COMP 355
Advanced Algorithms

Dijkstra's Algorithm for Shortest Paths
Sections 4.4 (KT)

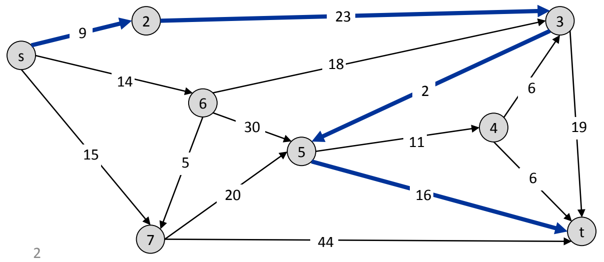
Shortest Path Problem

Shortest path network.

- Directed graph G = (V, E).
- Source s, destination 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
= 48.

Single Source Shortest Paths

Single Source Shortest Path Problem:

- Given a digraph G = (V, E)
- Numeric edge weights
- Source vertex, s ∈ V
- Determine the distance δ(s, v) from s to every vertex v in the graph

Are negative weight edges allowed? (could arise in financial transaction networks)

Dijkstra’s algorithm assumes no negative edge weights.

- Computing the distance from source to each vertex (not the actual path)

Shortest Paths and Relaxation

```
relax(u,v):
   if d[u] + w(u,v) < d[v]:       # is the path through u shorter?
      d[v] = d[u] + w(u,v)        # yes, then take it
      pred[v] = u                 # record that we go through u
```

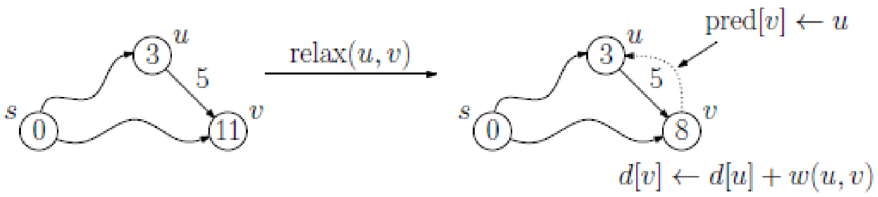


Fig. 15: Relaxation.

Dijkstra's Algorithm

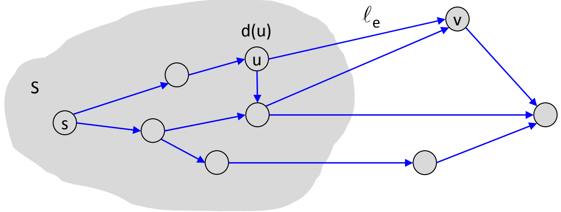
Dijkstra's algorithm.

- 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 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

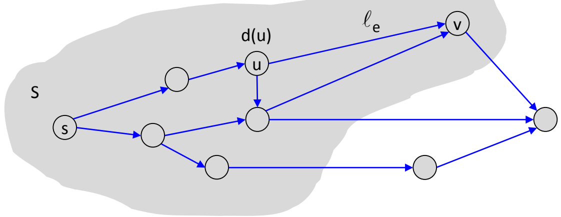
Dijkstra's algorithm.

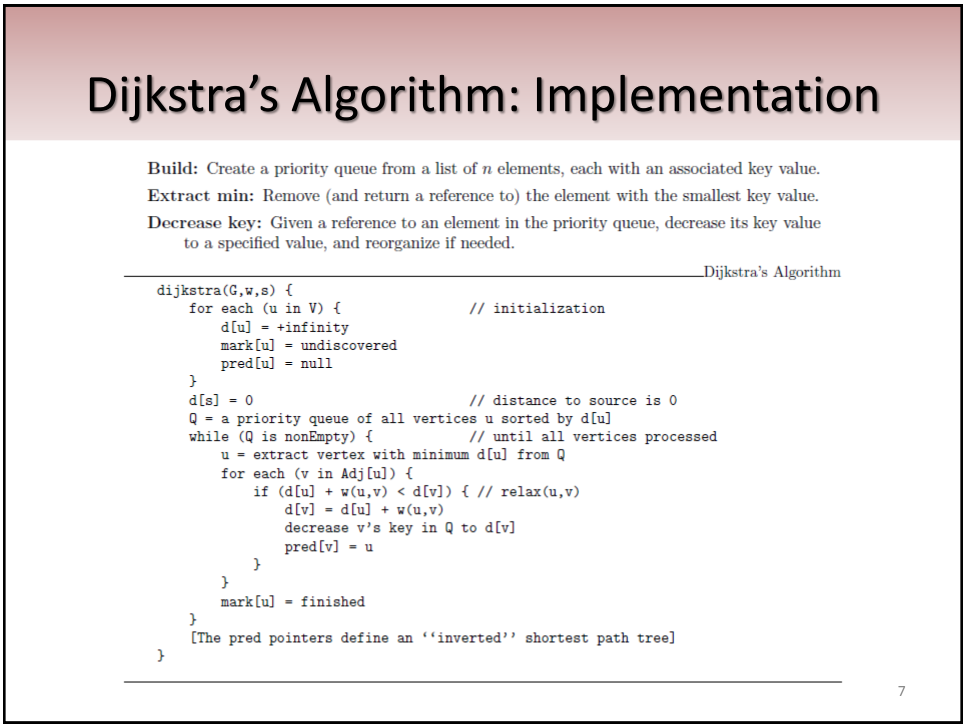
- Maintain a set of explored nodes S for which we have determined the shortest path distance d(u) from s to u.
- Initialize S = { s }, d(s) = 0.
- Repeatedly choose unexplored node v which minimizes

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

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: Implementation

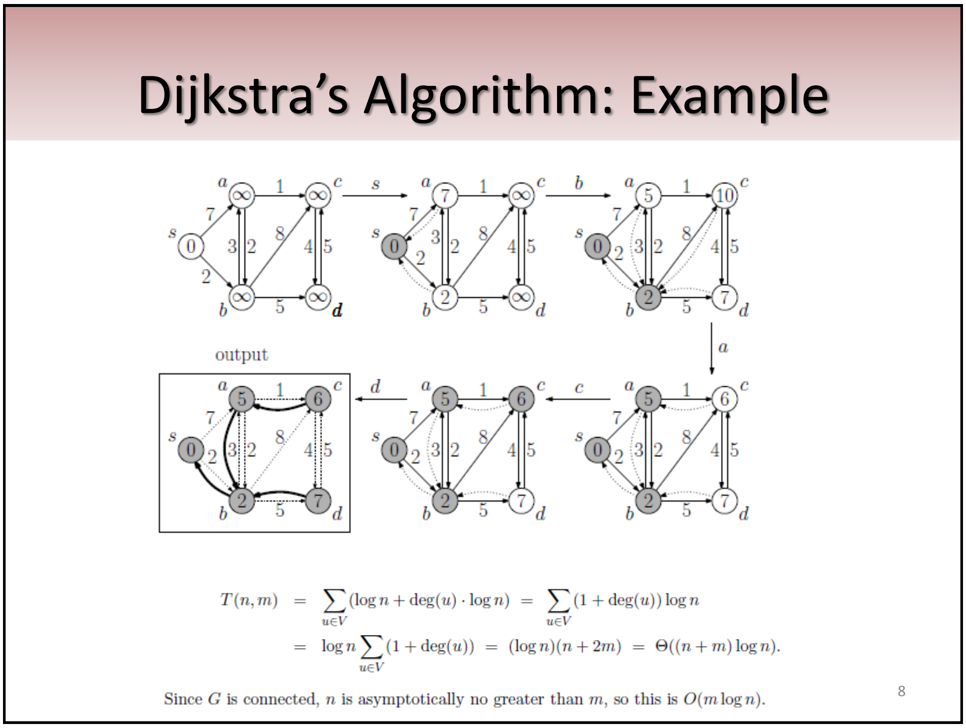
Build: Create a priority queue from a list of n elements, each with an associated key value.

Extract min: Remove (and return a reference to) the element with the smallest key value.

Decrease key: Given a reference to an element in the priority queue, decrease its key value to a specified value, and reorganize if needed.

Dijkstra’s Algorithm

```
dijkstra(G,w,s) {
    for each (u in V) {                 // initialization
        d[u] = +infinity
        mark[u] = undiscovered
        pred[u] = null
    }
    d[s] = 0                            // distance to source is 0
    Q = a priority queue of all vertices u sorted by d[u]
    while (Q is nonEmpty) {             // until all vertices processed
        u = extract vertex with minimum d[u] from Q
        for each (v in Adj[u]) {
            if (d[u] + w(u,v) < d[v]) { // relax(u,v)
                d[v] = d[u] + w(u,v)
                decrease v’s key in Q to d[v]
                pred[v] = u
            }
        }
        mark[u] = finished
    }
    [The pred pointers define an ‘‘inverted’’ shortest path tree]
}
```



Dijkstra’s Algorithm: Example

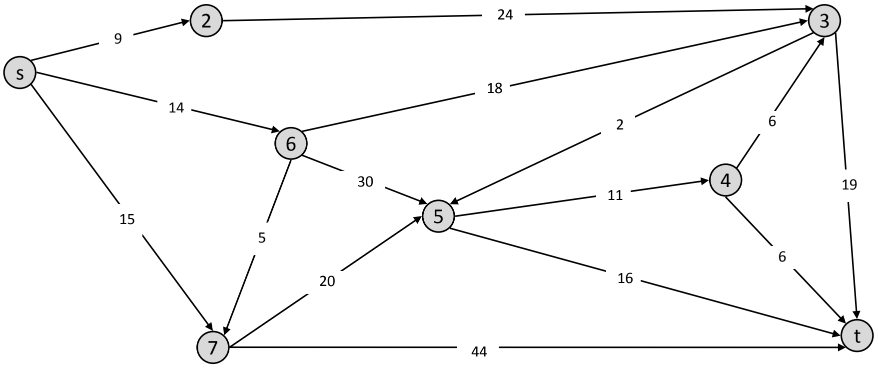
$$T(n,m) = \sum_{u\in V}(\log n + \deg(u)\cdot\log n) = \sum_{u\in V}(1+\deg(u))\log n$$

$$= \log n\sum_{u\in V}(1+\deg(u)) = (\log n)(n+2m) = \Theta((n+m)\log n).$$

Since G is connected, n is asymptotically no greater than m, so this is $O(m\log n)$.

Dijkstra's Shortest Path Algorithm

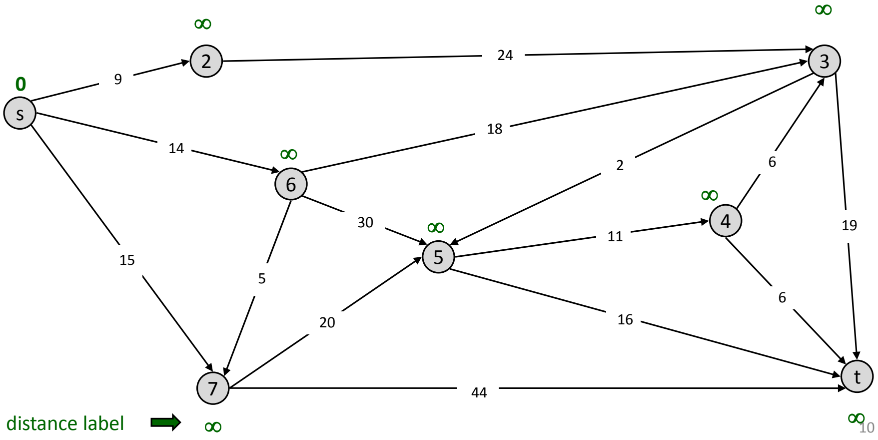
Find shortest path from s to t.

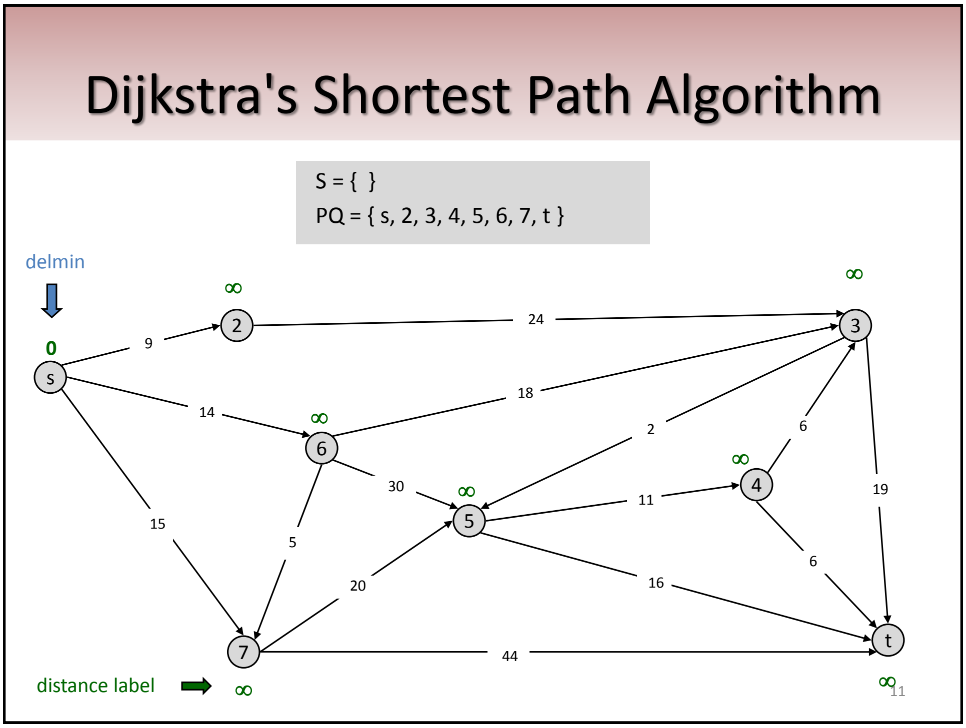


Dijkstra's Shortest Path Algorithm

S = { }

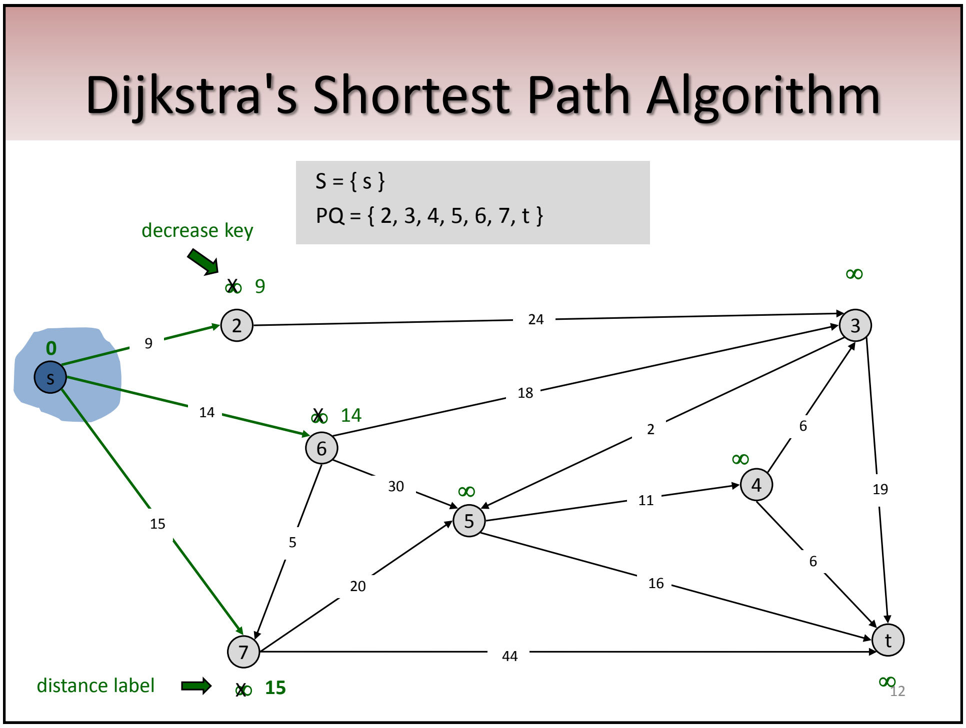
PQ = { s, 2, 3, 4, 5, 6, 7, t }





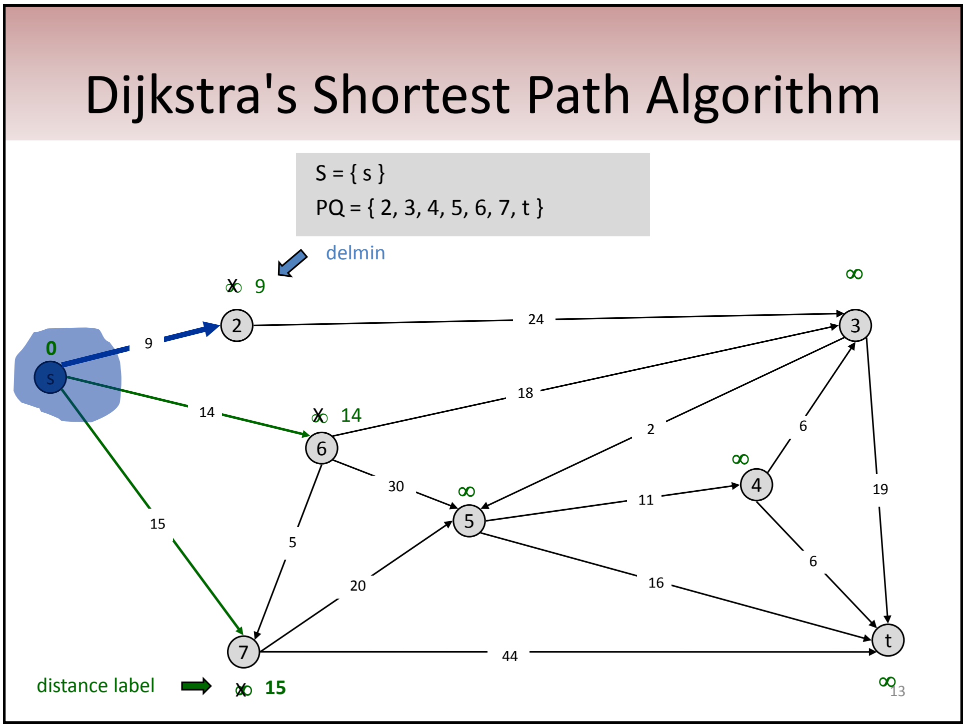
Dijkstra's Shortest Path Algorithm

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 }

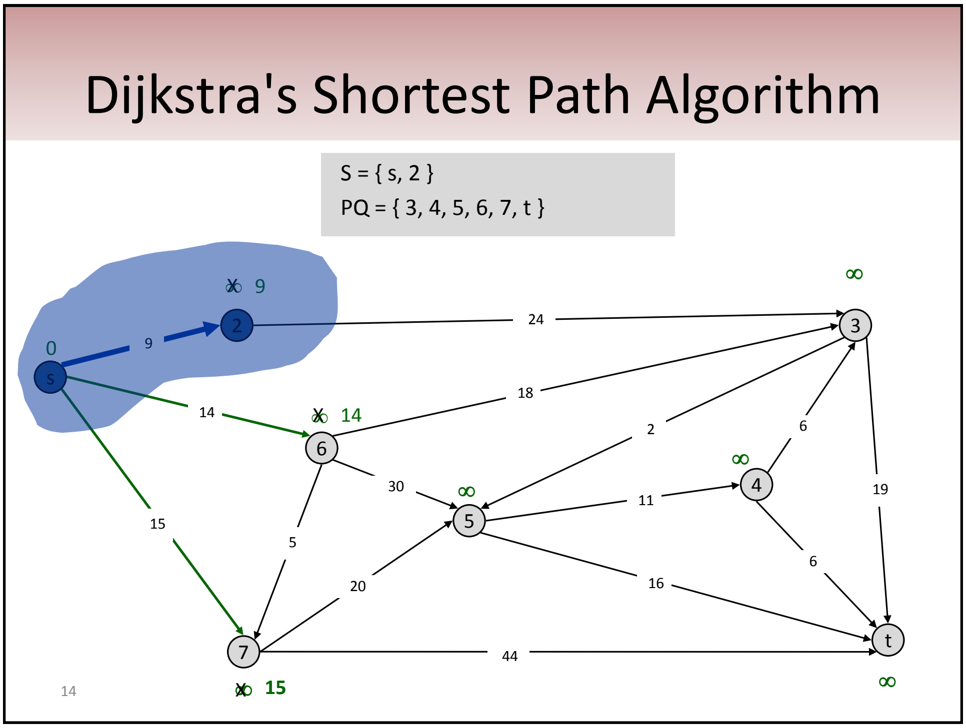


Dijkstra's Shortest Path Algorithm

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

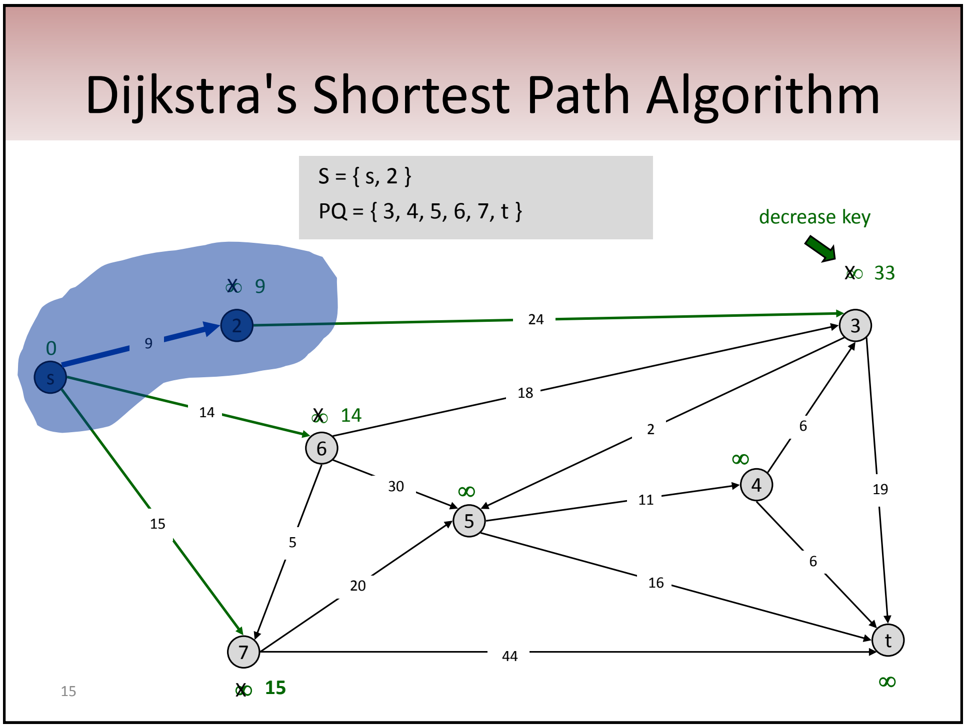
delmin

distance label



Dijkstra's Shortest Path Algorithm

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

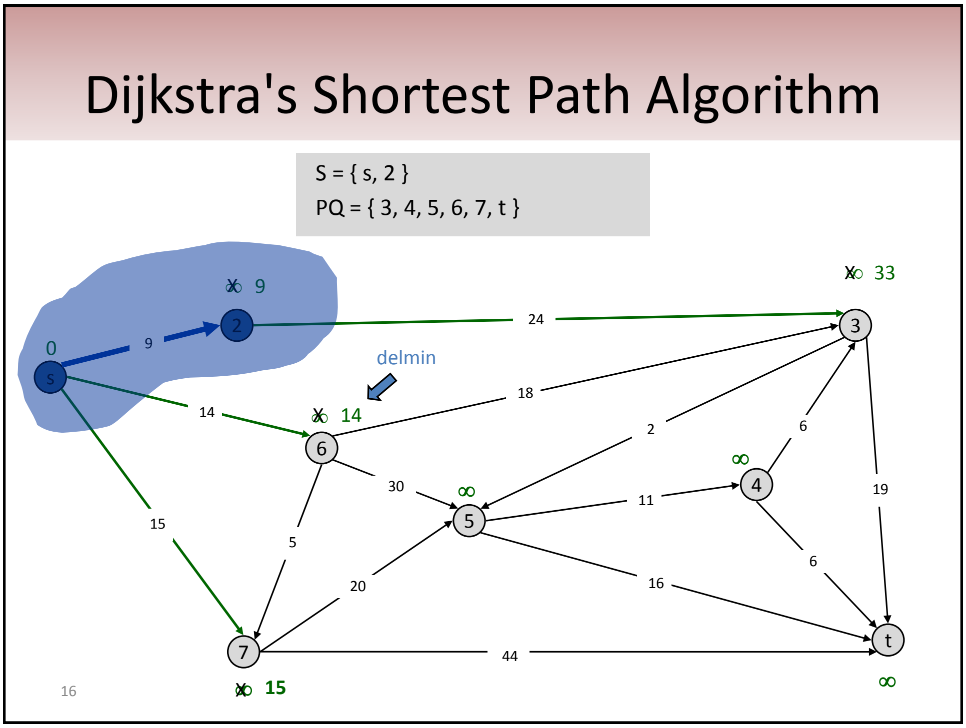


Dijkstra's Shortest Path Algorithm

S = { s, 2 }

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

15



Dijkstra's Shortest Path Algorithm

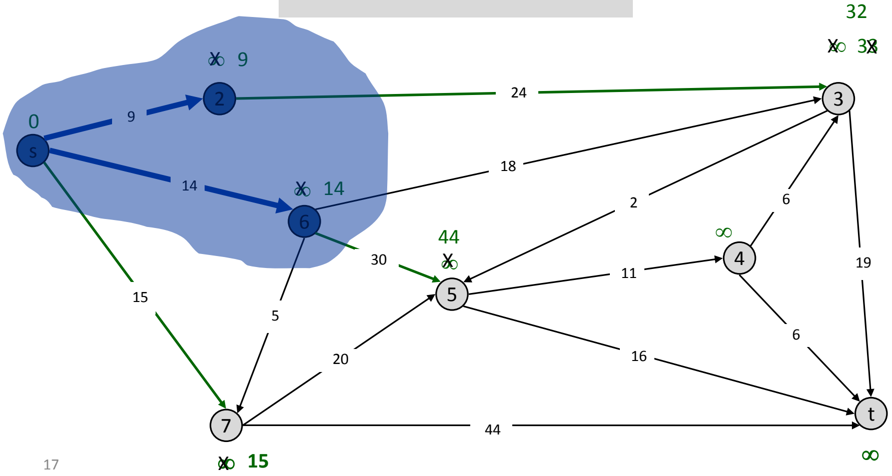
S = { s, 2 }

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

16

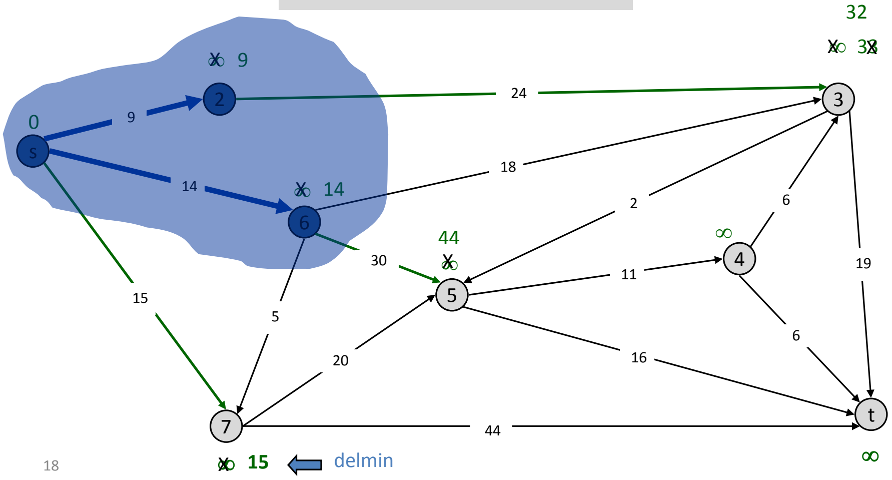
Dijkstra's Shortest Path Algorithm

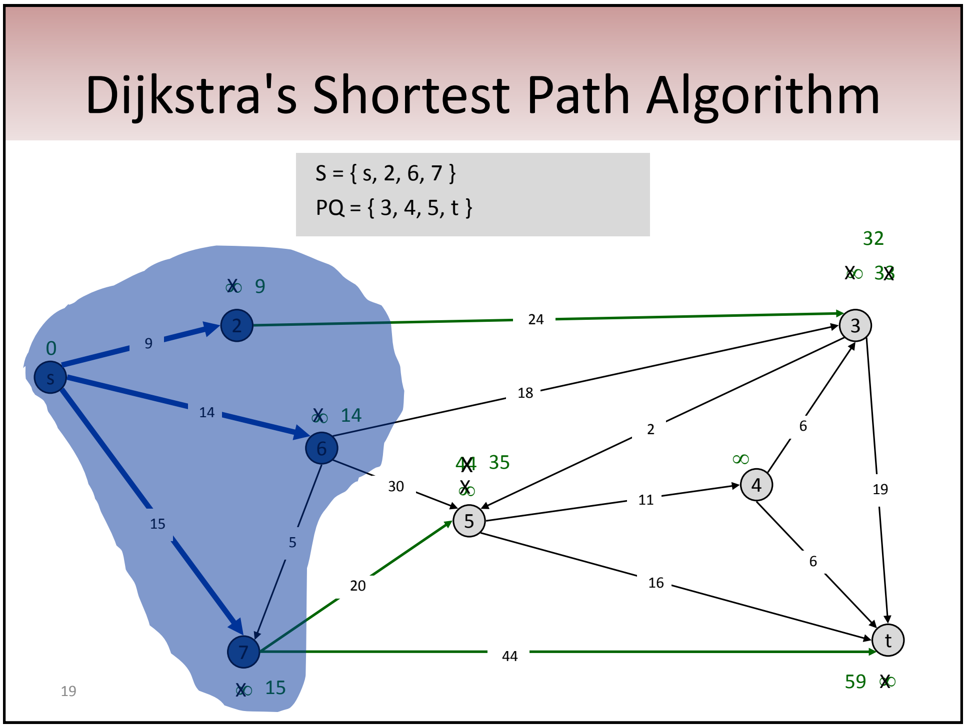
S = { s, 2, 6 }
PQ = { 3, 4, 5, 7, t }



Dijkstra's Shortest Path Algorithm

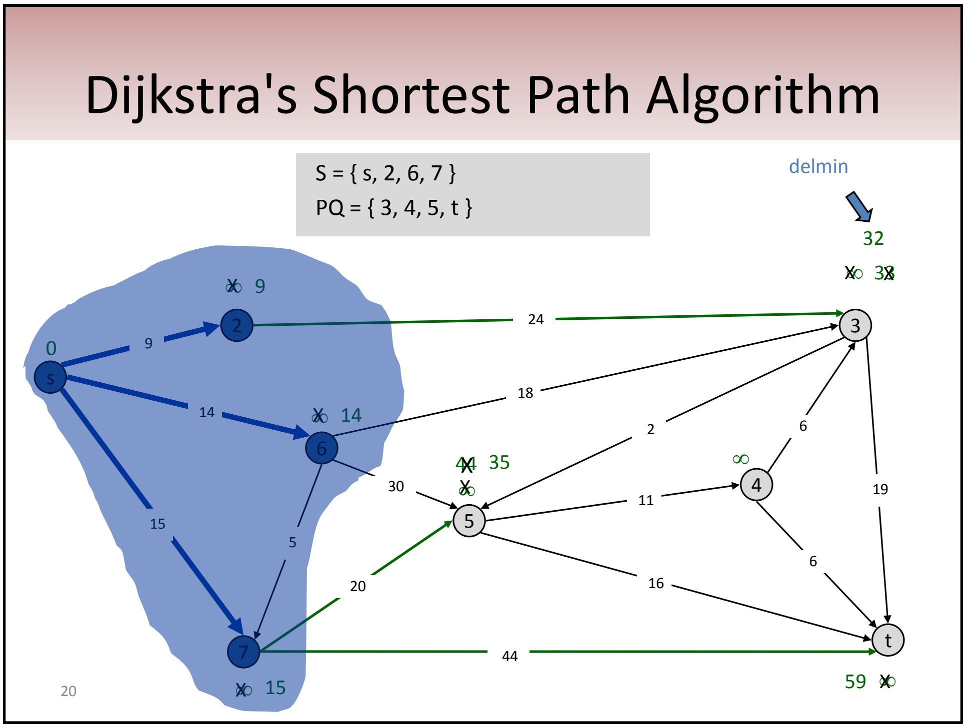
S = { s, 2, 6 }
PQ = { 3, 4, 5, 7, t }





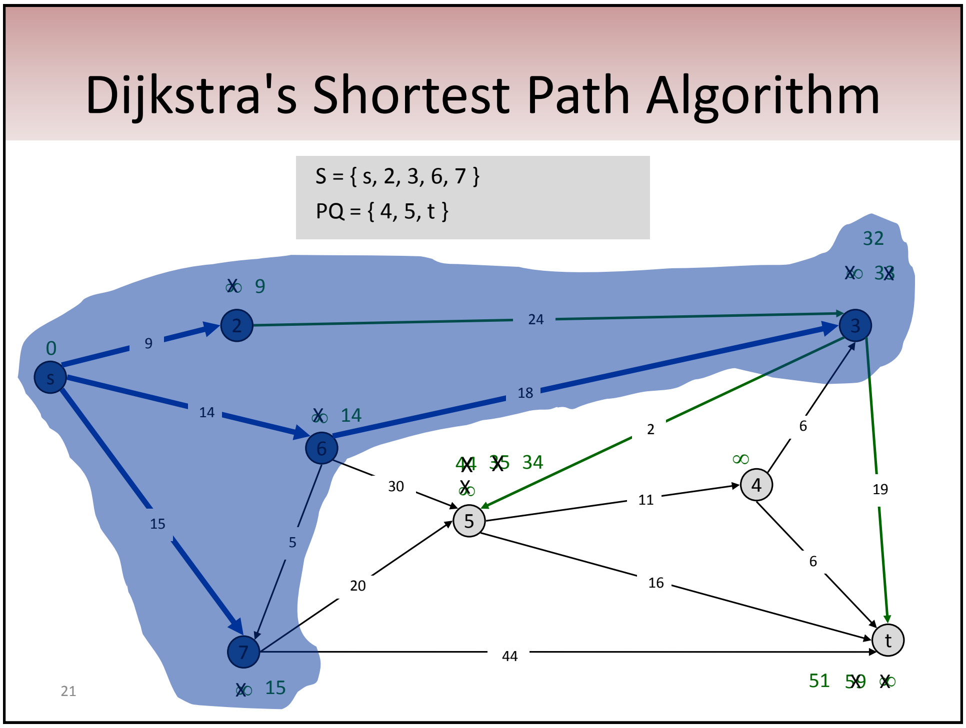
Dijkstra's Shortest Path Algorithm

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



Dijkstra's Shortest Path Algorithm

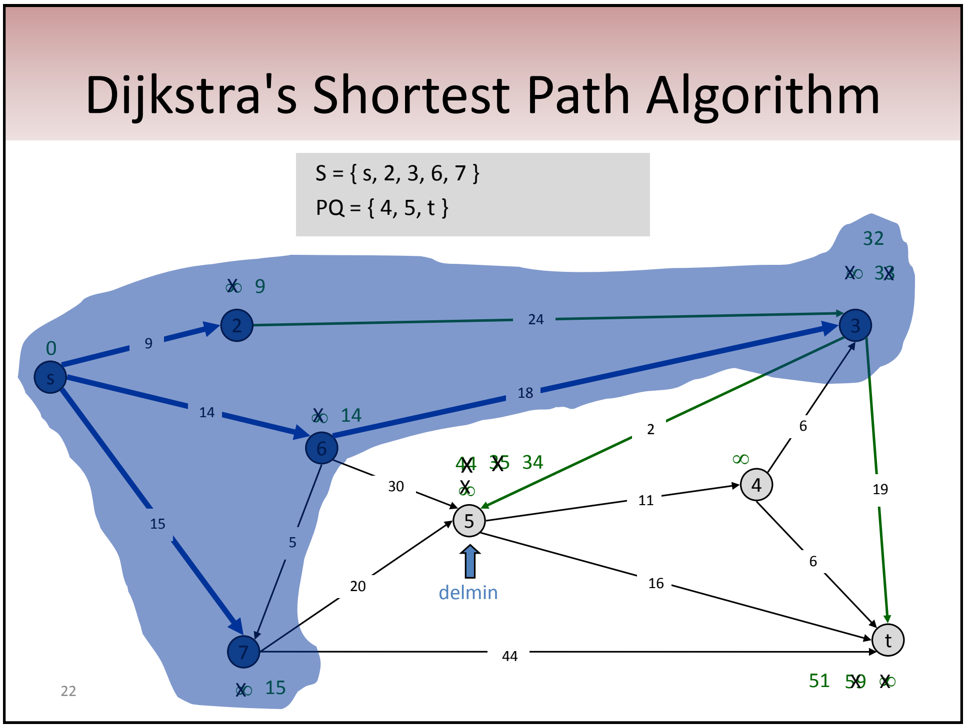
S = { s, 2, 6, 7 }
PQ = { 3, 4, 5, t }



Dijkstra's Shortest Path Algorithm

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

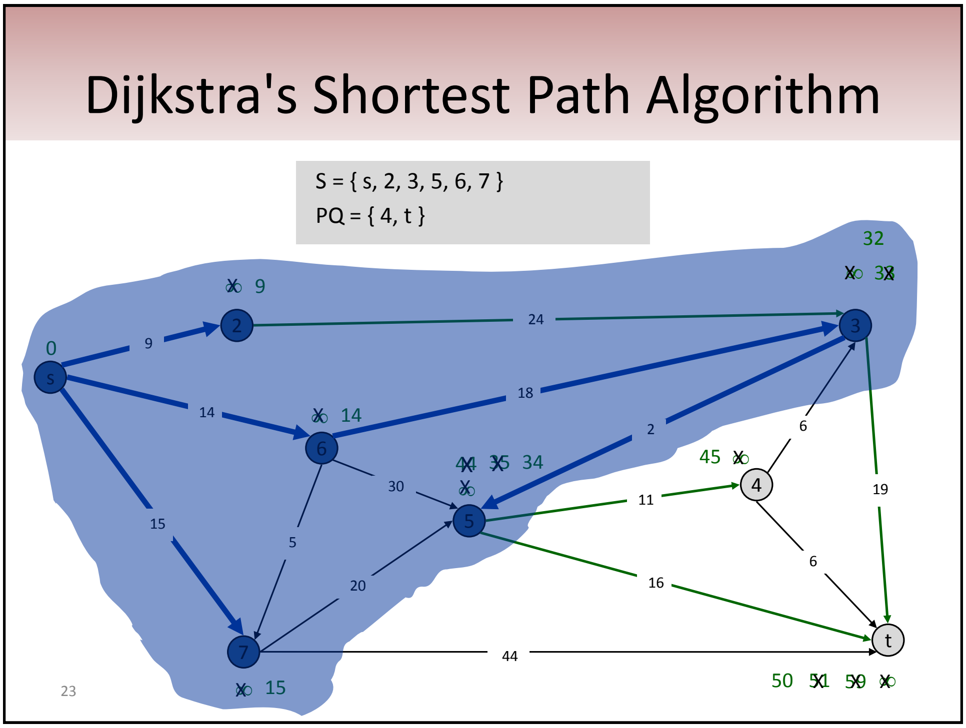
PQ = { 4, 5, t }



Dijkstra's Shortest Path Algorithm

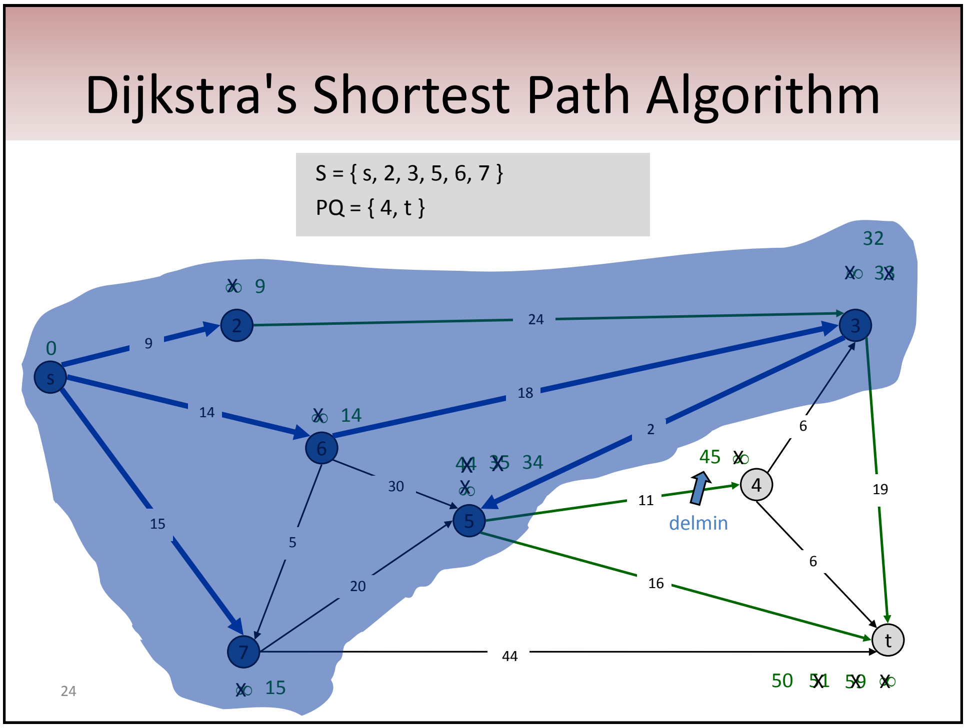
S = { s, 2, 3, 6, 7 }

PQ = { 4, 5, t }



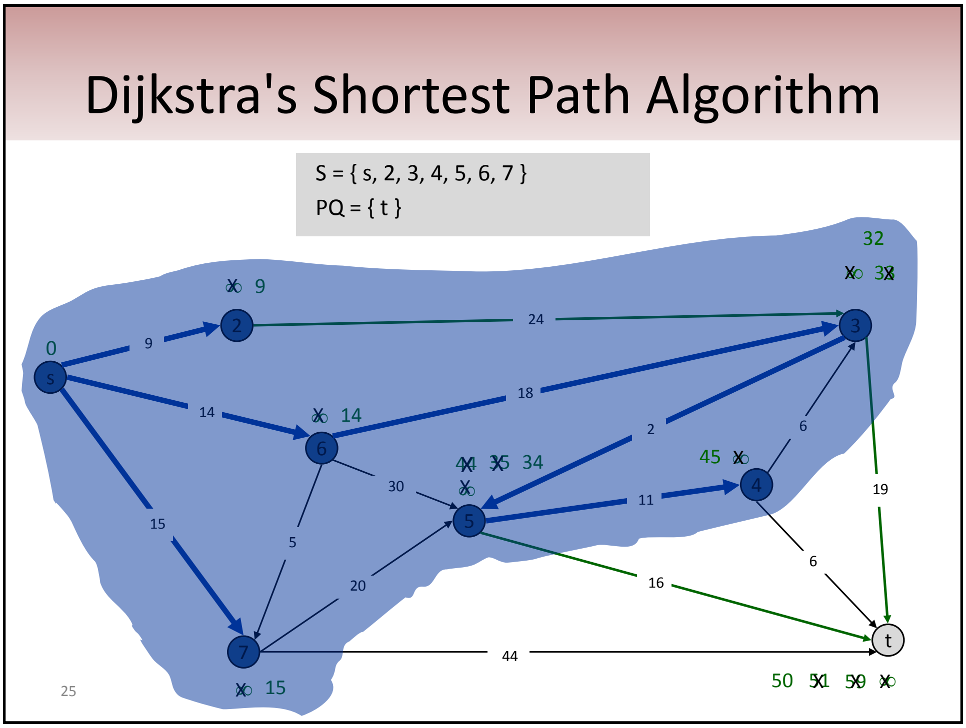
Dijkstra's Shortest Path Algorithm

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



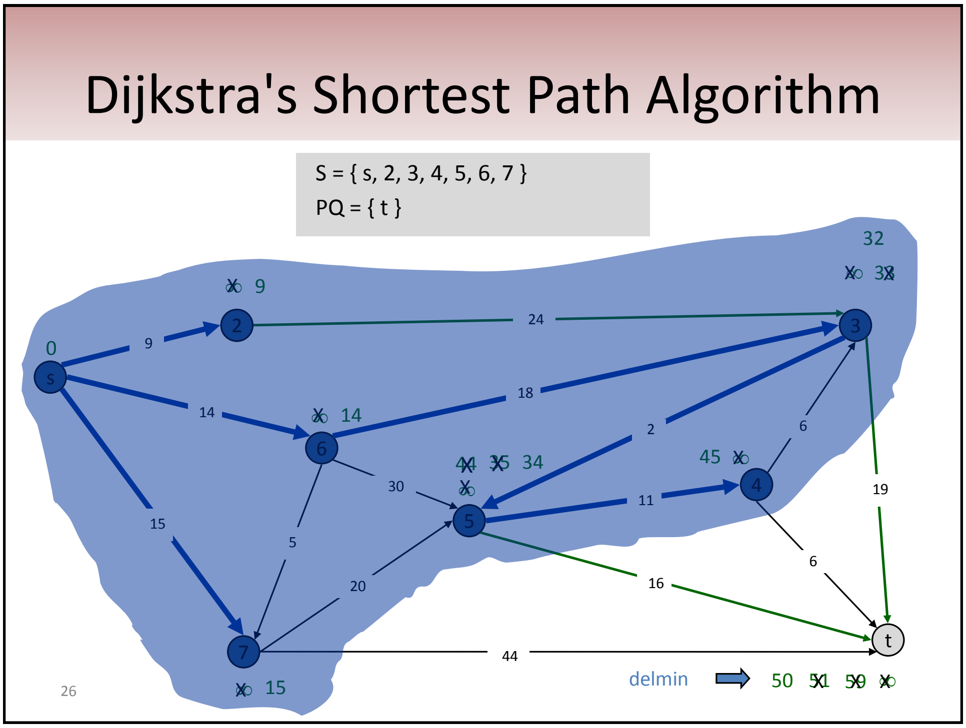
Dijkstra's Shortest Path Algorithm

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



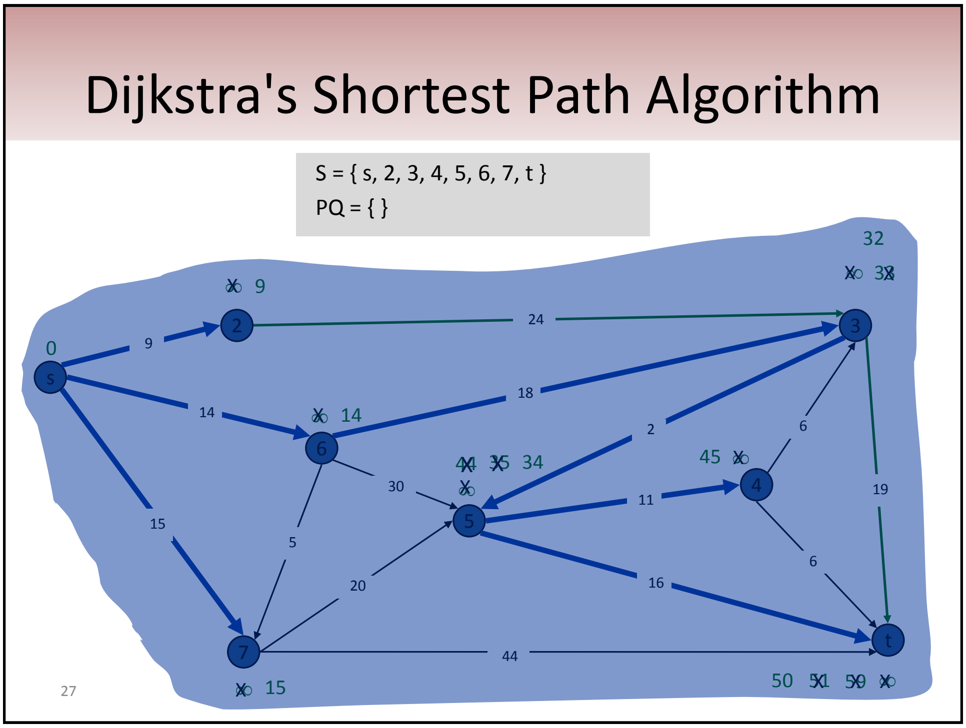
Dijkstra's Shortest Path Algorithm

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



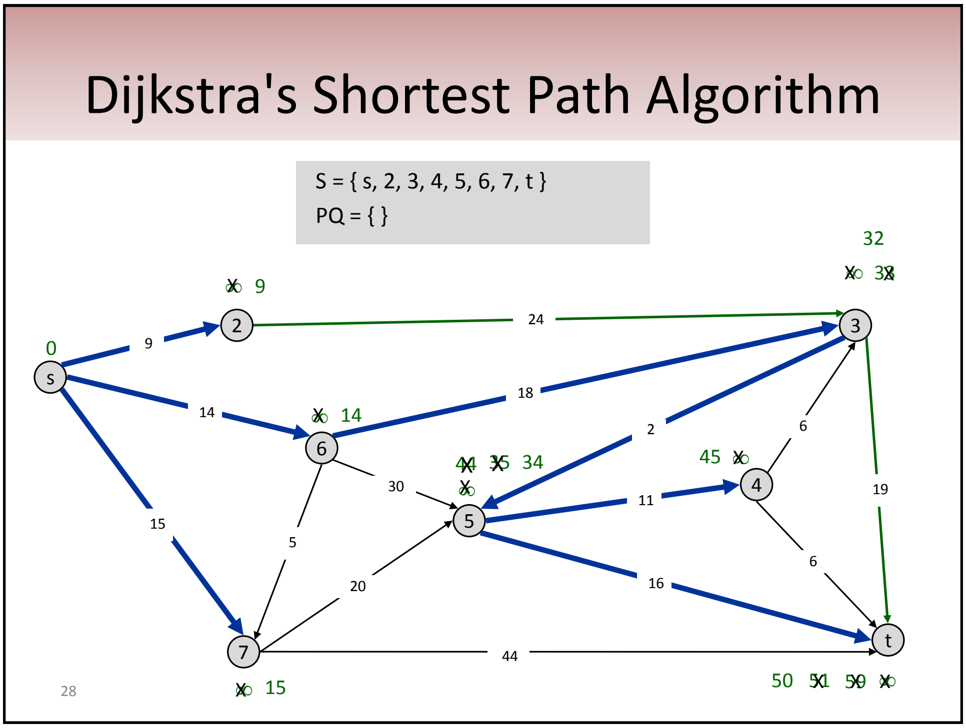
Dijkstra's Shortest Path Algorithm

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

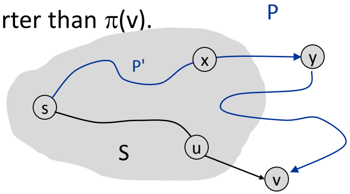
Invariant. For each node u ∈ S, d(u) is the length of the shortest s-u path.

Pf. (by induction on |S|)

Base case: |S| = 1 is trivial.

Inductive hypothesis: Assume true for |S| = k ≥ 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 π(v).
- Consider any s-v path P. We'll see that it's no shorter than π(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.



$$\ell(P) \underset{\substack{\uparrow \\ \text{nonnegative} \\ \text{weights}}}{\ge} \ell(P') + \ell(x,y) \underset{\substack{\uparrow \\ \text{inductive} \\ \text{hypothesis}}}{\ge} d(x) + \ell(x,y) \underset{\substack{\uparrow \\ \text{defn of } \pi(y)}}{\ge} \pi(y) \underset{\substack{\uparrow \\ \text{Dijkstra chose v} \\ \text{instead of y}}}{\ge} \pi(v)$$

Variants of Dijkstra’s

Vertex weights: There is a cost associated with each vertex. The overall cost is the sum of vertex and/or edge weights on the path.

Single-Sink Shortest Path: Find the shortest path from each vertex to a sink vertex t.

Multi-Source/Multi-Sink: You are given a collection of source vertices {s1, . . . , sk}. For each vertex find the shortest path from its nearest source. (Analogous for multi-sink.)

Multiplicative Cost: Define the cost of a path to be the product of the edge weights (rather than the sum.) If all the edge weights are at least 1, find the single-source shortest path.

Practice

Give the final d and predecessor values of the vertices obtained by running Dijkstra's algorithm on the directed graph below with source A.