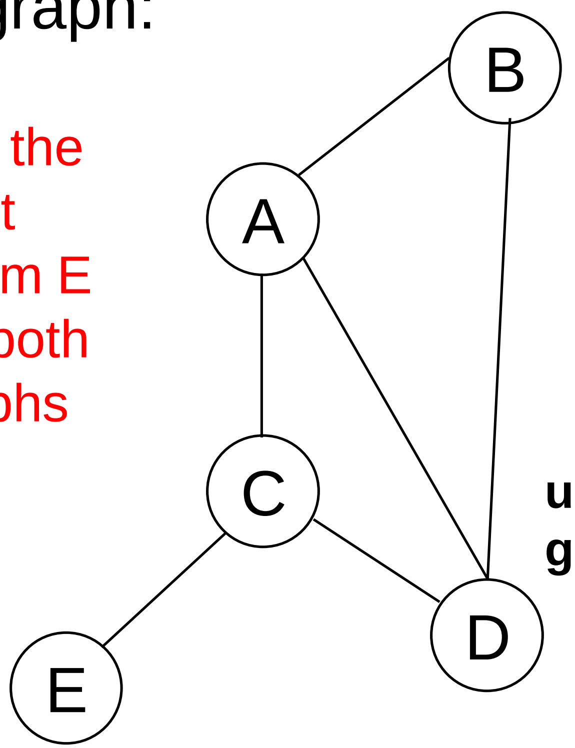
Dijkstra's Algorithm – Single Source Shortest Path

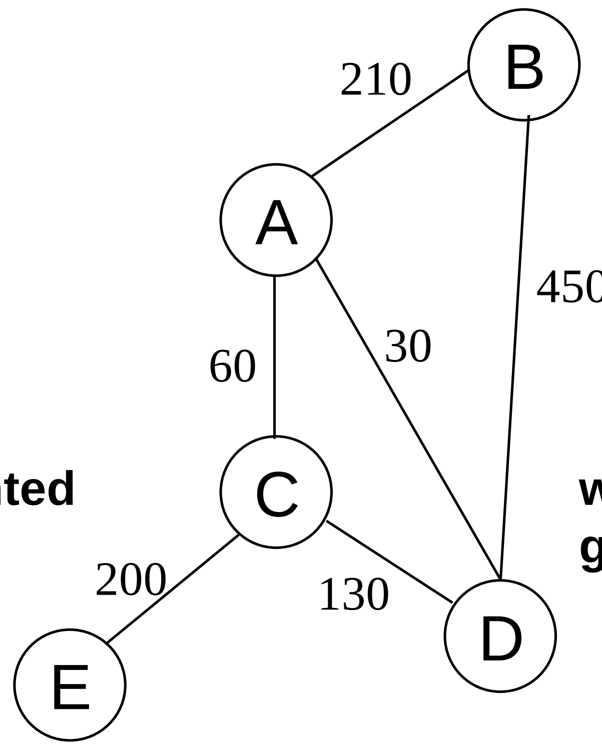
Shortest Path Problems

- **What is shortest path ?**
  - shortest length between two vertices for an unweighted graph:
  - smallest cost between two vertices *for* a weighted graph:

What is the Shortest path from E to D in both the graphs



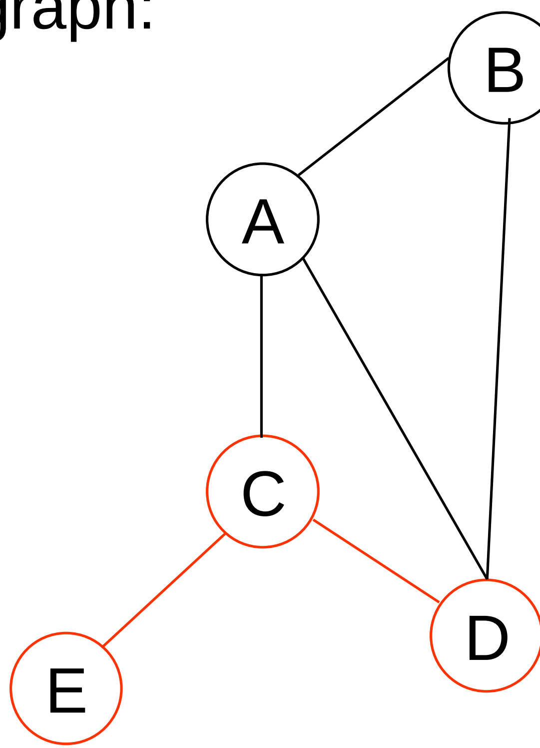
unweighted graph

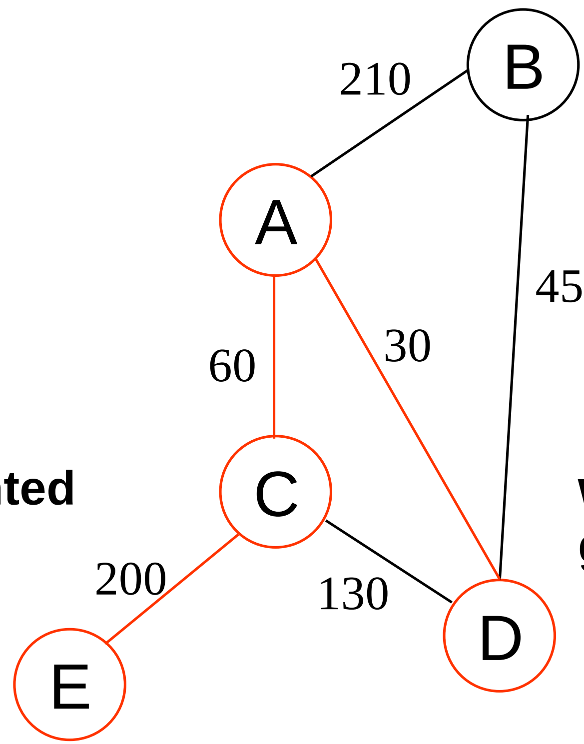


weighted graph

Shortest Path Problems

- **What is shortest path ?**
  - <u>shortest length between two vertices</u> for an unweighted graph:
  - <u>smallest cost between two vertices</u> *for* a weighted graph:





Shortest Path Problems

- How can we find the shortest route between two points on a map?
- Model the problem as a graph problem:
  – Road map is a weighted graph:

    vertices = cities

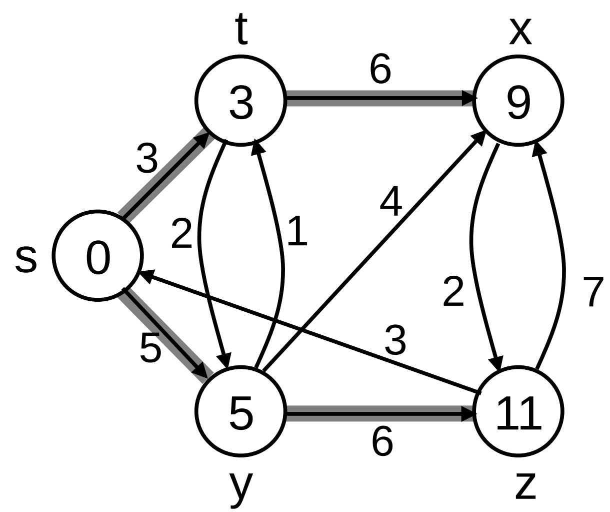
    edges = road segments between cities

    edge weights = road distances

  – Goal: find a shortest path between two vertices (cities)

Shortest Path Problems

- **Input:**
  - Directed graph G = (V, E)
  - Weight function w : E → **R**



- **Weight of path** $p = \langle v_0, v_1, \ldots, v_k \rangle$

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- **Shortest-path weight** from u to v:

$$\delta(u, v) = \min \begin{cases} w(p) : u \overset{p}{\rightsquigarrow} v & \text{if there exists a path from } u \text{ to } v \\ \infty & \text{otherwise} \end{cases}$$

- Shortest path u to v is any path p such that $w(p) = \delta(u, v)$

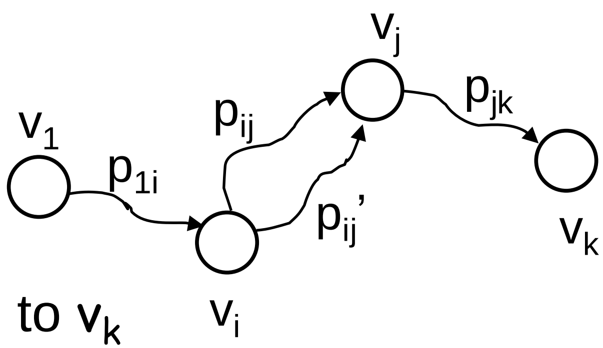
Variants of Shortest Paths

- **Single-source shortest path**
  - $G = (V, E) \Rightarrow$ find a shortest path from a given source vertex s to each vertex $v \in V$
- **Single-destination shortest path**
  - Find a shortest path to a given destination vertex t from each vertex v
  - Reverse the direction of each edge $\Rightarrow$ single-source
- **Single-pair shortest path**
  - Find a shortest path from u to v for given vertices u and v
  - Solve the single-source problem
- **All-pairs shortest-paths**
  - Find a shortest path from u to v for every pair of vertices u and v

Optimal Substructure of Shortest Paths

Given:

- A weighted, directed graph $G = (V, E)$
- A weight function $w: E \to \mathbf{R}$,
- A shortest path $p = \langle v_1, v_2, \ldots, v_k \rangle$ from v_1 to v_k
- A subpath of p: $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$, with $1 \le i \le j \le k$



Then: p_{ij} is a shortest path from v_i to v_j

Proof: $p = v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$

$$w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$$

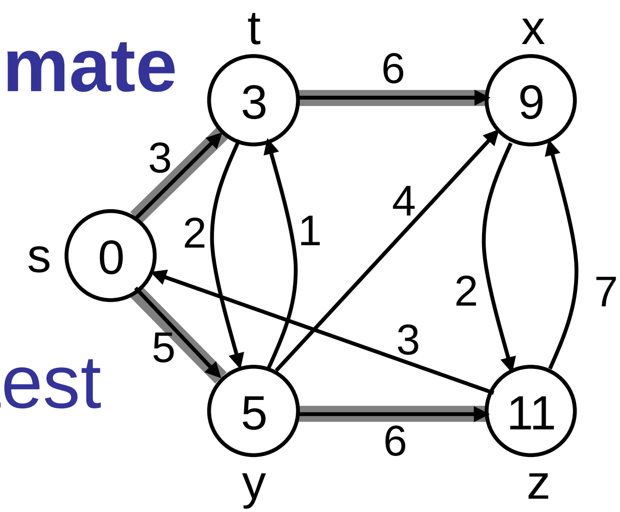
Assume $\exists$ p_{ij}' from v_i to v_j with $w(p_{ij}') < w(p_{ij})$

$\Rightarrow w(p') = w(p_{1i}) + w(p_{ij}') + w(p_{jk}) < w(p)$

Shortest-Path Representation

For each vertex v $\in$ V:

- d[v] = δ(s, v): a **shortest-path estimate**
  - Initially, d[v]=∞
  - Reduces as algorithms progress
- π[v] = **predecessor** of v on a shortest path from *s*
  - If no predecessor, π[v] = NIL
  - π induces a tree—**shortest-path tree**
- Shortest paths & shortest path trees are not unique



Initialization

Alg.: INITIALIZE-SINGLE-SOURCE(V, s)

1. **for** each v $\in$ V
2. **do** d[v] $\leftarrow \infty$
3. π[v] $\leftarrow$ NIL
4. d[s] $\leftarrow$ 0

- All the shortest-paths algorithms start with INITIALIZE-SINGLE-SOURCE

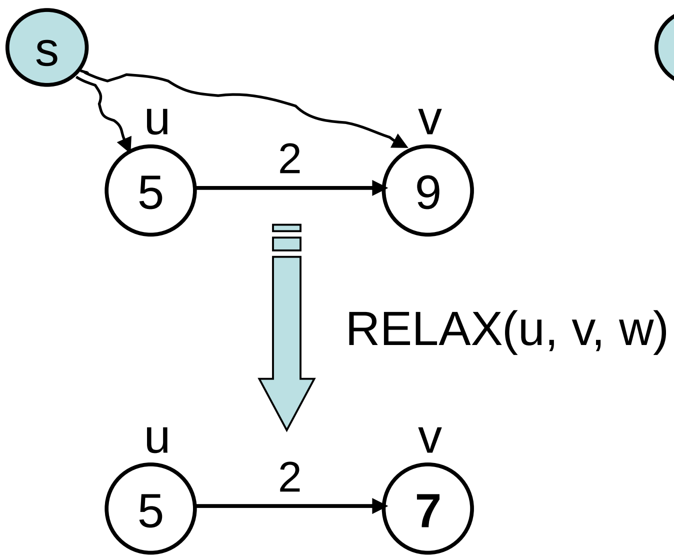
Relaxation

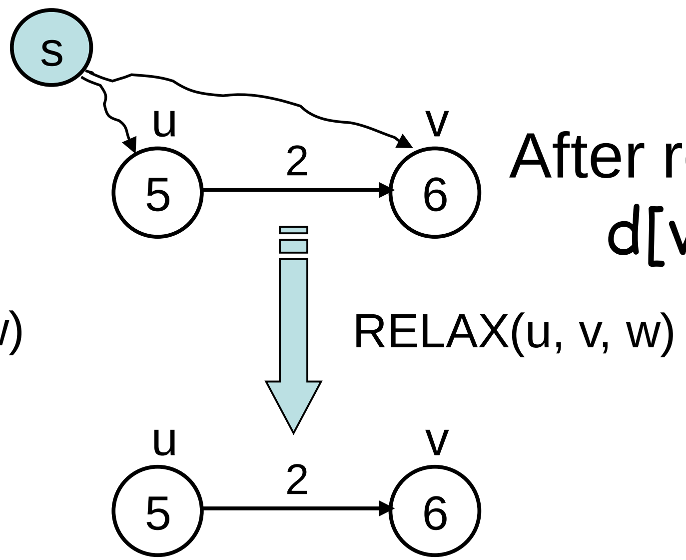
- **Relaxing** an edge (u, v) = testing whether we can improve the shortest path to v found so far by going through u

If $d[v] > d[u] + w(u, v)$

we can improve the shortest path to v

$\Rightarrow$ update d[v] and π[v]





After relaxation:

$d[v] \leq d[u] + w(u, v)$

RELAX(u, v, w)

1. **if** $d[v] > d[u] + w(u, v)$
2. **then** $d[v] \leftarrow d[u] + w(u, v)$
3. $\pi[v] \leftarrow u$

- All the single-source shortest-paths algorithms
  - start by calling INIT-SINGLE-SOURCE
  - then relax edges
- The algorithms differ in the order and how many times they relax each edge

Dijkstra's Algorithm

- Single-source shortest path problem:
  - No negative-weight edges: $w(u, v) > 0 \; \forall \; (u, v) \in E$
- Maintains two sets of vertices:
  - S = vertices whose final shortest-path weights have already been determined
  - Q = vertices in V – S: min-priority queue
    - Keys in Q are estimates of shortest-path weights (d[v])
- Repeatedly select a vertex $u \in V - S$, with the minimum shortest-path estimate d[v]

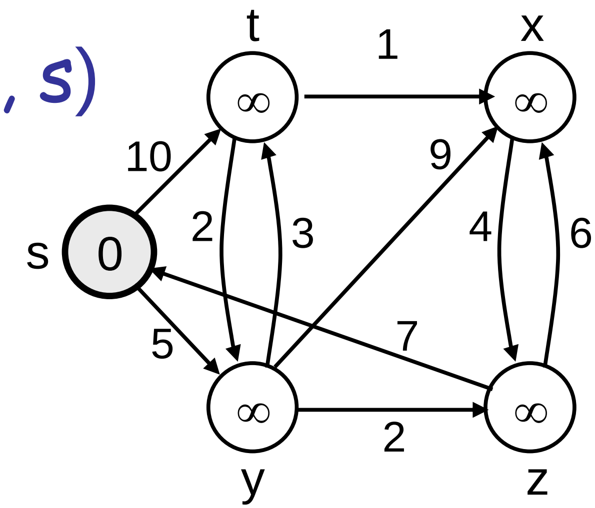
Dijkstra's Algorithm

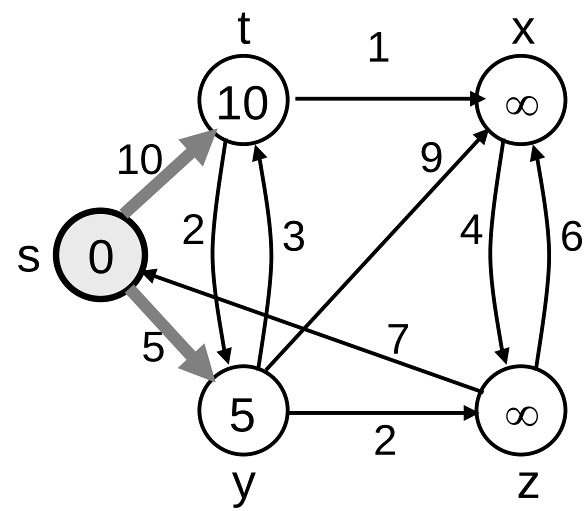
```
Algorithm ShortestPaths(v, cost, dist, n)
// dist[j], 1 ≤ j ≤ n, is set to the length of the shortest
// path from vertex v to vertex j in a digraph G with n
// vertices. dist[v] is set to zero. G is represented by its
// cost adjacency matrix cost[1 : n, 1 : n].
{
    for i := 1 to n do
    { // Initialize S.
        S[i] := false; dist[i] := cost[v, i];
    }
    S[v] := true; dist[v] := 0.0; // Put v in S.
    for num := 2 to n − 1 do
    {
        // Determine n − 1 paths from v.
        Choose u from among those vertices not
        in S such that dist[u] is minimum;
        S[u] := true; // Put u in S.
        for (each w adjacent to u with S[w] = false) do
            // Update distances.
            if (dist[w] > dist[u] + cost[u, w])) then
                    dist[w] := dist[u] + cost[u, w];
    }
}
```

Greedy algorithm to generate shortest paths

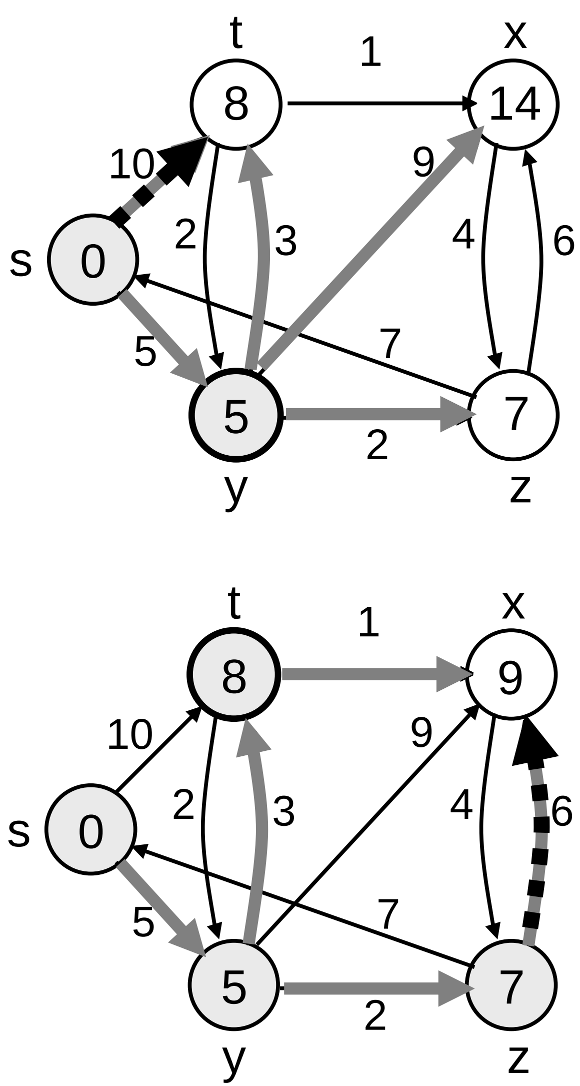
Dijkstra (G, w, s)

```
INITIALIZE-SINGLE-SOURCE(V, s)
S ← ∅
Q ← V[G]
while Q ≠ ∅
      do u ← EXTRACT-MIN(Q)
            S ← S ∪ {u}
            for each vertex v ∈ Adj[u]
                  do RELAX(u, v, w)
```





Example


t
x
s
y
z
8
14
0
5
7
1
10
9
2
3
4
6
5
7
2
t
x
s
y
z
8
13
0
5
7
1
10
9
2
3
4
6
5
7
2
t
x
s
y
z
8
9
0
5
7
1
10
9
2
3
4
6
5
7
2
t
x
s
y
z
8
9
0
5
7
1
10
9
2
3
4
6
5
7
2

Dijkstra's Pseudo Code

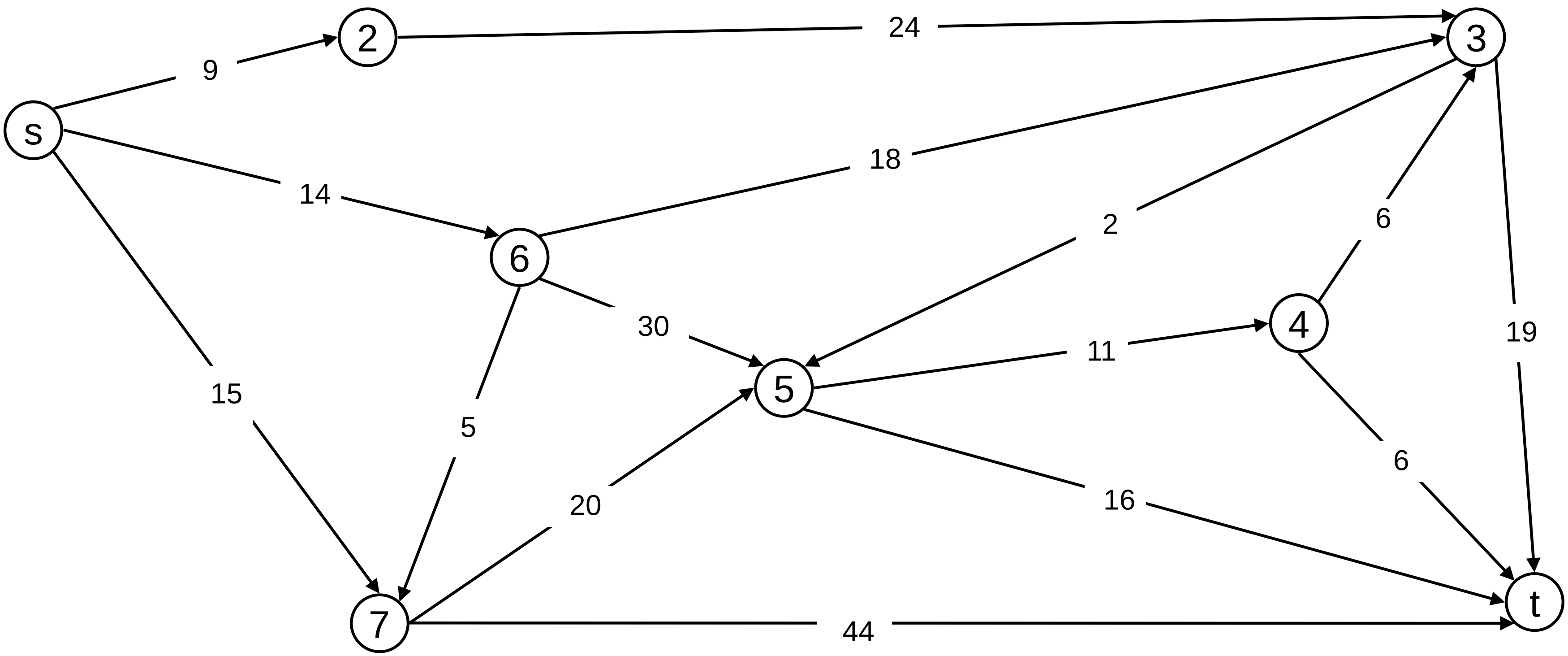
- Graph *G*, weight function *w*, root *s*

```
DIJKSTRA(G, w, s)
for each v ∈ V
        do d[v] ← ∞
d[s] ← 0
S ← ∅   ▷ Set of discovered nodes
Q ← V
while Q ≠ ∅
        do u ← EXTRACT-MIN(Q)
           S ← S ∪ {u}
           for each v ∈ Adj[u]
                do if d[v] > d[u] + w(u, v)
                      then d[v] ← d[u] + w(u, v)
```

relaxing edges

Dijkstra's Shortest Path Algorithm

- Find shortest path from s to t.

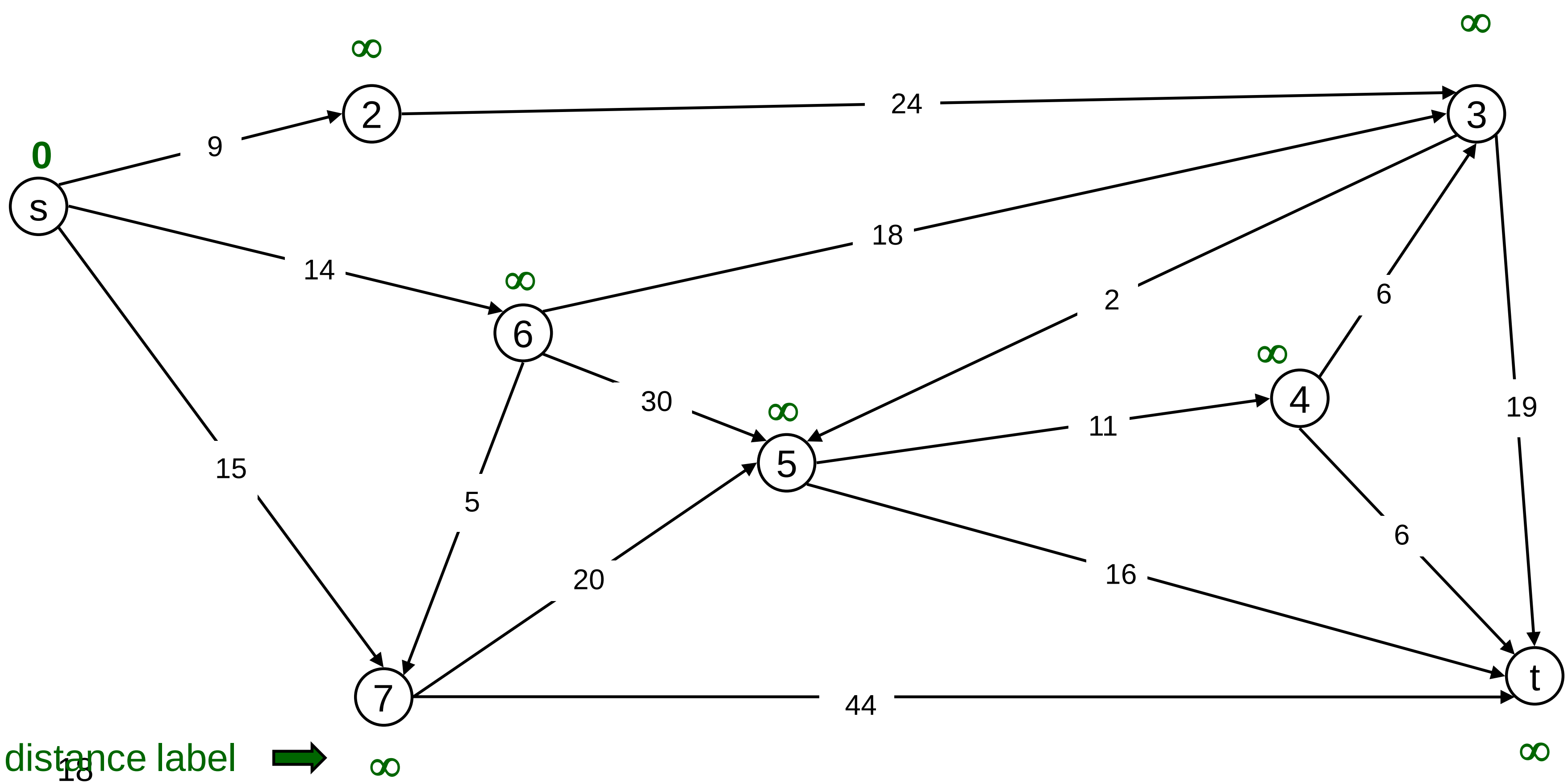


Dijkstra's Shortest Path Algorithm

S = { }

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

d	s	2	3	4	5	6	7	t
	0	∞	∞	∞	∞	∞	∞	∞

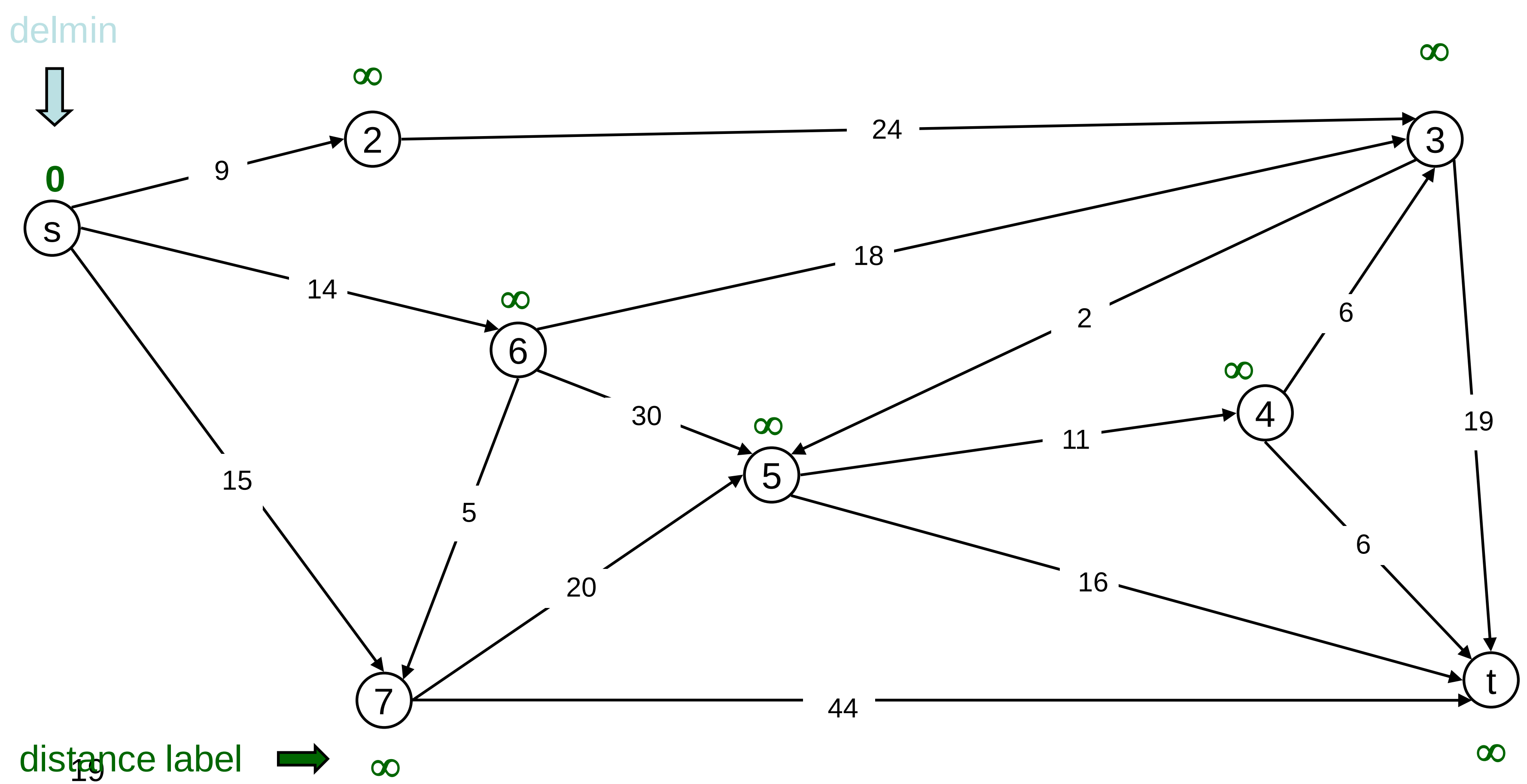


Dijkstra's Shortest Path Algorithm

S = { }

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

d	s	2	3	4	5	6	7	t
	0	∞	∞	∞	∞	∞	∞	∞

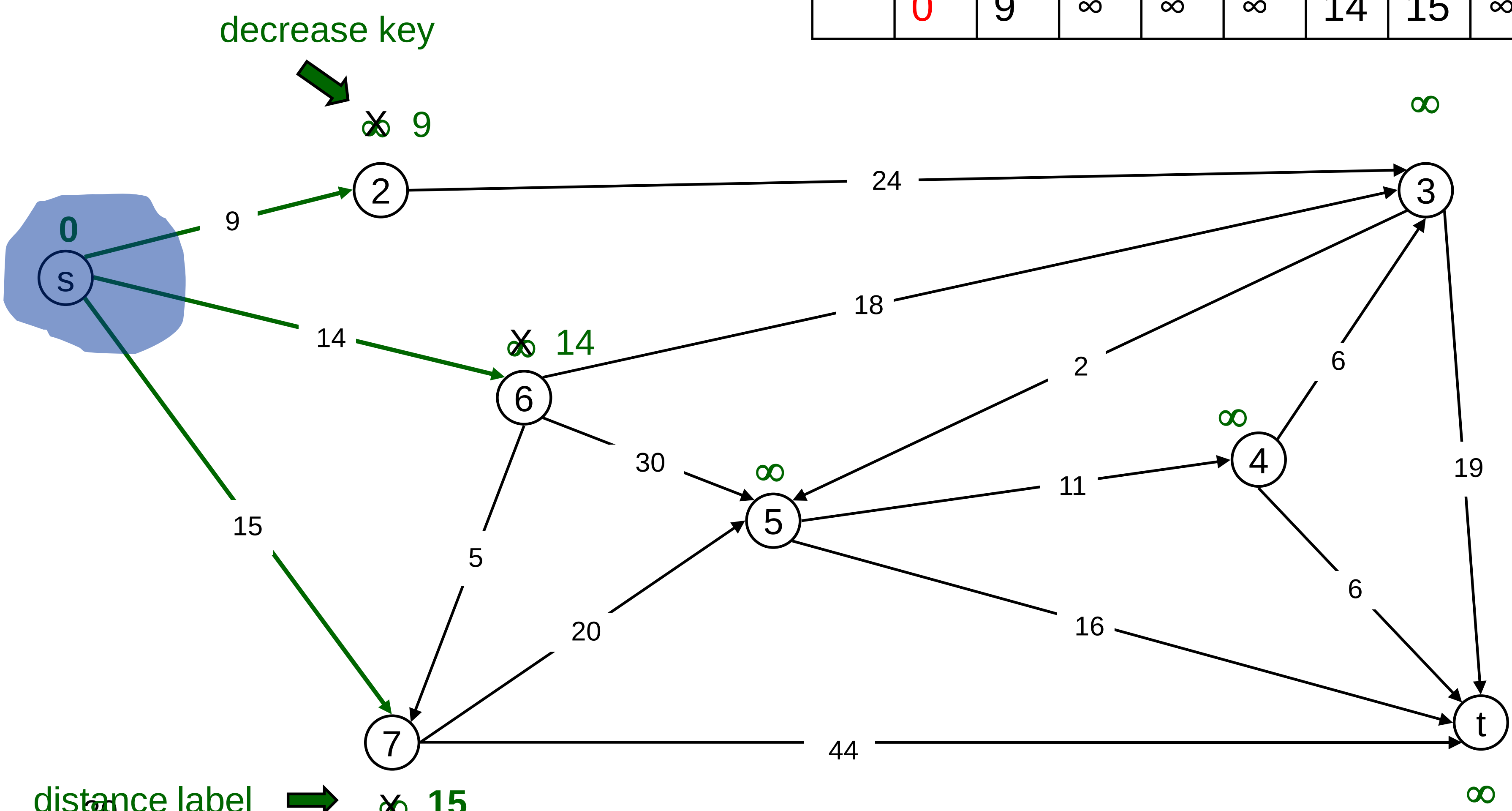


Dijkstra's Shortest Path Algorithm

S = { s }

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

d	s	2	3	4	5	6	7	t
	0	9	∞	∞	∞	14	15	∞

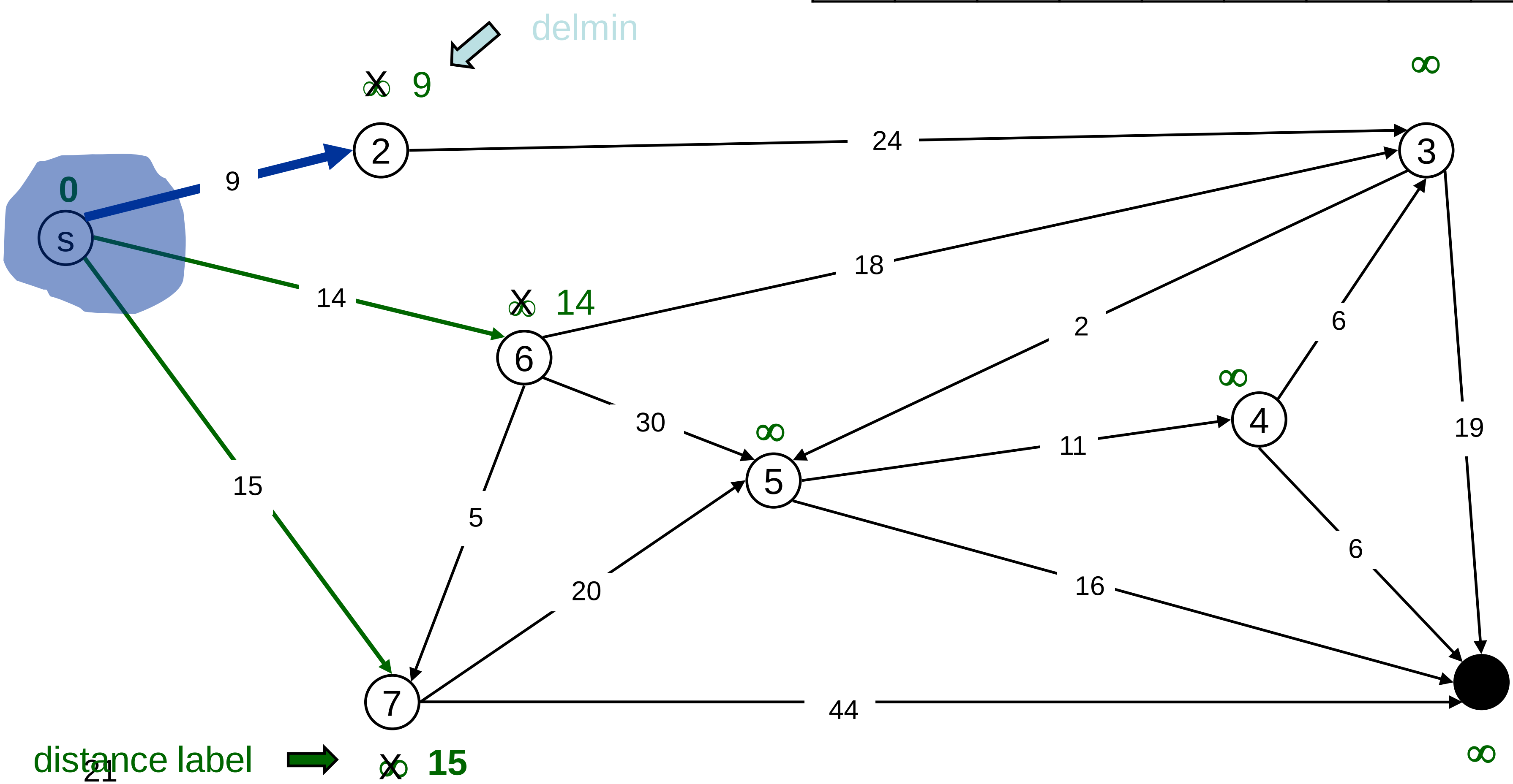


Dijkstra's Shortest Path Algorithm

S = { s }

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

d	s	2	3	4	5	6	7	t
	0	9	∞	∞	∞	14	15	∞

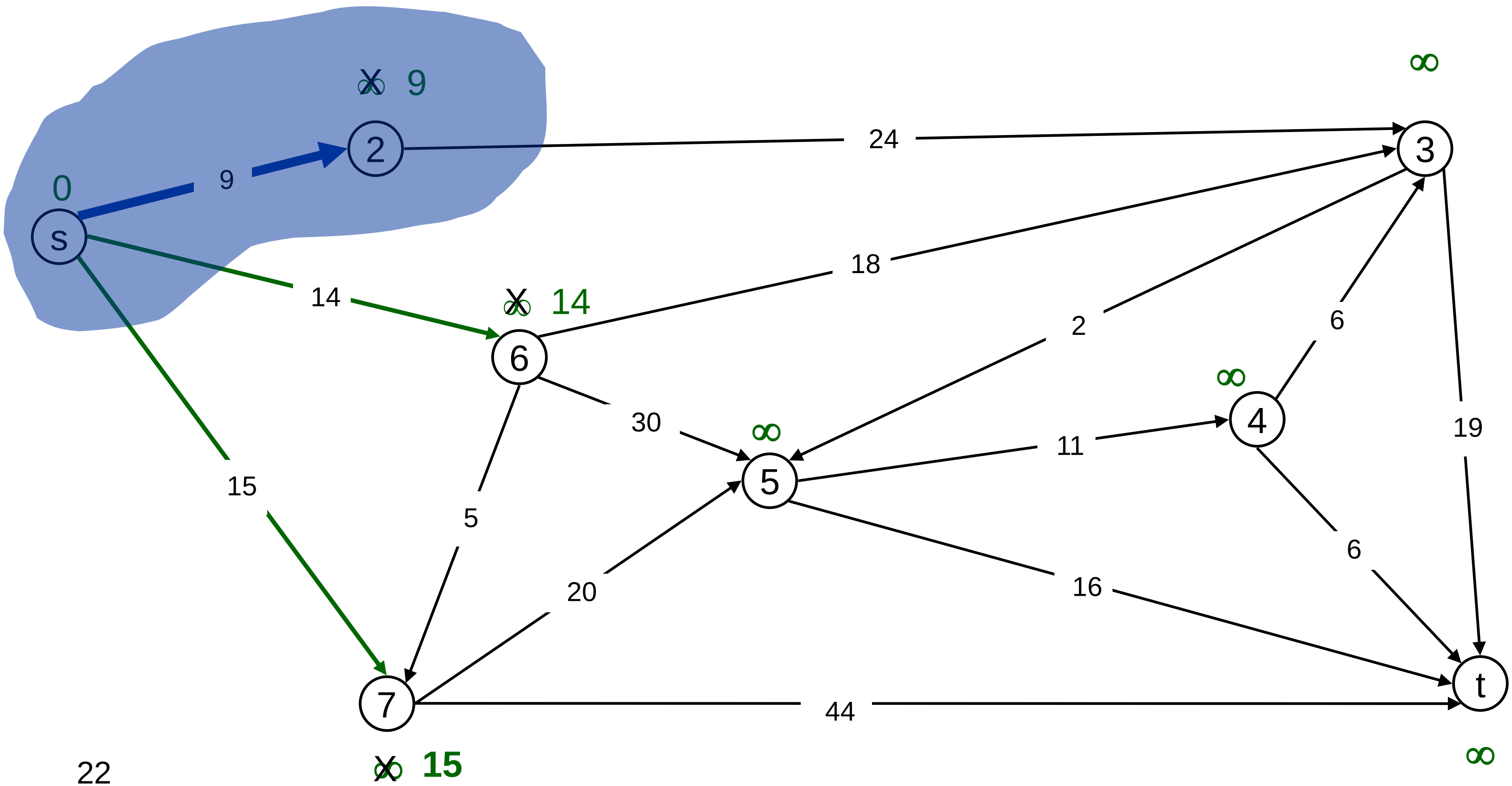


Dijkstra's Shortest Path Algorithm

S = { s, 2 }

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

d	s	2	3	4	5	6	7	t
	0	9	∞	∞	∞	14	15	∞

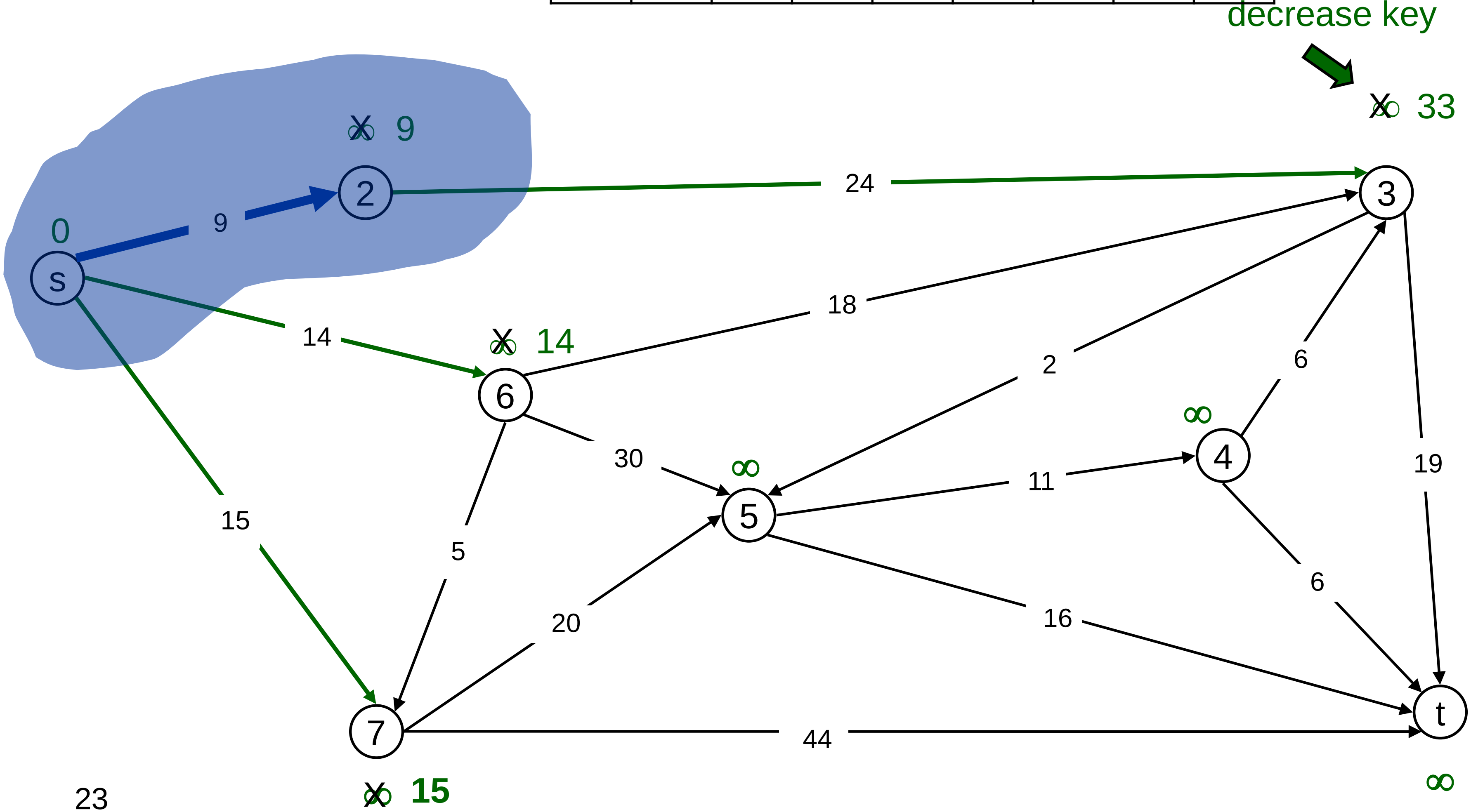


Dijkstra's Shortest Path Algorithm

S = { s, 2 }

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

d	s	2	3	4	5	6	7	t
	0	9	33	∞	∞	14	15	∞

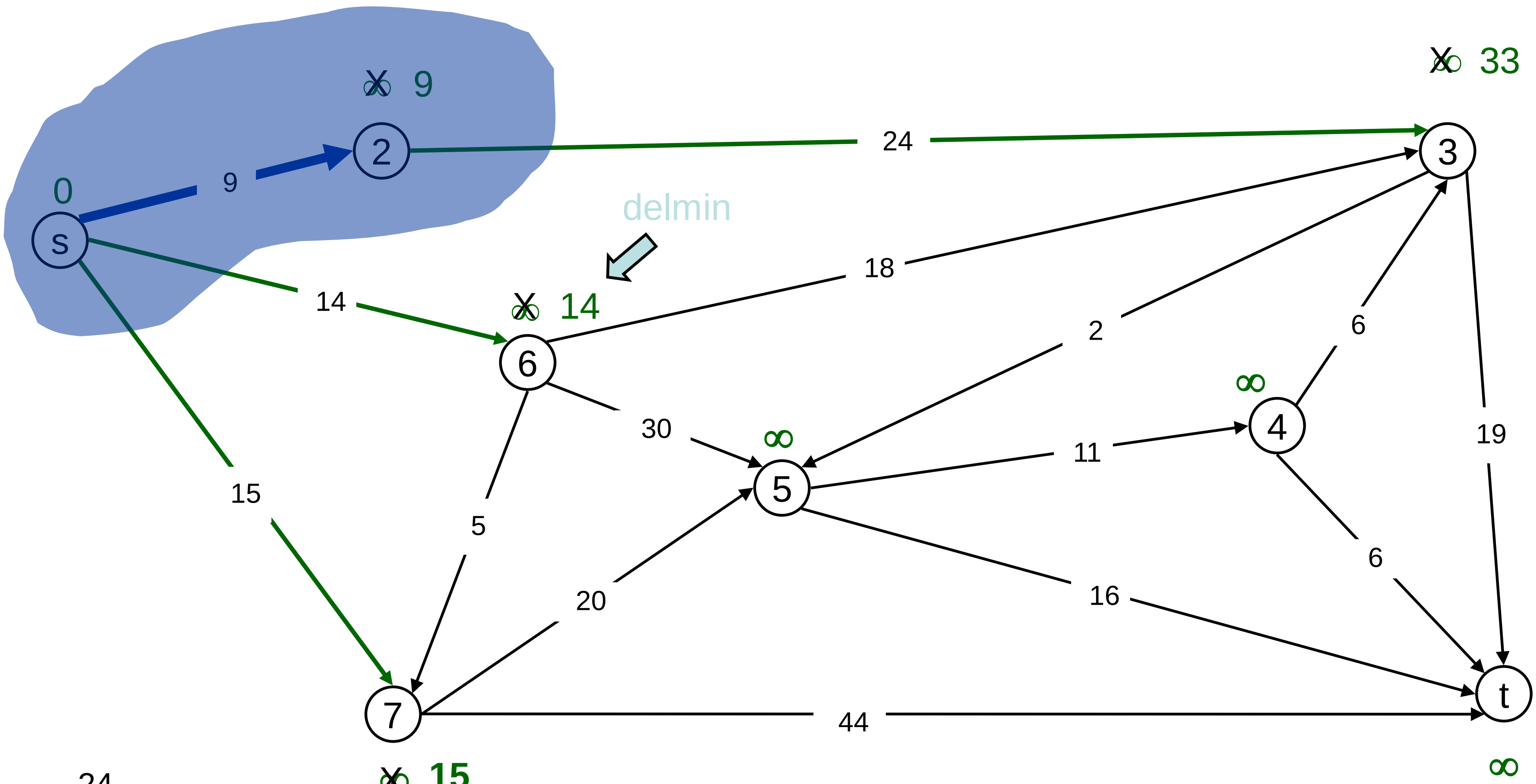


Dijkstra's Shortest Path Algorithm

S = { s, 2 }

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

d	s	2	3	4	5	6	7	t
	0	9	33	∞	∞	14	15	∞

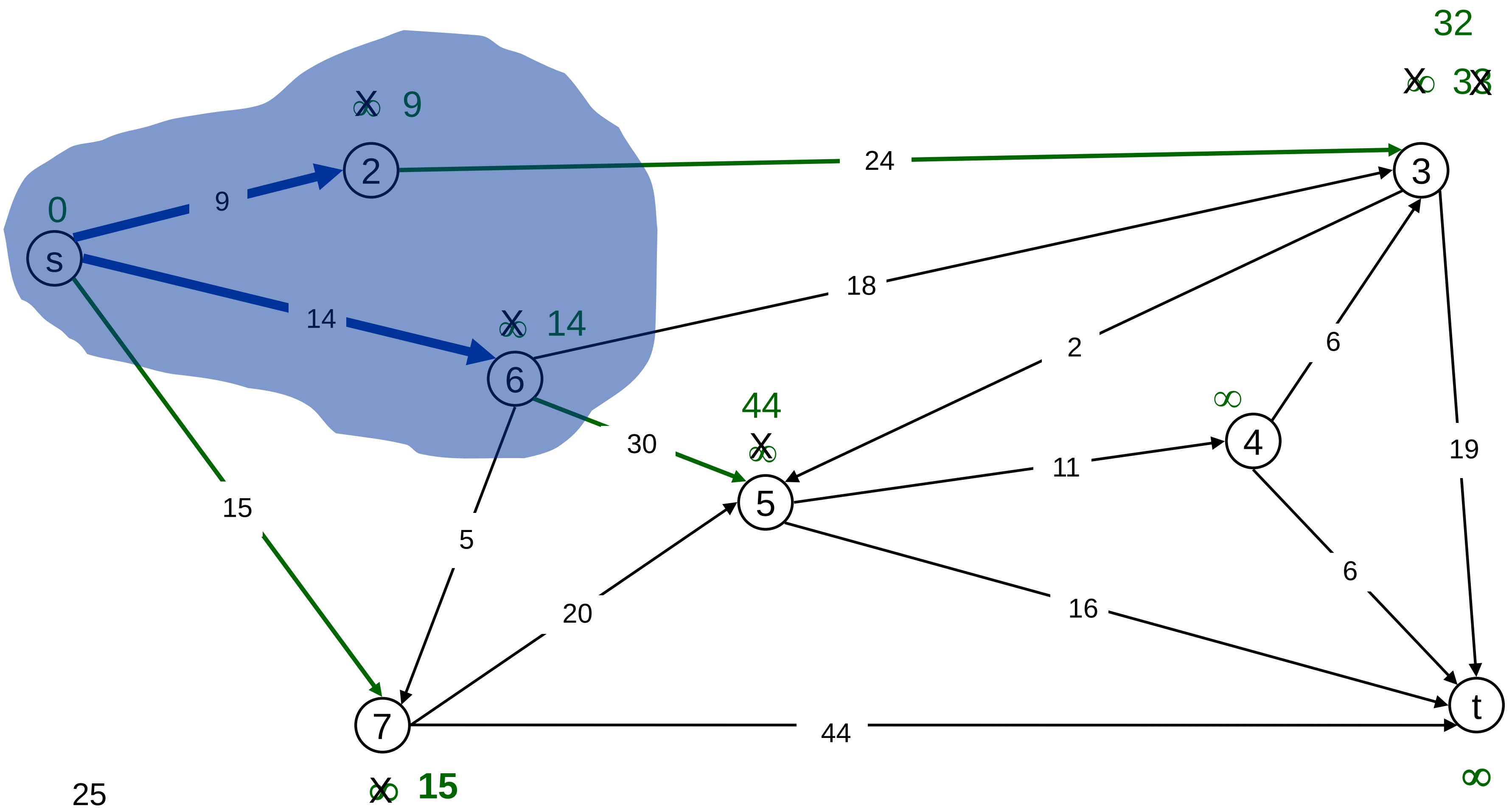


Dijkstra's Shortest Path Algorithm

S = { s, 2, 6 }

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

d	s	2	3	4	5	6	7	t
	0	9	32	∞	44	14	15	∞

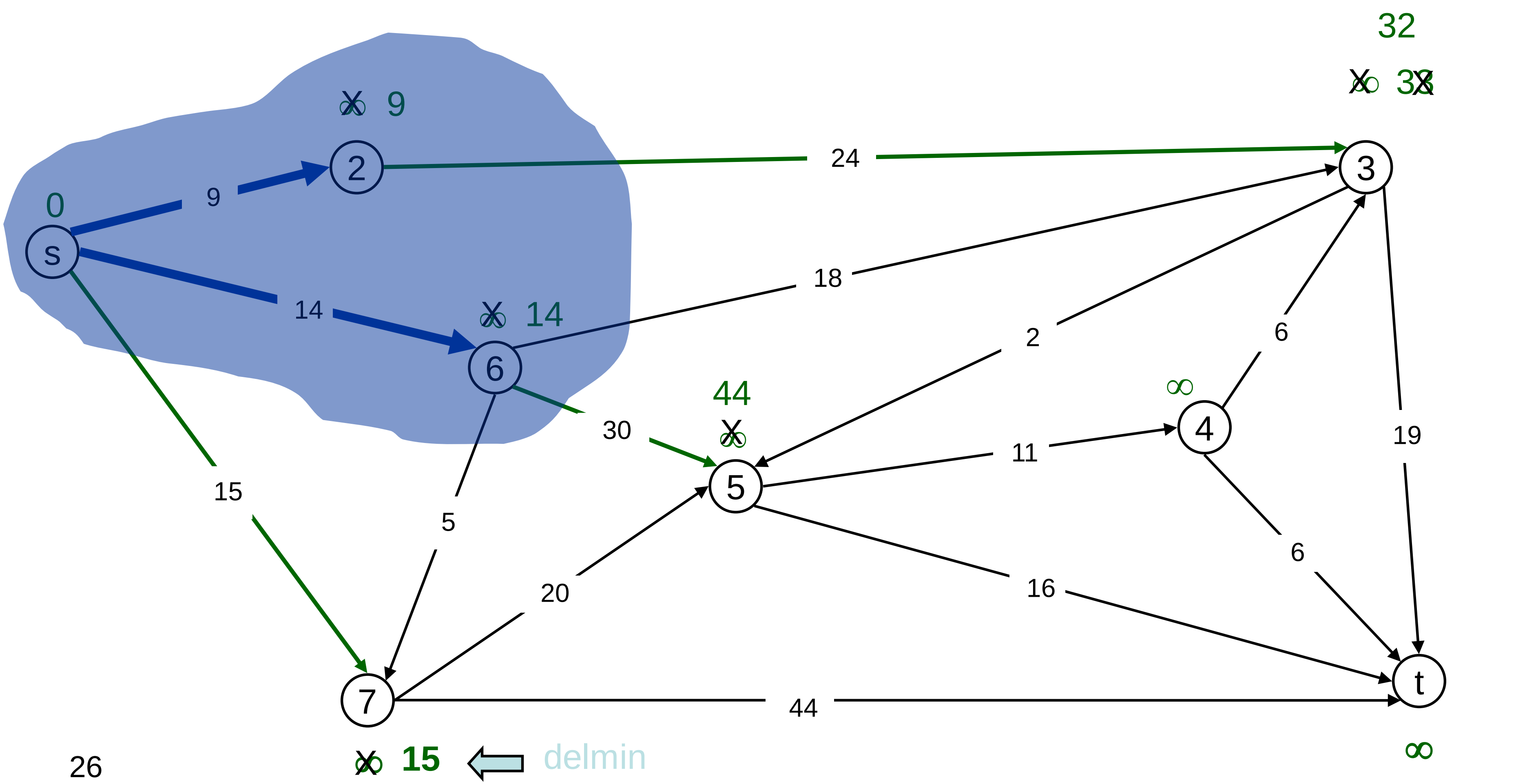


Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7 }

Q = { 3, 4, 5, t }

d	s	2	3	4	5	6	7	t
	0	9	32	∞	44	14	15	∞

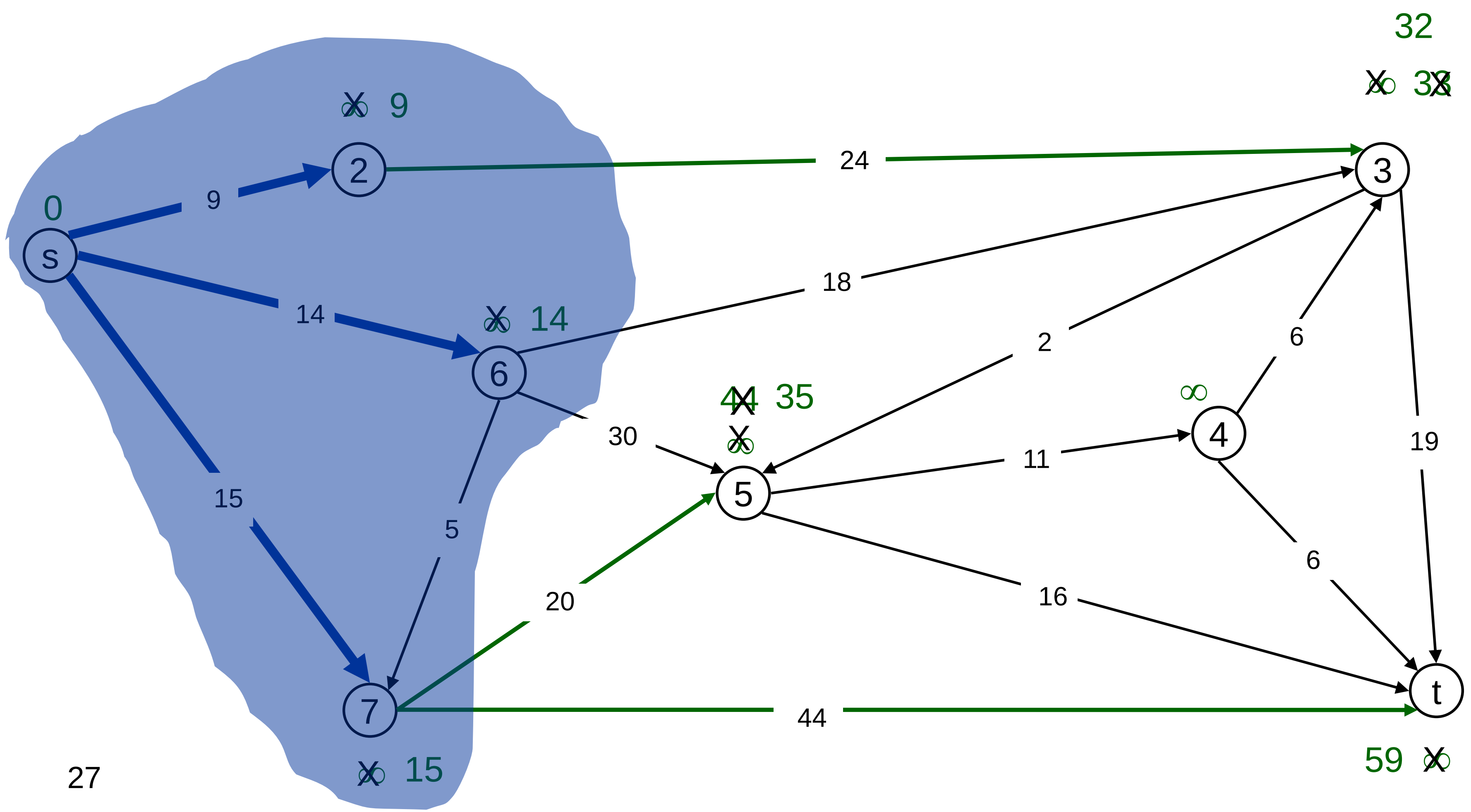


Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7 }

Q = { 3, 4, 5, t }

d	s	2	3	4	5	6	7	t
	0	9	32	∞	35	14	15	59

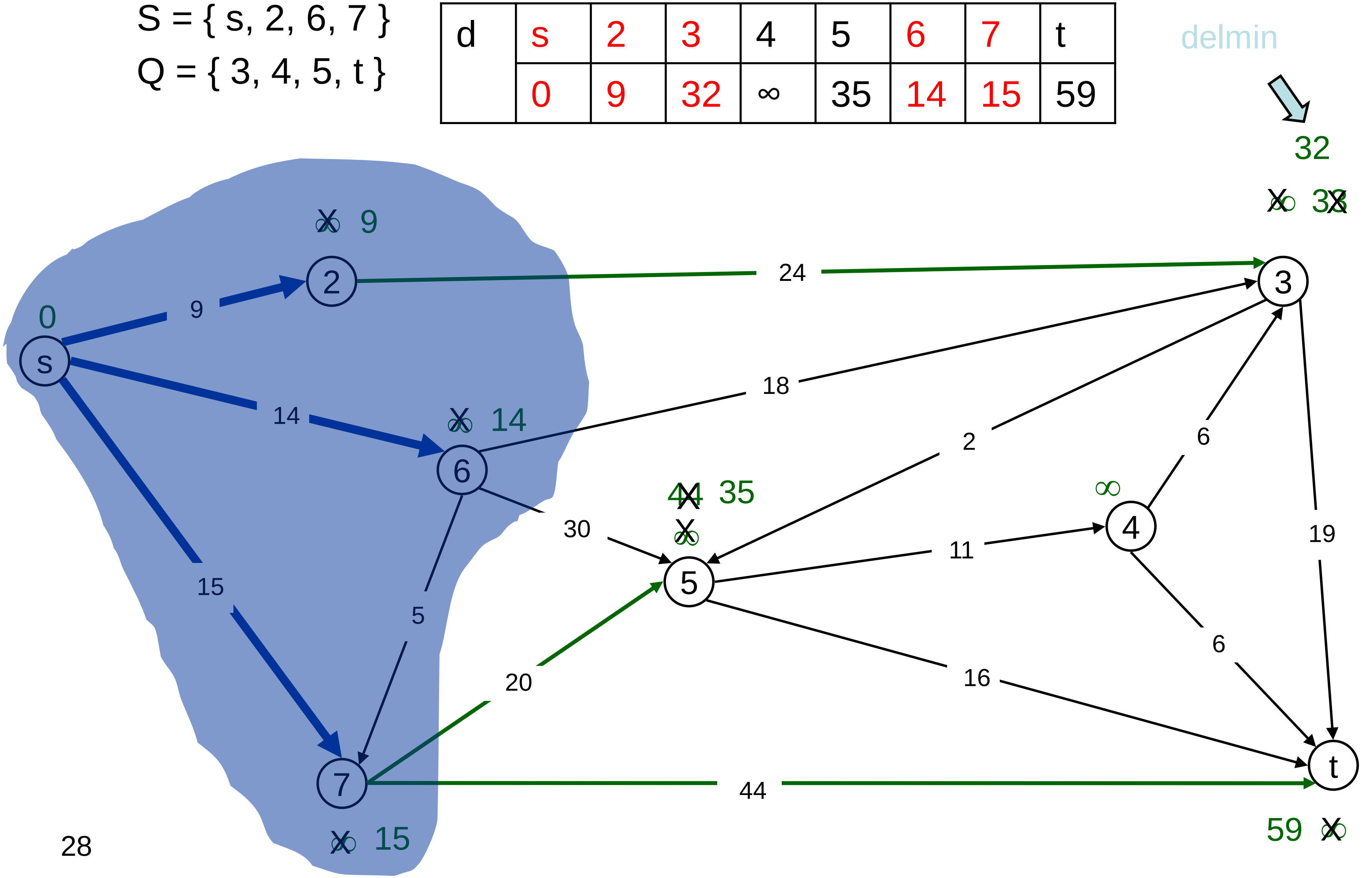


Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7 }

Q = { 3, 4, 5, t }

d	s	2	3	4	5	6	7	t
	0	9	32	∞	35	14	15	59

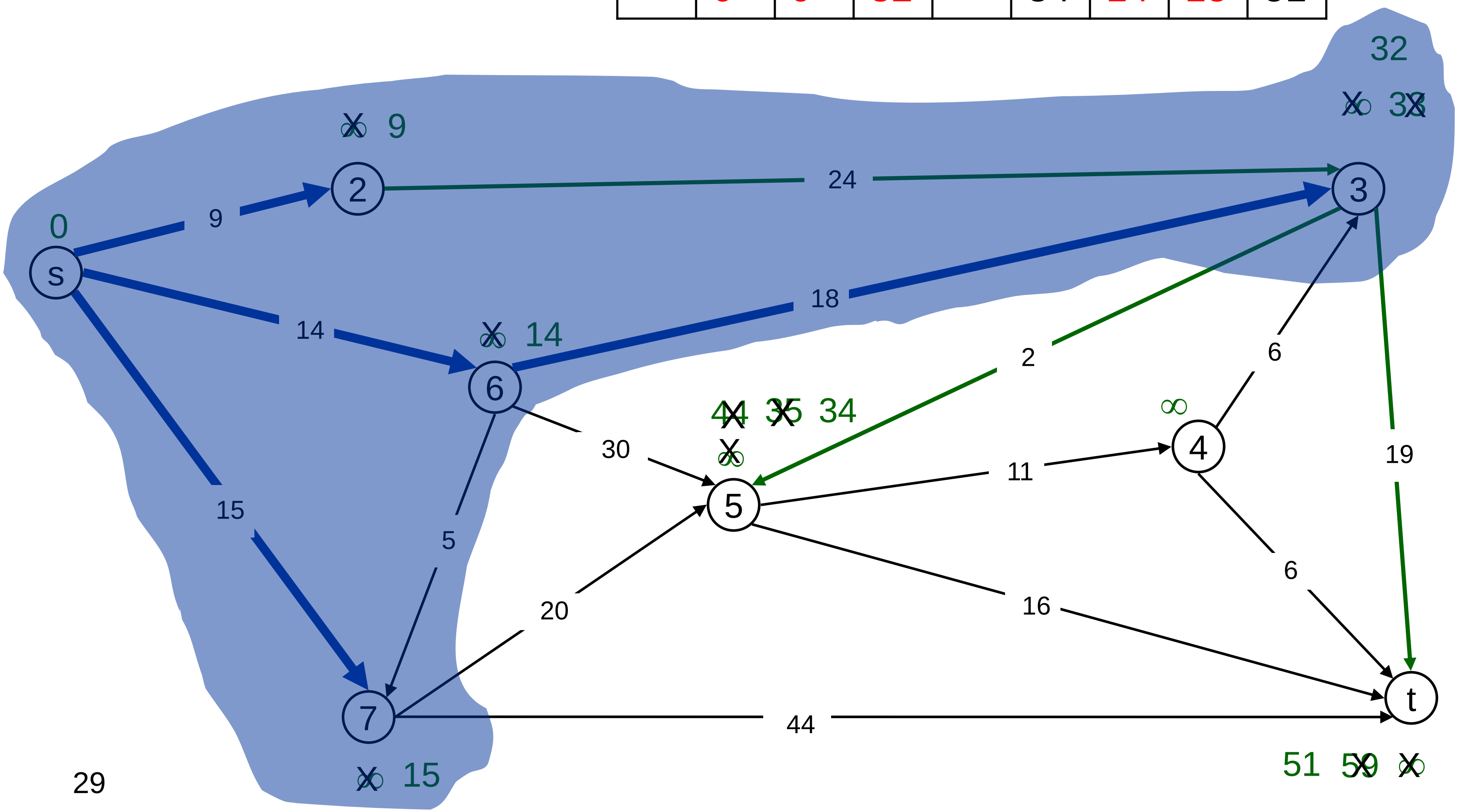


Dijkstra's Shortest Path Algorithm

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

Q = { 4, 5, t }

d	s	2	3	4	5	6	7	t
	0	9	32	∞	34	14	15	51

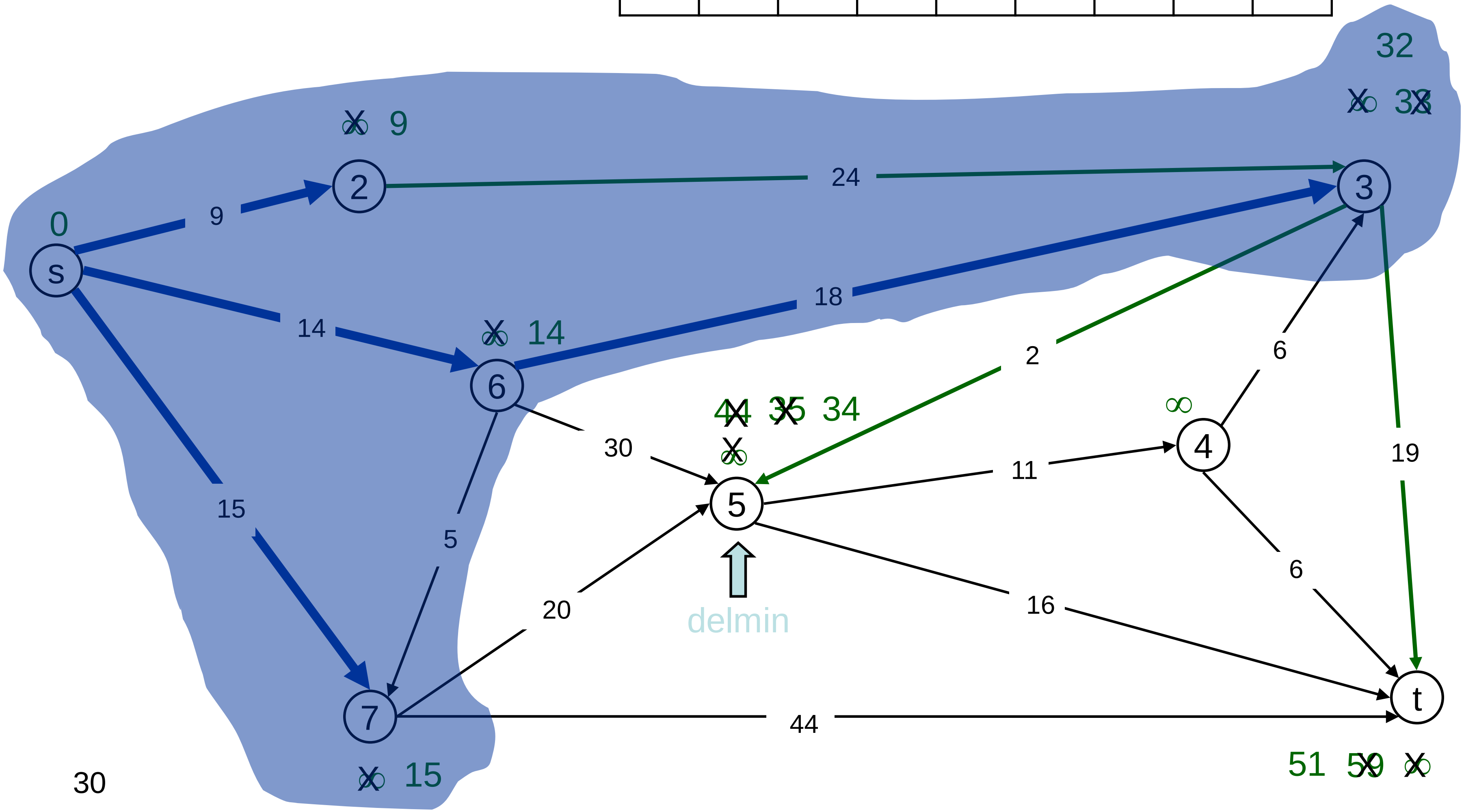


Dijkstra's Shortest Path Algorithm

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

Q = { 4, 5, t }

d	s	2	3	4	5	6	7	t
	0	9	32	∞	34	14	15	51

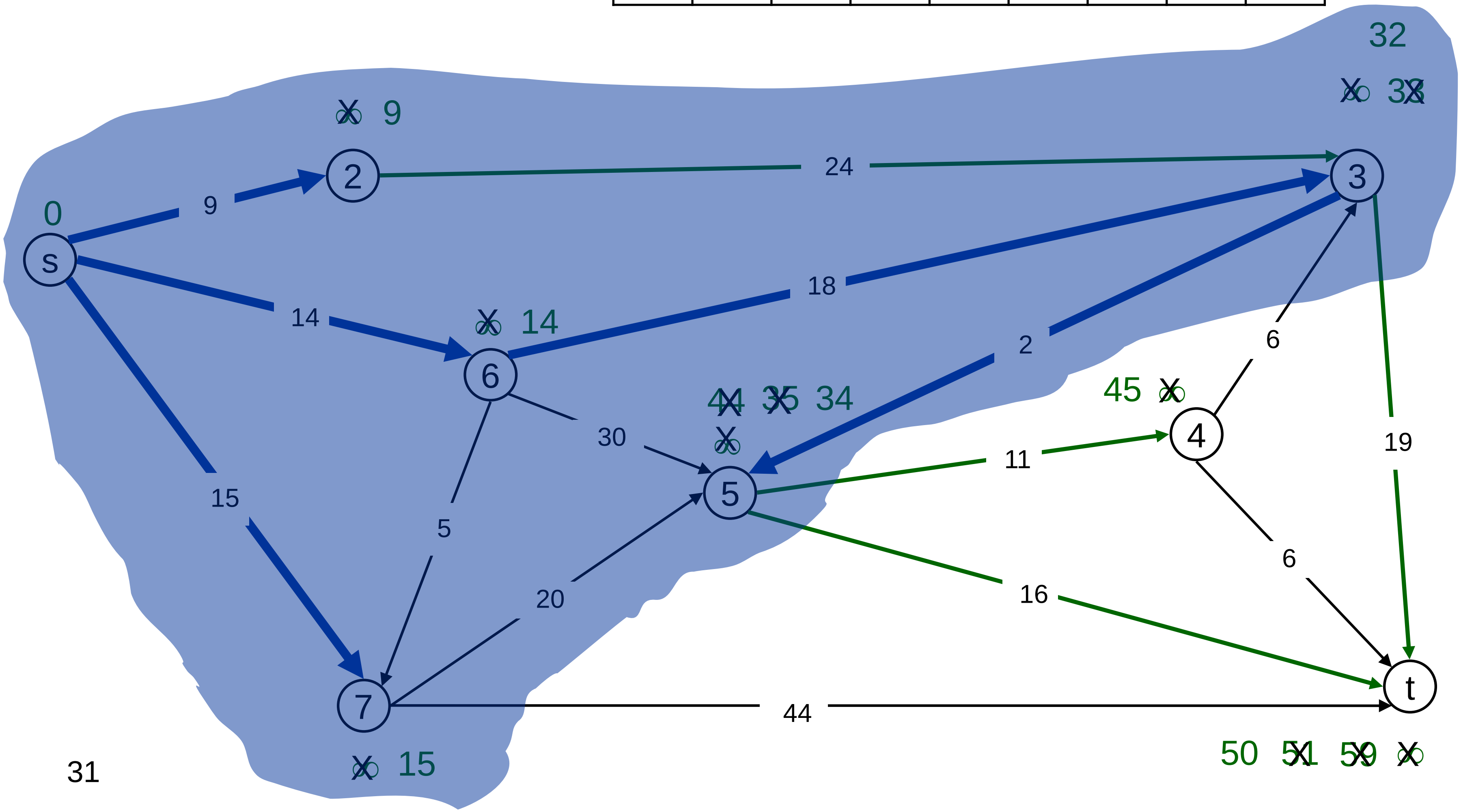


Dijkstra's Shortest Path Algorithm

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

Q = { 4, t }

d	s	2	3	4	5	6	7	t
	0	9	32	45	34	14	15	50

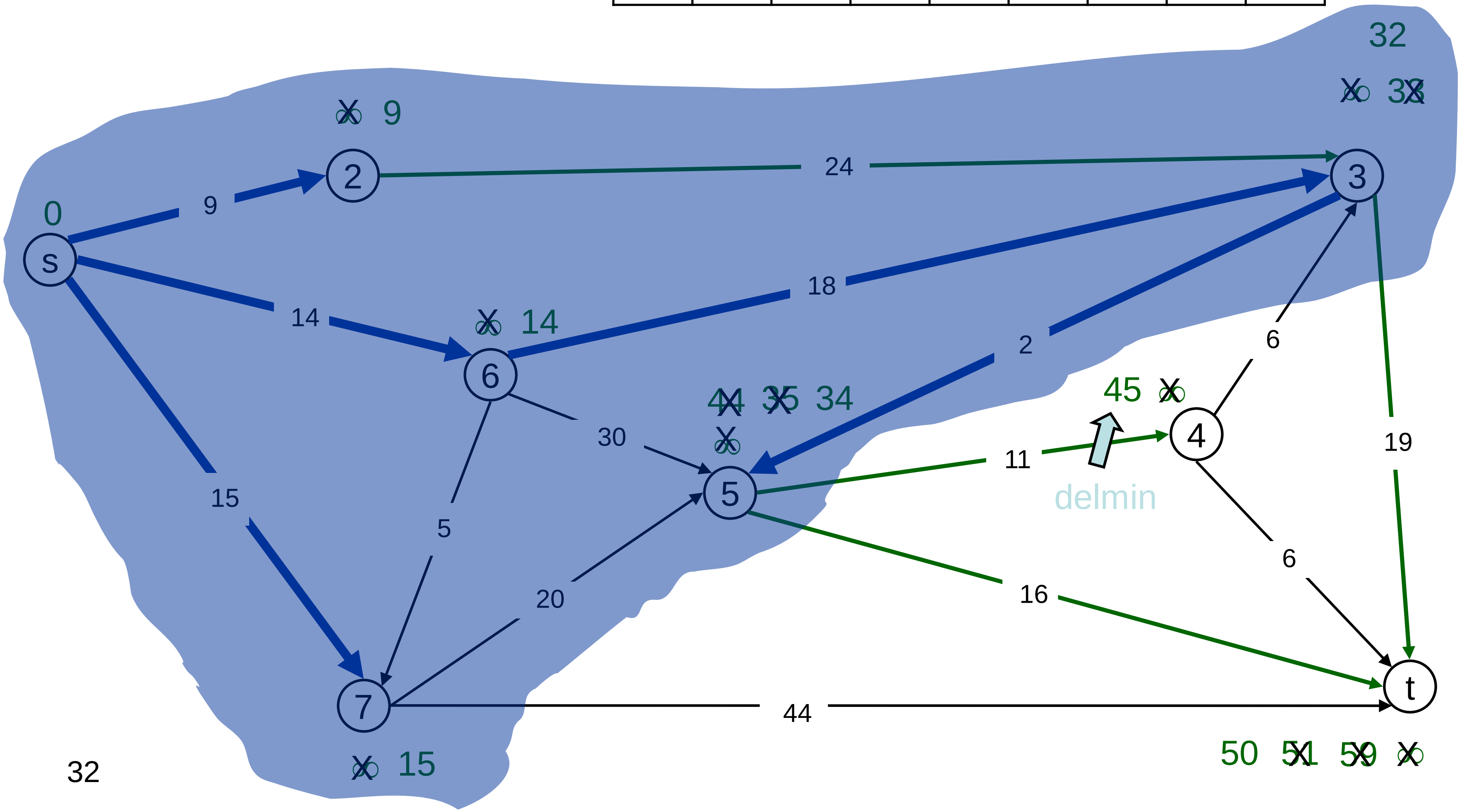


Dijkstra's Shortest Path Algorithm

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

Q = { 4, t }

d	s	2	3	4	5	6	7	t
	0	9	32	45	34	14	15	50

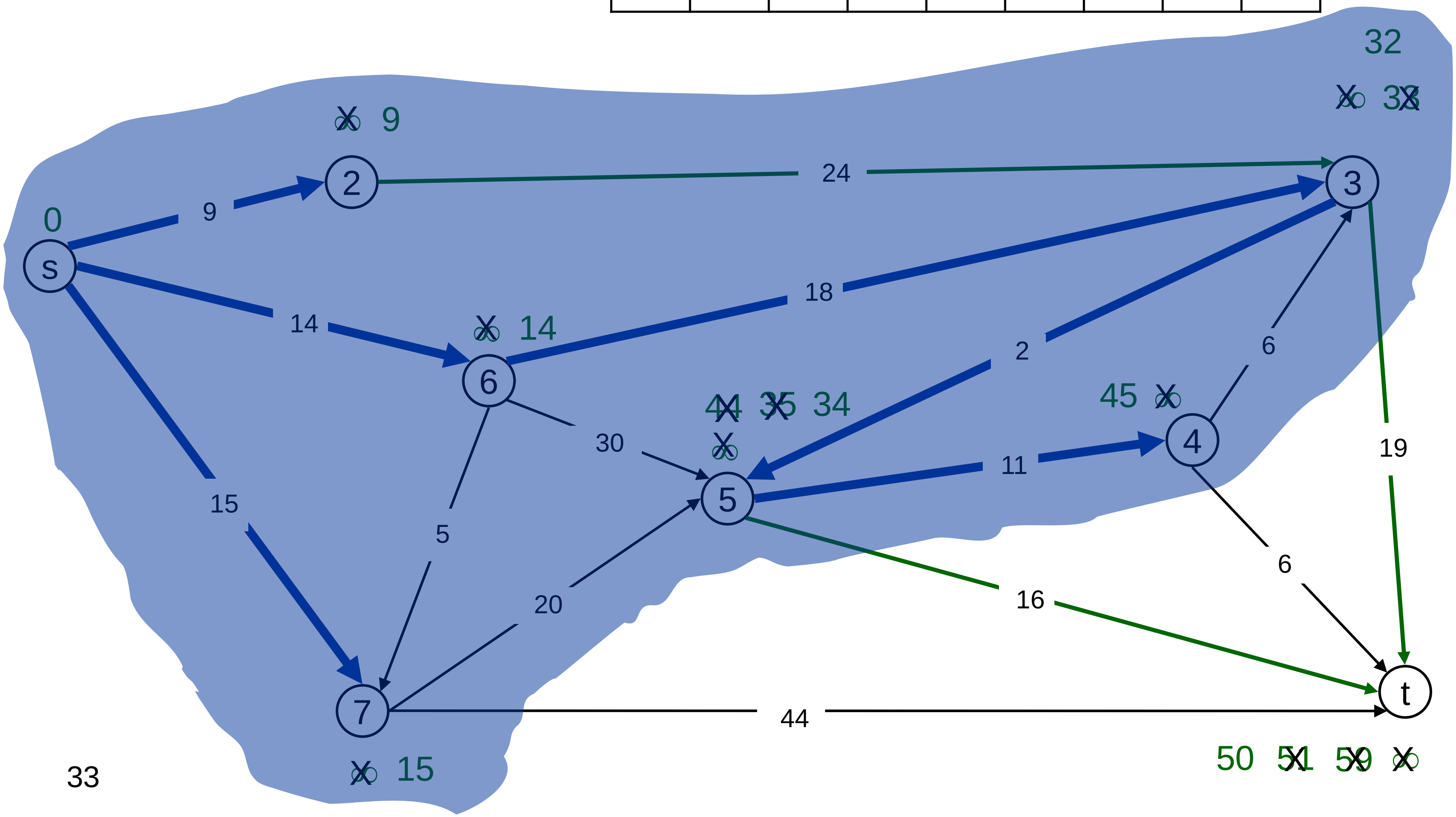


Dijkstra's Shortest Path Algorithm

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

Q = { t }

d	s	2	3	4	5	6	7	t
	0	9	32	45	34	14	15	50

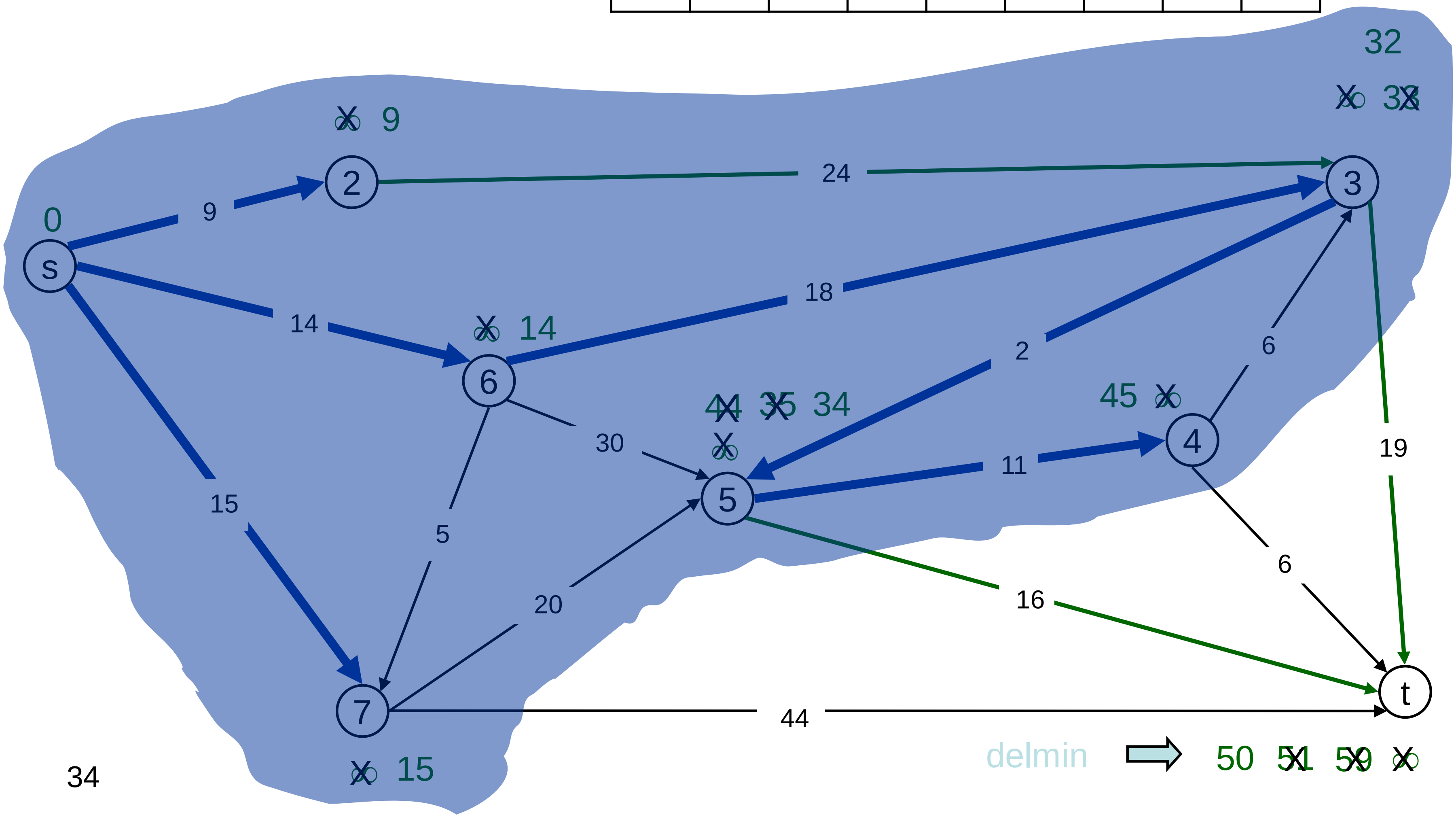


Dijkstra's Shortest Path Algorithm

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

Q = { t }

d	s	2	3	4	5	6	7	t
	0	9	32	45	34	14	15	50

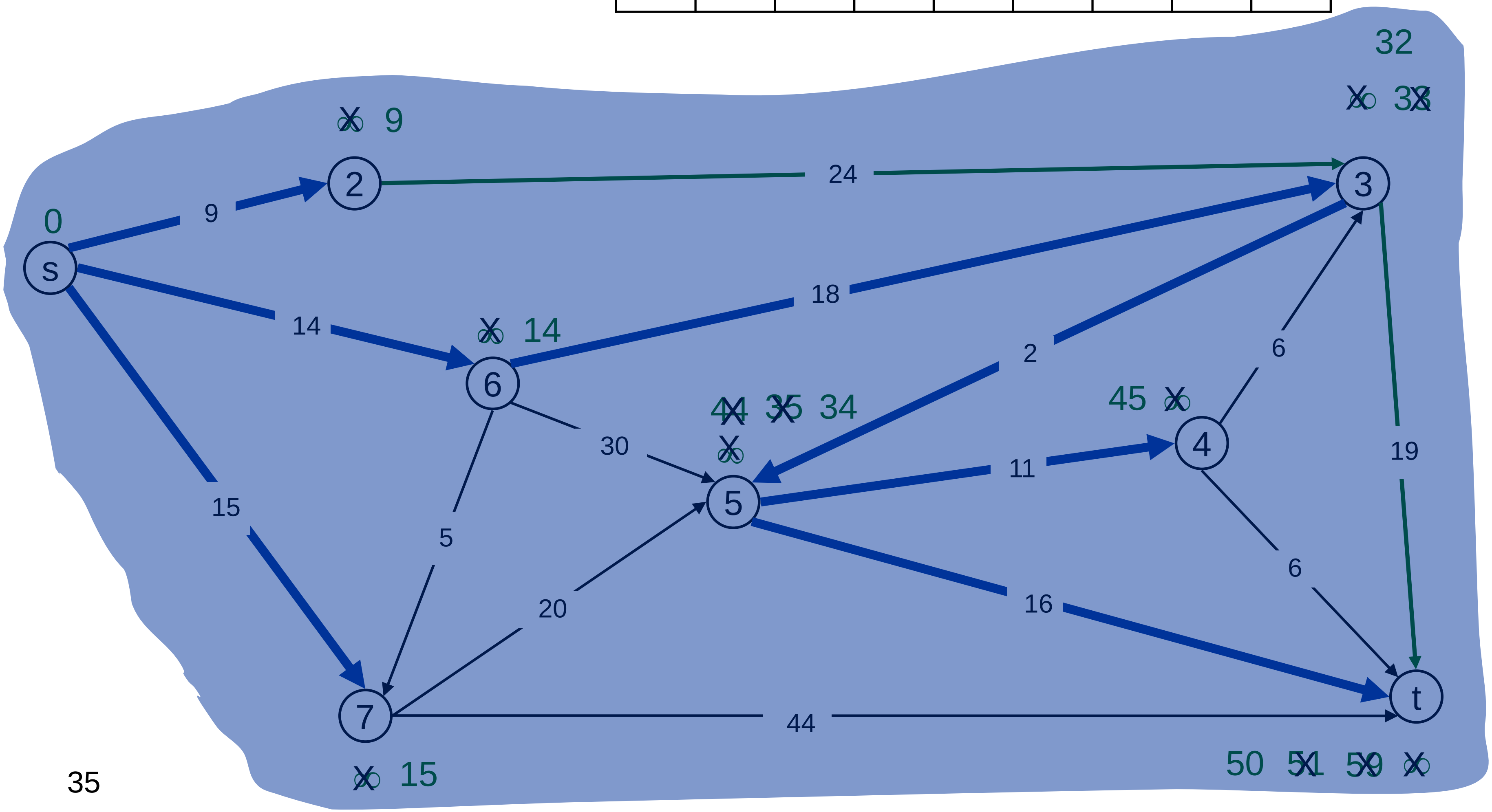


Dijkstra's Shortest Path Algorithm

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

Q = { }

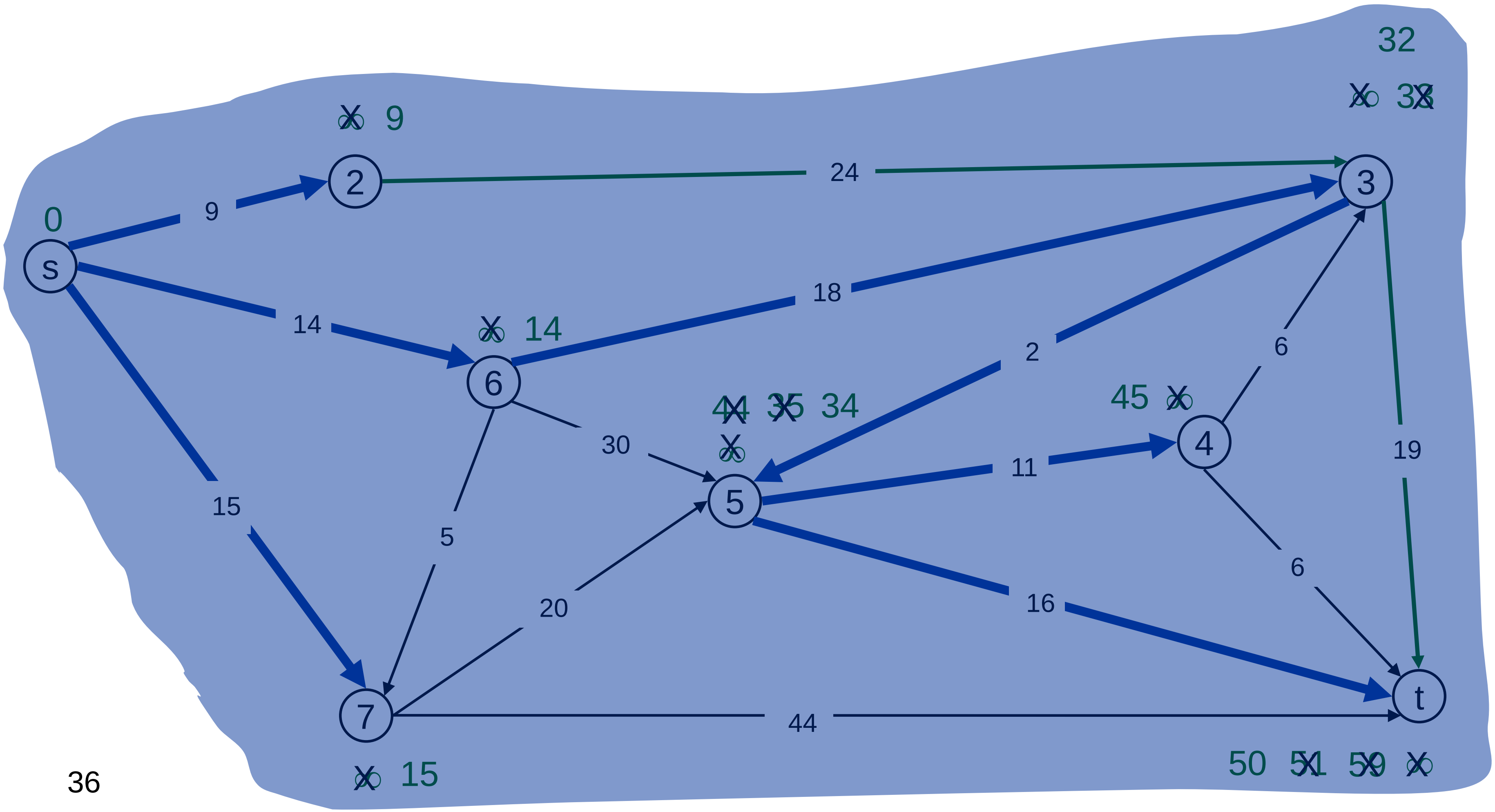
d	s	2	3	4	5	6	7	t
	0	9	32	45	34	14	15	50



Dijkstra's Shortest Path Algorithm

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

Q = { }



Time Complexity: Using List

The simplest implementation of the Dijkstra's algorithm stores vertices in an ordinary linked list or array

- Good for dense graphs (many edges)

- |V| vertices and |E| edges
- Initialization O(|V|)
- While loop O(|V|)
  - Find and remove min distance vertices O(|V|)
- Potentially |E| updates
    - Update costs O(1)

Total time $O(|V^2| + |E|) = O(|V^2|)$

Time Complexity: Priority Queue

For sparse graphs, (i.e. graphs with much less than $|V^2|$ edges) Dijkstra's implemented more efficiently by *priority queue*

- Initialization O(|V|) using O(|V|) buildHeap
- While loop O(|V|)
  - Find and remove min distance vertices O(log |V|) using O(log |V|) deleteMin

- Potentially |E| updates
  - Update costs O(log |V|) using decreaseKey

Total time O(|V|log|V| + |E|log|V|) = O(|E|log|V|)

- |V| = O(|E|) assuming a connected graph

Dijkstra (G, w, s)

INITIALIZE-SINGLE-SOURCE(V, s) $\longleftarrow$ $\Theta(V)$

$S \leftarrow \emptyset$

$Q \leftarrow V[G]$ $\longleftarrow$ O(V) build min-heap

while $Q \neq \emptyset$ $\longleftarrow$ Executed O(V) times

    do $u \leftarrow$ EXTRACT-MIN(Q) $\longleftarrow$ O(lgV)

        $S \leftarrow S \cup \{u\}$

        for each vertex $v \in Adj[u]$

            do RELAX(u, v, w) $\longleftarrow$ O(E) times; O(lgV)

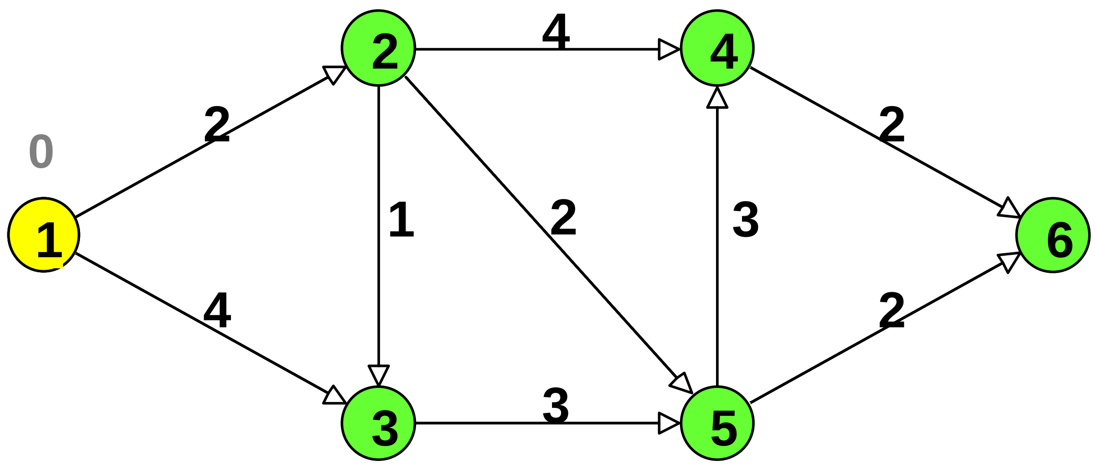
Running time: $O(V\lg V + E\lg V) = O(E\lg V)$

Dijkstra's Running Time

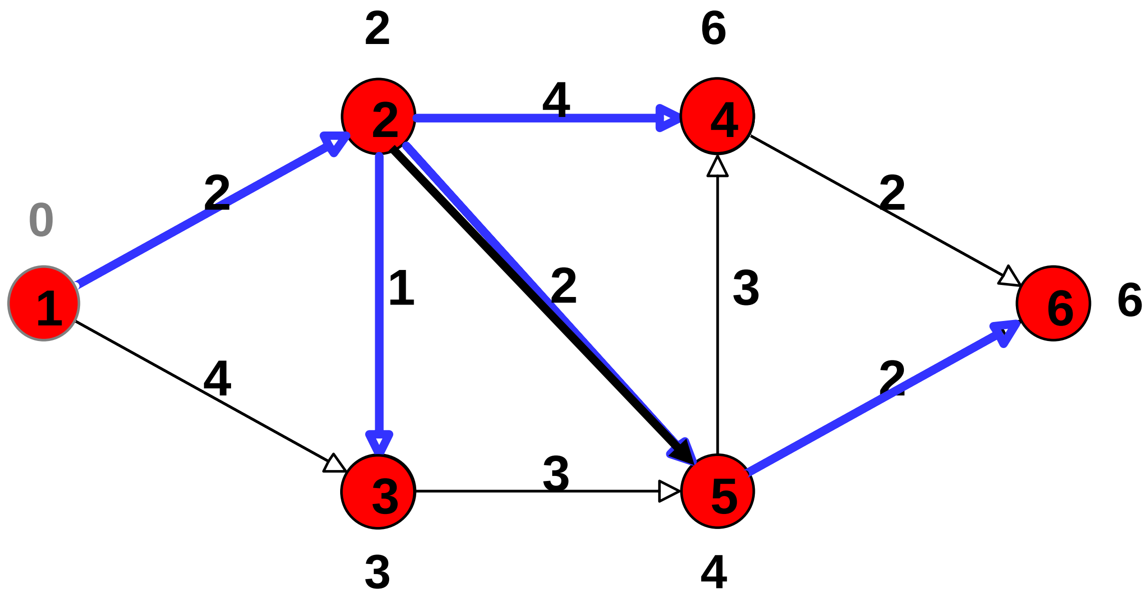
- Extract-Min executed $|V|$ time
- Decrease-Key executed $|E|$ time
- Time = $|V|\ T_{\text{Extract-Min}} + |E|\ T_{\text{Decrease-Key}}$
- T depends on different Q implementations

Q	T(Extract-Min)	T(Decrease-Key)	Total
array	$O(V)$	$O(1)$	$O(V^2)$
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$	$O(1)$ (amort.)	$O(V \lg V + E)$

An Example

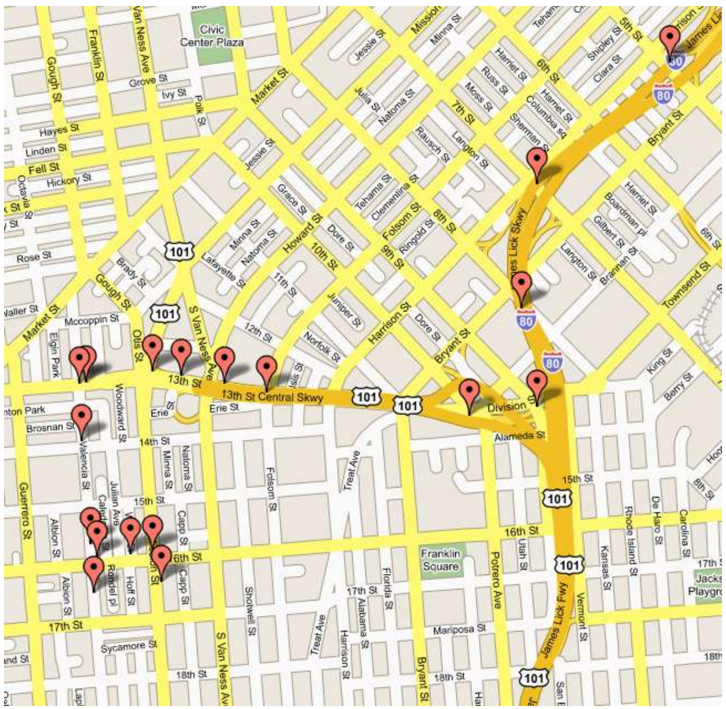


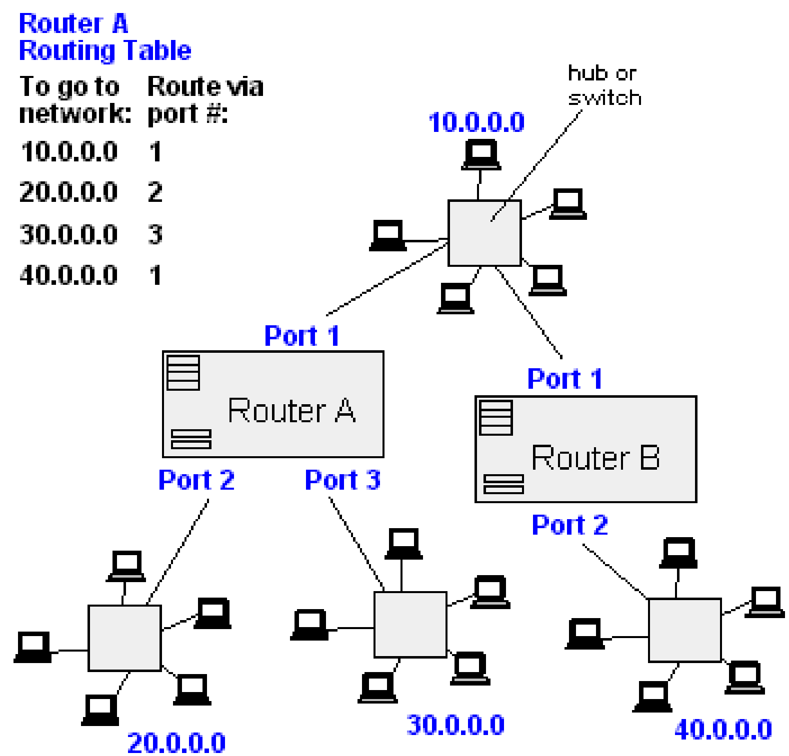
An Example



Applications of Dijkstra's Algorithm

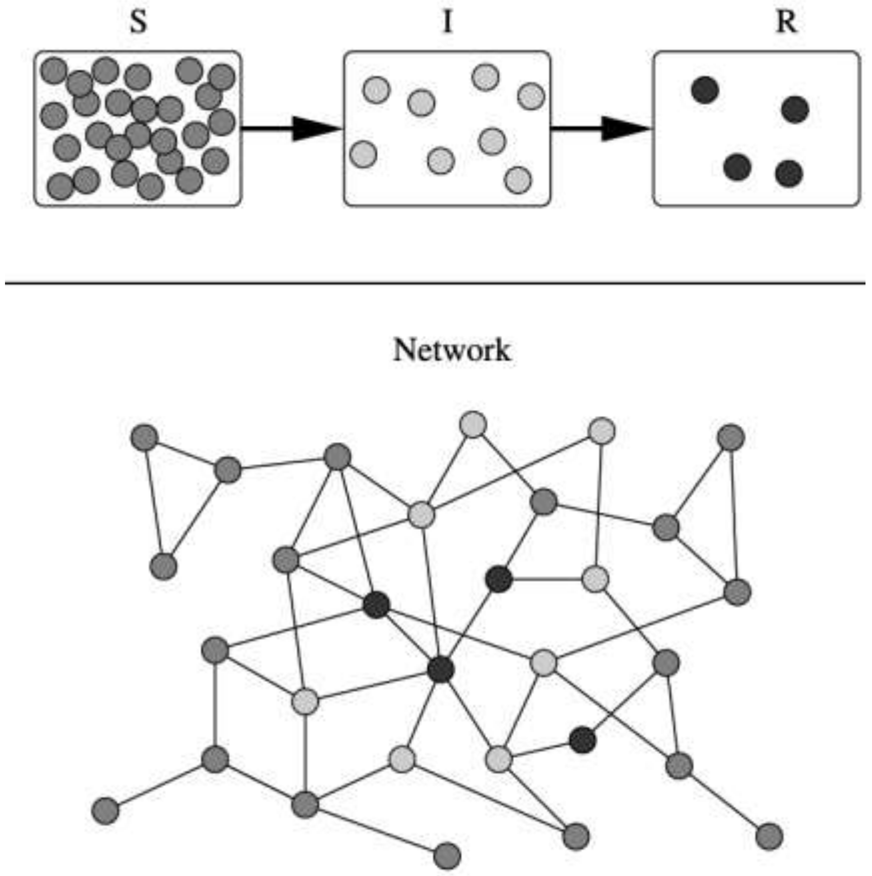
- Traffic Information Systems are most prominent use
- Mapping (Map Quest, Google Maps)
- Routing Systems





Applications of Dijkstra's Algorithm

- One particularly relevant this week: epidemiology
- Prof. Lauren Meyers (MIT, Biology Dept.) uses networks to model the spread of infectious diseases and design prevention and response strategies.
- Vertices represent individuals, and edges their possible contacts. It is useful to calculate how a particular individual is connected to others.
- Knowing the shortest path lengths to other individuals can be a relevant indicator of the potential of a particular individual to infect others.



Drawbacks of Dijkstra algorithm

- The major disadvantage of the algorithm is the fact that it does a blind search there by consuming a lot of time waste of necessary resources.
- Another disadvantage is that it cannot handle negative edges. This leads to acyclic graphs and most often cannot obtain the right shortest path.

References

http://staff.ustc.edu.cn/~csli/graduate/algorithms/book6/chap21.htm

https://www.youtube.com/watch?v=nZ0nFTvQez0

https://www.youtube.com/watch?v=E_AgyAI8lsc

https://brilliant.org/wiki/fibonacci-heap/

https://www.academia.edu/37584931/9_Dijkstra_pptx?auto=download