MA 515: Introduction to Algorithms & MA353 : Design and Analysis of Algorithms [3-0-0-6]

Lecture 24

http://www.iitg.ernet.in/psm/indexing_ma353/y09/index.html

Partha Sarathi Manal

psm@iitg.ernet.in

Dept. of Mathematics, IIT Guwahati

Mon 10:00-10:55 Tue 11:00-11:55 Fri 9:00-9:55

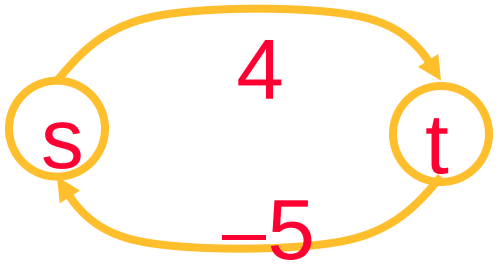
Class Room : 2101

Shortest Paths in a Graph

- Two important *single-source* shortest path algorithms:
  - Dijkstra's algorithm
  - Bellman-Ford algorithm

Single Source Shortest Path Problem

- Given: directed or undirected graph G = (V,E,w) and source node s in V
- Find: For each t in V, a path in G from s to t with minimum weight
- Warning! Negative weights are a problem:



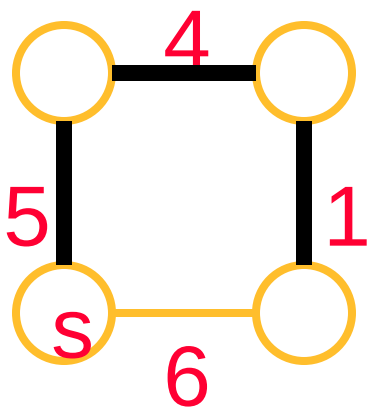
Shortest Path Tree

- Result of a SSSP algorithm can be viewed as a tree rooted at the source
- Why not use breadth-first search?
- Works fine if all weights are the same:
  - weight of each path is (a multiple of) the number of edges in the path
- Doesn't work when weights are different

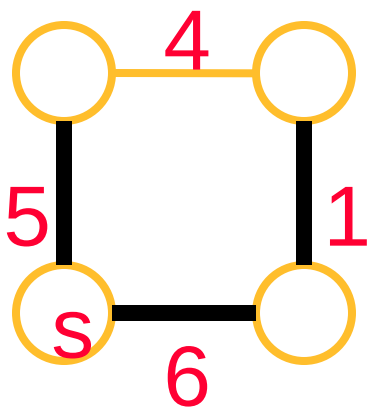
Dijkstra's SSSP Algorithm

- Assumes all edge weights are nonnegative
- Similar to Prim's MST algorithm
- Start with source node s and iteratively construct a tree rooted at s
- Each node keeps track of tree node that provides cheapest path from s (not just cheapest path from any tree node)
- At each iteration, include the node whose cheapest path from s is the overall cheapest

Prim's vs. Dijkstra's



Prim's
MST



Dijkstra's
SSSP

Implementing Dijkstra's Alg.

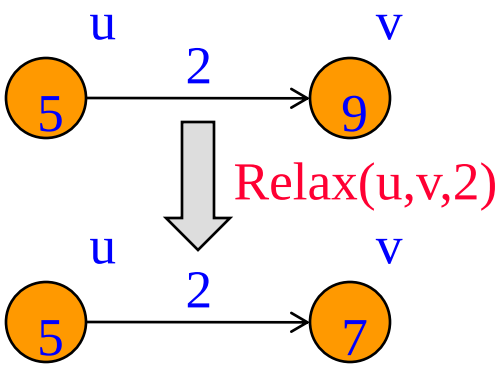
- How can each node u keep track of its best path from s?
- Keep an estimate, d[u], of shortest path distance from s to u
- Use d as a key in a priority queue
- When u is added to the tree, check each of u's neighbors v to see if u provides v with a cheaper path from s:
  - compare d[v] to d[u] + w(u,v)

Dijkstra's Algorithm

- input: G = (V,E,w) and source node s
- d[s] := 0 // initialization
- d[v] := ∞ for all other nodes v
- Parent(v):=NIL
- initialize priority queue Q to contain all nodes using d values as keys
- while Q is not empty do
  - u := extract-min(Q)
  - for each neighbor v of u do

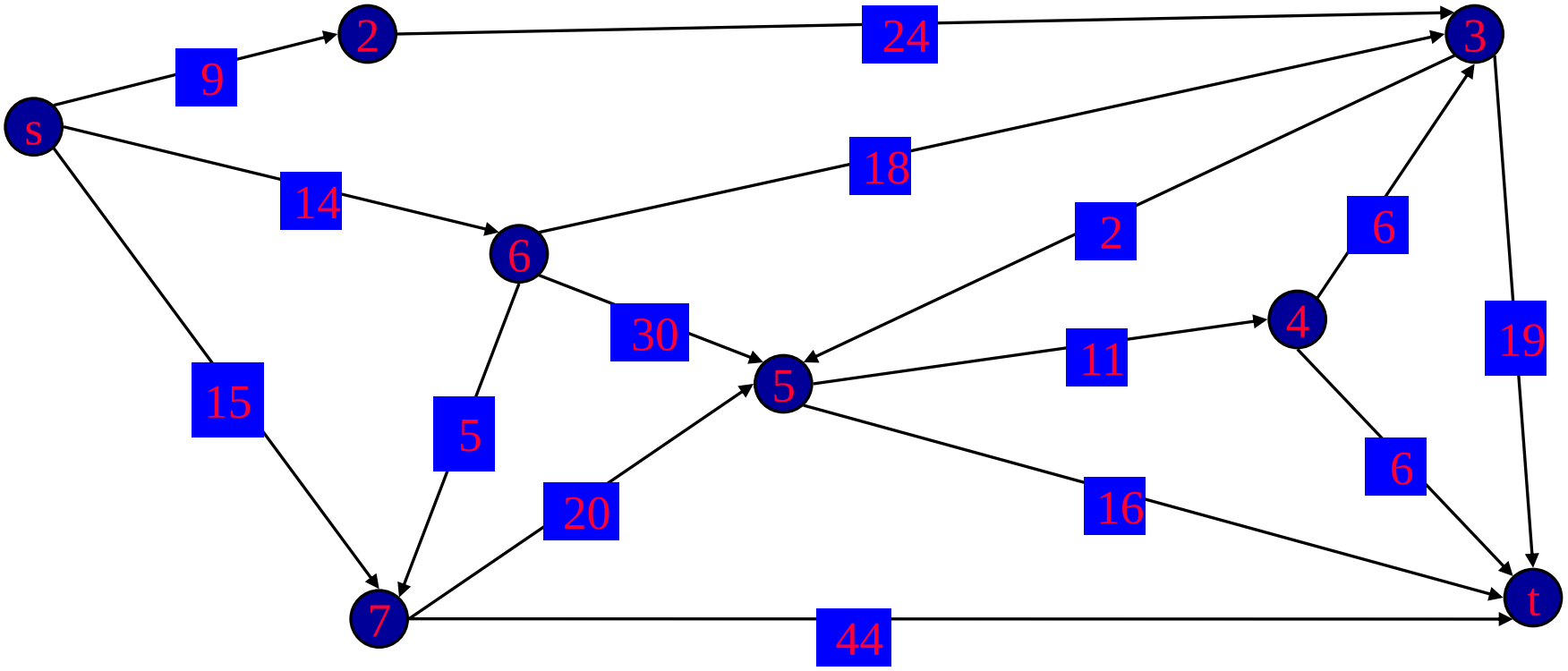
Relax(u,v,w)

- if d[u] + w(u,v) < d[v] then
  - d[v] := d[u] + w(u,v)
  - decrease-key(Q,v,d[v])
  - parent(v) := u



Dijkstra's Algorithm Example 1

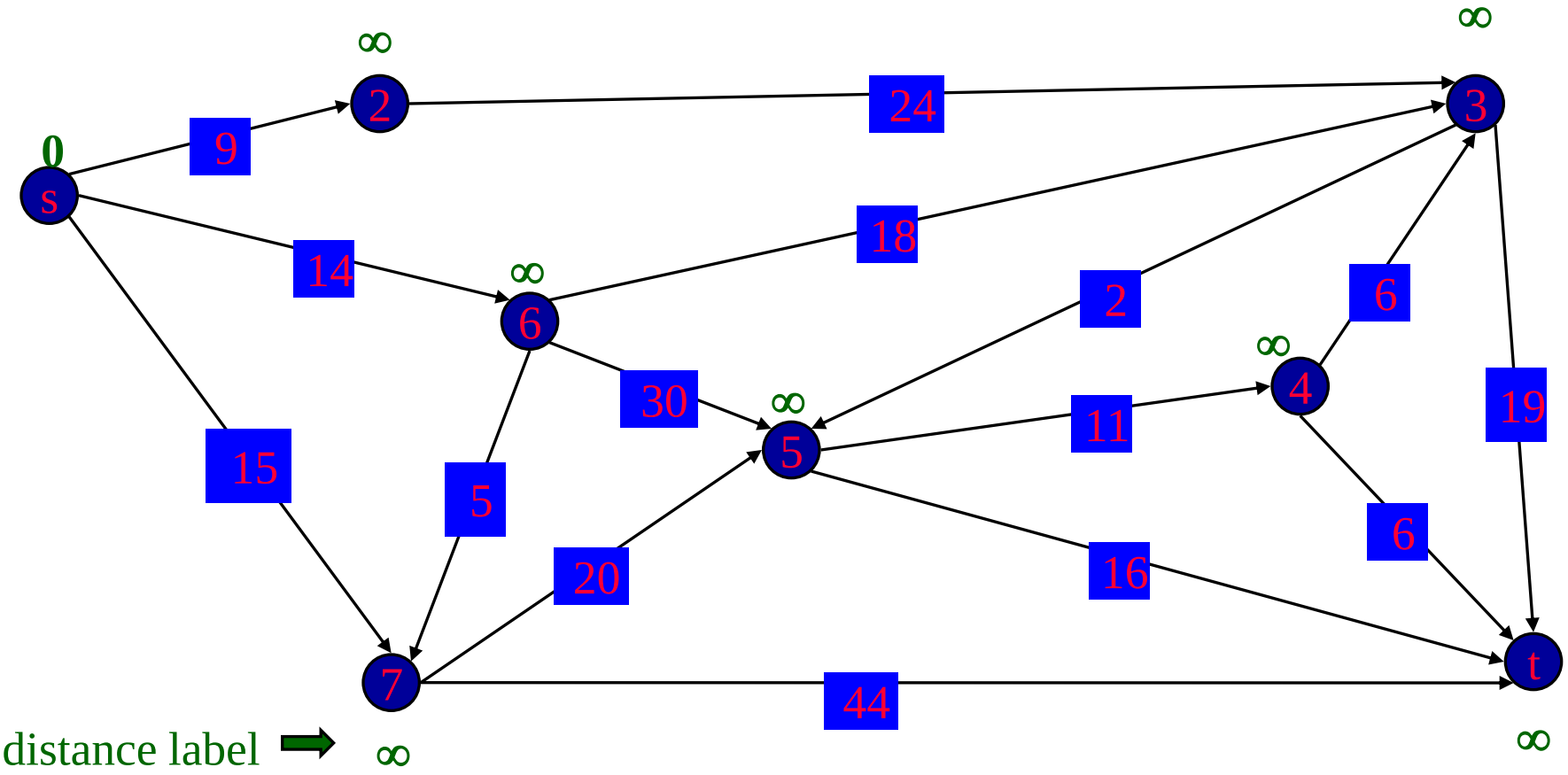
- Find shortest path from s to all other nodes.



Dijkstra's Algorithm Example 1

S = { }

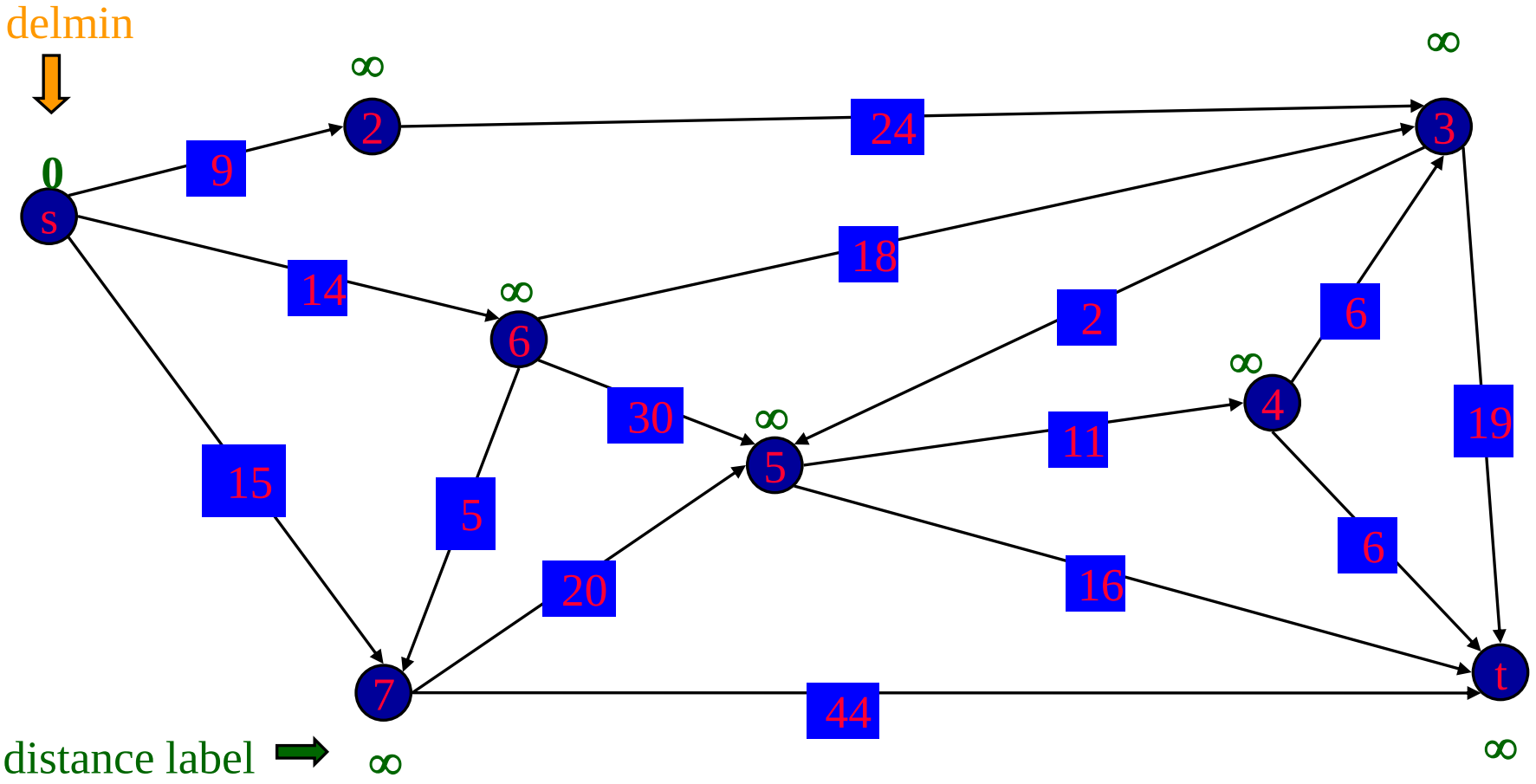
PQ = { s, 2, 3, 4, 5, 6, 7, t }



Dijkstra's Algorithm Example 1

S = { }

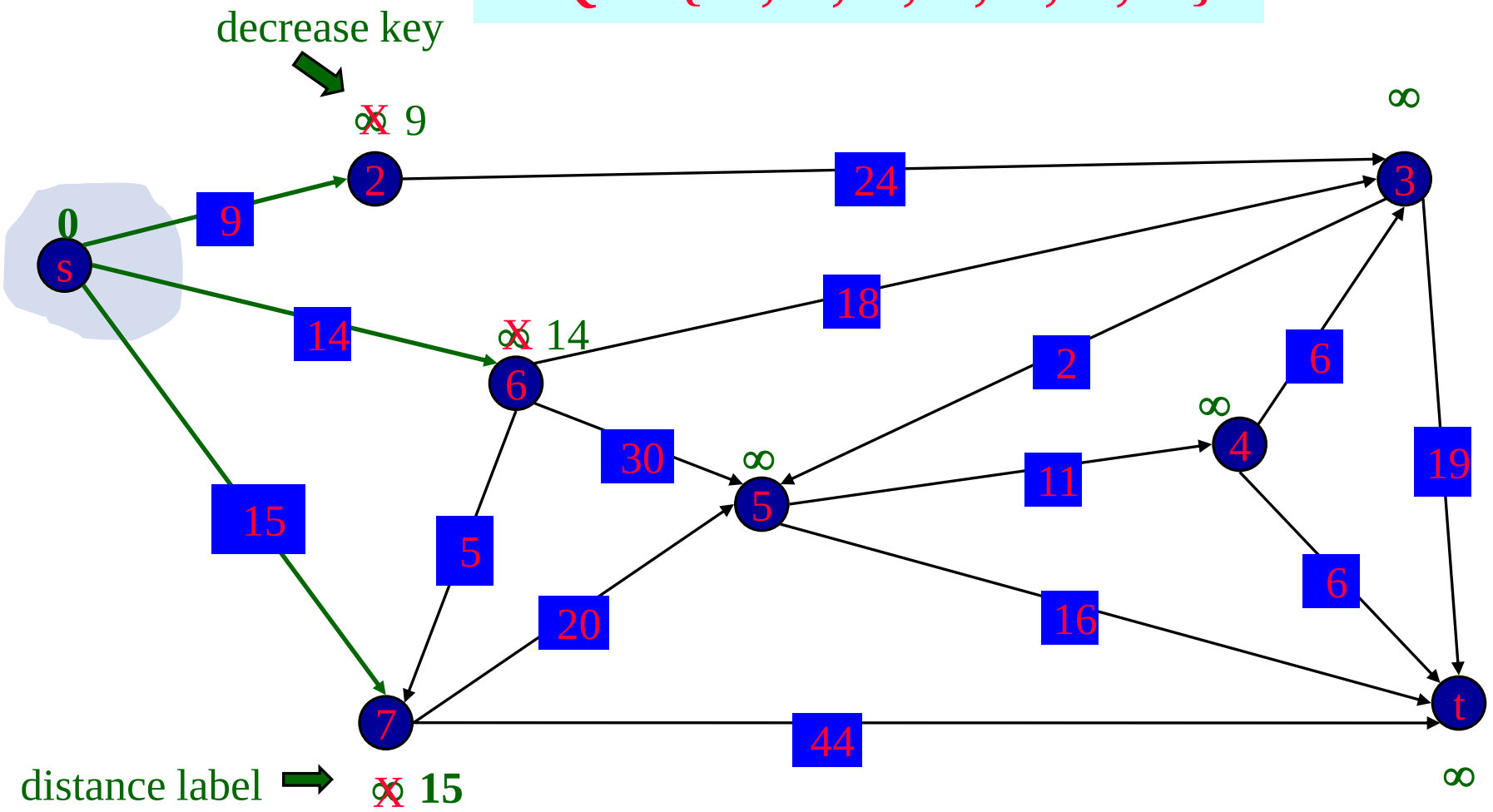
PQ = { s, 2, 3, 4, 5, 6, 7, t }



Dijkstra's Algorithm Example 1

S = { s }

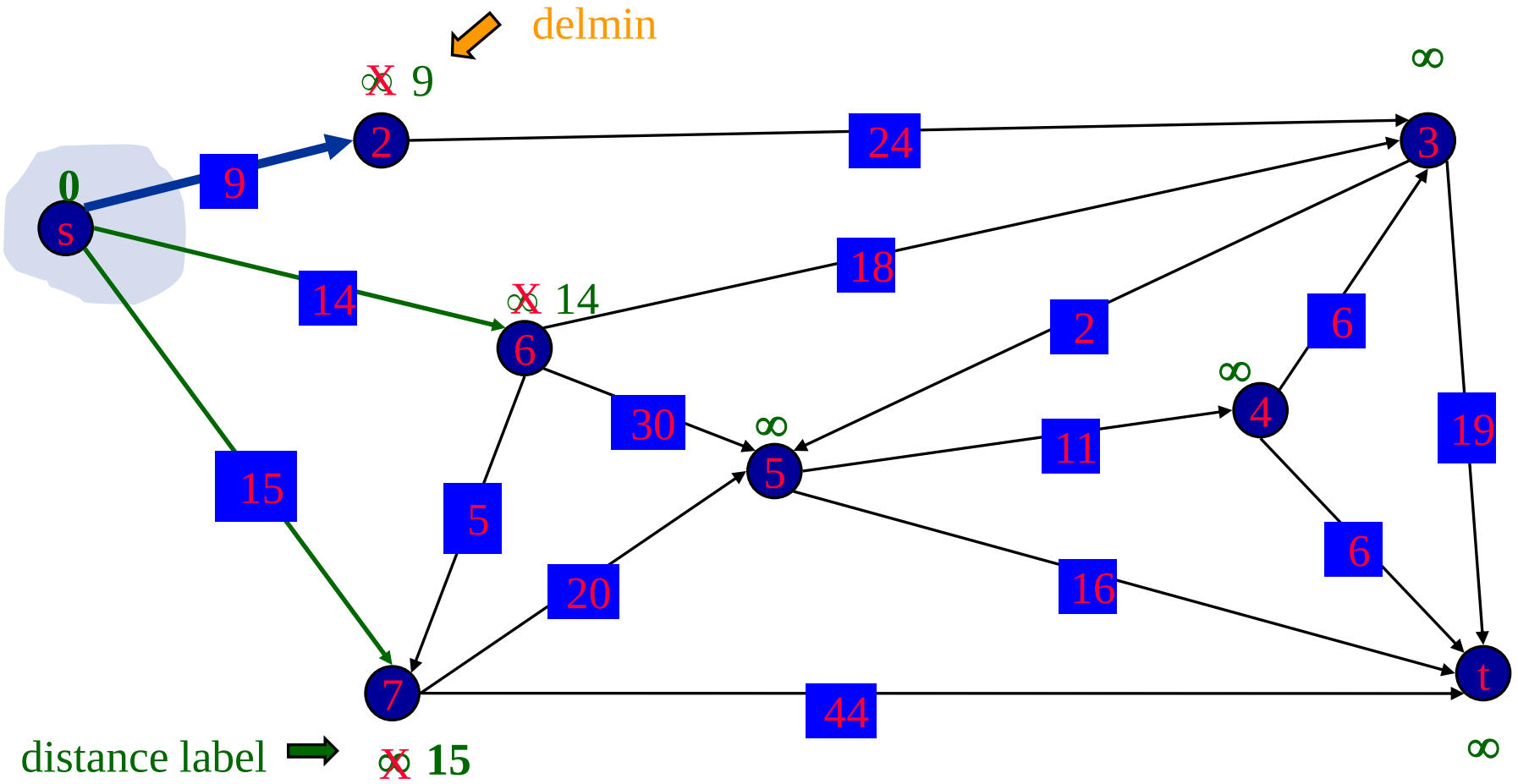
PQ = { 2, 3, 4, 5, 6, 7, t }



Dijkstra's Algorithm Example 1

S = { s }

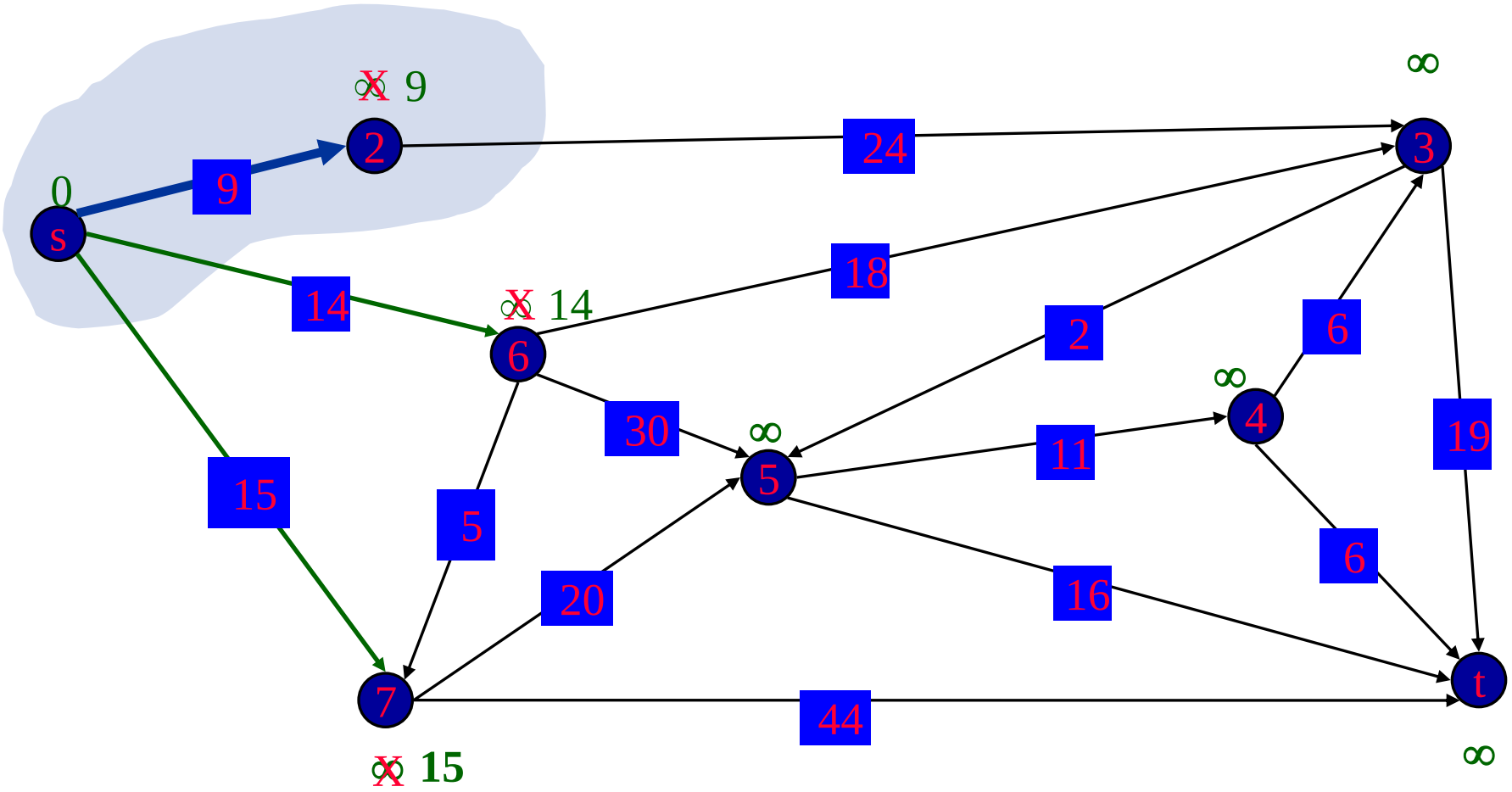
PQ = { 2, 3, 4, 5, 6, 7, t }



Dijkstra's Algorithm Example 1

S = { s, 2 }

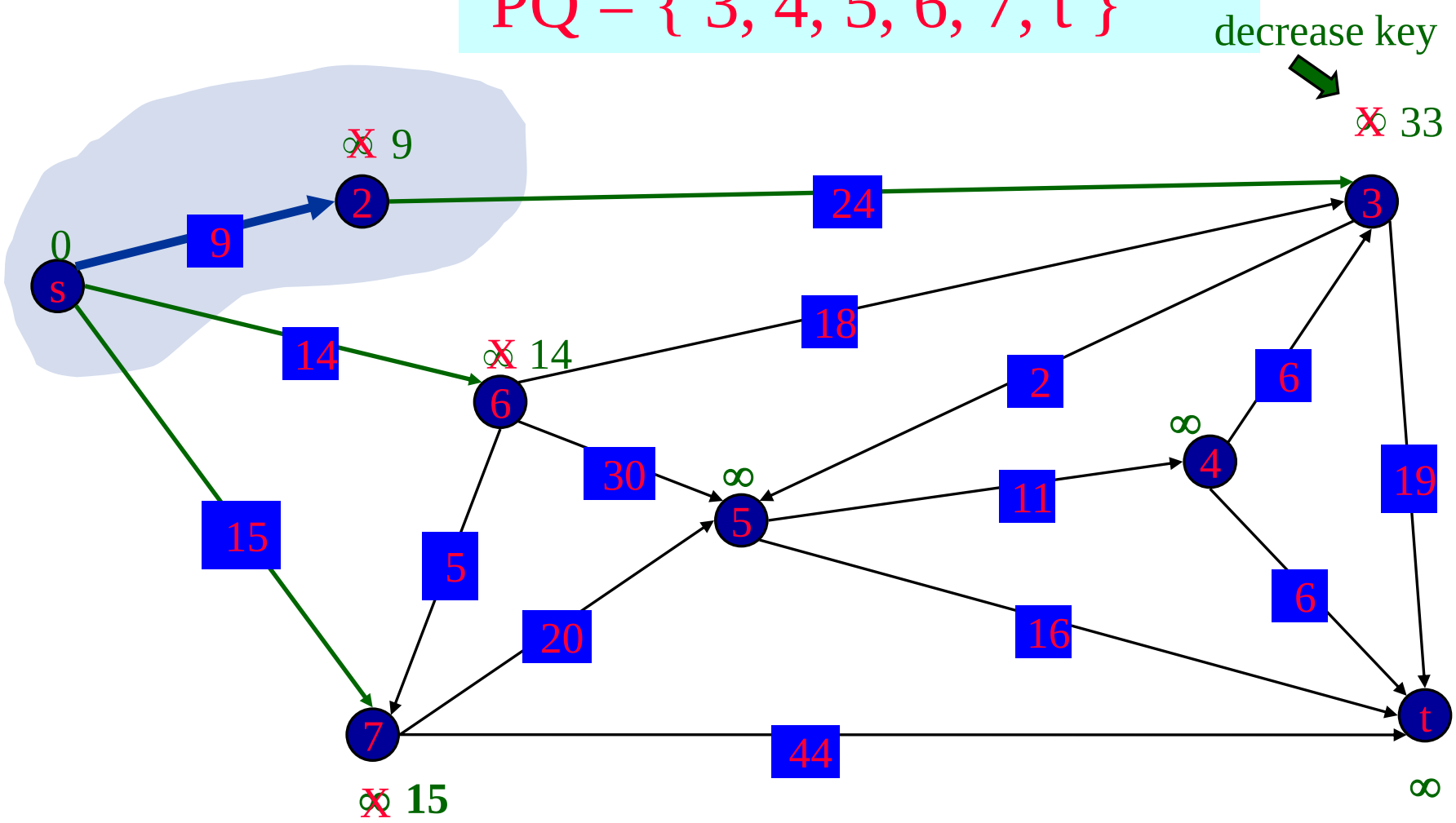
PQ = { 3, 4, 5, 6, 7, t }



Dijkstra's Algorithm Example 1

S = { s, 2 }

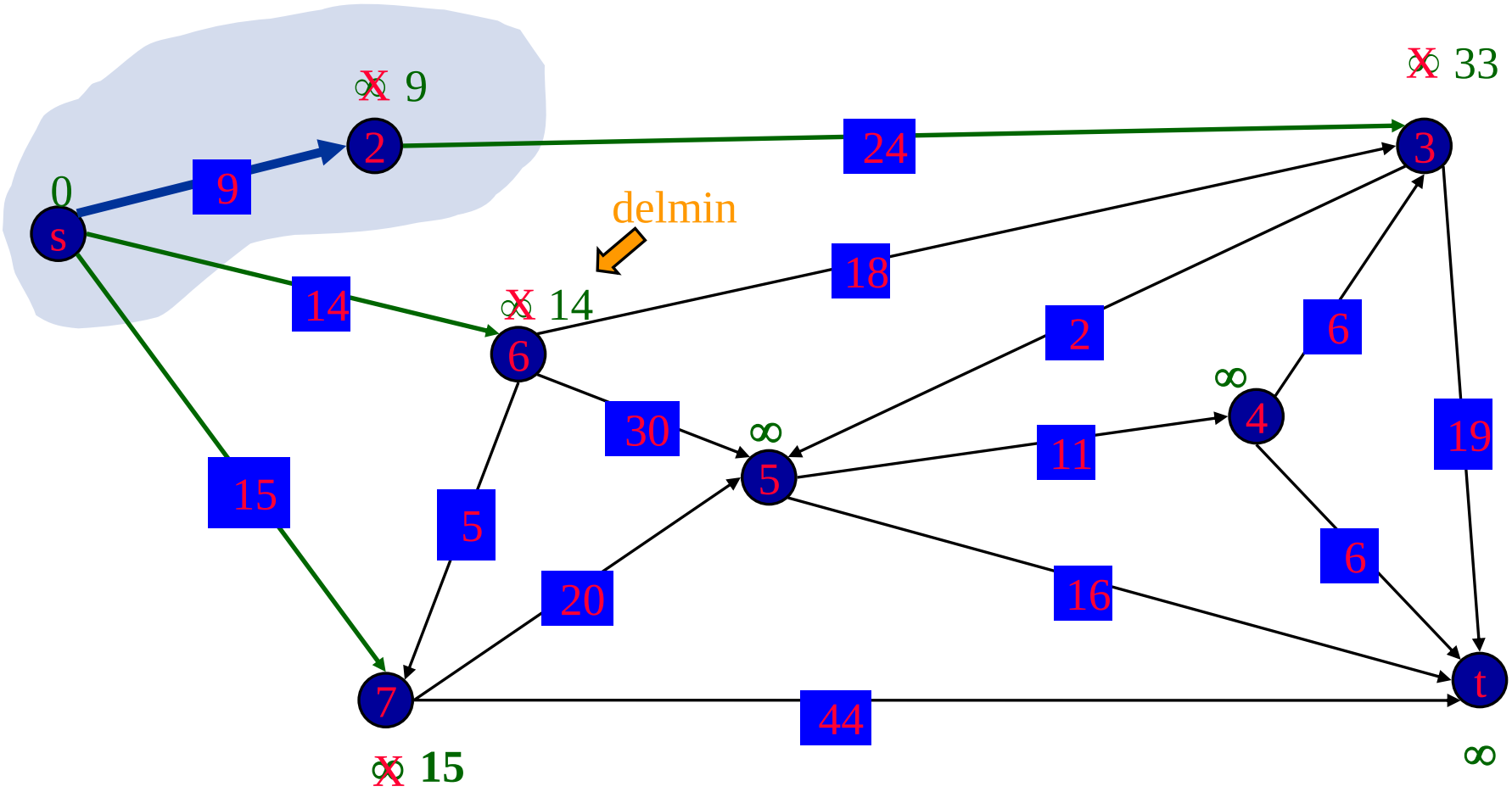
PQ = { 3, 4, 5, 6, 7, t }



Dijkstra's Algorithm Example 1

S = { s, 2 }

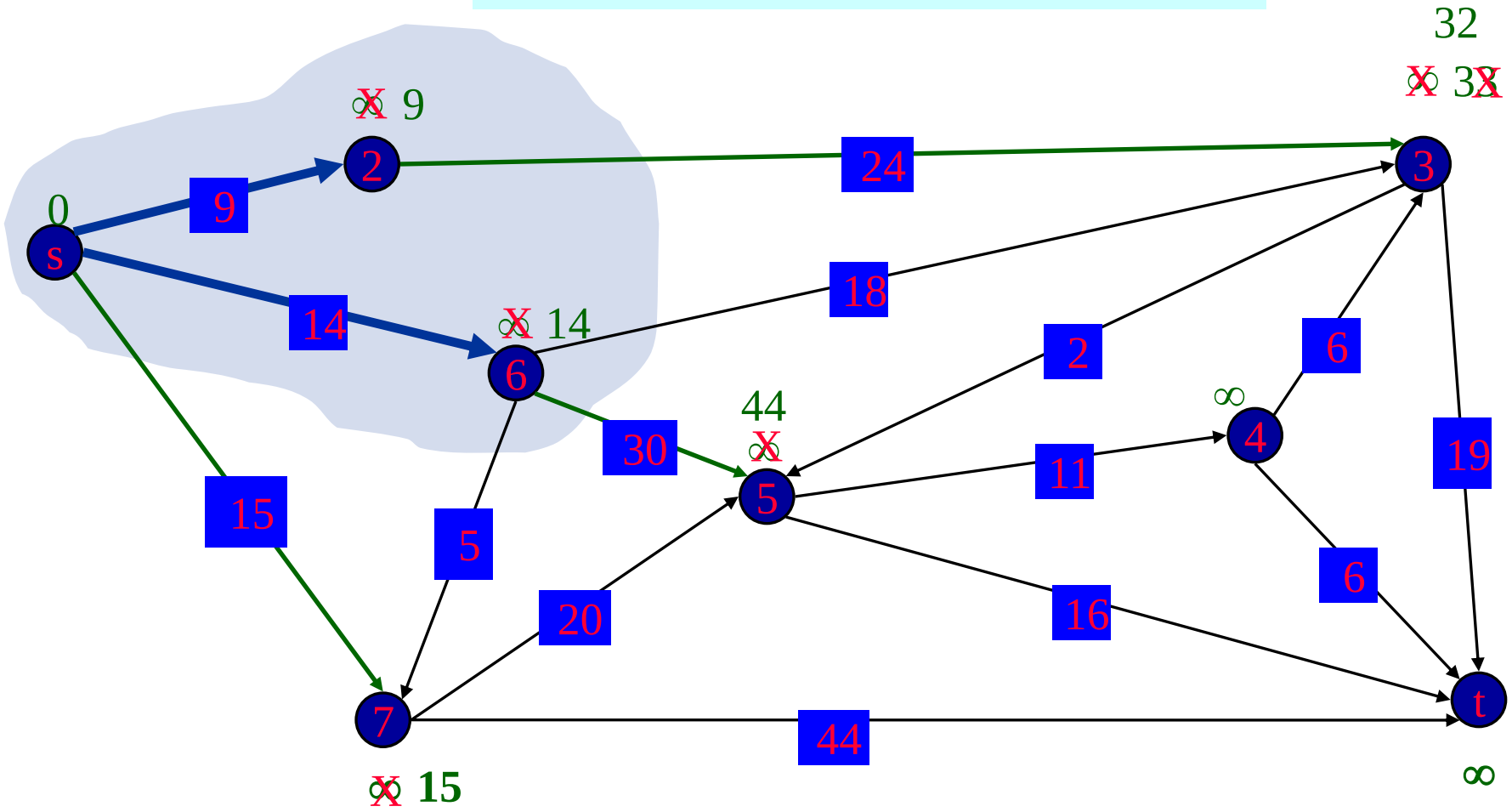
PQ = { 3, 4, 5, 6, 7, t }



Dijkstra's Algorithm Example 1

S = { s, 2, 6 }

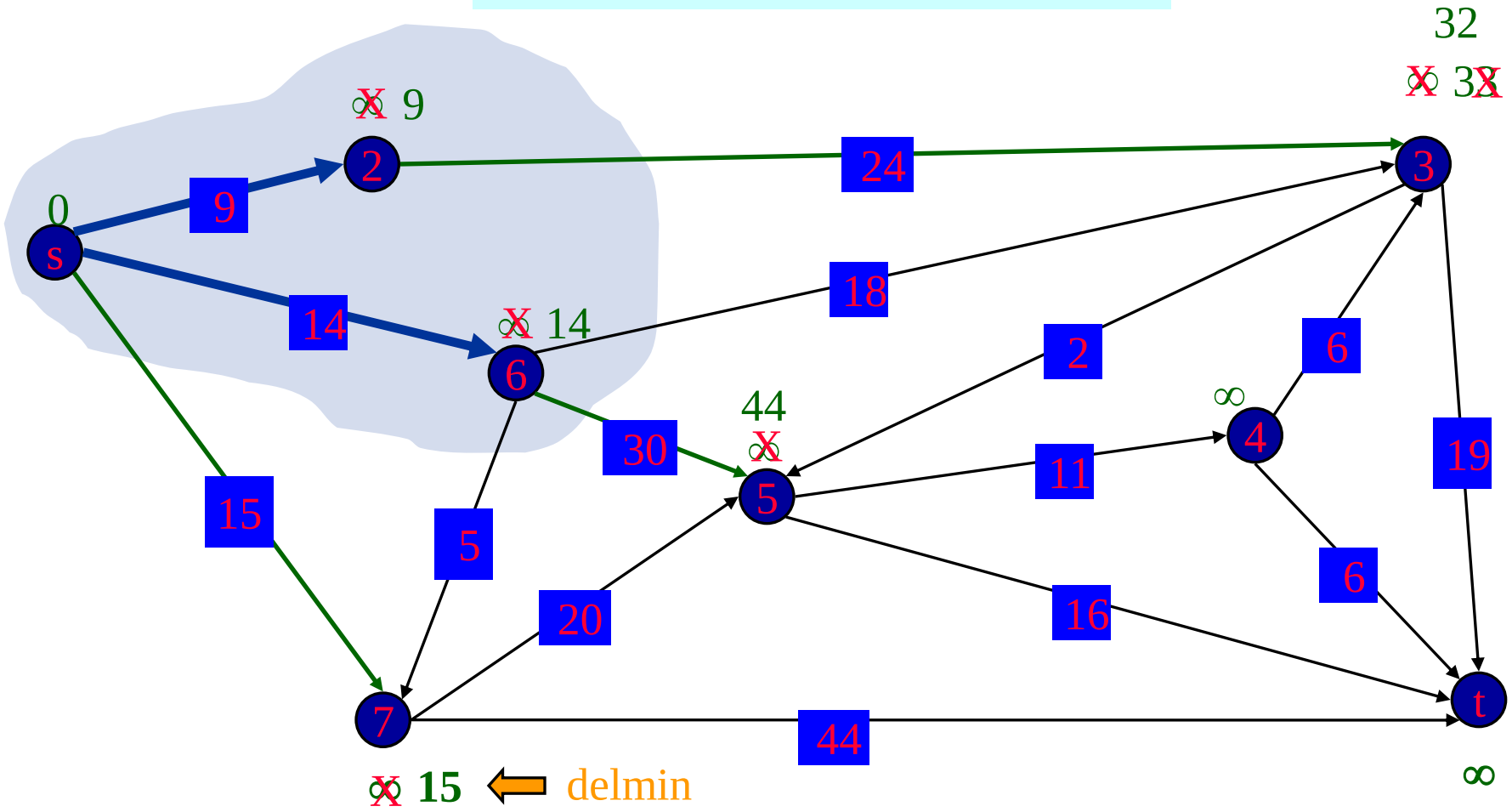
PQ = { 3, 4, 5, 7, t }



Dijkstra's Algorithm Example 1

S = { s, 2, 6 }

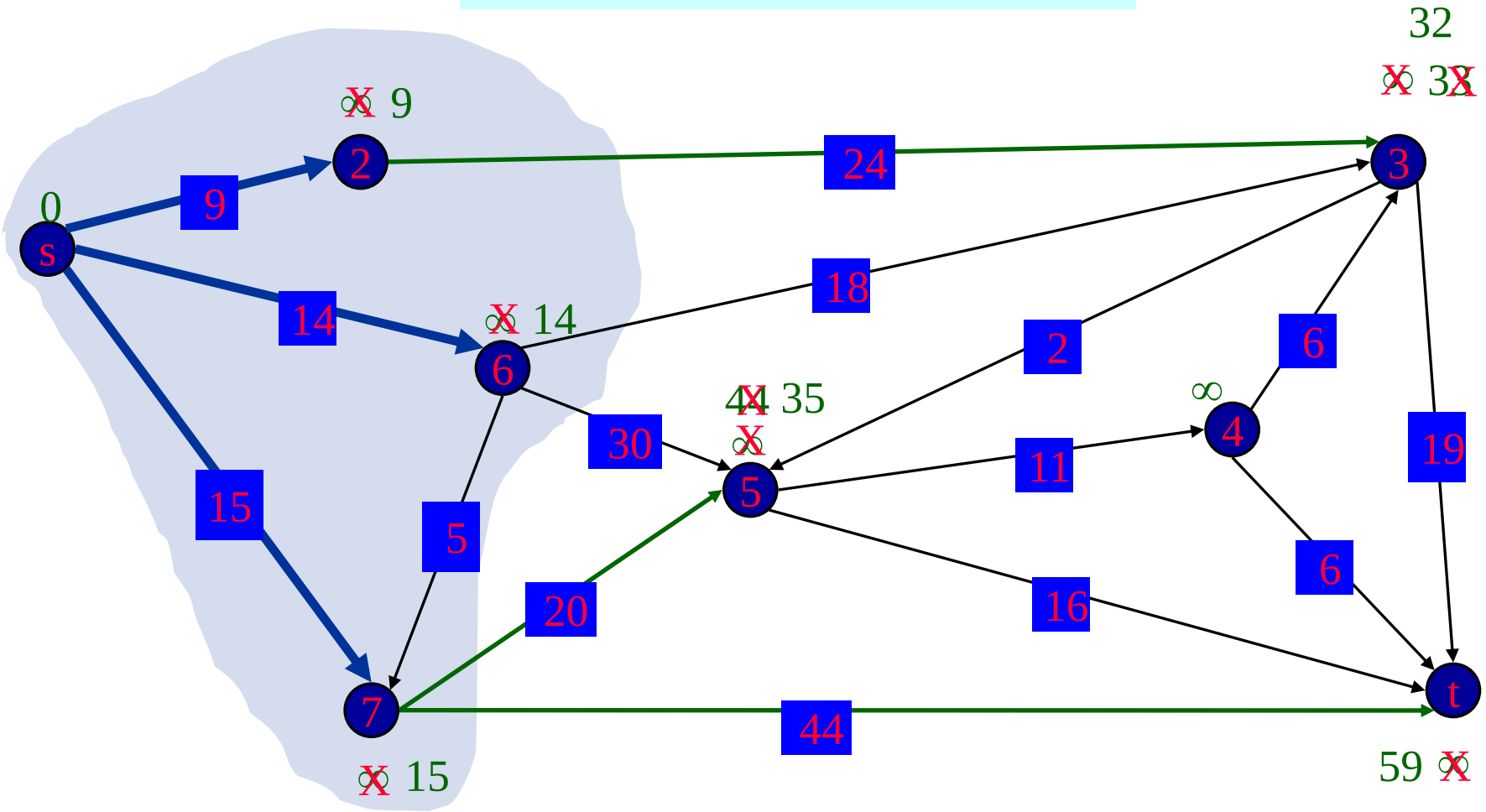
PQ = { 3, 4, 5, 7, t }



Dijkstra's Algorithm Example 1

S = { s, 2, 6, 7 }

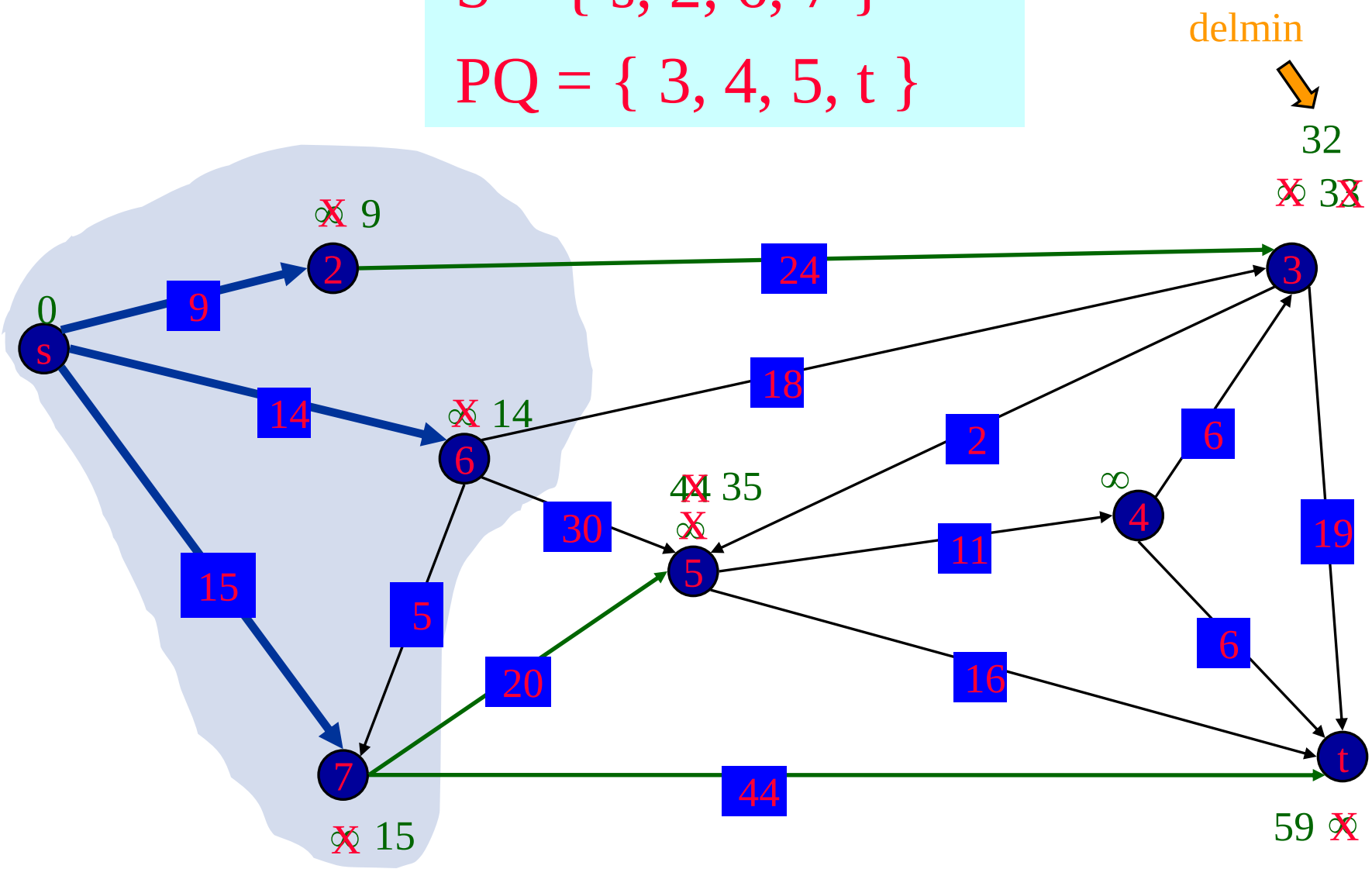
PQ = { 3, 4, 5, t }



Dijkstra's Algorithm Example 1

S = { s, 2, 6, 7 }

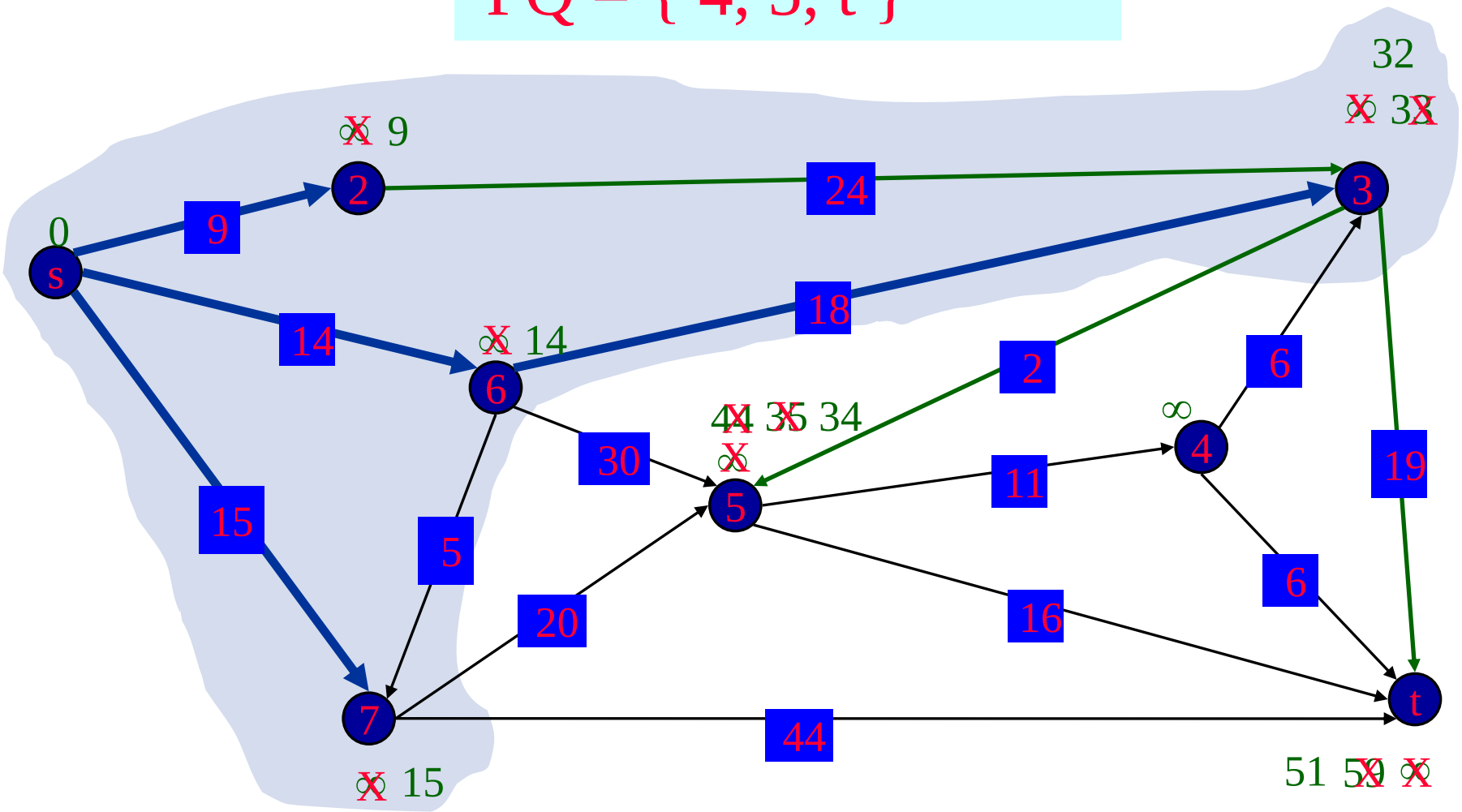
PQ = { 3, 4, 5, t }



Dijkstra's Algorithm Example 1

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

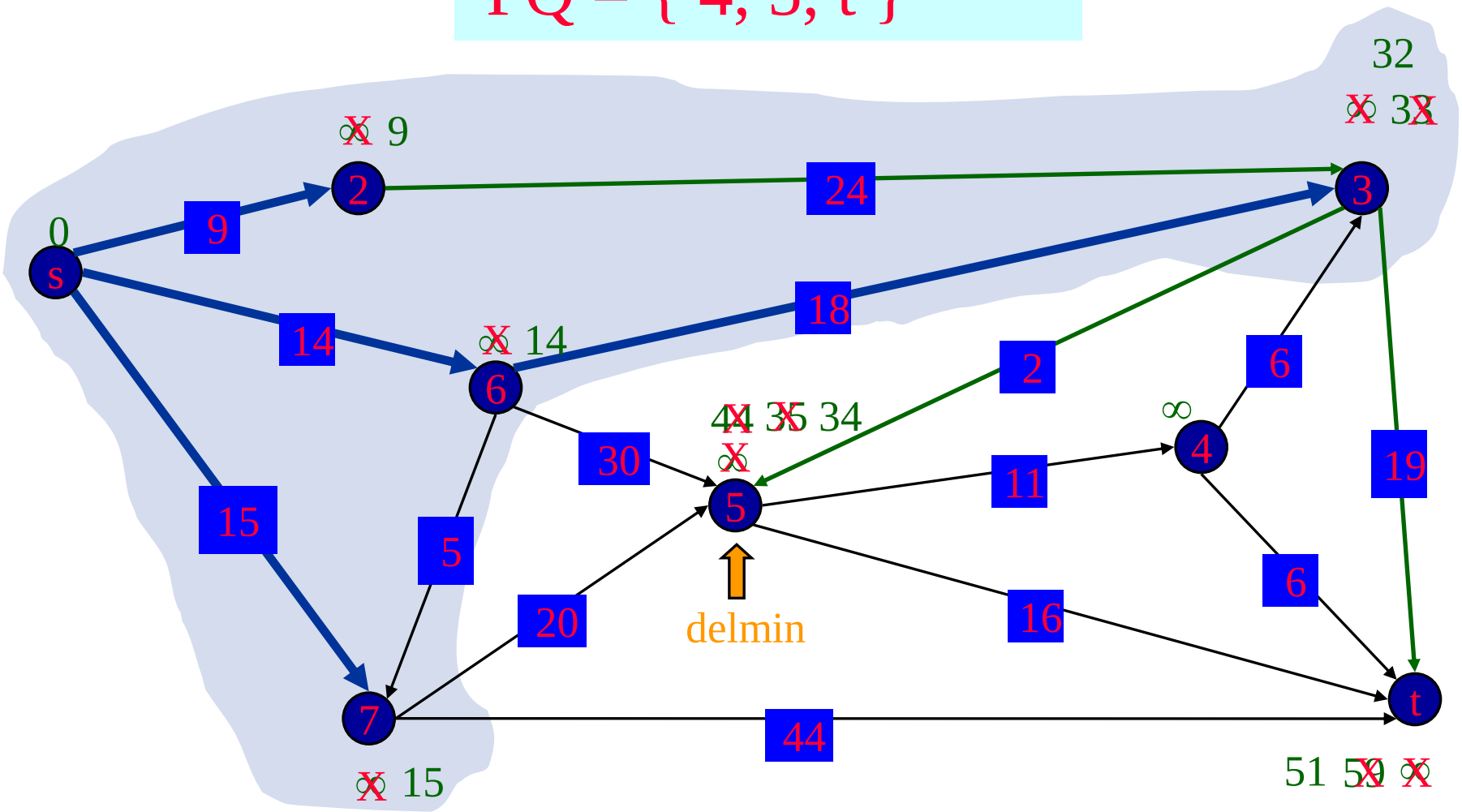
PQ = { 4, 5, t }



Dijkstra's Algorithm Example 1

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

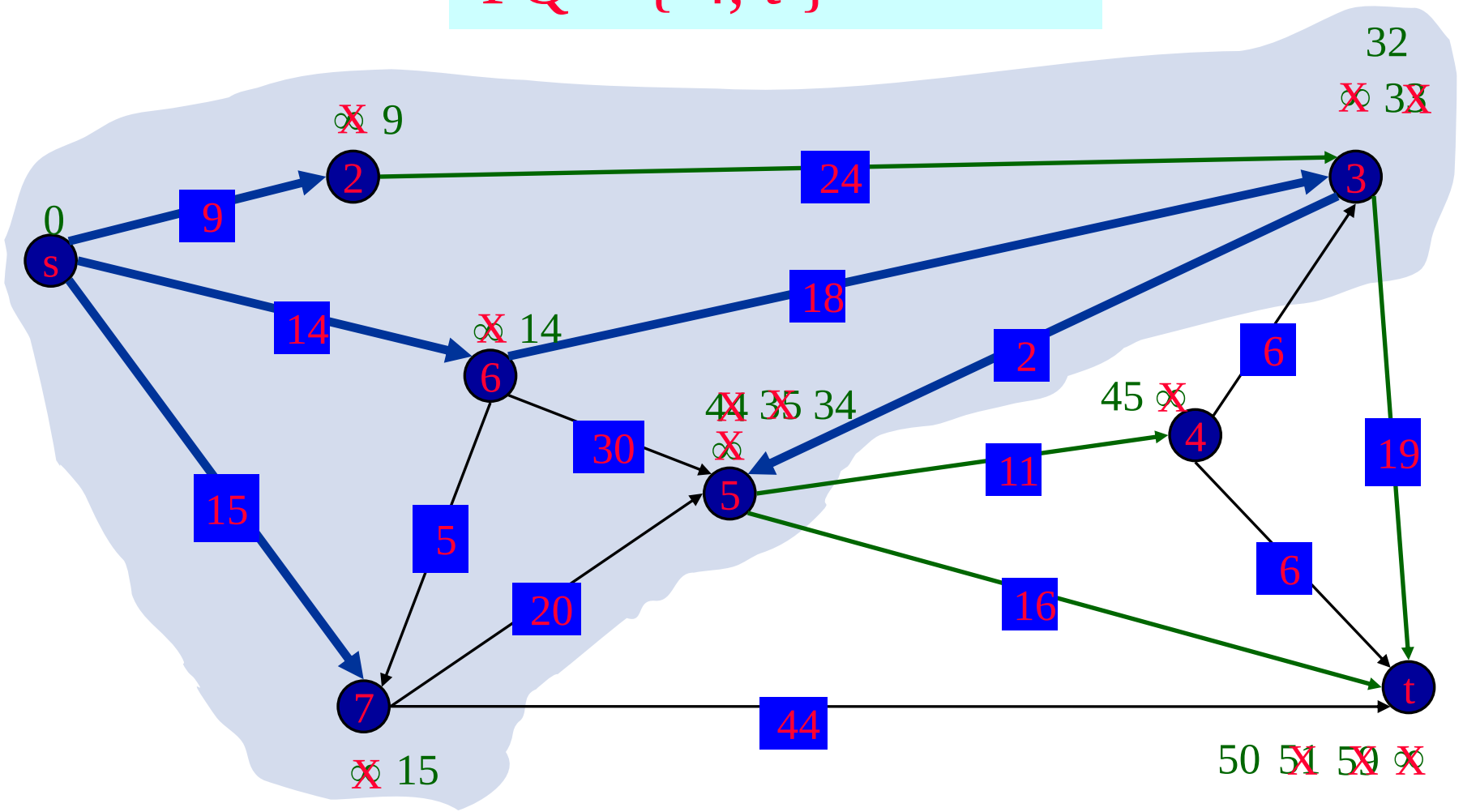
PQ = { 4, 5, t }



Dijkstra's Algorithm Example 1

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

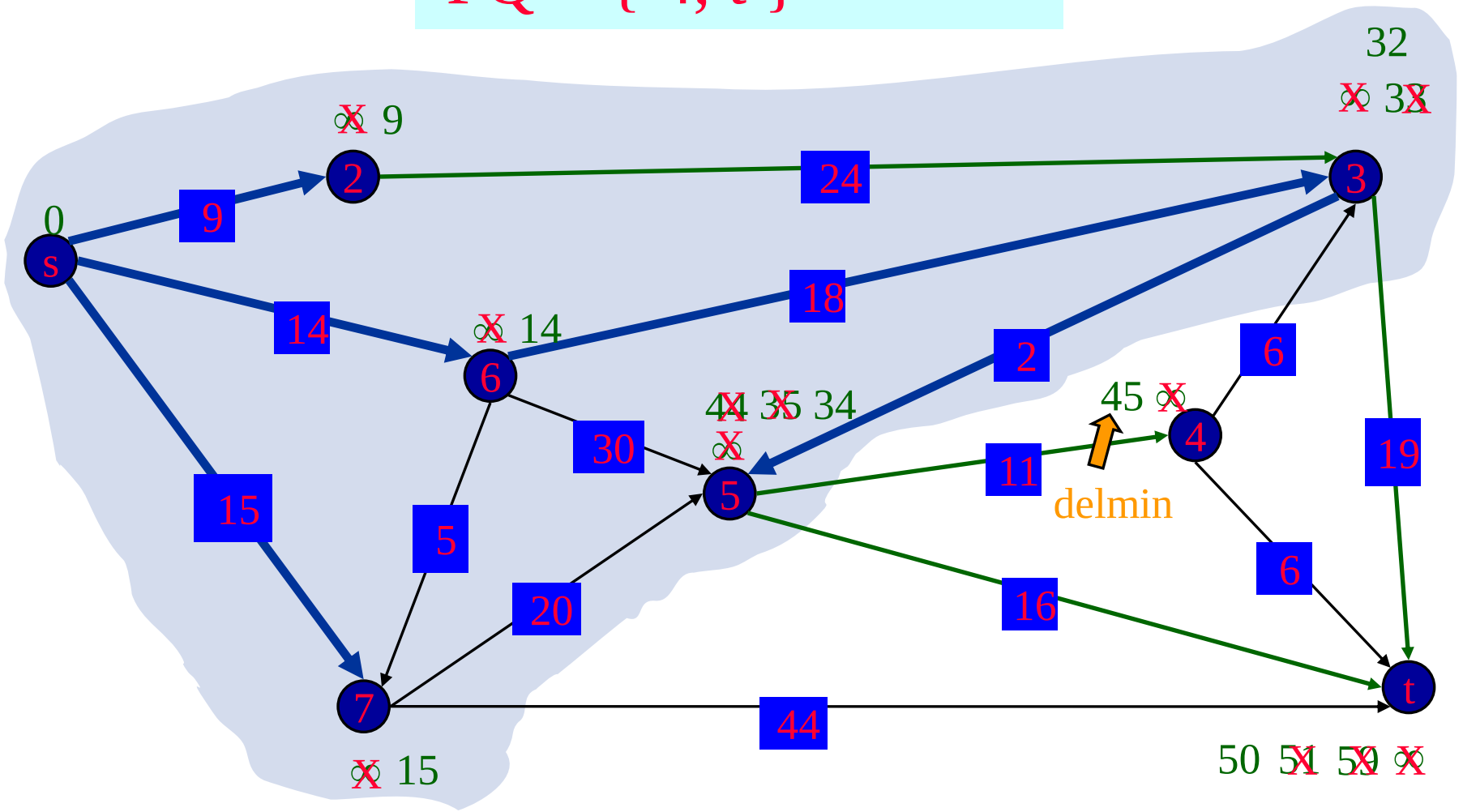
PQ = { 4, t }



Dijkstra's Algorithm Example 1

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

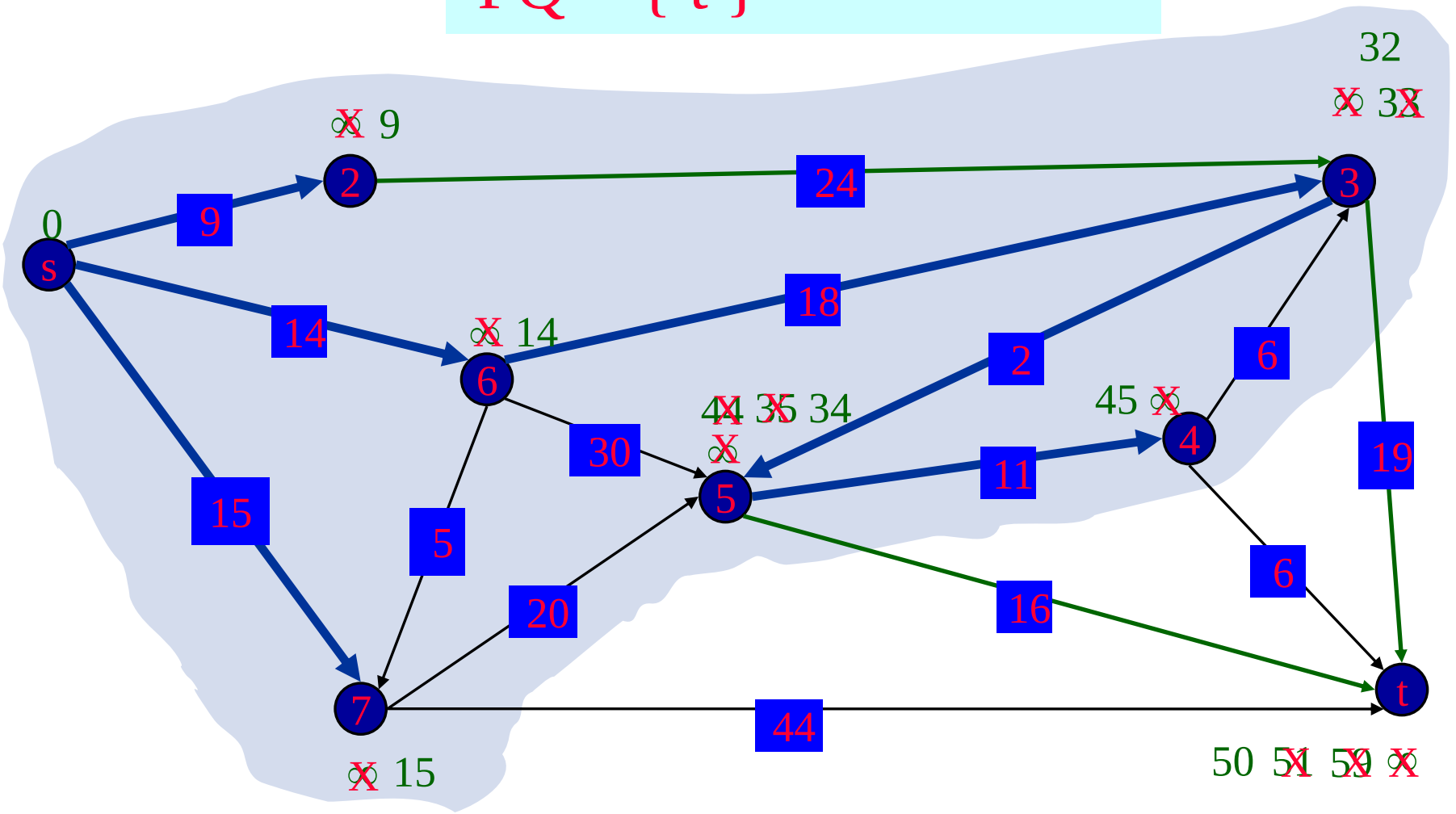
PQ = { 4, t }



Dijkstra's Algorithm Example 1

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

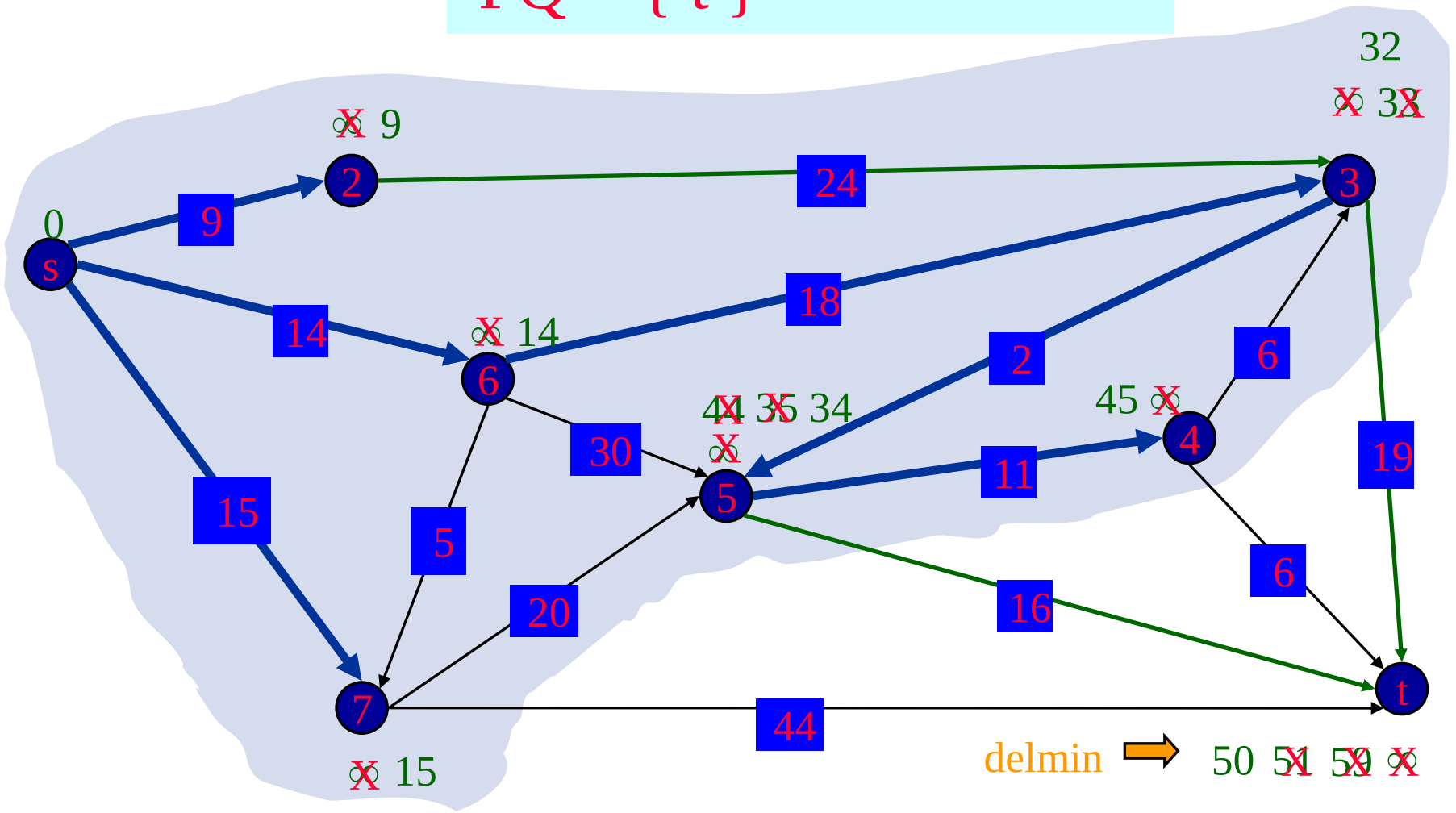
PQ = { t }



Dijkstra's Algorithm Example 1

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

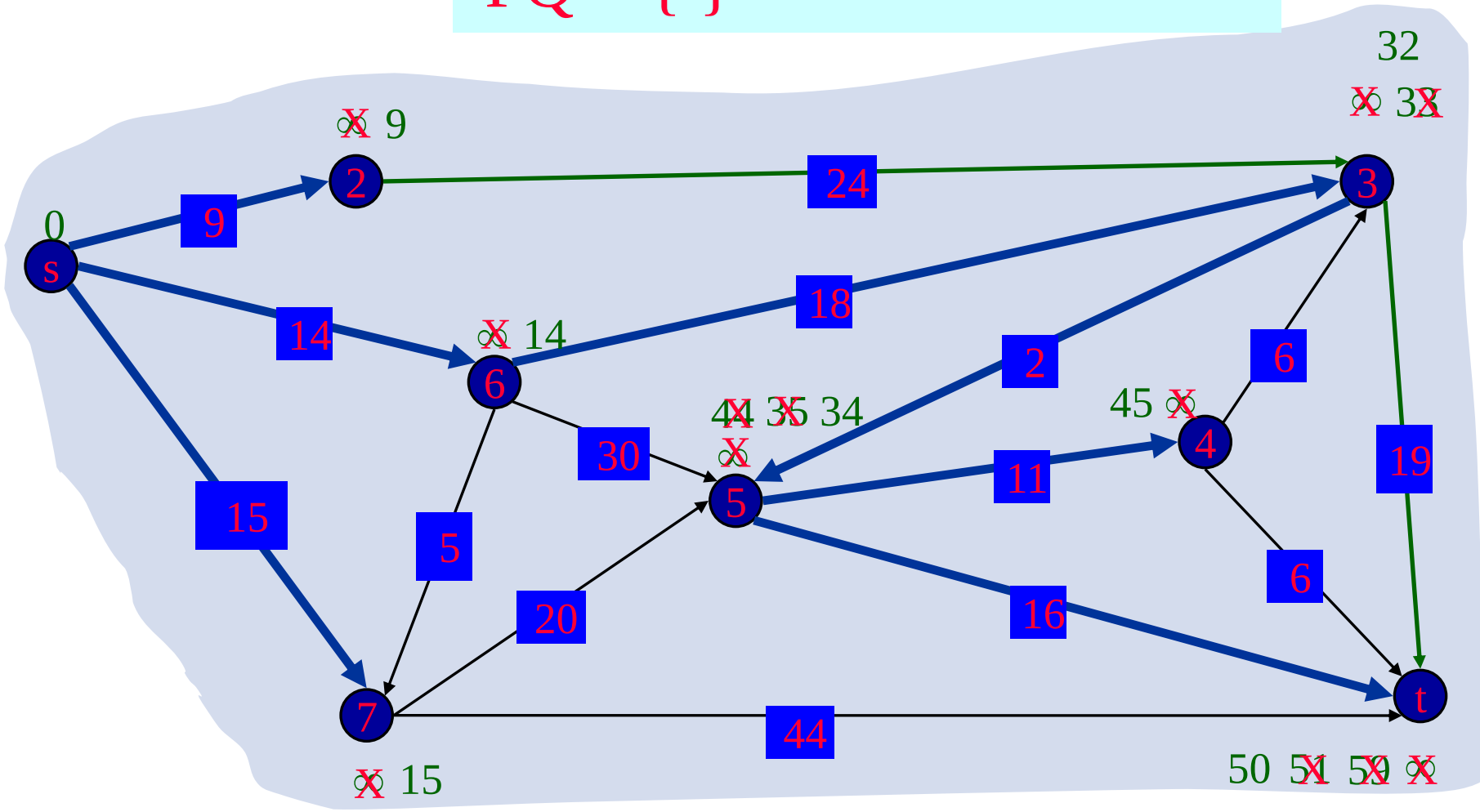
PQ = { t }



Dijkstra's Algorithm Example 1

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

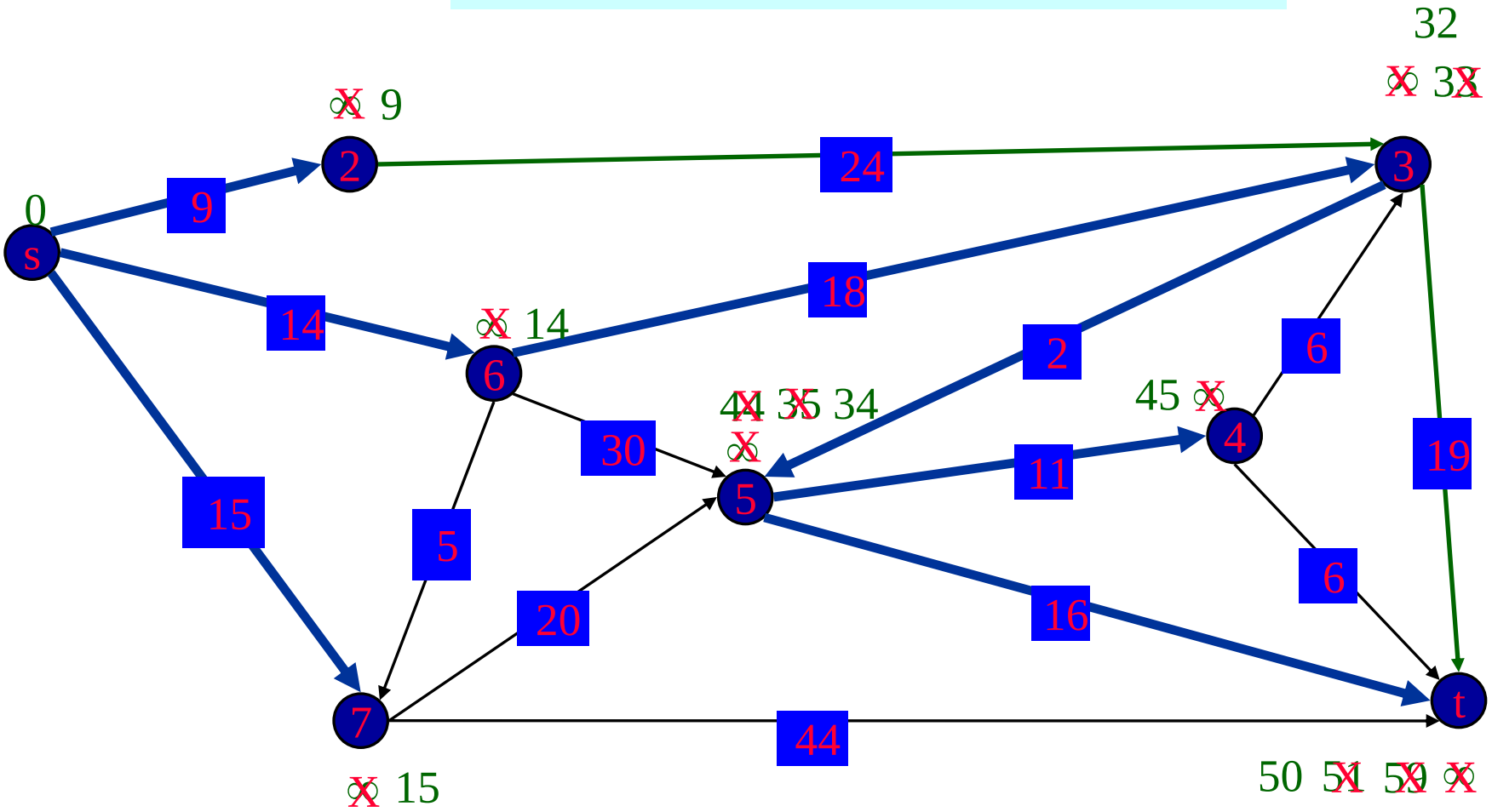
PQ = { }



Dijkstra's Algorithm Example 1

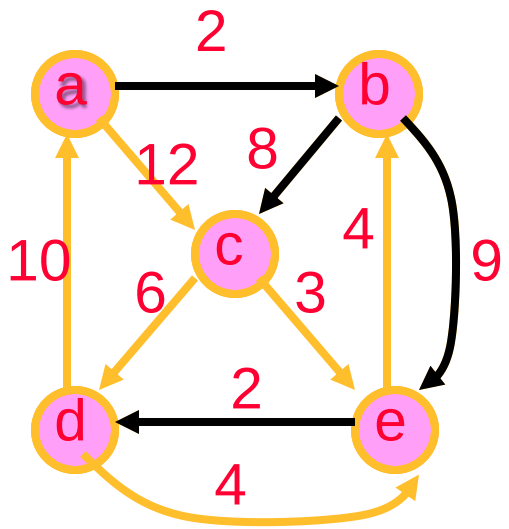
S = { s, 2, 3, 4, 5, 6, 7, t }

PQ = { }



Dijkstra's Algorithm Example 2

source is node a




```
d[s] := 0
d[v] := infinity for all other nodes v
while Q is not empty do
        u := extract-min(Q)
        for each neighbor v of u do
                if d[u] + w(u,v) < d[v] then
                        d[v] := d[u] + w(u,v)
                        decrease-key(Q,v,d[v])
                        parent(v) := u
```

iteration

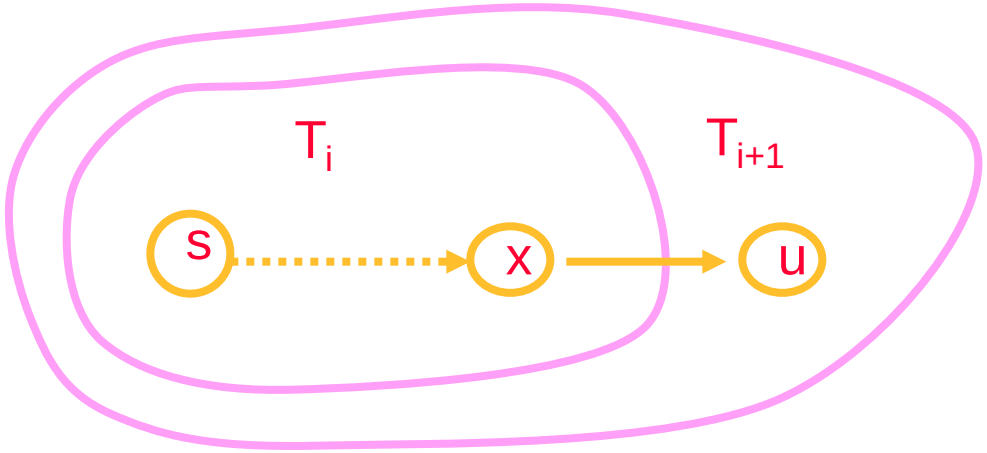
	0	1	2	3	4	5
Q	abcde	bcde	cde	de	d	Ø
d[a]	0	0	0	0	0	0
d[b]	∞	2	2	2	2	2
d[c]	∞	12	10	10	10	10
d[d]	∞	∞	∞	16	13	13
d[e]	∞	∞	11	11	11	11

Correctness of Dijkstra's Alg.

- Let T_i be the tree constructed after i-th iteration of while loop:
  - nodes not in Q
  - edges indicated by parent variables
- Show by induction on i that the path in T_i from s to u is a shortest path and has distance d[u], for all u in T_i.
- Basis: i = 1. s is the only node in T_1 and d[s] = 0.

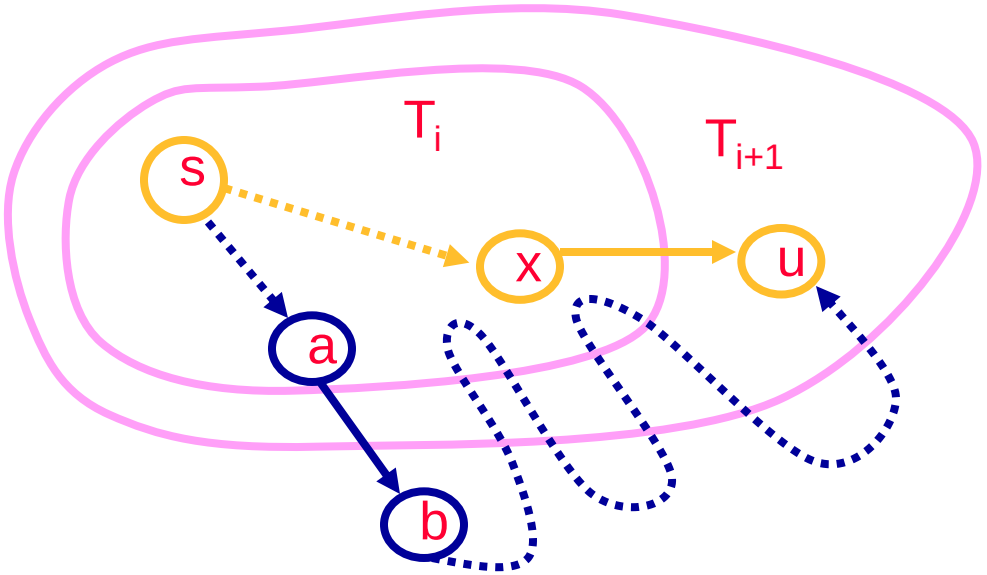
Correctness of Dijkstra's Alg.

- Induction: Assume T_i is a correct shortest path tree and show for T_{i+1}.
- Let u be the node added in iteration i.
- Let x = parent(u).



Need to show path in T_{i+1} from s to u is a shortest path, and has distance d[u]

Correctness of Dijkstra's Alg

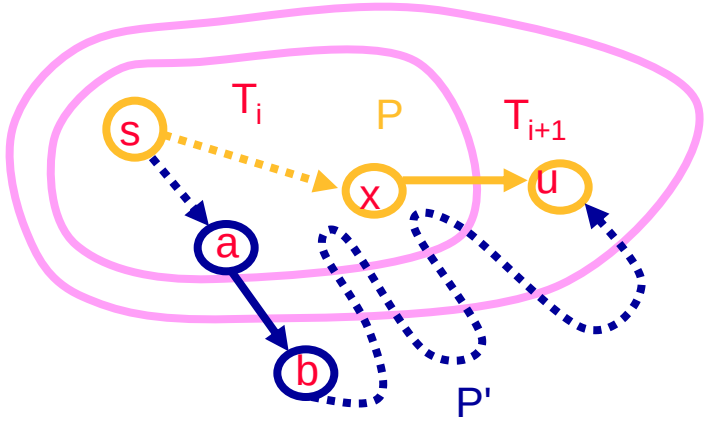


P, path in T_{i+1} from s to u

P', another path from s to u

(a,b) is first edge in P' that leaves T_i

Correctness of Dijkstra's Alg



Let P1 be part of P' before (a,b.)

Let P2 be part of P' after (a,b).

w(P') = w(P1) + w(a,b) + w(P2)

≥ w(P1) + w(a,b) (nonneg wts)

≥ wt of path in T_i from s to a + w(a,b) (inductive hypothesis)

≥ w(s->x path in T_i) + w(x,u) (alg chose u in iteration i and d-values are accurate, by inductive hypothesis

= w(P).

So P is a shortest path, and d[u] is accurate after iteration i+1.

Running Time of Dijstra's Alg.

- initialization: insert each node once
  - $O(V\ T_{ins})$
- O(V) iterations of while loop
  - one extract-min per iteration => $O(V\ T_{ex})$
  - for loop inside while loop has variable number of iterations…
- For loop has O(E) iterations total
  - one decrease-key per iteration => $O(E\ T_{dec})$
- Total is $O(V\ (T_{ins} + T_{ex}) + E\ T_{dec})$

Using Fancier Heap Implementations

- $O(V(T_{ins} + T_{ex}) + E \bullet T_{dec})$
- If priority queue is implemented with a binary heap, then
  - $T_{ins} = T_{ex} = T_{dec} = O(\log V)$
  - total time is $O(E \log V)$
- If priority queue is implemented with a Fibonacci heap, then
  - total time is $O(V \log V + E)$ [Exercises]