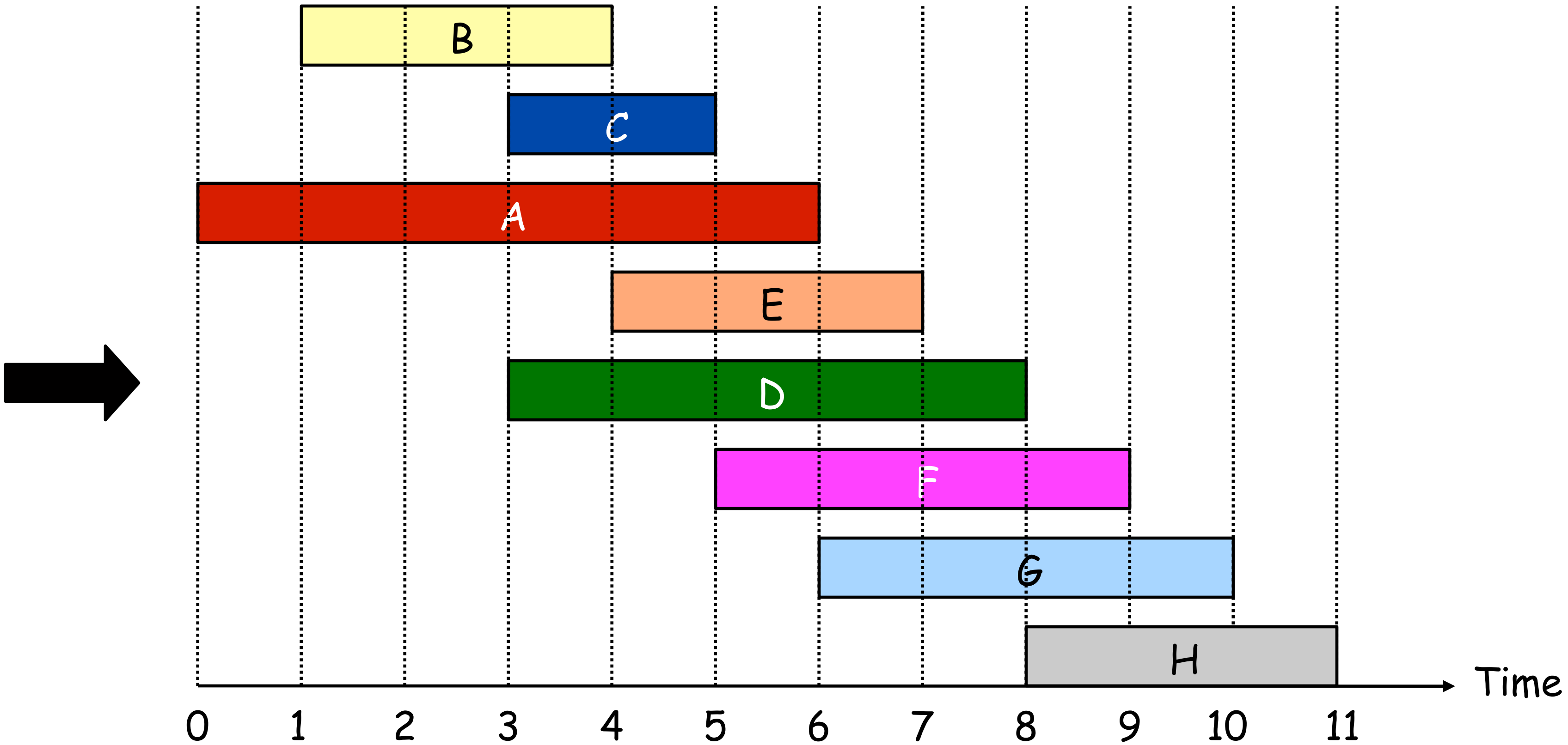
Interval Scheduling

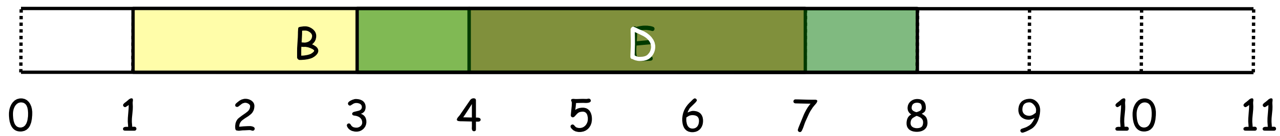


Interval Scheduling

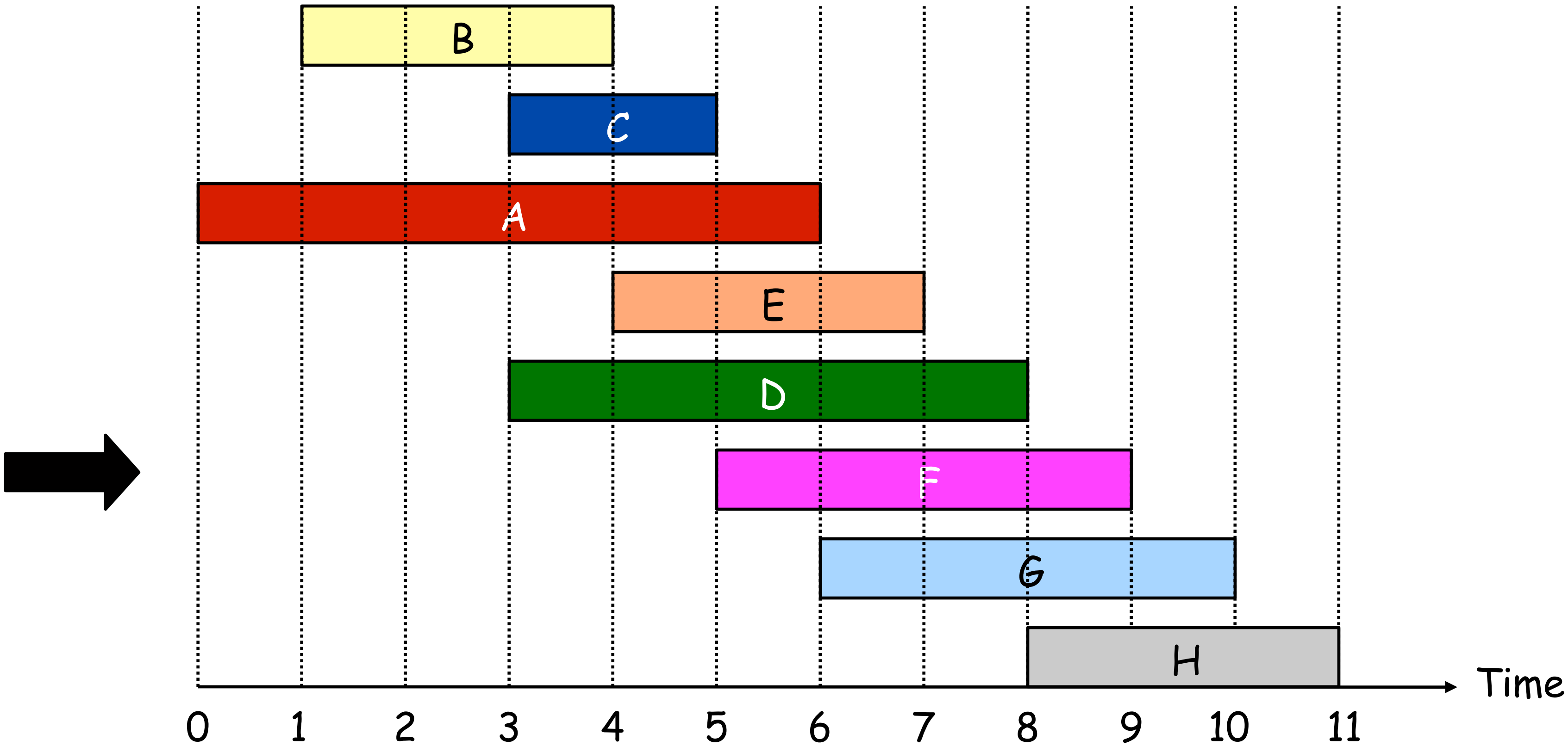


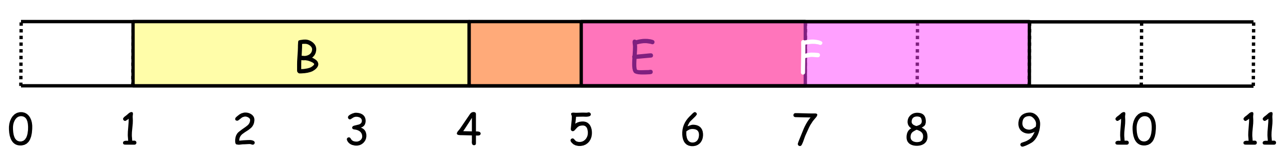
Interval Scheduling



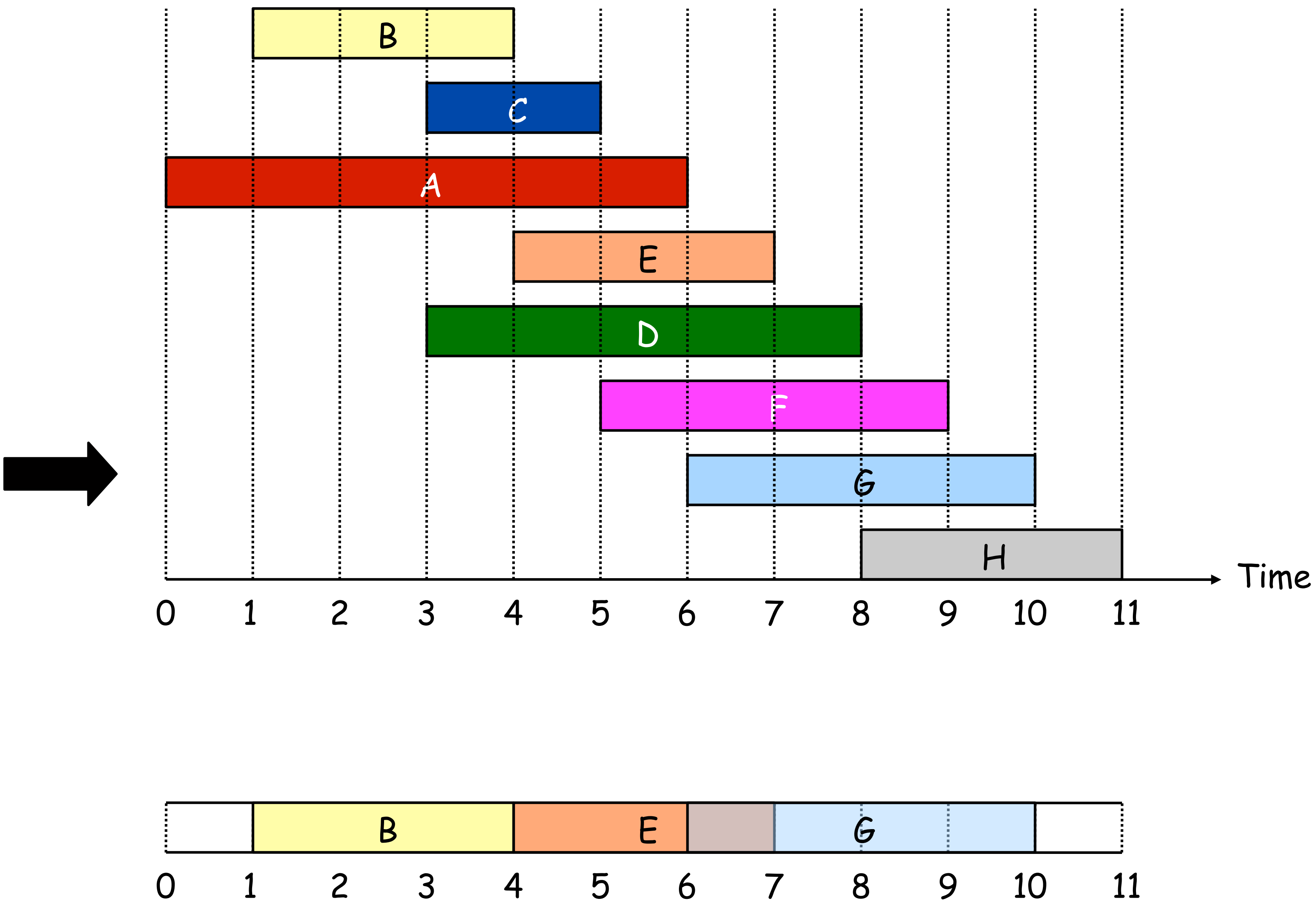


Interval Scheduling

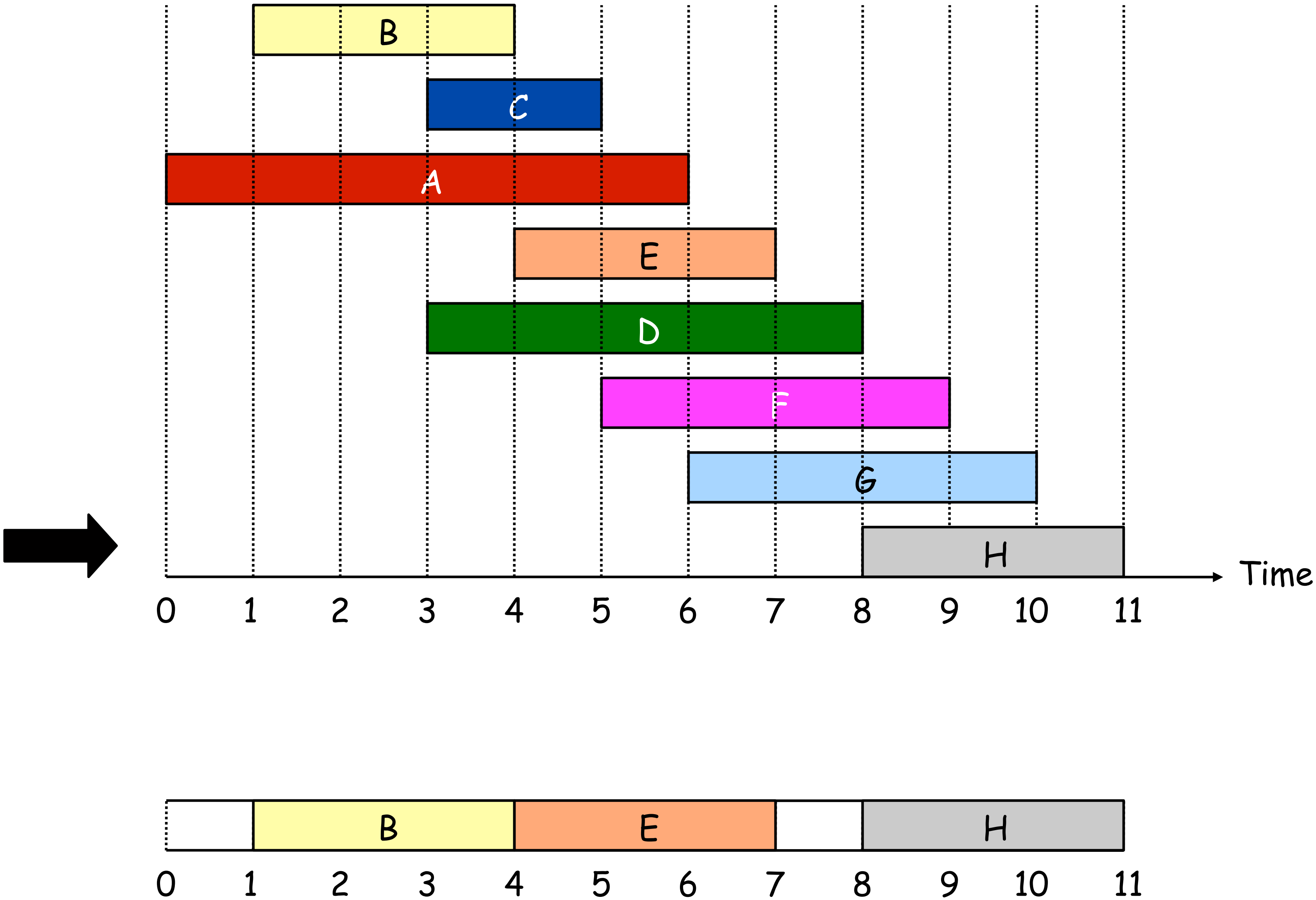




Interval Scheduling



Interval Scheduling

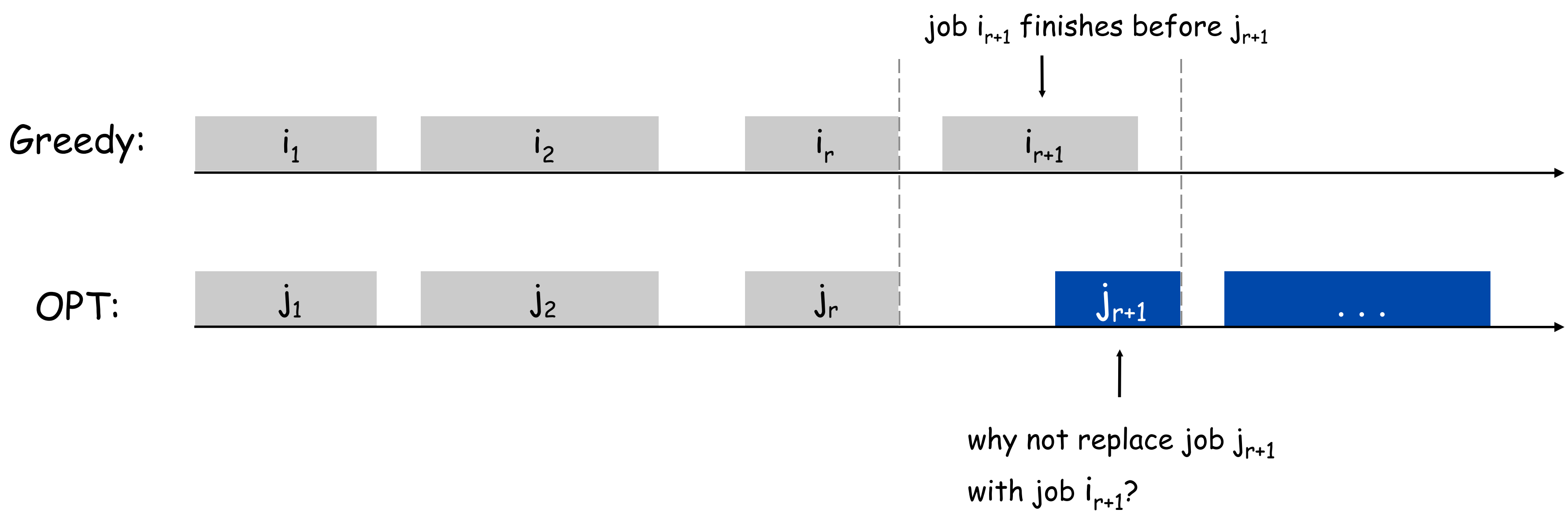


Interval Scheduling: Analysis

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let $i_1, i_2, \ldots i_k$ denote a set of jobs selected by greedy.
- Let $j_1, j_2, \ldots j_m$ denote a set of jobs in an optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of r.

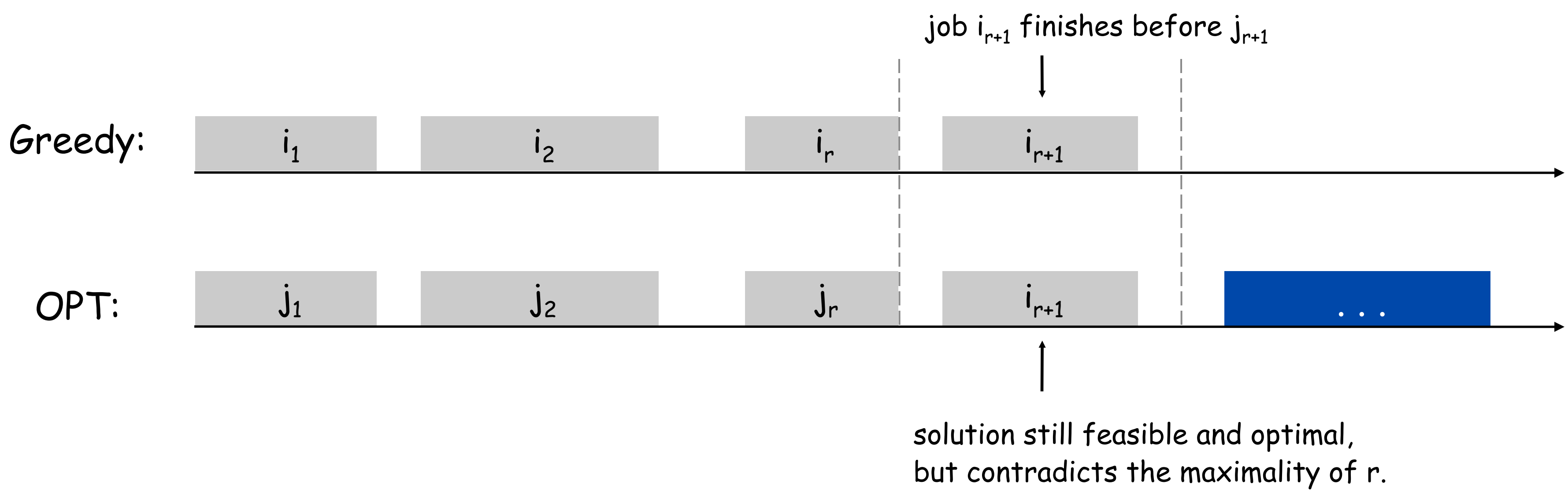


Interval Scheduling: Analysis

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let $i_1, i_2, \ldots i_k$ denote a set of jobs selected by greedy.
- Let $j_1, j_2, \ldots j_m$ denote a set of jobs in an optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of r.



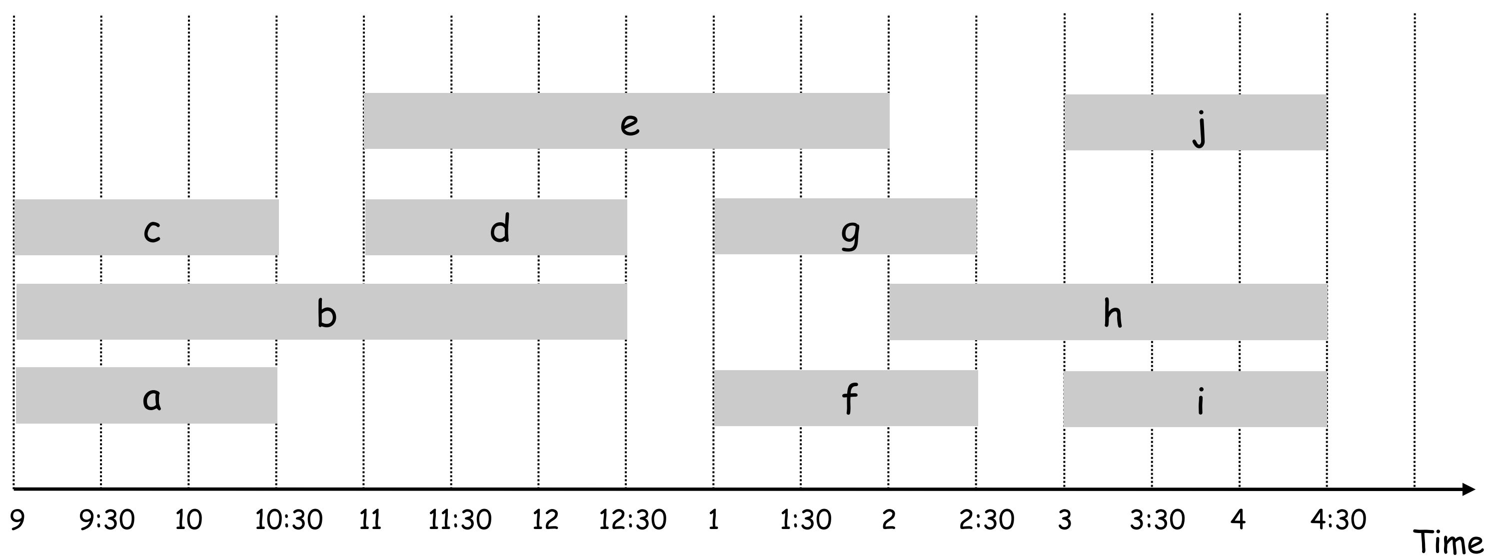
4.1 Interval Partitioning

Interval Partitioning

Interval partitioning.

- Lecture j starts at s_j and finishes at f_j.
- Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

Ex: This schedule uses 4 classrooms to schedule 10 lectures.

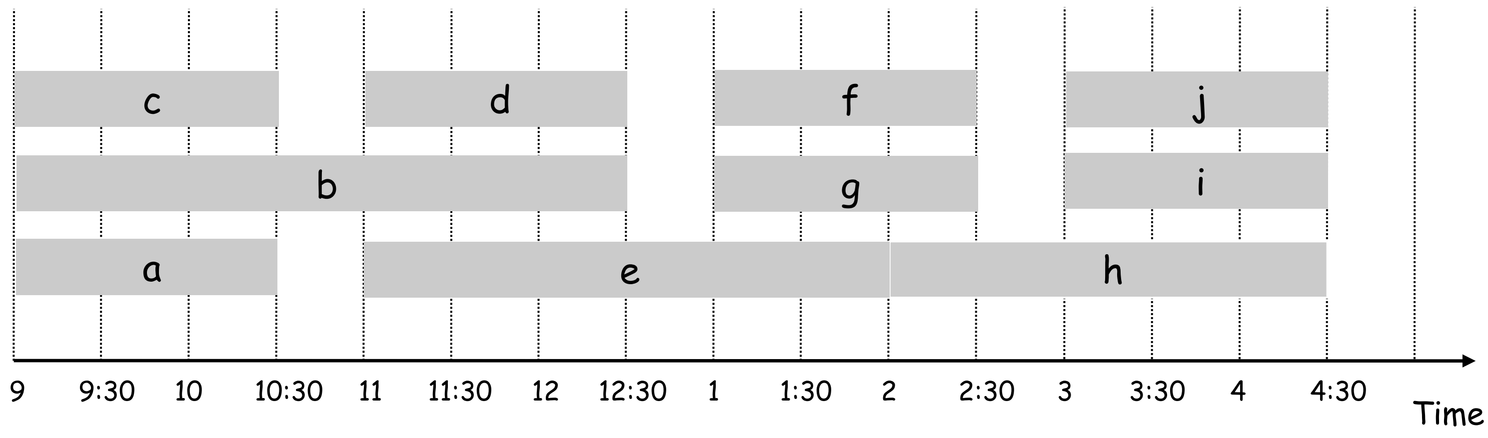


Interval Partitioning

Interval partitioning.

- Lecture j starts at s_j and finishes at f_j.
- Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

Ex: This schedule uses only 3.



Interval Partitioning: Lower Bound on Optimal Solution

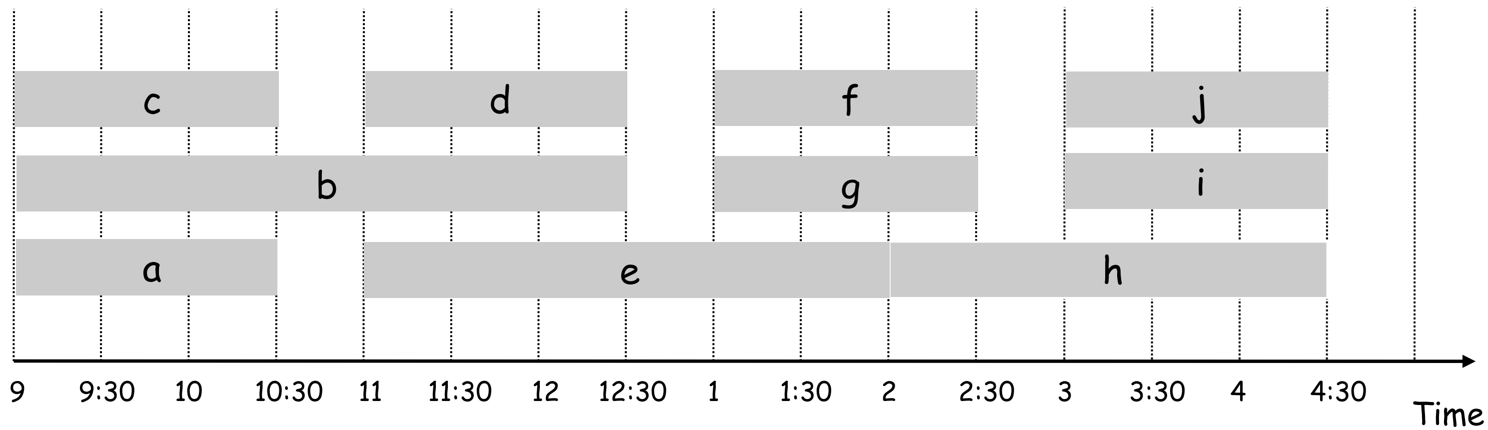
Def. The depth of a set of open intervals is the maximum number that contain any given time.

Key observation. Number of classrooms needed $\geq$ depth.

Ex: Depth of schedule below = 3 $\Rightarrow$ schedule below is optimal.

a, b, c all contain 9:30

Q. Does there always exist a schedule equal to depth of intervals?



Interval Partitioning: Greedy Algorithm

Greedy algorithm. Consider lectures in increasing order of start time: assign lecture to any compatible classroom.

```
Sort intervals by starting time so that s1 ≤ s2 ≤ ... ≤ sn.
d ← 0    ← number of allocated classrooms

for j = 1 to n {
   if (lecture j is compatible with some classroom k)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d ← d + 1
}
```

Implementation. O(n log n).

- For each classroom k, maintain the finish time of the last job added.
- Keep the classrooms in a priority queue.

Interval Partitioning: Greedy Analysis

Observation. Greedy algorithm never schedules two incompatible lectures in the same classroom.

Theorem. Greedy algorithm is optimal.
Pf.

- Let d = number of classrooms that the greedy algorithm allocates.
- Classroom d was allocated because we needed to schedule a job, say j, that is incompatible with all d-1 other classrooms.
- Since we sorted by start time, all these incompatibilities are caused by lectures that start no later than s_j.
- Thus, we have d lectures overlapping at time $s_j + \varepsilon$.
- Key observation $\Rightarrow$ all schedules use $\geq$ d classrooms. ▪

4.2 Scheduling to Minimize Lateness

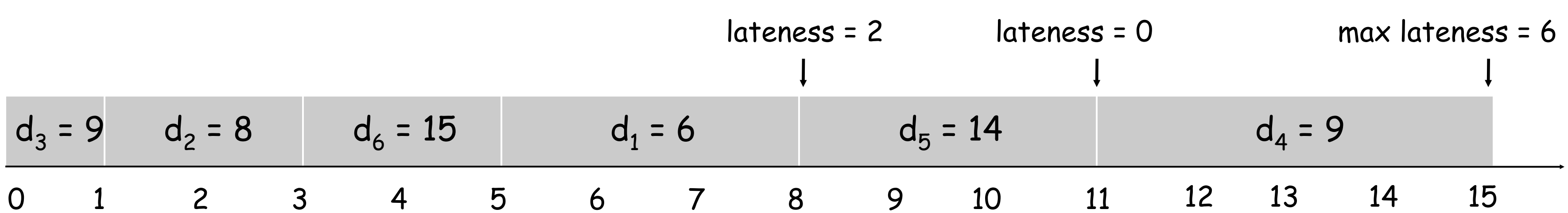
Scheduling to Minimizing Lateness

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.

- [Earliest deadline first] Consider jobs in ascending order of deadline d_j.

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

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	10	1
d_j	10	2

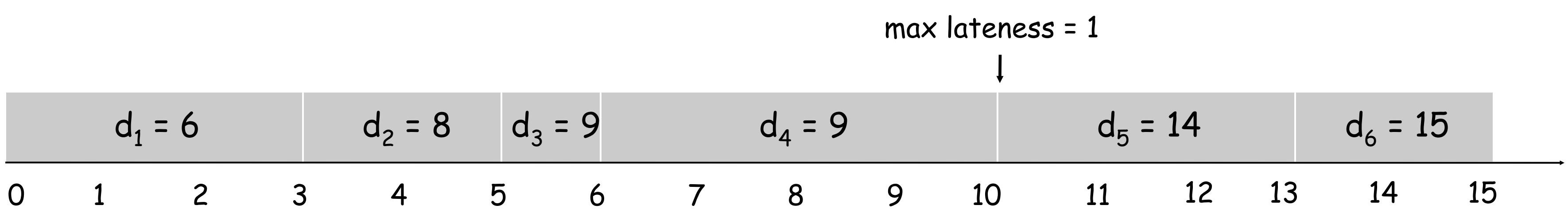
counterexample

Minimizing Lateness: Greedy Algorithm

Greedy algorithm. Earliest deadline first.

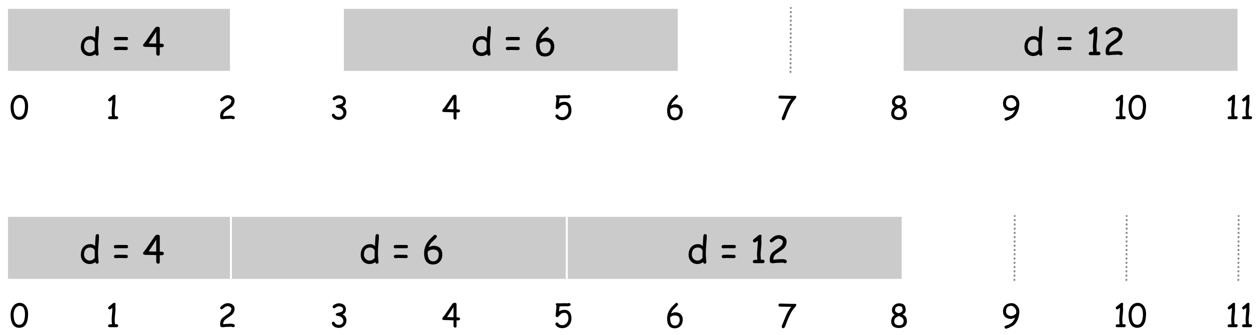
Sort n jobs by deadline so that $d_1 \le d_2 \le \ldots \le d_n$

$t \leftarrow 0$
for $j = 1$ to n
    (Assign job j to interval $[t, t + t_j]$)
    $s_j \leftarrow t,\ f_j \leftarrow t + t_j$
    $t \leftarrow t + t_j$
output intervals $[s_j, f_j]$



Minimizing Lateness: No Idle Time

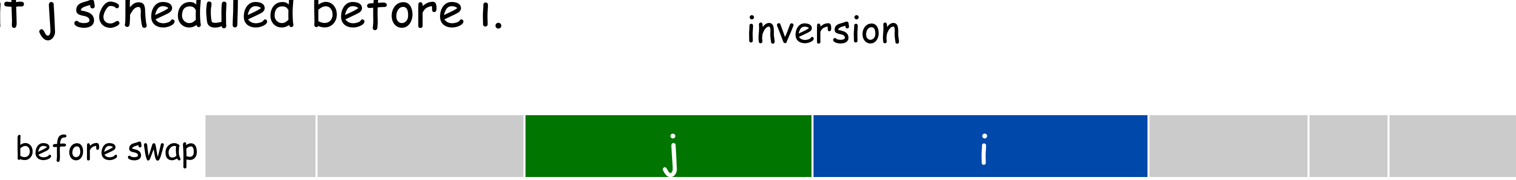
Observation. There exists an optimal schedule with no idle time.



Observation. The greedy schedule has no idle time.

Minimizing Lateness: Inversions

Def. An inversion in schedule S is a pair of jobs i and j such that: i < j but j scheduled before i.

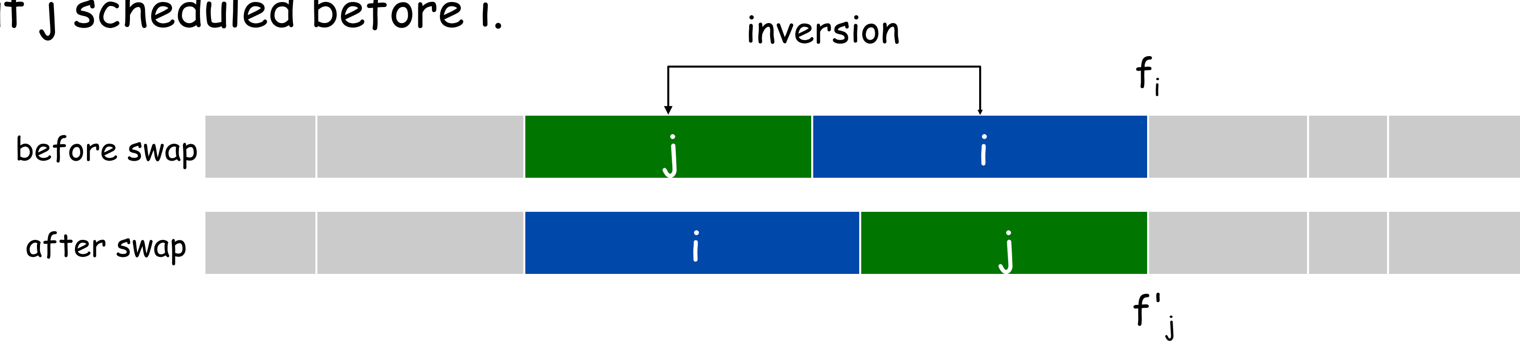


Observation. Greedy schedule has no inversions.

Observation. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.

Minimizing Lateness: Inversions

Def. An inversion in schedule S is a pair of jobs i and j such that: i < j but j scheduled before i.



Claim. Swapping two adjacent, 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 $k \neq i, j$
- $\ell'_i \leq \ell_i$
- If job j is late:

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

Minimizing Lateness: Analysis of Greedy Algorithm

Theorem. Greedy schedule S is optimal.

Pf. Define S* to be an optimal schedule that has the fewest number of inversions, and let's see what happens.

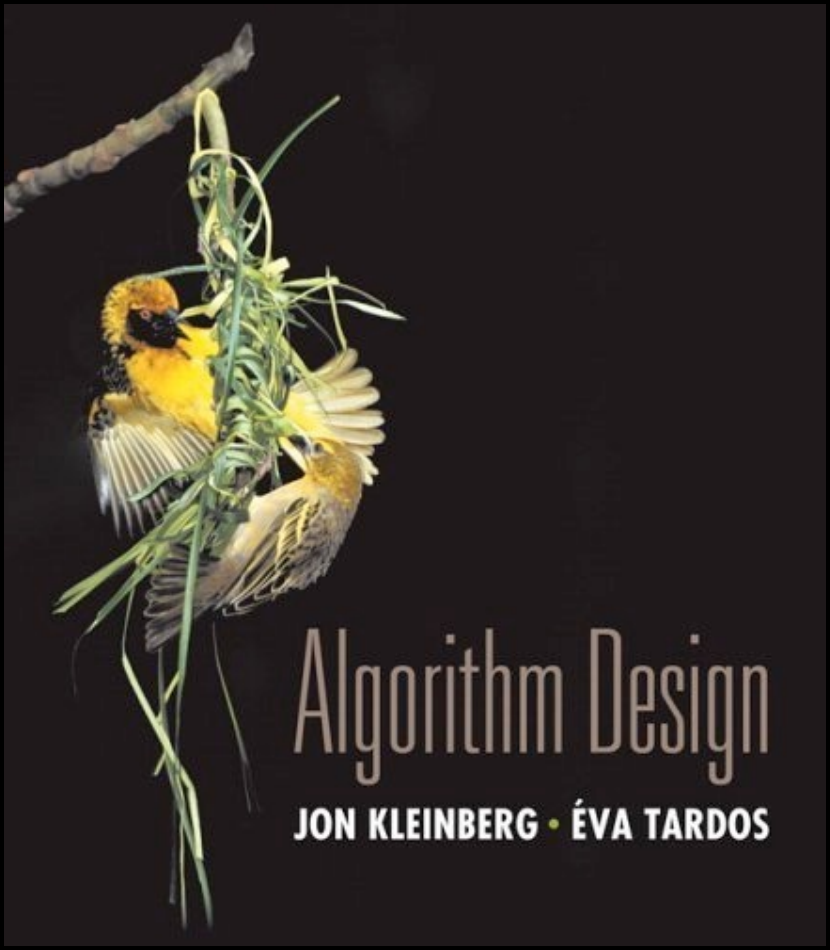
- Can assume S* has no idle time.
- If S* has no inversions, then S = S*.
- If S* has an inversion, let i-j be an adjacent inversion.
  - swapping i and j does not increase the maximum lateness and strictly decreases the number of inversions
  - this contradicts definition of S* ▪

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.

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

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.

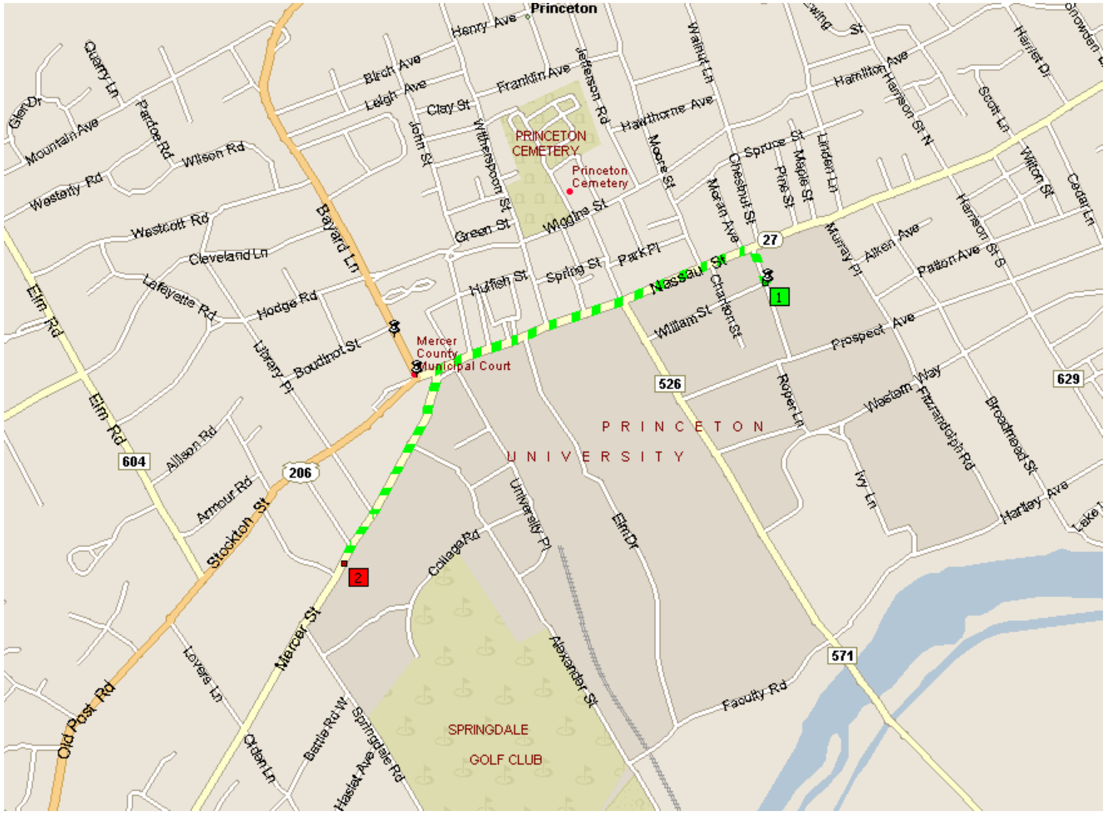


Chapter 4

Greedy Algorithms

Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.

4.4 Shortest Paths in a Graph



shortest path from Princeton CS department to Einstein's house

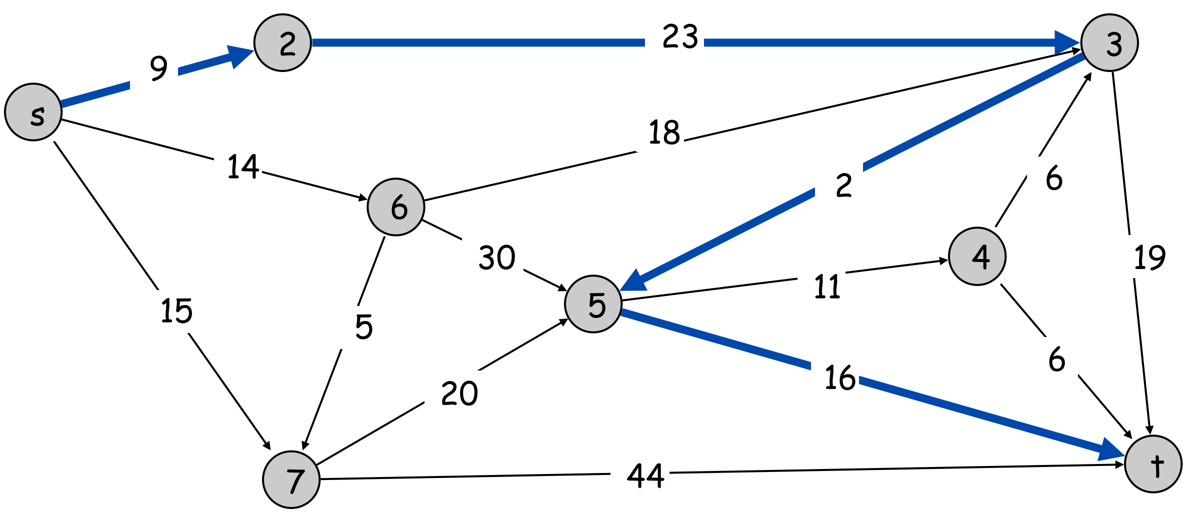
Shortest Path Problem

Shortest path network.

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

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

cost of path = sum of edge costs in path



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

Dijkstra's Algorithm

Dijkstra's algorithm.

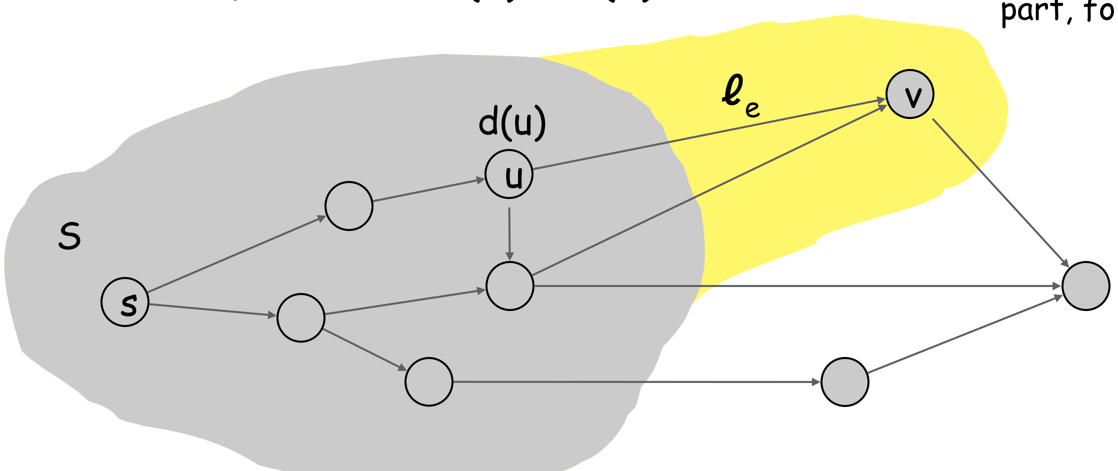
- Maintain a set of explored nodes S for which we have determined the shortest path distance d(u) from s to u.

- Initialize S = { s }, d(s) = 0.
- Repeatedly (greedily) choose unexplored node $v \notin S$ which minimizes

$$\partial(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e ,$$

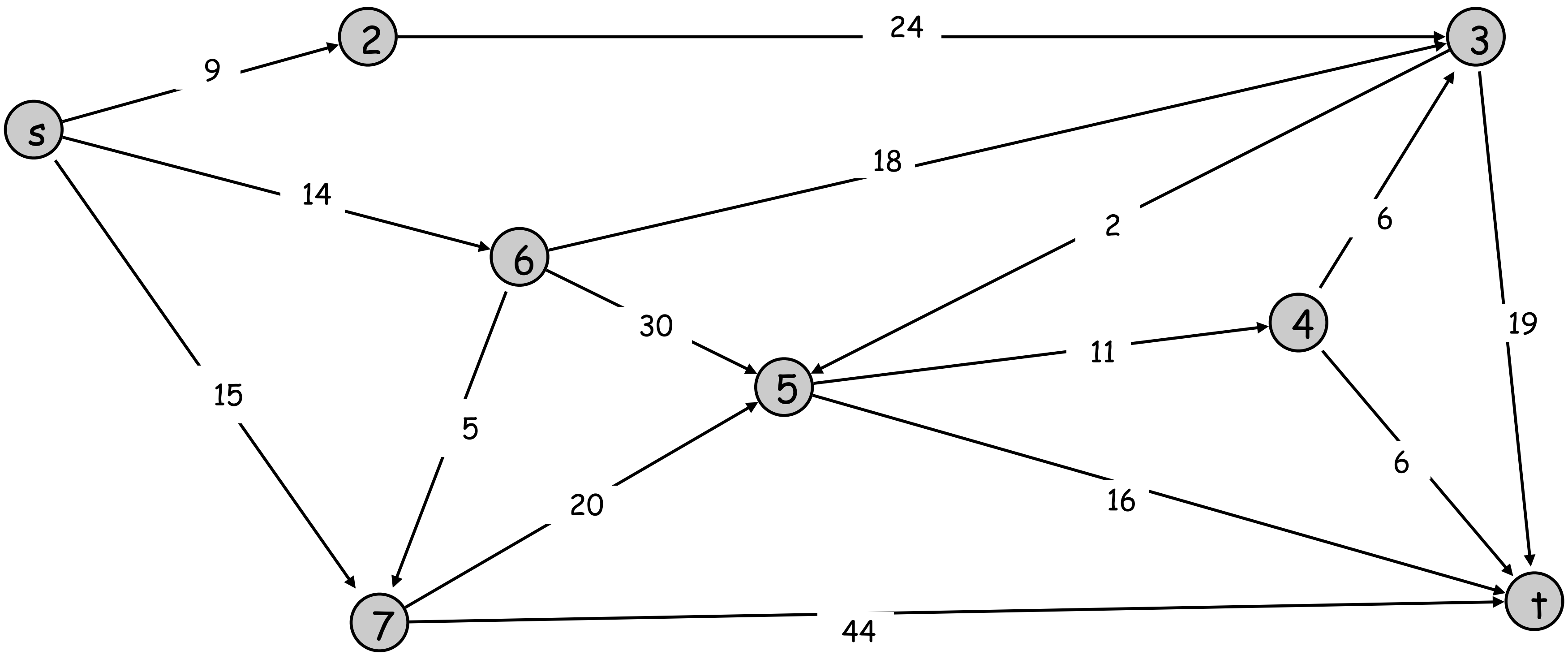
shortest path to some u in explored part, followed by a single edge (u, v)

add v to S, and set d(v) = ∂(v).



Dijkstra's Shortest Path Algorithm

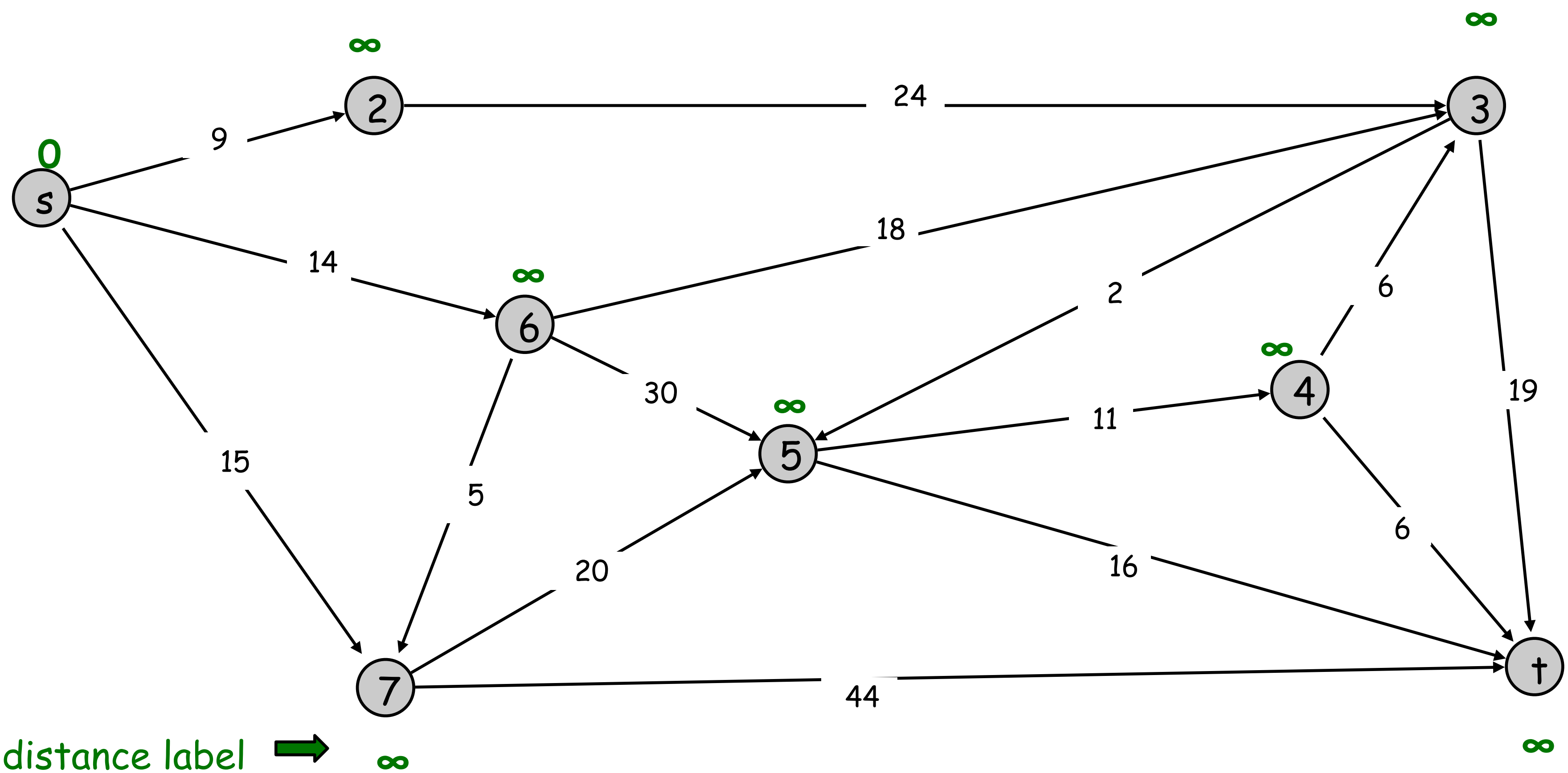
Find shortest path from s to t.



Dijkstra's Shortest Path Algorithm

S = { }

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

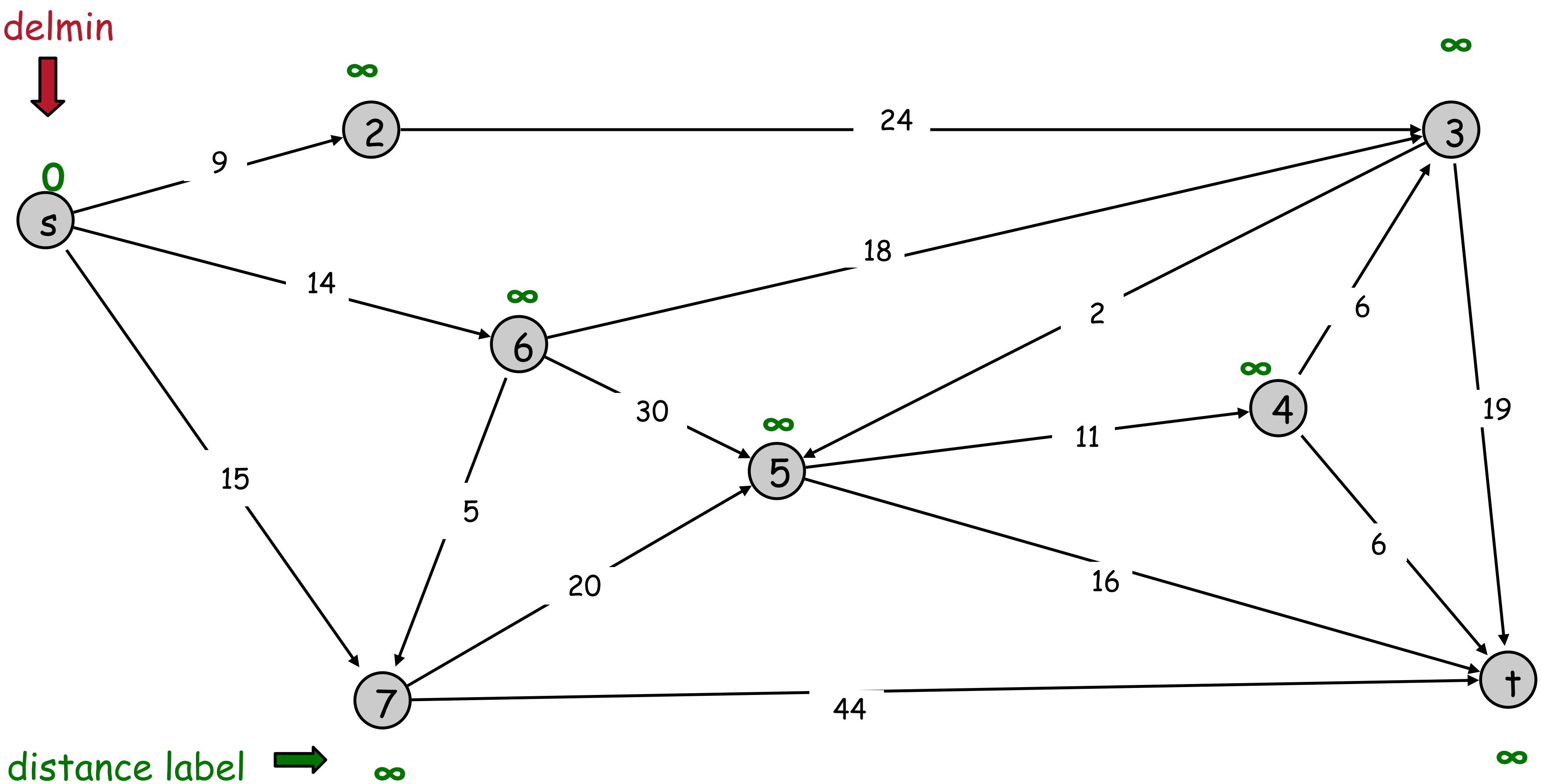


distance label

Dijkstra's Shortest Path Algorithm

S = { }

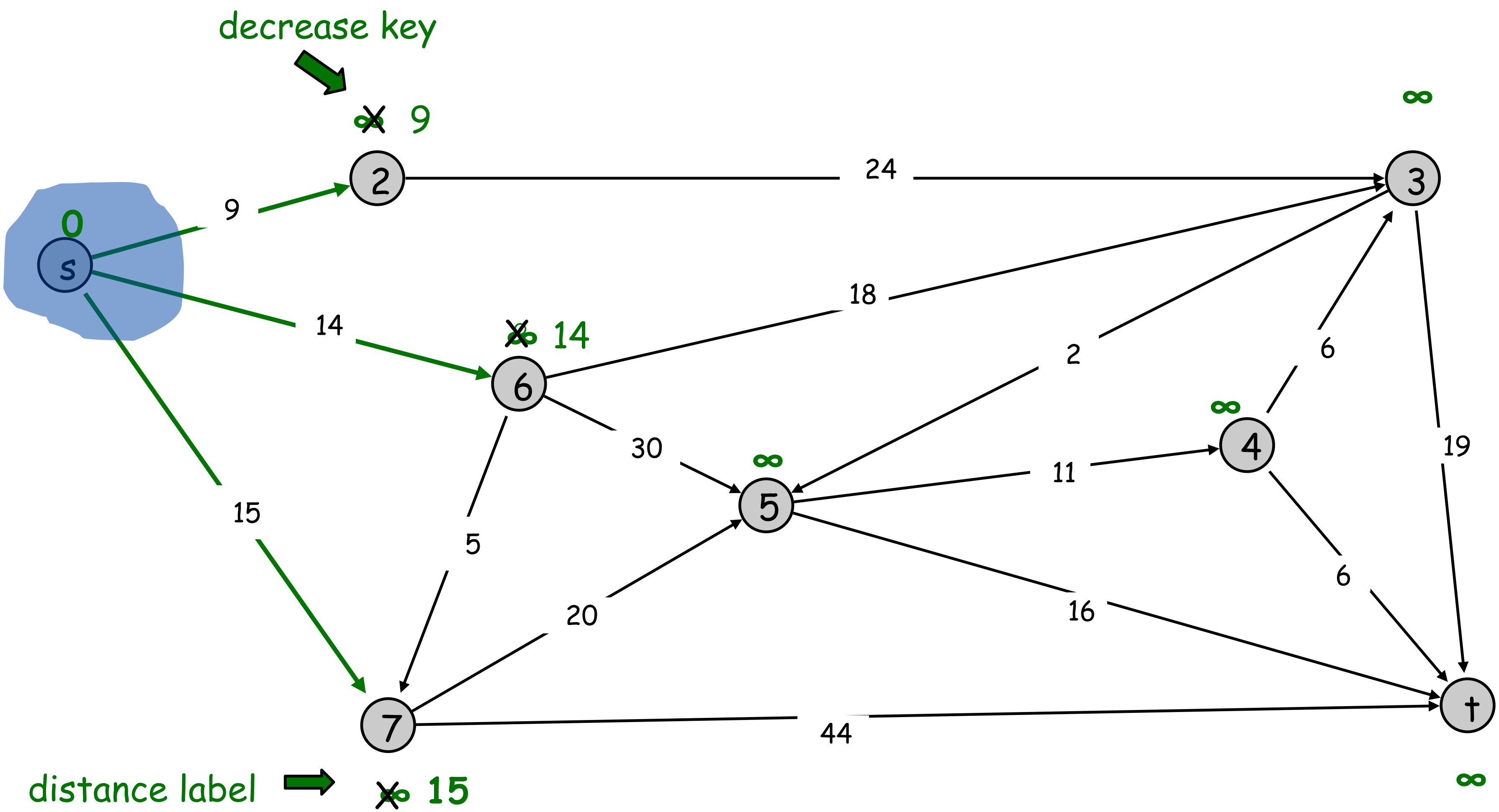
PQ = { s, 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s }

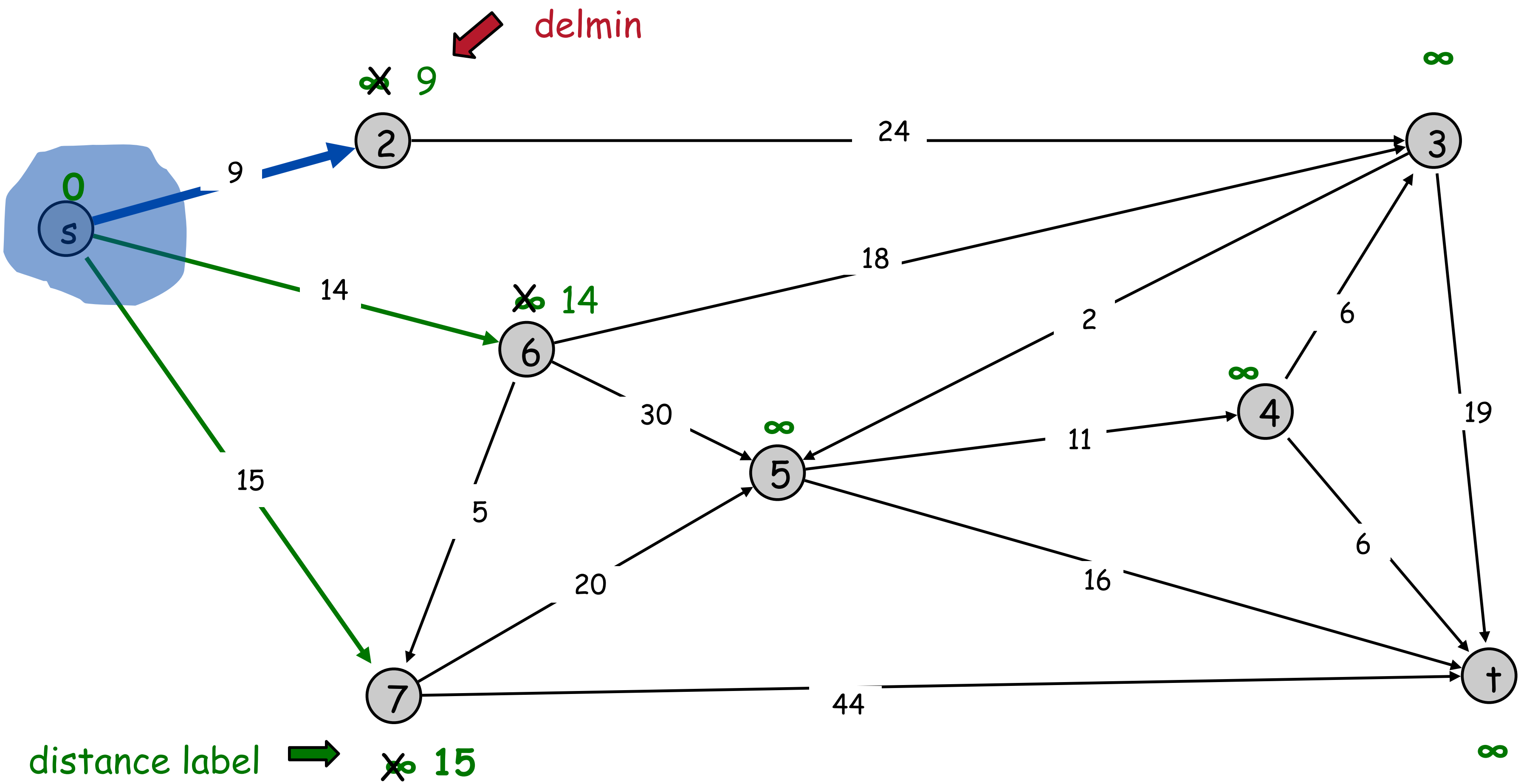
PQ = { 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s }

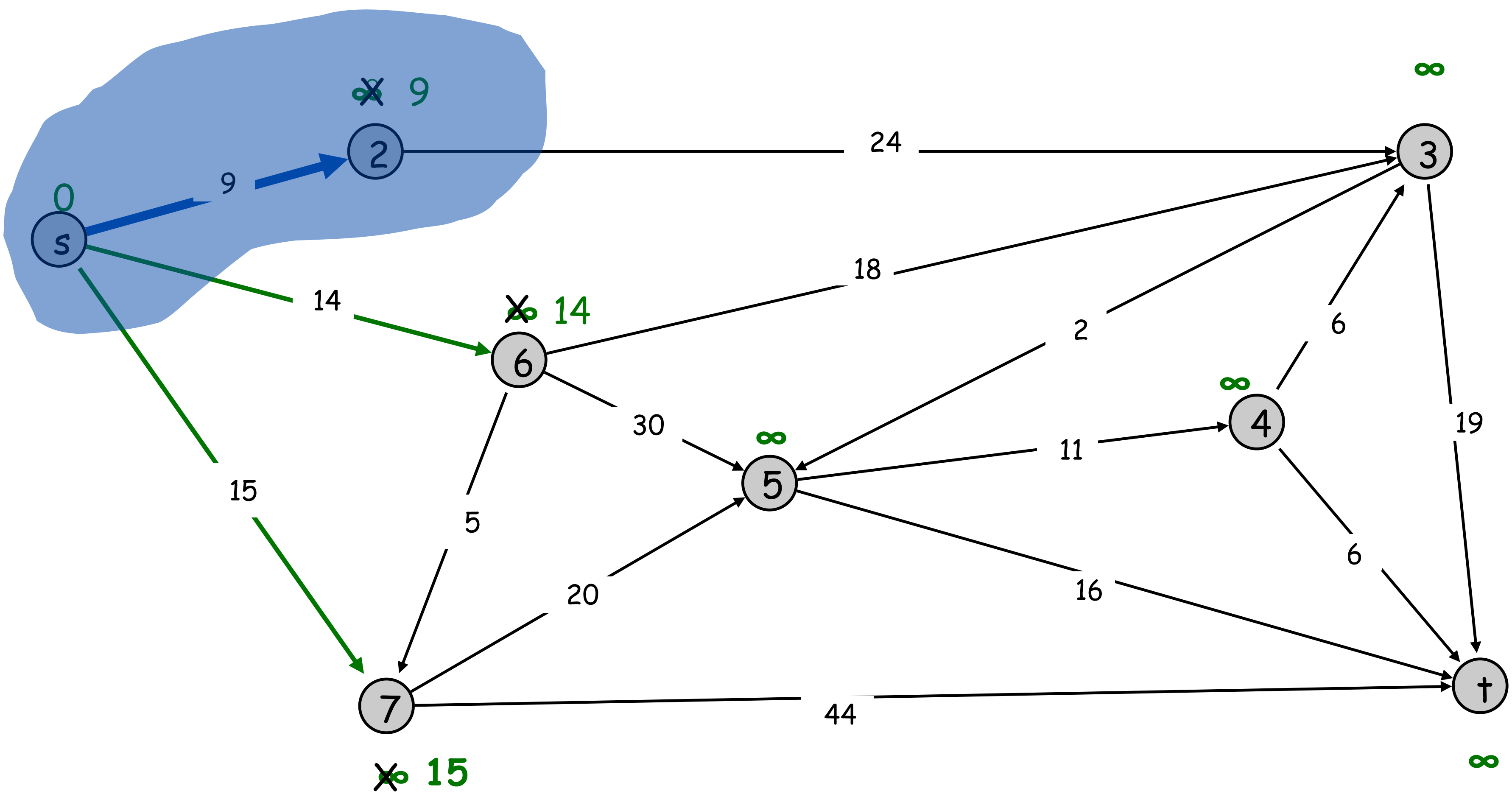
PQ = { 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2 }

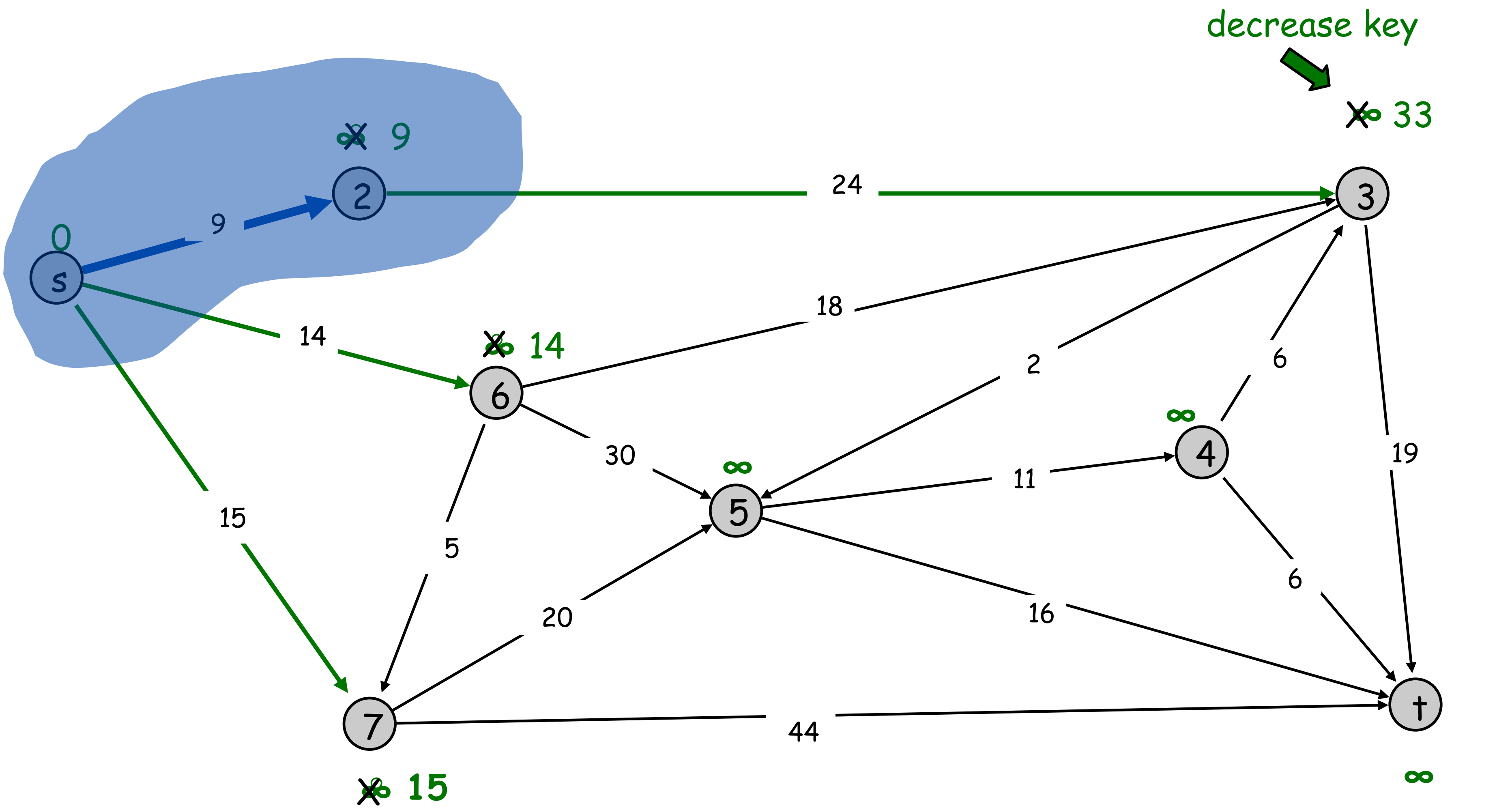
PQ = { 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2 }

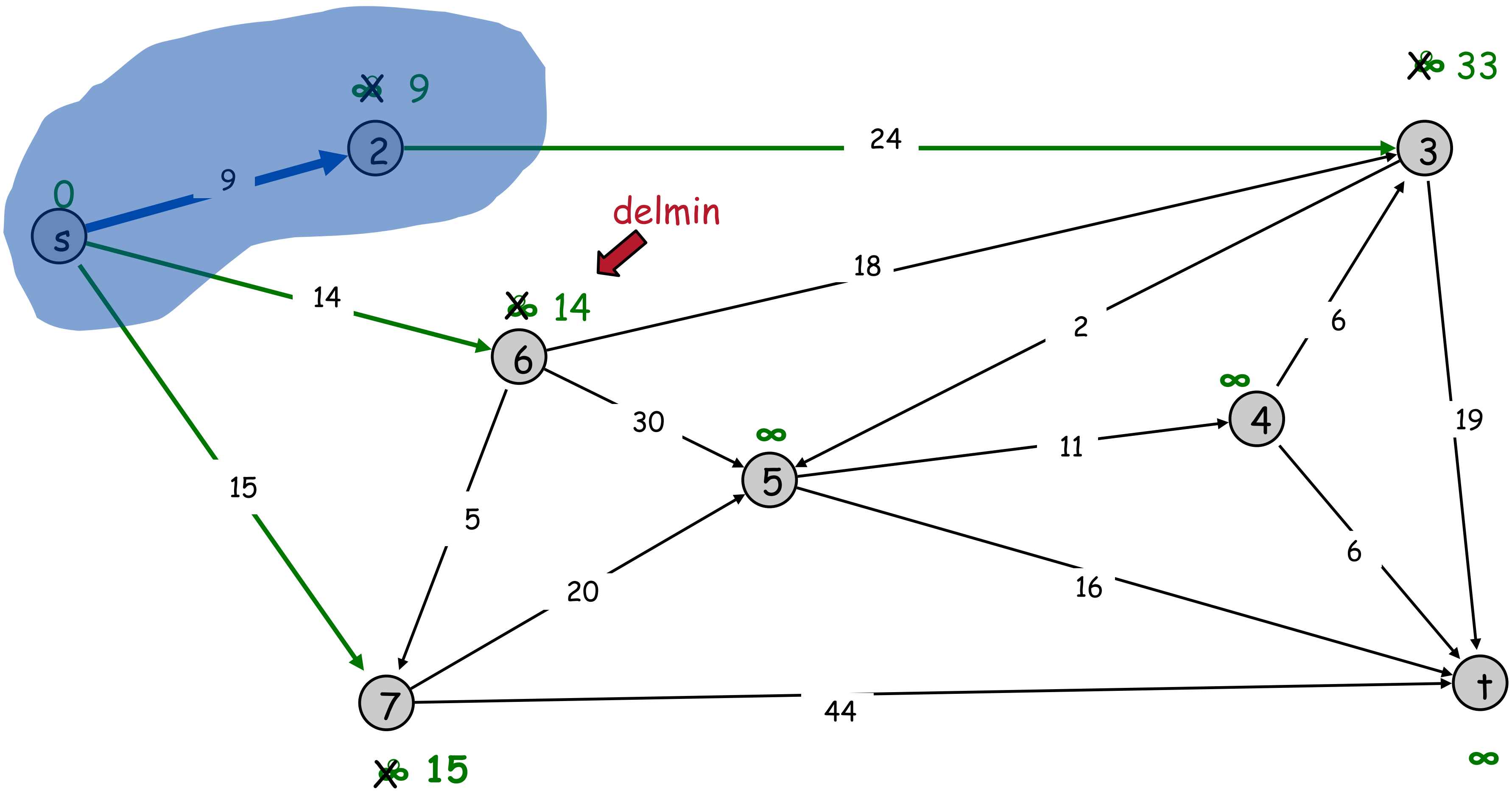
PQ = { 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2 }

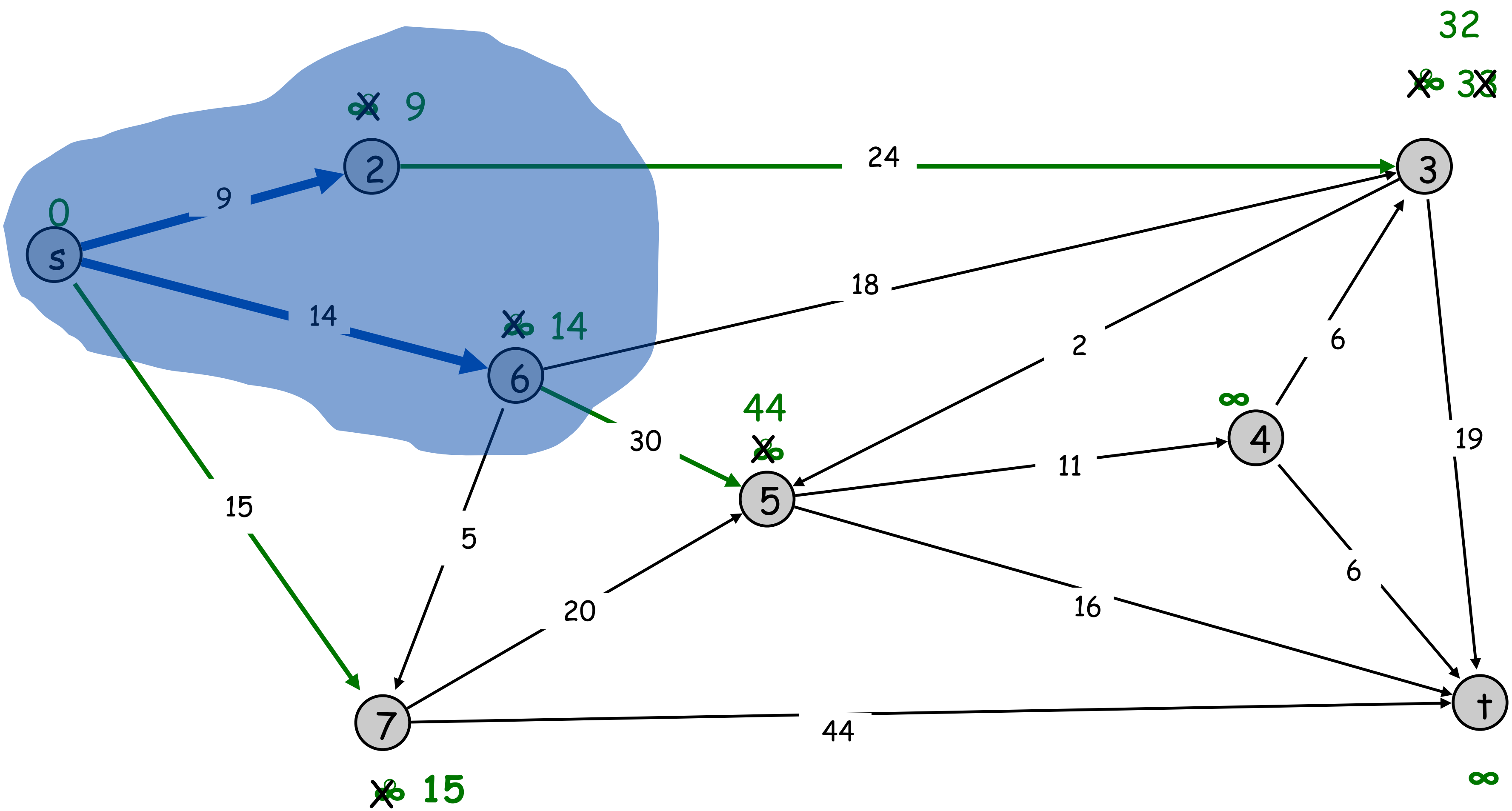
PQ = { 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2, 6 }

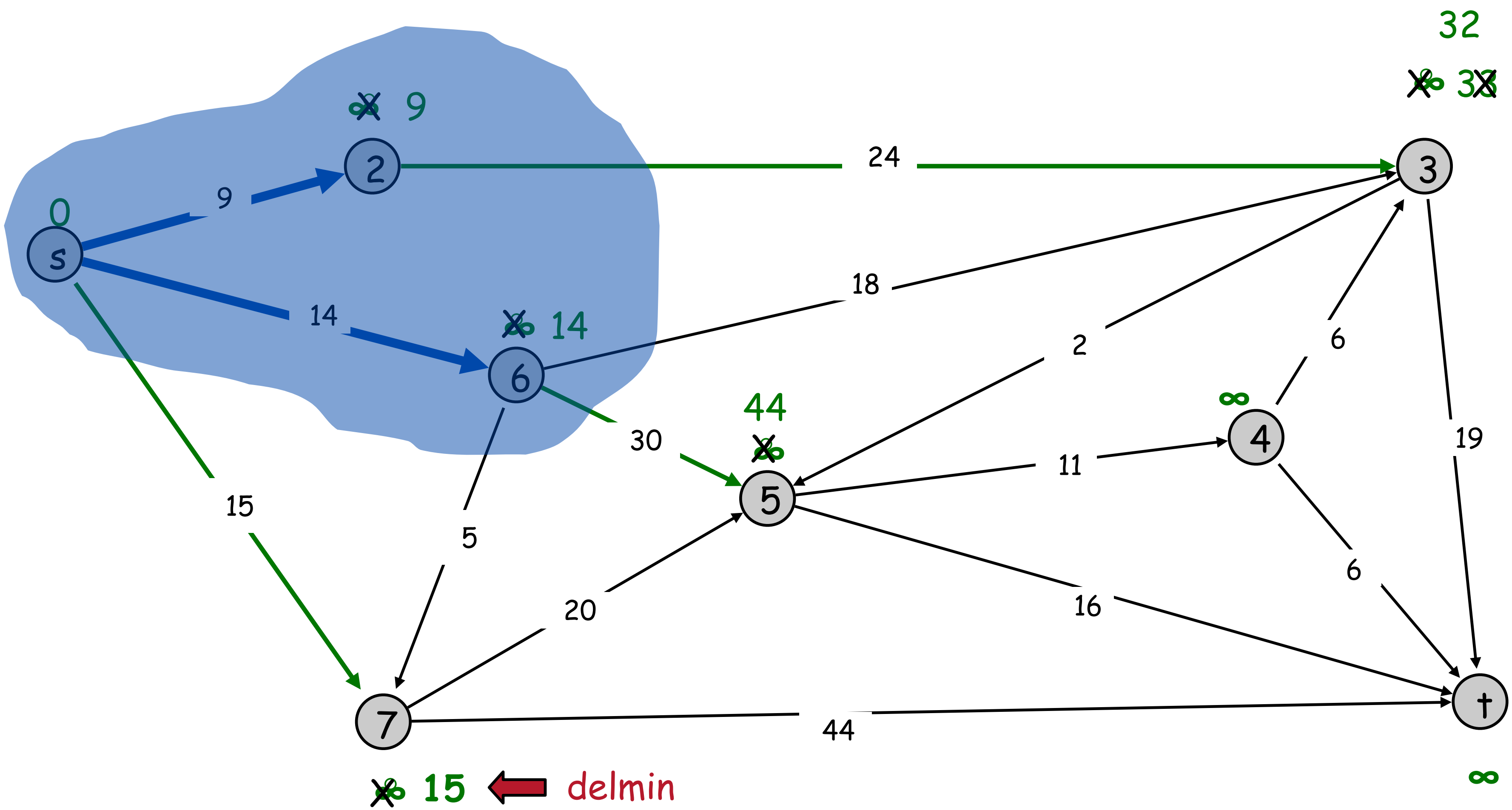
PQ = { 3, 4, 5, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2, 6 }

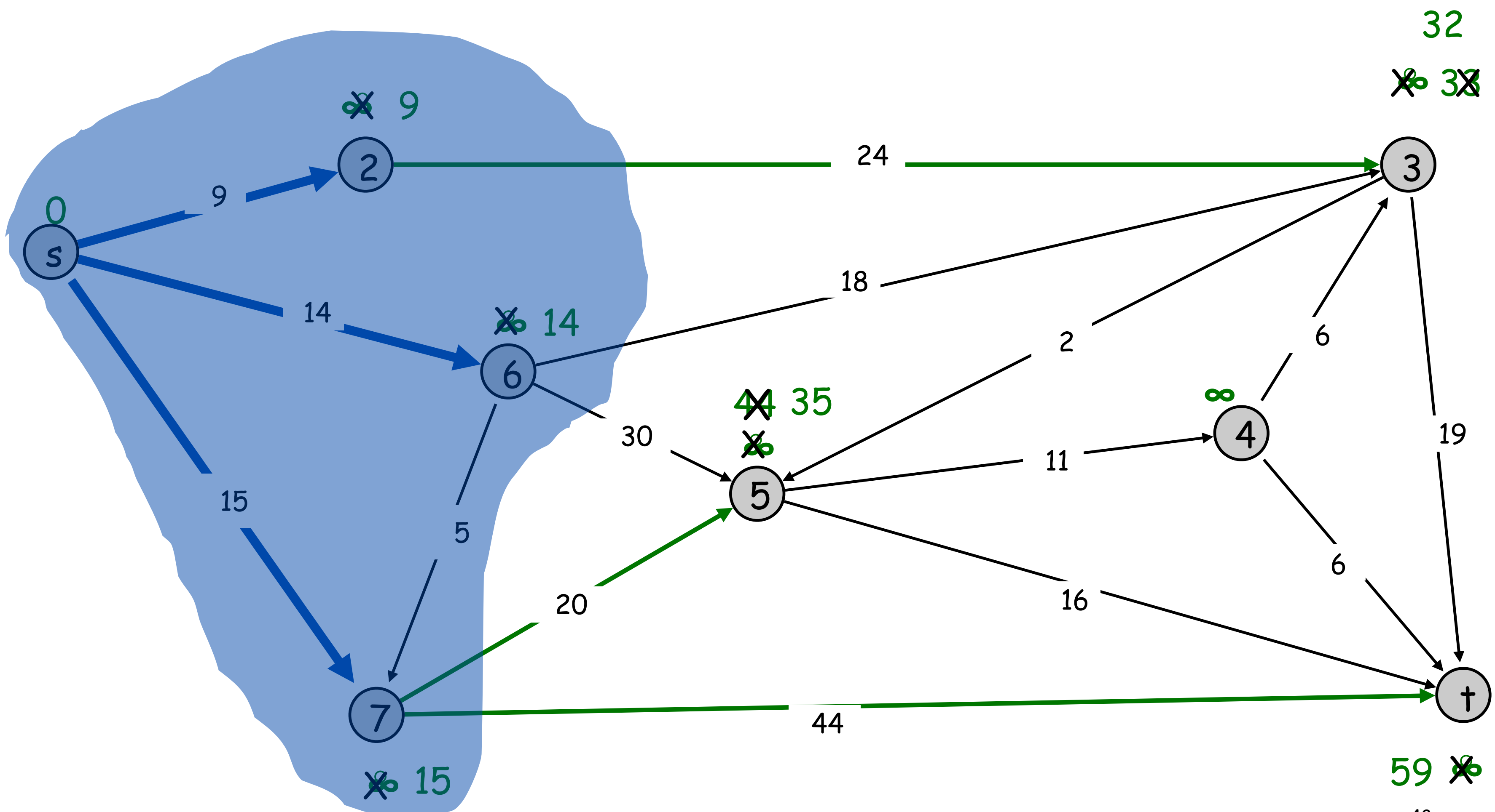
PQ = { 3, 4, 5, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7 }

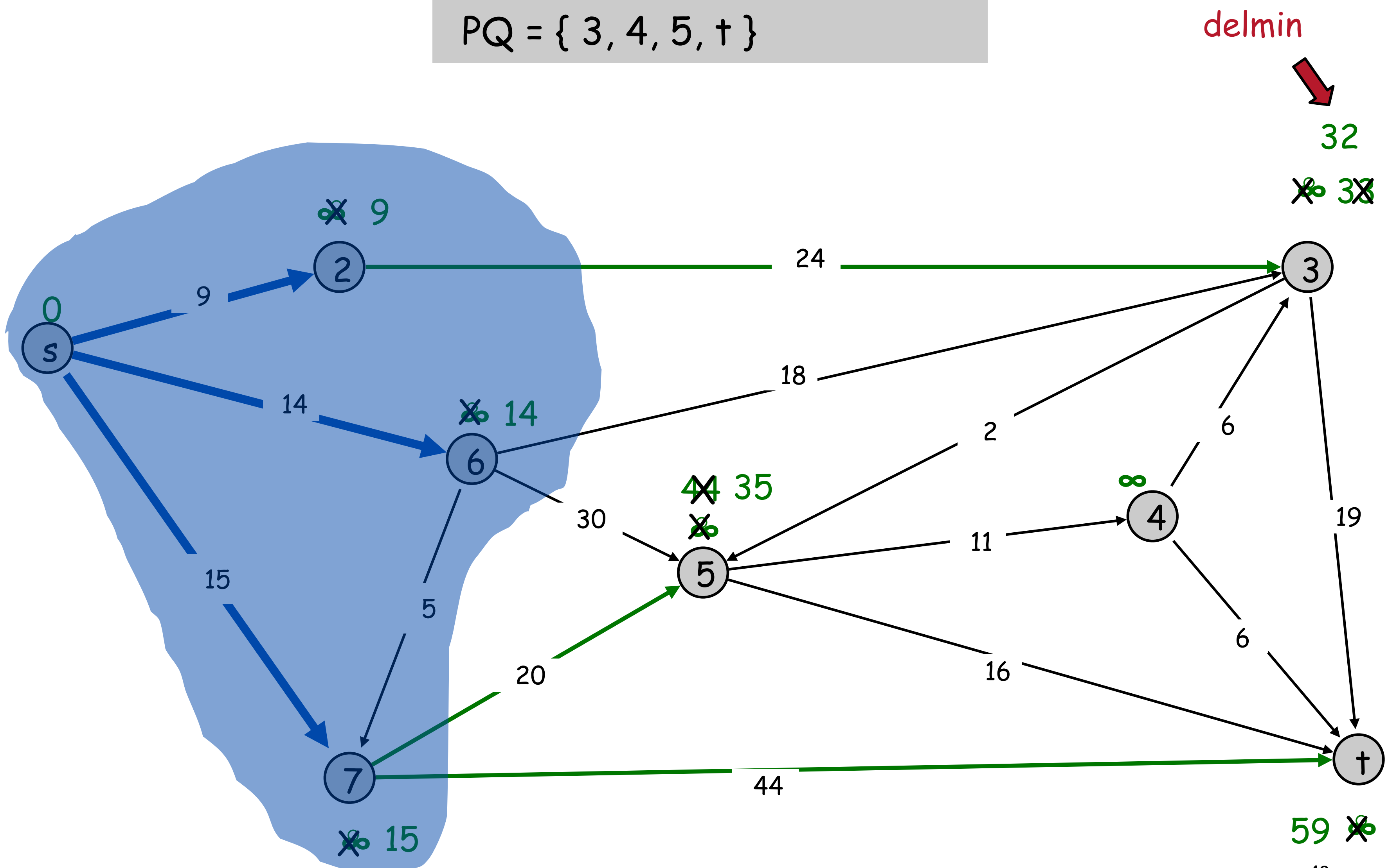
PQ = { 3, 4, 5, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7 }

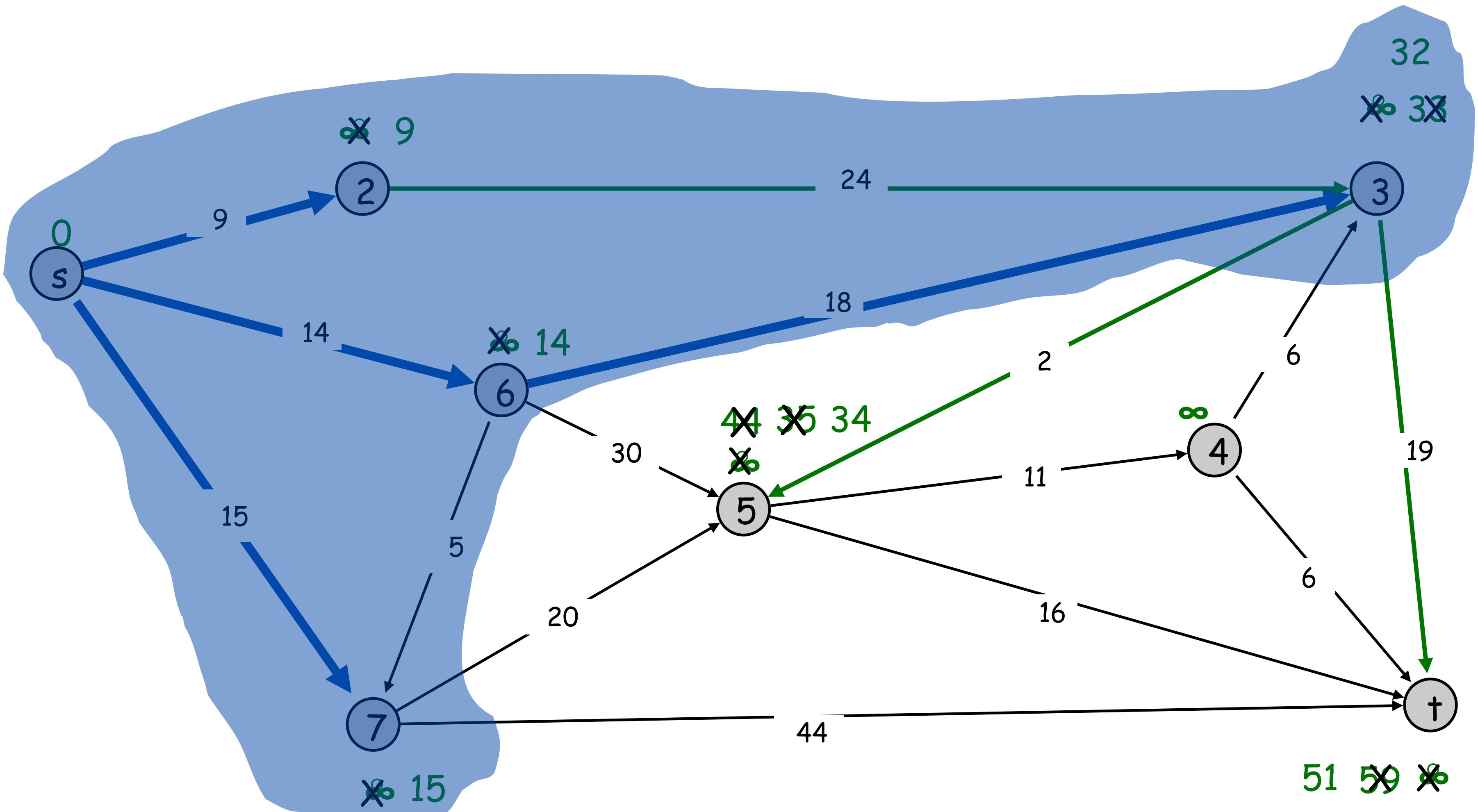
PQ = { 3, 4, 5, t }



Dijkstra's Shortest Path Algorithm

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

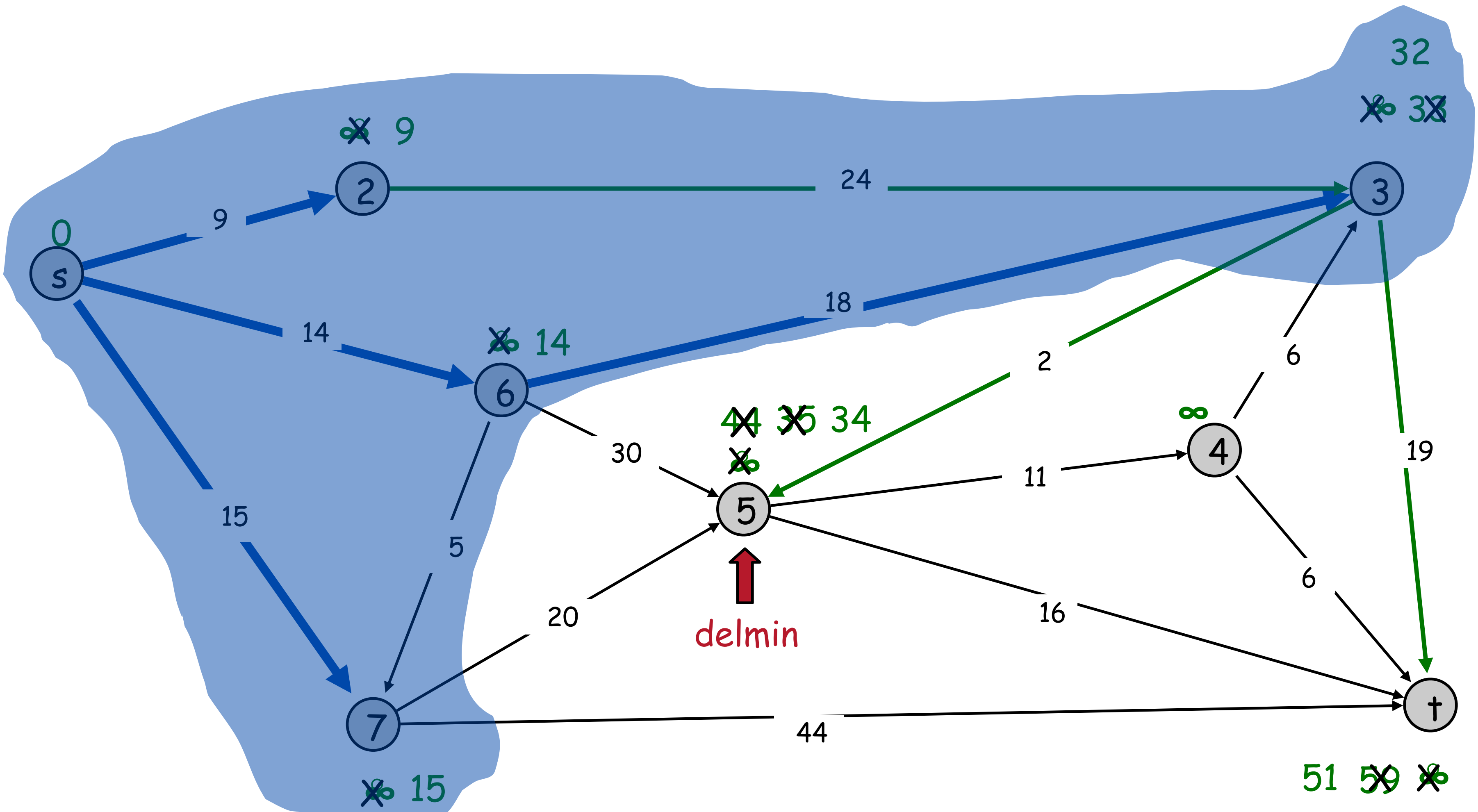
PQ = { 4, 5, t }



Dijkstra's Shortest Path Algorithm

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

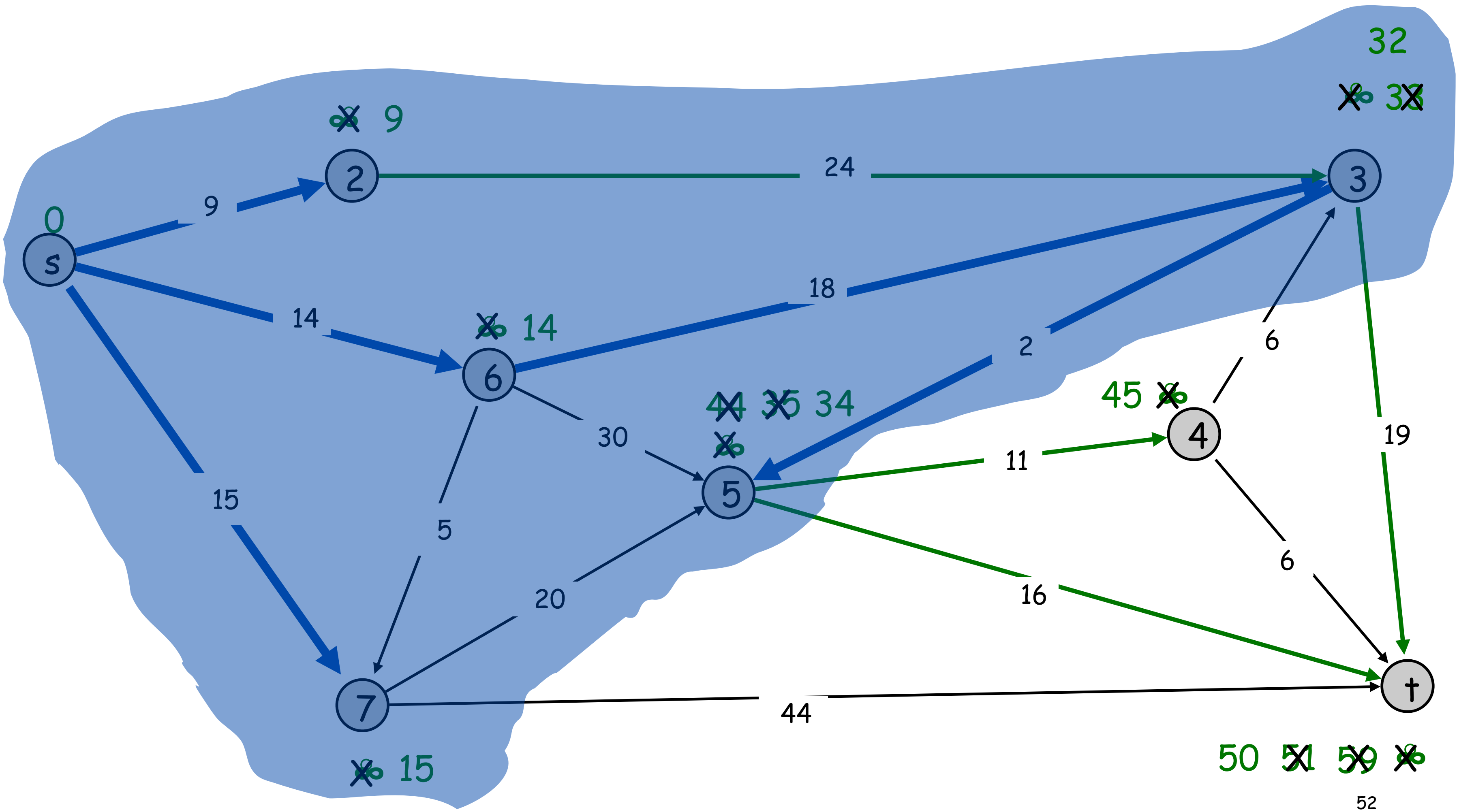
PQ = { 4, 5, t }



Dijkstra's Shortest Path Algorithm

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

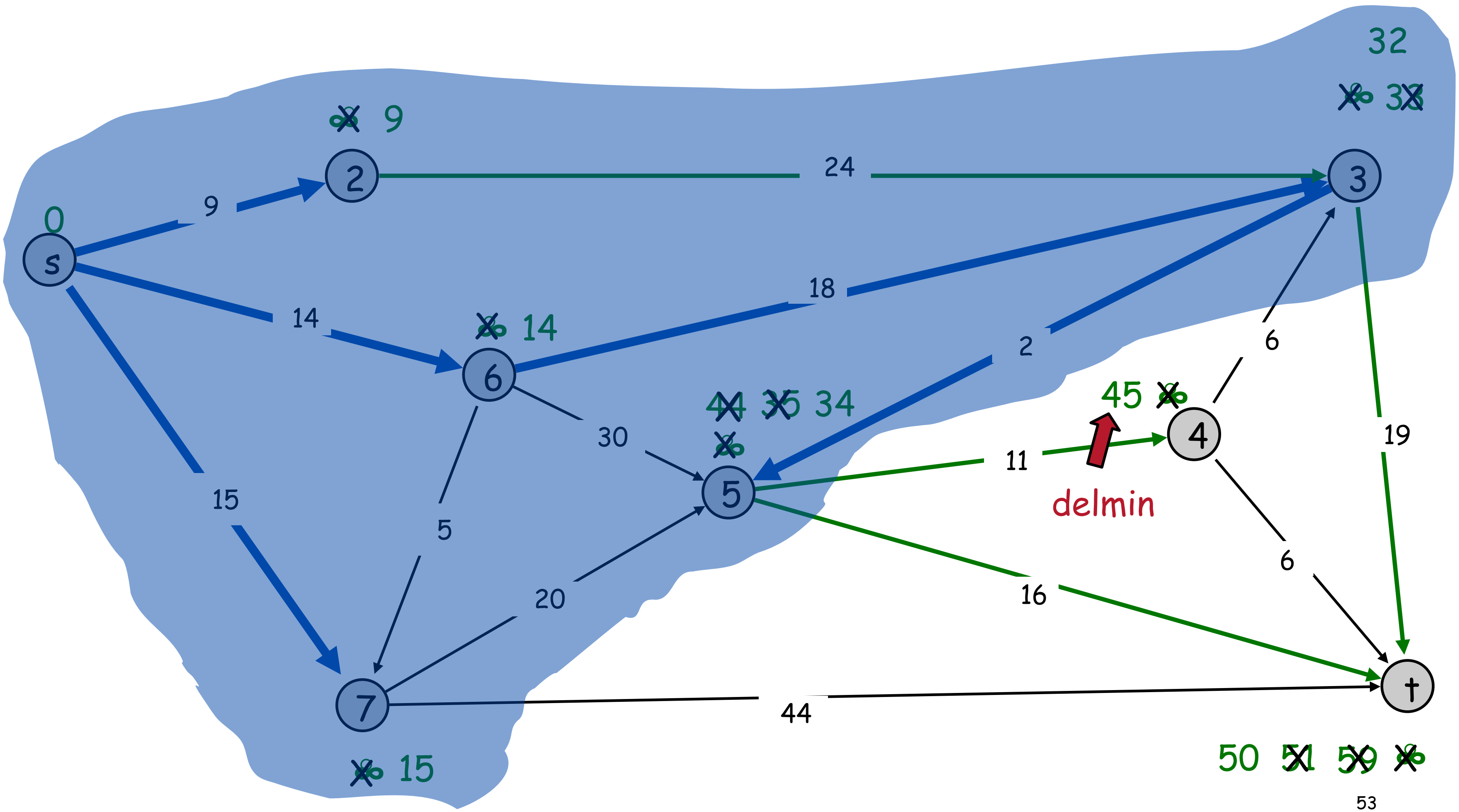
PQ = { 4, t }



Dijkstra's Shortest Path Algorithm

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

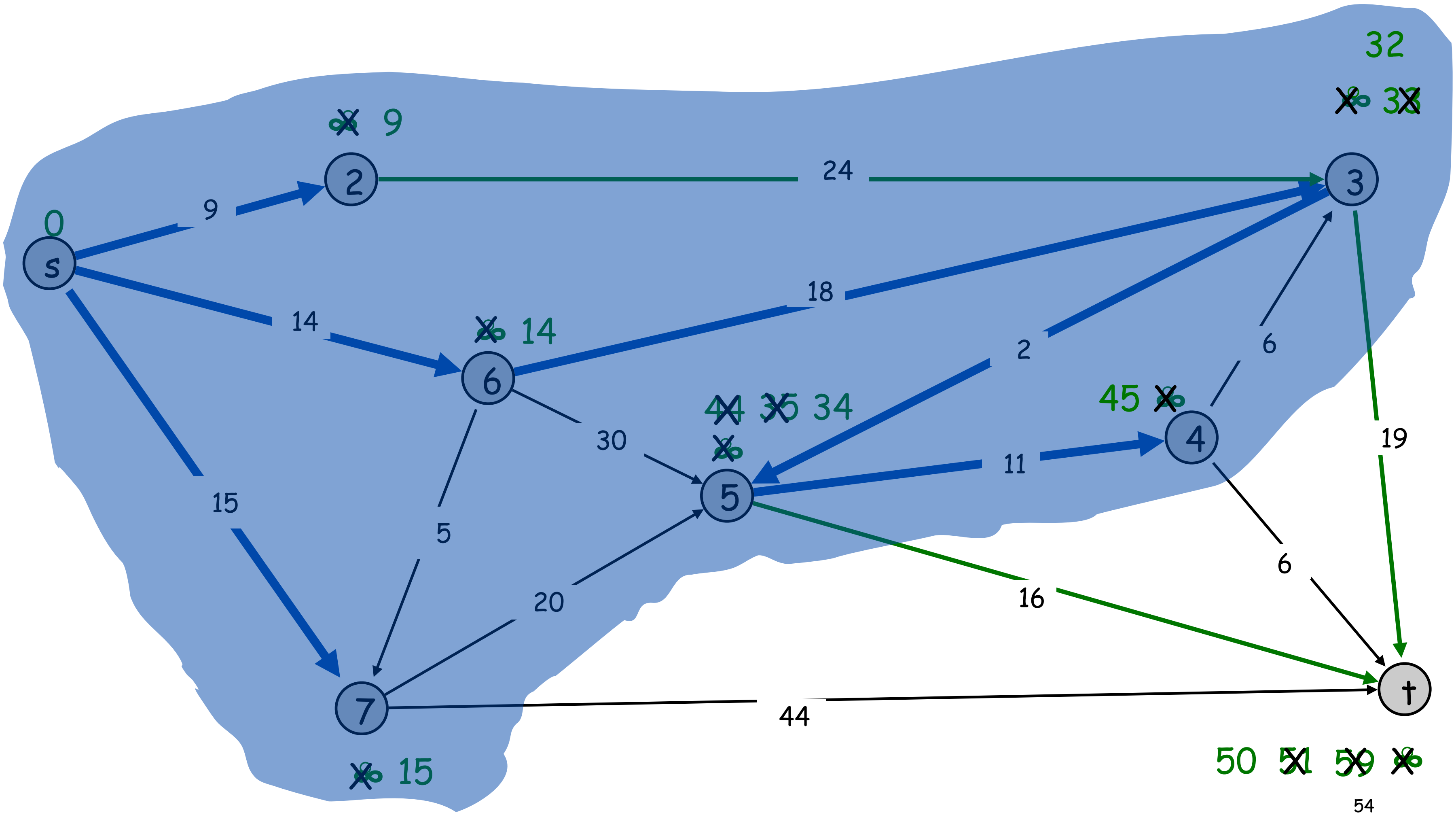
PQ = { 4, t }



Dijkstra's Shortest Path Algorithm

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

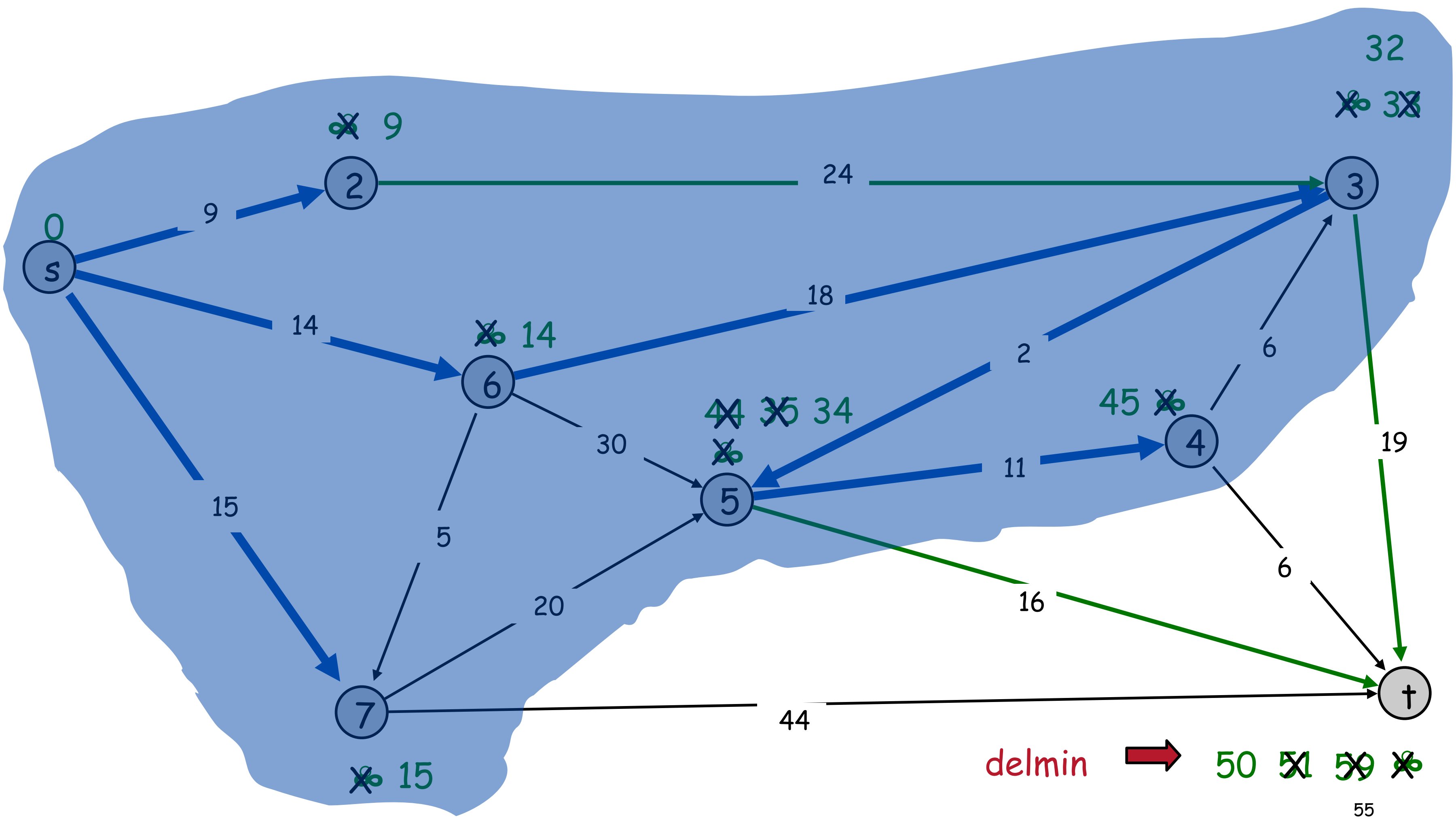
PQ = { t }



Dijkstra's Shortest Path Algorithm

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

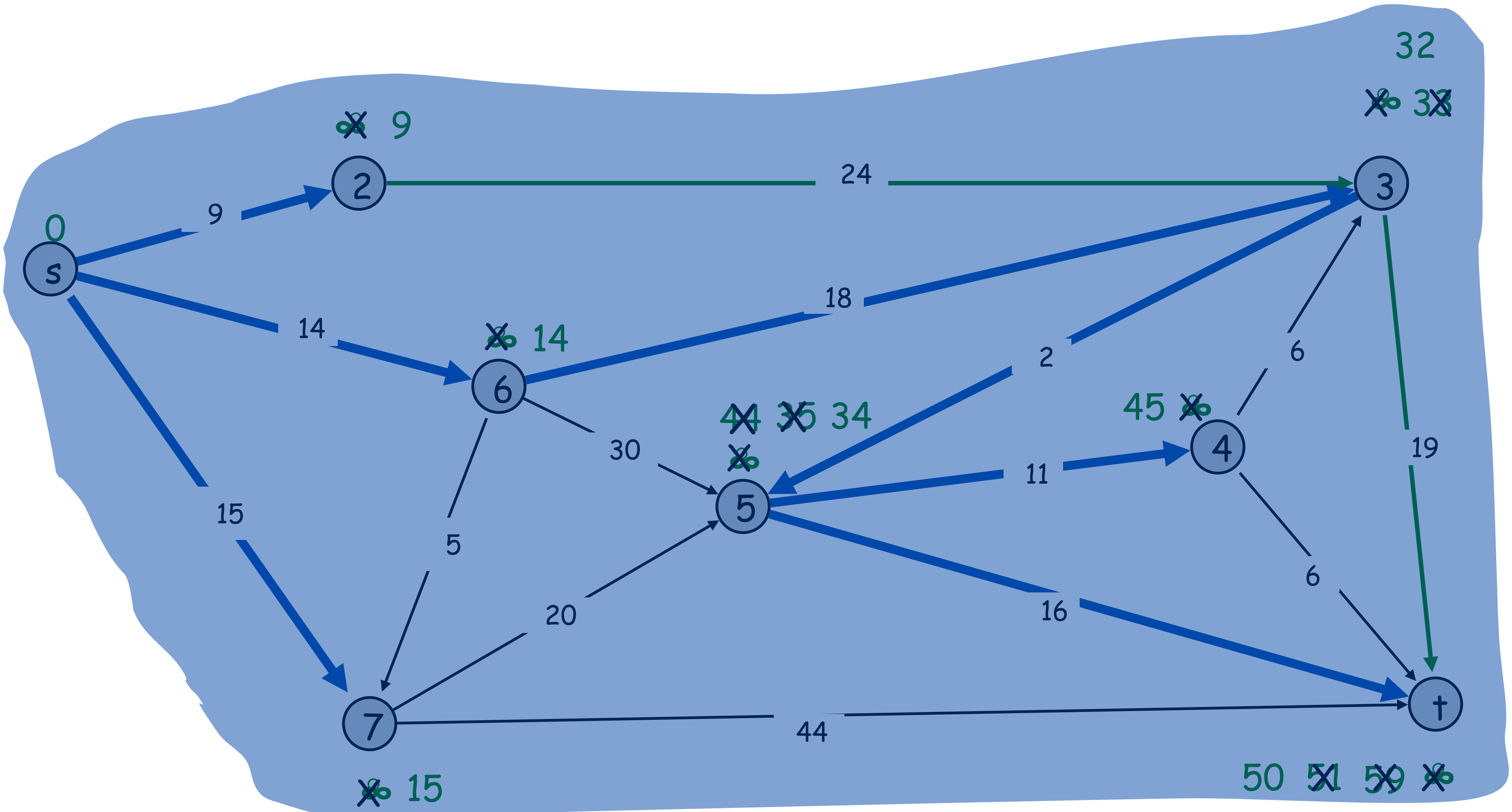
$PQ = \{ t \}$



Dijkstra's Shortest Path Algorithm

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

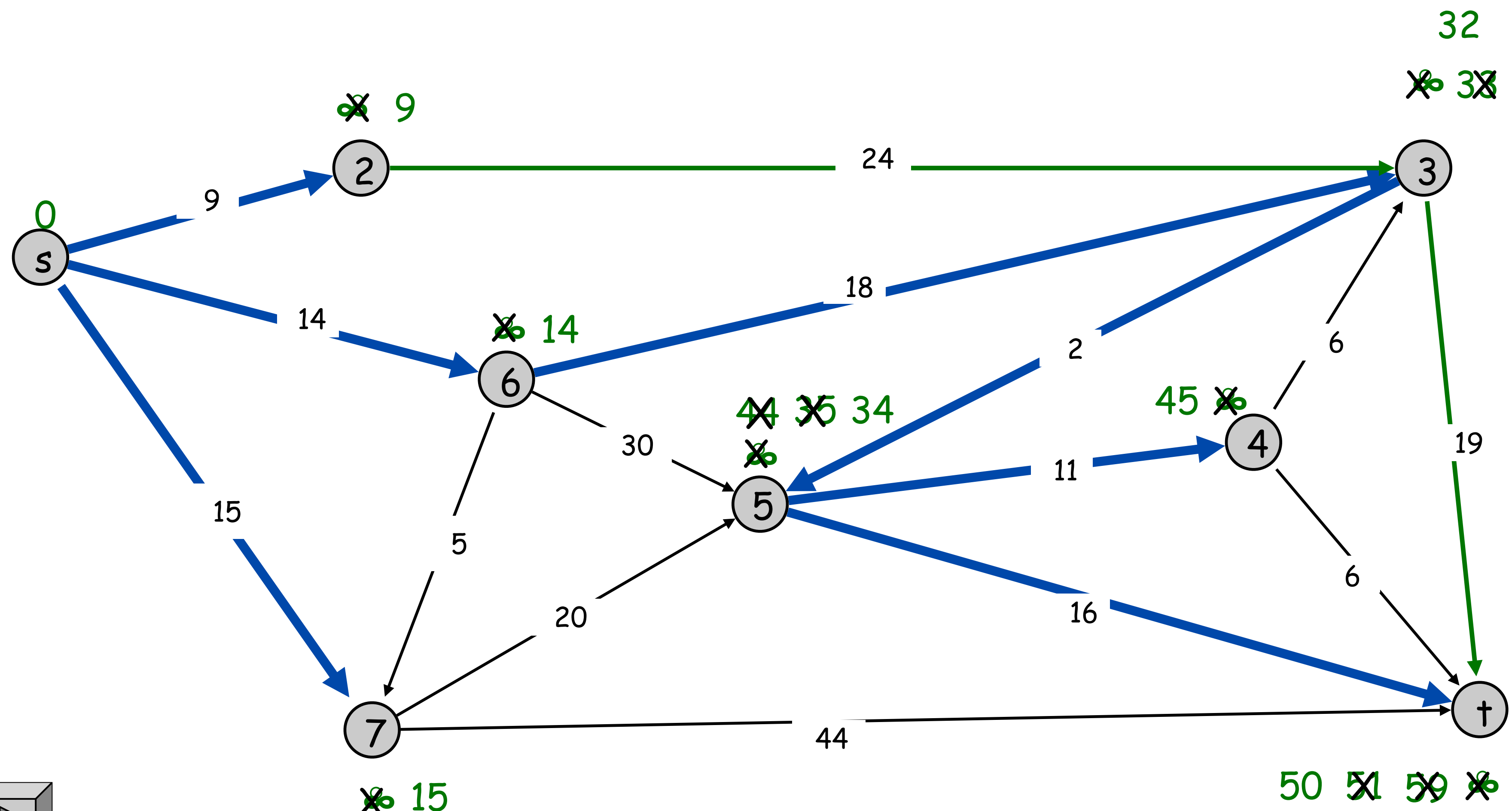
PQ = { }



Dijkstra's Shortest Path Algorithm

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

PQ = { }



Dijkstra's Algorithm: Proof of Correctness

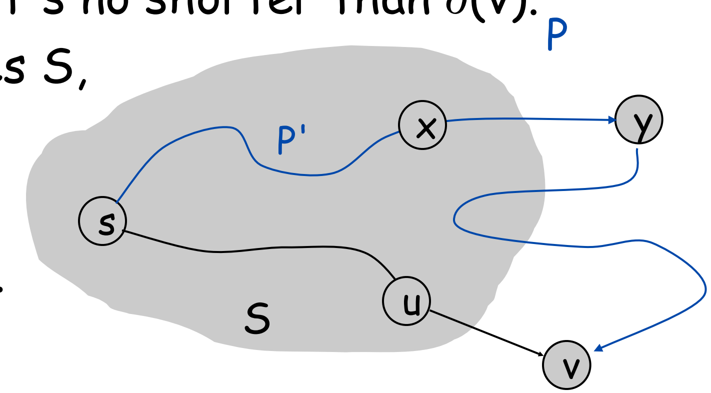
Invariant. For each node $u \in S$, $d(u)$ is the length of the shortest s-u path.

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 $\partial(v)$.
- Consider any s-v path P. We'll see that it's no shorter than $\partial(v)$.
- 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.



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

nonnegative weights

inductive hypothesis

defn of $\partial(y)$

Dijkstra chose v instead of y

Dijkstra's Algorithm: Implementation

For each unexplored node $v \notin S$, explicitly maintain $\pi(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e,$

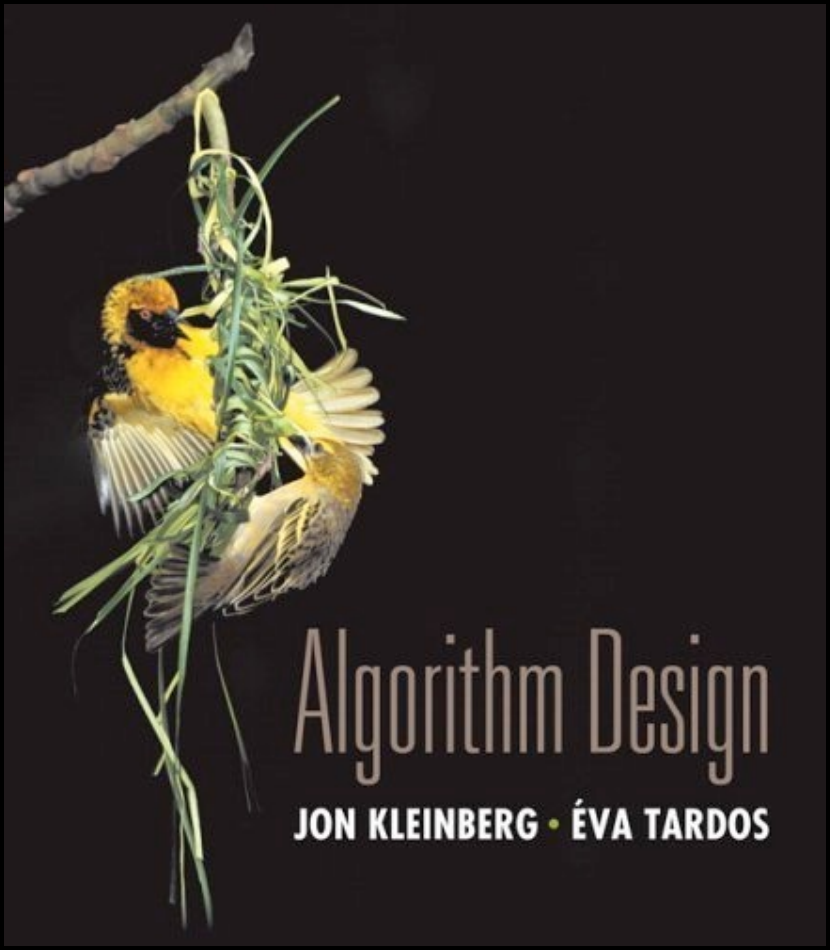
- Next node to explore = node $v \notin S$ with minimum $\pi(v)$.
- When exploring v, for each incident edge e = (v, w), $w \notin S$, update

$$\pi(w) = \min \{ \ \pi(w), \ \pi(v) + \ell_e \}.$$

Efficient implementation. Maintain a priority queue of unexplored nodes, prioritized by $\pi(v)$.

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



Chapter 4

Greedy Algorithms

Slides by Kevin Wayne.
Copyright © 2005 Pearson-Addison Wesley.
All rights reserved.