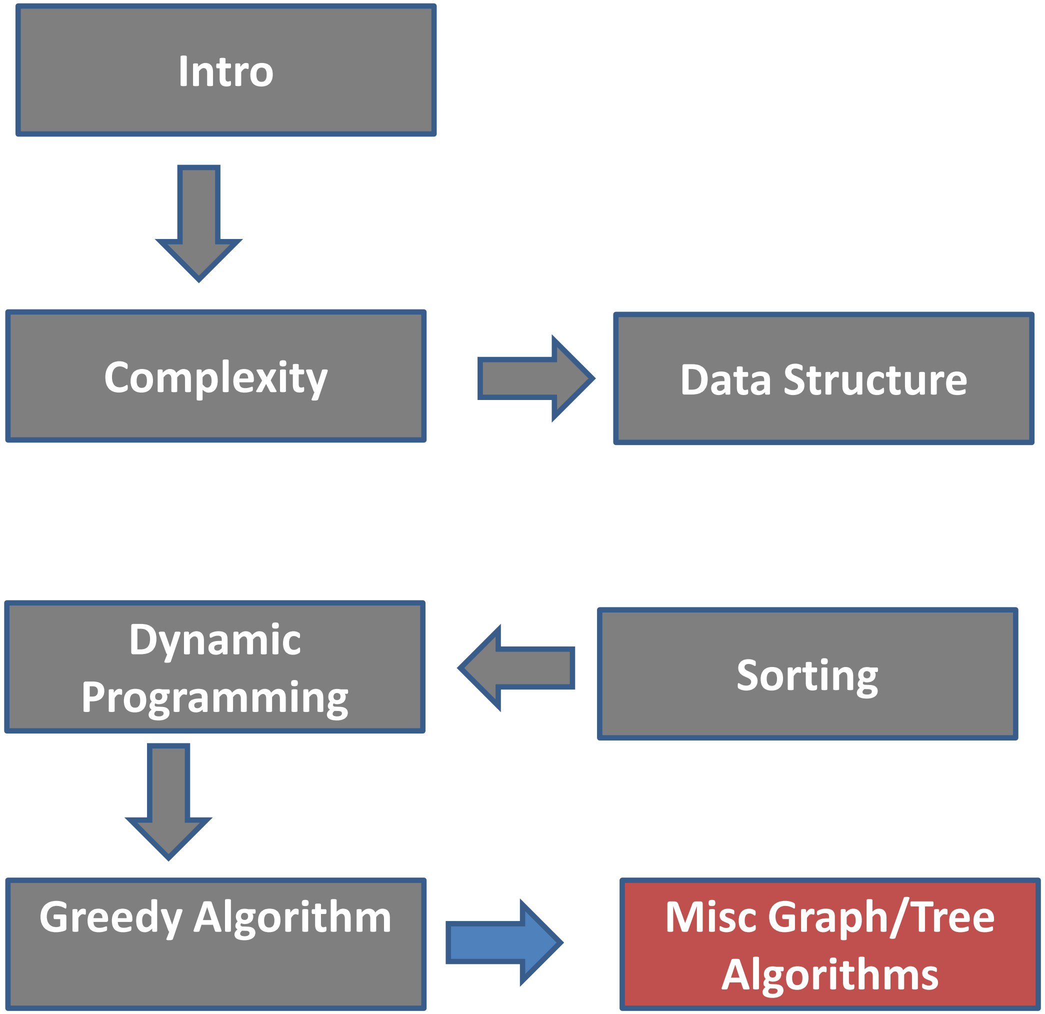
An Introduction to Algorithms

By

Hossein Rahmani

h_rahmani@iust.ac.ir
http://webpages.iust.ac.ir/h_rahmani/


Intro
Complexity
Data Structure
Trees
Hash Functions
Sorting
Dynamic Programming
Greedy Algorithm
Misc Graph/Tree Algorithms

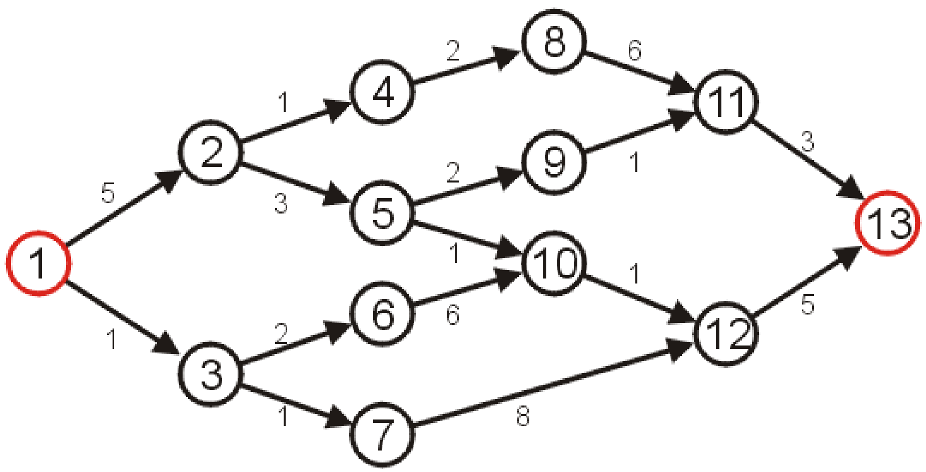
Shortest Path

- Given a weighted directed graph, one common problem is finding the shortest path between two given vertices
- Recall that in a weighted graph, the *length* of a path is the sum of the weights of each of the edges in that path

Shortest Path

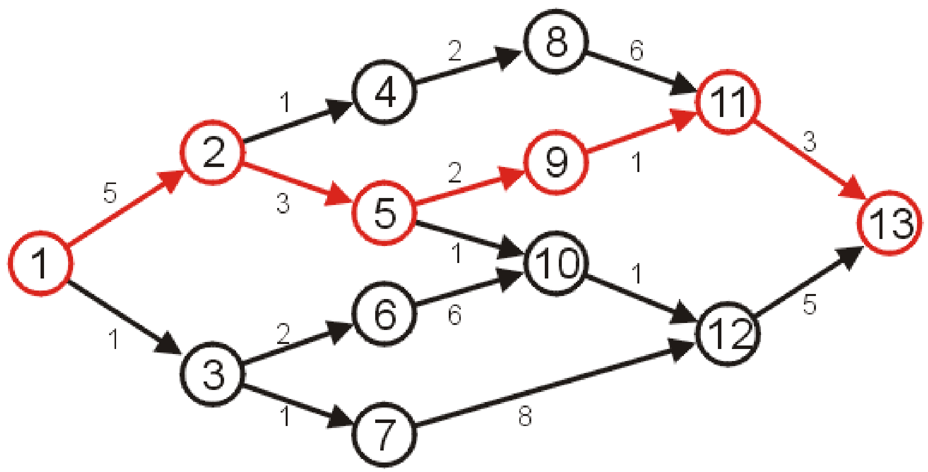
- Given the graph below, suppose we wish to find the shortest path

  from <u>vertex 1</u> to <u>vertex 13</u>



Shortest Path

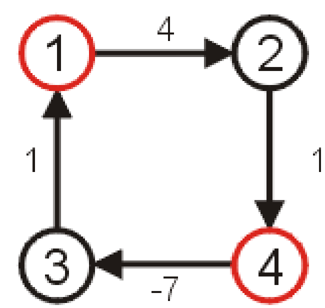
- After some consideration, we may determine that the shortest path is as follows, with <u>length 14</u>



- Other paths exists, but they are longer

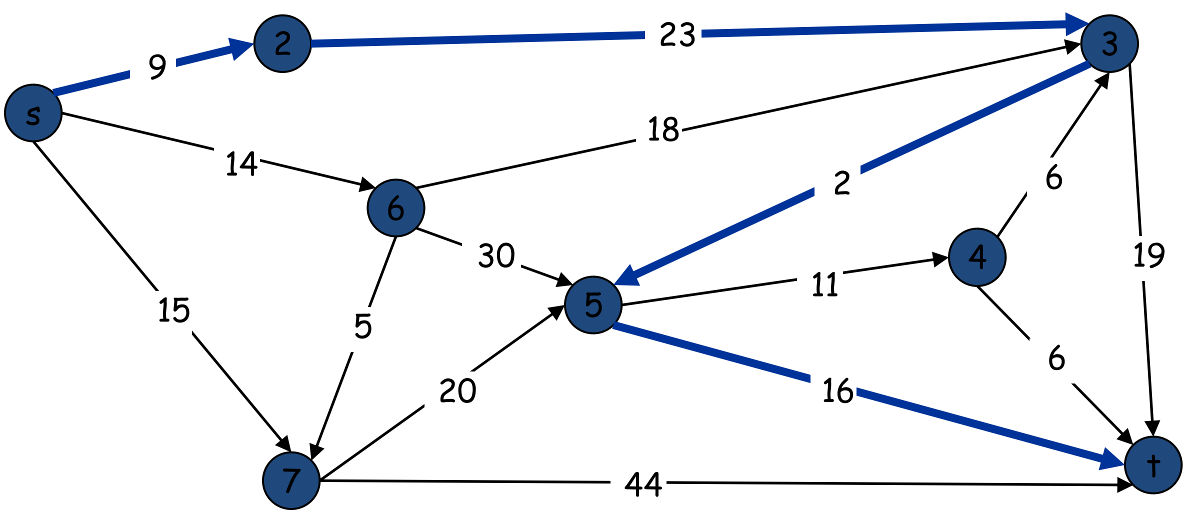
Negative Cycles

- Clearly, if we have negative vertices, it may be possible to end up in a cycle whereby each pass through the cycle decreases the total *length*
- Thus, a shortest length would be undefined for such a graph
- Consider the shortest path from vertex 1 to 4...
- We will only consider non-negative weights.



Shortest Path Example

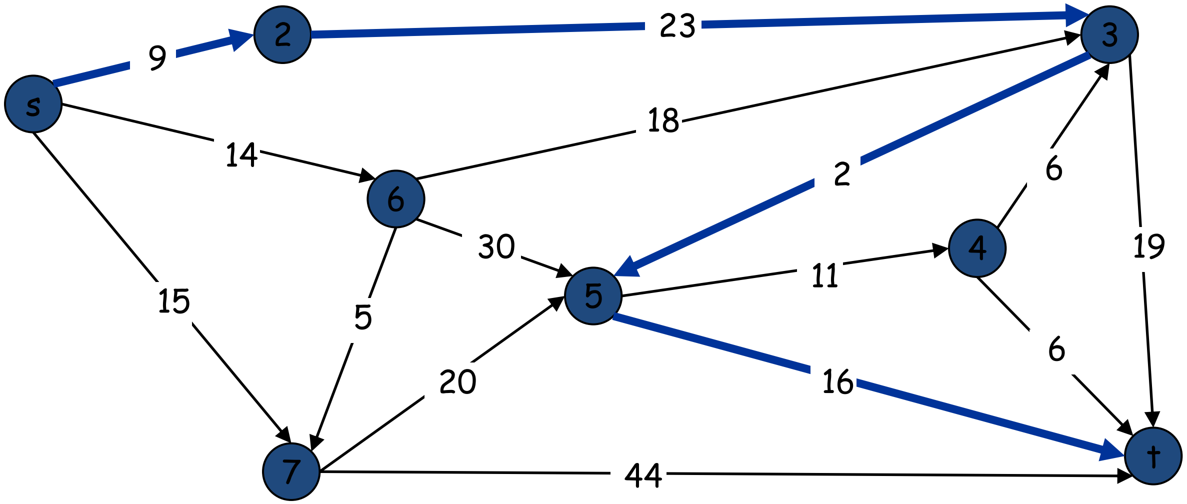
- Given:
  - Weighted Directed graph G = (V, E).
  - Source *s*, destination *t*.
- Find shortest directed path from *s* to *t*.



Cost of path s-2-3-5-t
= 9 + 23 + 2 + 16
= 48.

Discussion Items

- How many possible paths are there from *s* to *t*?
- Can we safely ignore cycles? If so, how?
- Any suggestions on how to reduce the set of possibilities?
- Can we determine a lower bound on the complexity like we did for comparison sorting?



Key Observation

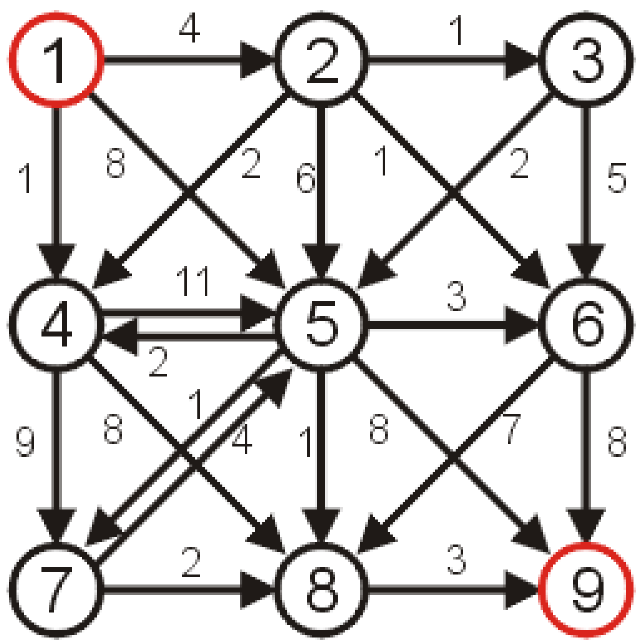
- A key observation is that if the shortest path contains the node *v*, then:
  - It will only contain *v* once, as any cycles will only add to the length.
  - The path from *s* to *v* must be the shortest path to *v* from *s*.
  - The path from *v* to *t* must be the shortest path to *t* from *v*.
- Thus, if we can determine the shortest path to all other vertices that are incident to the target vertex we can easily compute the shortest path.
  - Implies a set of sub-problems on the graph with the target vertex removed.

Dijkstra’s Algorithm

- Works when all of the <u>weights</u> are <u>positive</u>.
- Provides the <u>shortest</u> paths from a source to <u>**all**</u> other vertices in the graph.
  - Can be terminated early once the shortest path to *t* is found if desired.

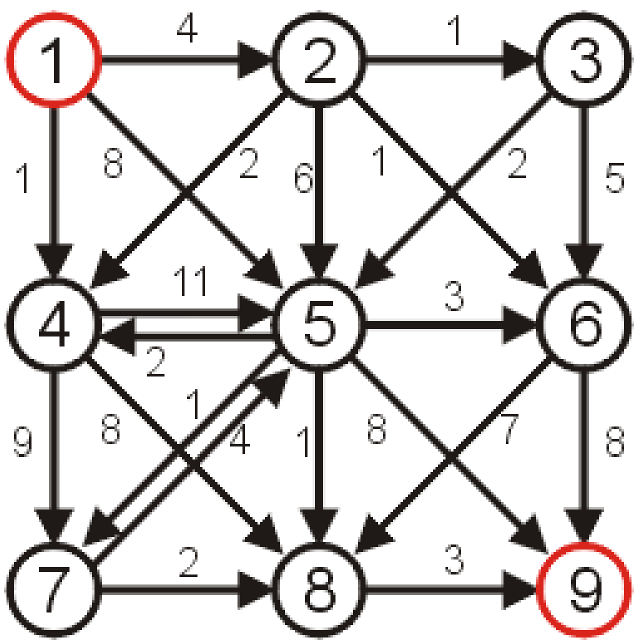
Shortest Path

- Consider the following graph with positive weights and cycles.



Dijkstra's Algorithm

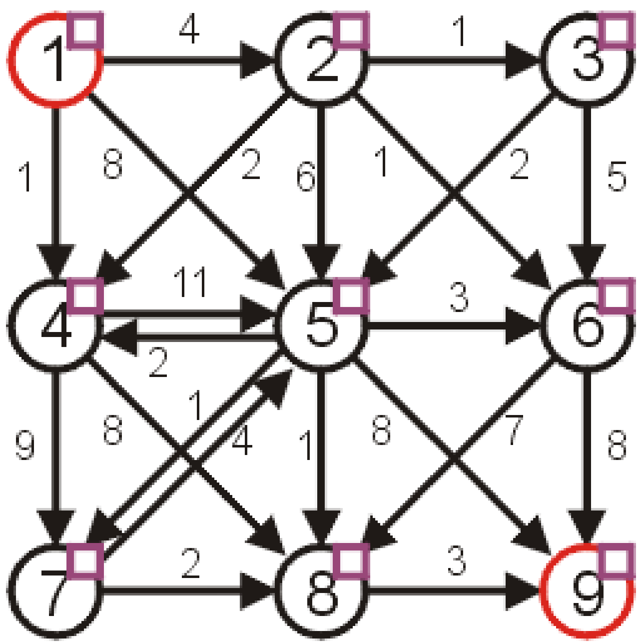
- A first attempt at solving this problem might require an array of Boolean values, all initially false, that indicate whether we have found a path from the source.



1	F
2	F
3	F
4	F
5	F
6	F
7	F
8	F
9	F

Dijkstra's Algorithm

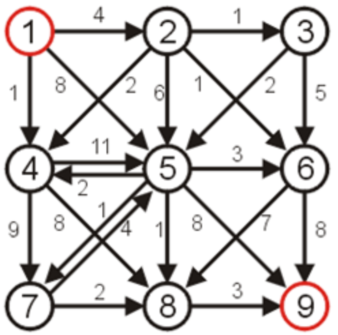
- Graphically, we will denote this with <u>check boxes</u> next to each of the vertices (initially unchecked)



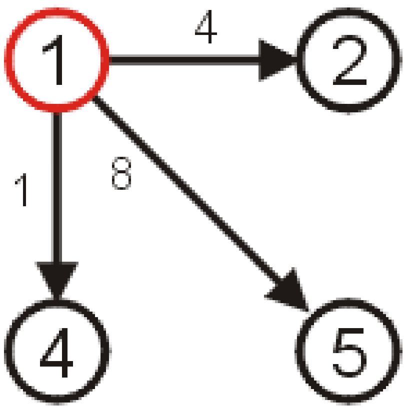
Dijkstra's Algorithm

- We will work bottom up.
  - Note that if the <u>starting vertex</u> has any <u>adjacent</u> edges, then there will be one vertex that is the <u>shortest</u> distance from the starting vertex. This is the shortest reachable vertex of the graph.
- We will then try to <u>extend any ***existing*** paths</u> to new vertices.
- <u>Initially</u>, we will start with the <u>path of length 0</u>
  - this is the trivial <u>path</u> from vertex <u>1</u> to itself

Dijkstra’s Algorithm

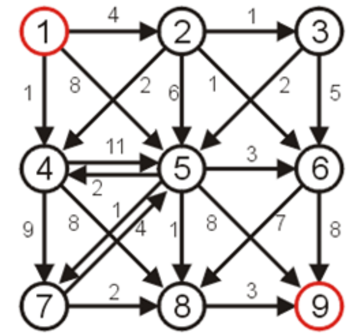


- If we now extend this path, we should consider the paths
  - (1, 2) length 4
  - (1, 4) length 1
  - (1, 5) length 8

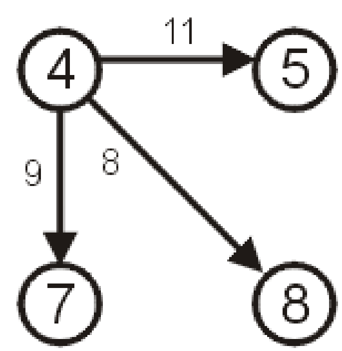


The *shortest* path so far is <u>(1, 4)</u> which is of <u>length 1</u>.

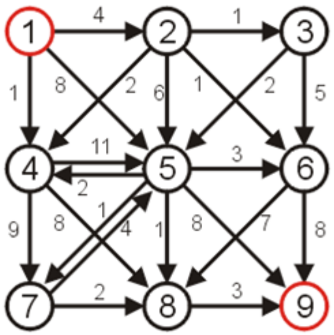
Dijkstra's Algorithm



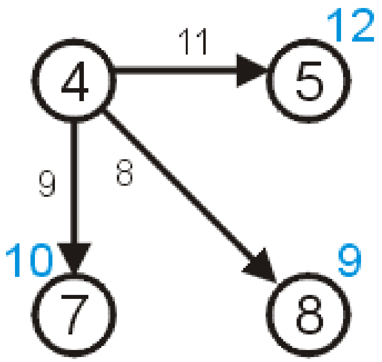
- Thus, if we now examine vertex 4, we may deduce that there exist the following paths:
  - (1, 4, 5) length 12
  - (1, 4, 7) length 10
  - (1, 4, 8) length 9



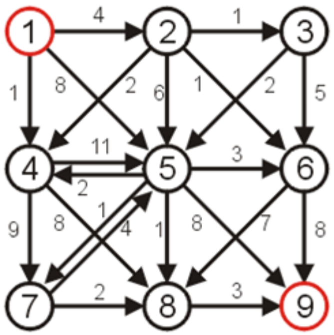
Dijkstra's Algorithm



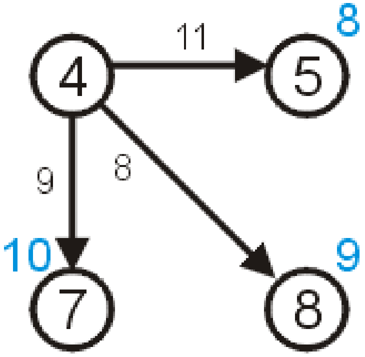
- We need to remember that the length of that path from node 1 to node 4 is 1
- Thus, we need to store the length of a path that goes through node 4:
  - 5 of length 12
  - 7 of length 10
  - 8 of length 9



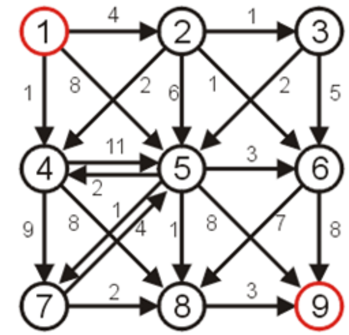
Dijkstra's Algorithm



- We have <u>already</u> discovered that there is a path of length 8 to vertex 5 with the path (1, 5).
- Thus, we can safely <u>ignore</u> this <u>longer</u> path.



Dijkstra's Algorithm



- We now know that:
  - There exist paths from vertex 1 to vertices {2,4,5,7,8}.
  - We know that the shortest path from vertex 1 to vertex 4 is of length 1.
  - We know that the shortest path to the other vertices {2,5,7,8} is at most the length listed in the table to the right.

Vertex	Length
1	0
2	4
4	1
5	8
7	10
8	9

Dijkstra’s Algorithm

- There cannot exist a shorter path to either of the vertices 1 or 4, since the distances can only increase at each iteration.
- We consider these vertices to be ***visited***

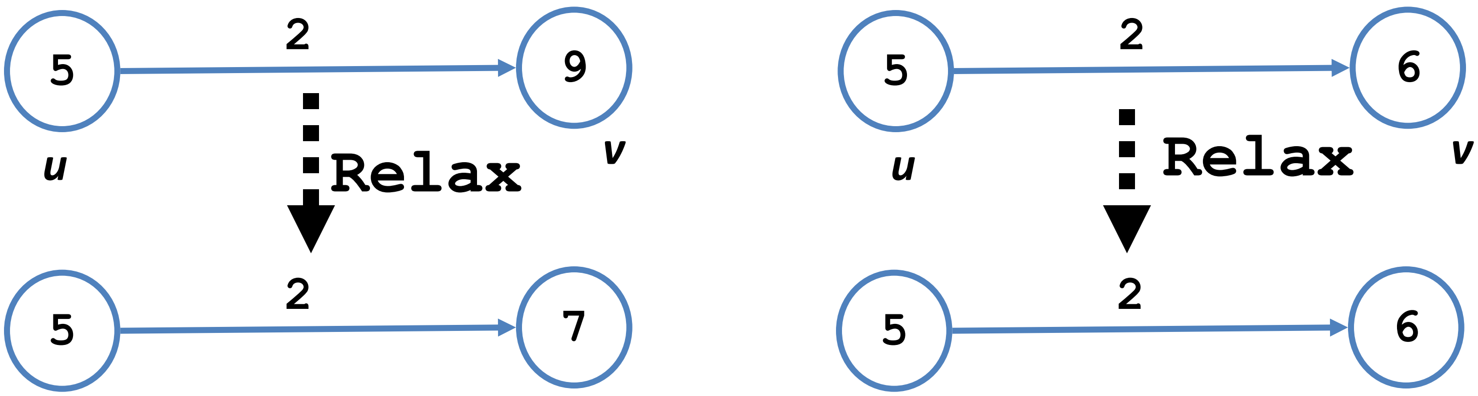
Vertex	Length
1	0
2	4
4	1
5	8
7	10
8	9

If you only knew this information and nothing else about the graph, what is the possible lengths from vertex 1 to vertex 2? What about to vertex 7?

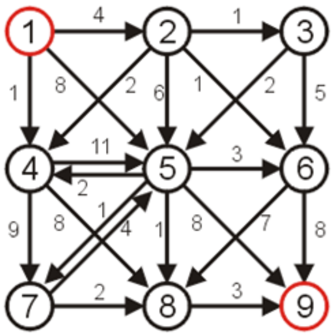
Relaxation

- Maintaining this shortest discovered distance d[*v*] is called **relaxation**:

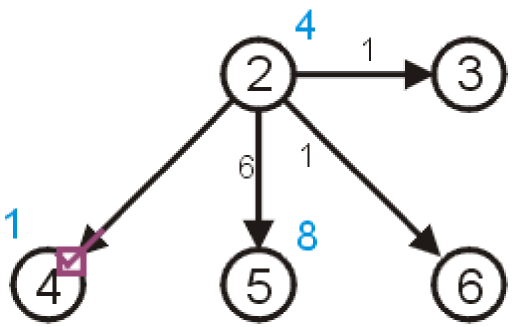
```
Relax(u,v,w) {
    if (d[v] > d[u]+w) then
        d[v]=d[u]+w;
}
```



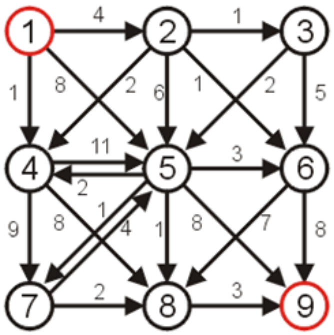
Dijkstra's Algorithm



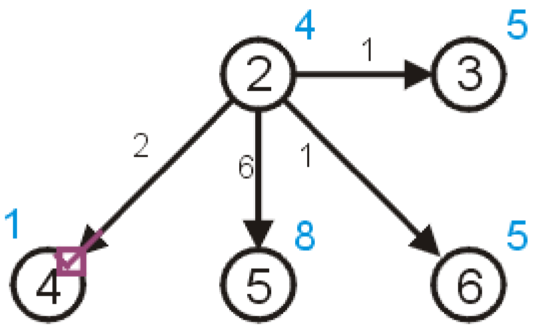
- In Dijkstra's algorithm, we always take the next unvisited vertex which has the current shortest path from the starting vertex in the table.
- This is vertex 2



Dijkstra’s Algorithm



- We can try to <u>update the shortest paths</u> to vertices 3 and 6 (both of length 5) however:
  – there already exists a path of length 8 < 10 to vertex 5 (10 = 4 + 6)
  – we already know the shortest path to 4 is 1



Dijkstra’s Algorithm

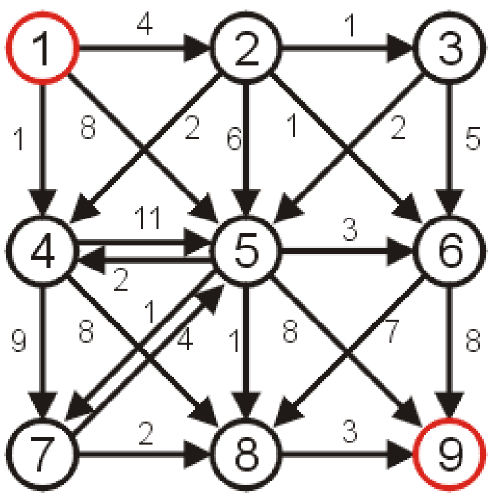
- To keep track of those vertices to which no path has reached, we can assign those vertices an initial distance of either
  - infinity (∞),
  - a number larger than any possible path, or
  - a negative number
- For demonstration purposes, we will use ∞

Dijkstra's Algorithm

- As well as finding the length of the shortest path, we'd like to find the corresponding shortest path
- Each time we <u>update</u> the <u>shortest</u> distance to a particular vertex, we will <u>keep track</u> of the <u>predecessor</u> used to reach this vertex on the shortest path.

Dijkstra's Algorithm

- We will store a table of pointers, each initially 0
- This table will be updated each time a distance is updated



1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Dijkstra's Algorithm

- Graphically, we will display the reference to the <u>preceding</u> vertex by a <u>red arrow</u>
  - if the distance to a vertex is ∞, there will be no preceding vertex
  - otherwise, there will be <u>exactly one preceding</u> vertex

Dijkstra's Algorithm

- Thus, for our initialization:
  - we set the current distance to the initial vertex as 0
  - for all other vertices, we set the current distance to ∞
  - all vertices are initially marked as unvisited
  - set the previous pointer for all vertices to null

Dijkstra's Algorithm

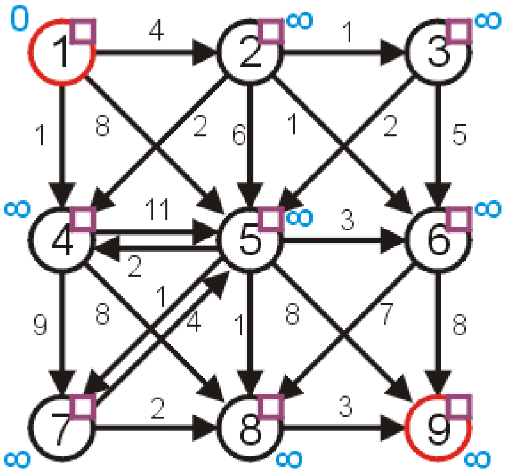
- Thus, we iterate:
  - find an unvisited vertex which has the shortest distance to it
  - mark it as visited
  - for each unvisited vertex which is adjacent to the current vertex:
    - add the distance to the current vertex to the weight of the connecting edge
    - if this is less than the current distance to that vertex, update the distance and set the parent vertex of the adjacent vertex to be the current vertex

Dijkstra’s Algorithm

- Halting condition:
  - we successfully halt when the vertex we are visiting is the target vertex
  - if at some point, all remaining unvisited vertices have distance ∞, then no path from the starting vertex to the end vertex exits
- Note: We do not halt just because we have updated the distance to the end vertex, we have to **visit** the target vertex.

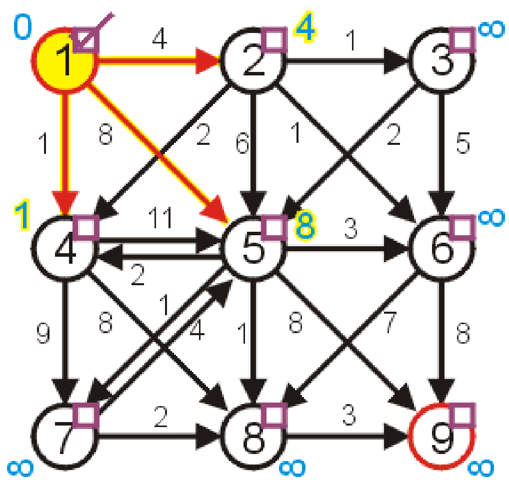
Example

- Consider the graph:
  - the distances are appropriately initialized
  - all vertices are marked as being unvisited



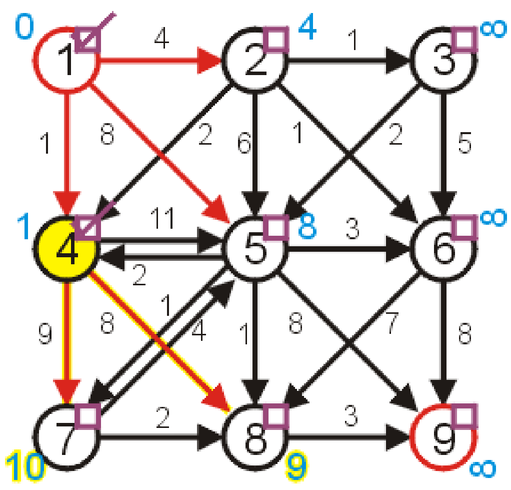
Example

- Visit vertex 1 and update its neighbours, marking it as visited
  - the shortest paths to 2, 4, and 5 are updated



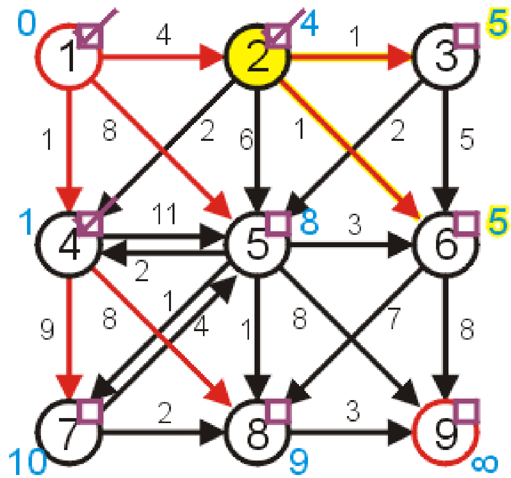
Example

- The next vertex we visit is vertex 4
  - vertex 5 $1 + 11 \geq 8$ don't update
  - vertex 7 $1 + 9 < \infty$ update
  - vertex 8 $1 + 8 < \infty$ update



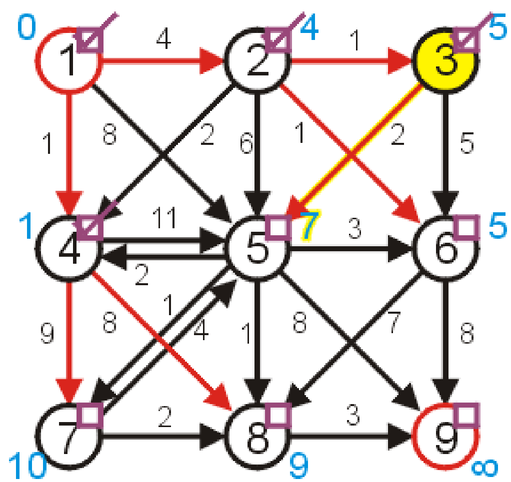
Example

- Next, visit vertex 2
  - vertex 3 $4 + 1 < \infty$ update
  - vertex 4 already visited
  - vertex 5 $4 + 6 \geq 8$ don't update
  - vertex 6 $4 + 1 < \infty$ update



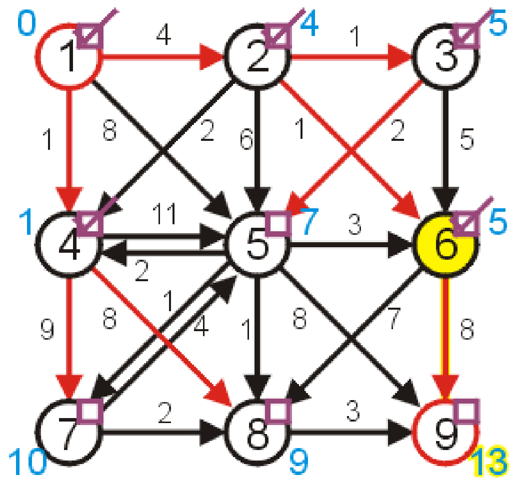
Example

- Next, we have a choice of either 3 or 6
- We will choose to visit 3
  - vertex 5 $5 + 2 < 8$ update
  - vertex 6 $5 + 5 \geq 5$ don't update



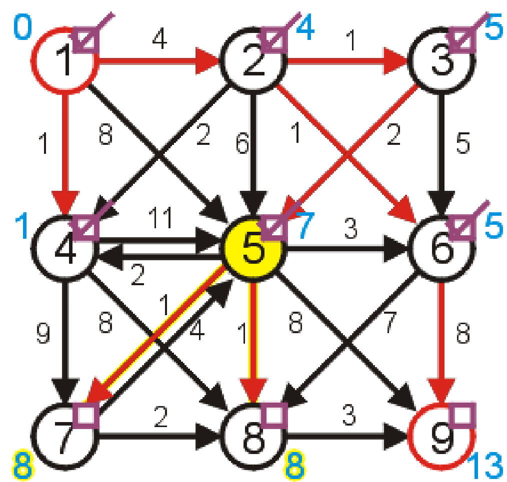
Example

- We then visit 6
  - vertex 8 $5 + 7 \geq 9$ don't update
  - vertex 9 $5 + 8 < \infty$ update



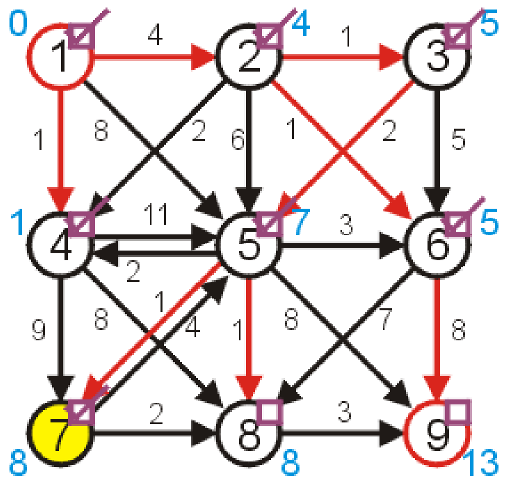
Example

- Next, we finally visit vertex 5:
  - vertices 4 and 6 have already been visited
  - vertex 7 $\quad 7 + 1 < 10 \quad$ update
  - vertex 8 $\quad 7 + 1 < 9 \quad$ update
  - vertex 9 $\quad 7 + 8 \geq 13 \quad$ don’t update



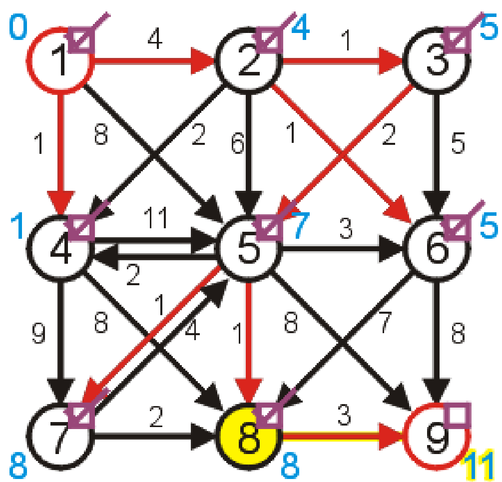
Example

- Given a choice between vertices 7 and 8, we choose vertex 7
  - vertices 5 has already been visited
  - vertex 8 $\quad 8 + 2 \geq 8 \quad$ don’t update



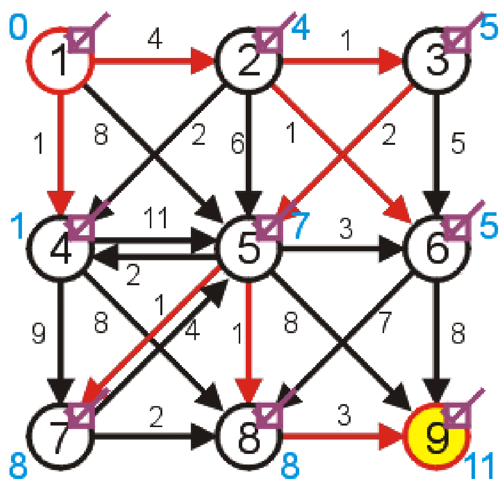
Example

- Next, we visit vertex 8:
  - vertex 9 8 + 3 < 13 update



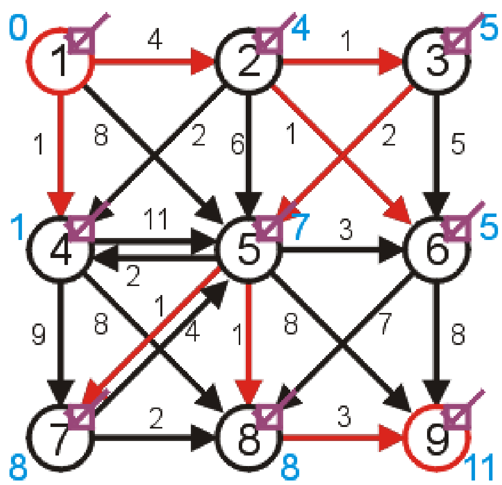
Example

- Finally, we visit the end vertex
- Therefore, the shortest path from 1 to 9 has length 11



Example

- We can find the shortest path by working back from the final vertex:
  - 9, 8, 5, 3, 2, 1
- Thus, the shortest path is (1, 2, 3, 5, 8, 9)

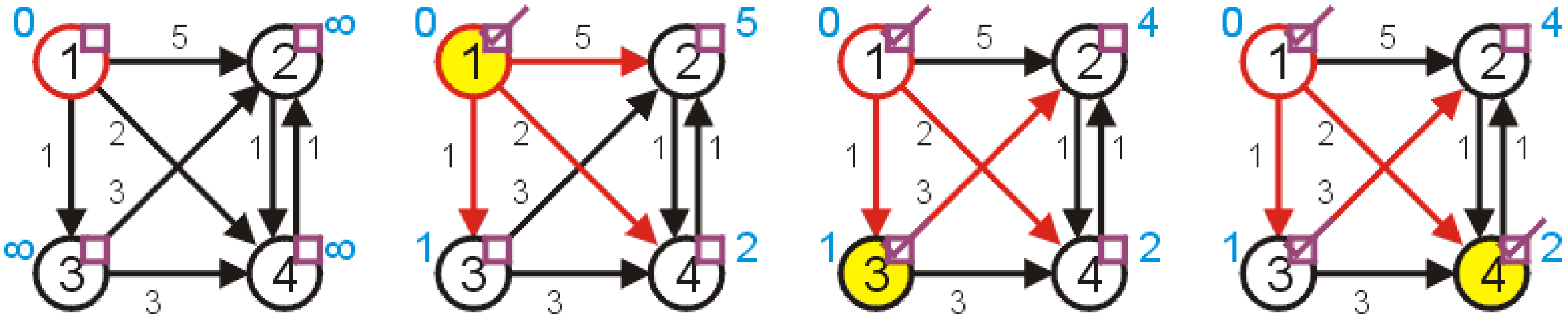


Example

- In the example, we visited all vertices in the graph before we finished
- This is not always the case, consider the next example

Example

- Find the shortest path from 1 to 4:
  - the shortest path is found after only three vertices are visited
  - we terminated the algorithm as soon as we reached vertex 4
  - we only have useful information about 1, 3, 4
  - we don’t have the shortest path to vertex 2



Dijkstra's algorithm

$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
    do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ ⊳ Q is a priority queue maintaining $V - S$
while $Q \neq \varnothing$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        $S \leftarrow S \cup \{u\}$
        for each $v \in Adj[u]$
            do if $d[v] > d[u] + w(u, v)$
                then $d[v] \leftarrow d[u] + w(u, v)$
                $p[v] \leftarrow u$

Dijkstra's algorithm

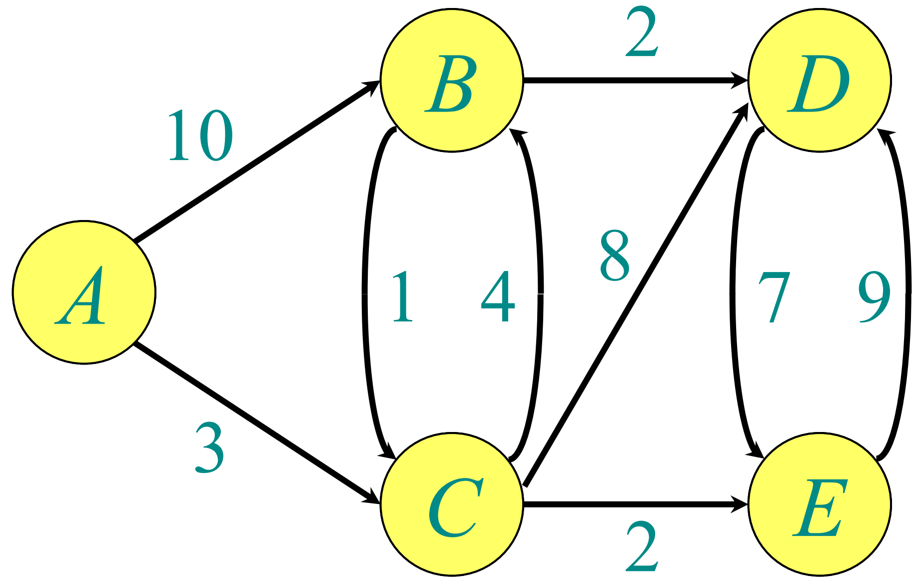
$d[s] \leftarrow 0$
for each $v \in V - \{s\}$
  do $d[v] \leftarrow \infty$
$S \leftarrow \varnothing$
$Q \leftarrow V$ $\triangleright$ Q is a priority queue maintaining $V - S$
while $Q \neq \varnothing$
  do $u \leftarrow$ EXTRACT-MIN(Q)
    $S \leftarrow S \cup \{u\}$
    for each $v \in Adj[u]$
      do if $d[v] > d[u] + w(u, v)$
        then $d[v] \leftarrow d[u] + w(u, v)$
          $p[v] \leftarrow u$

relaxation step

Implicit DECREASE-KEY

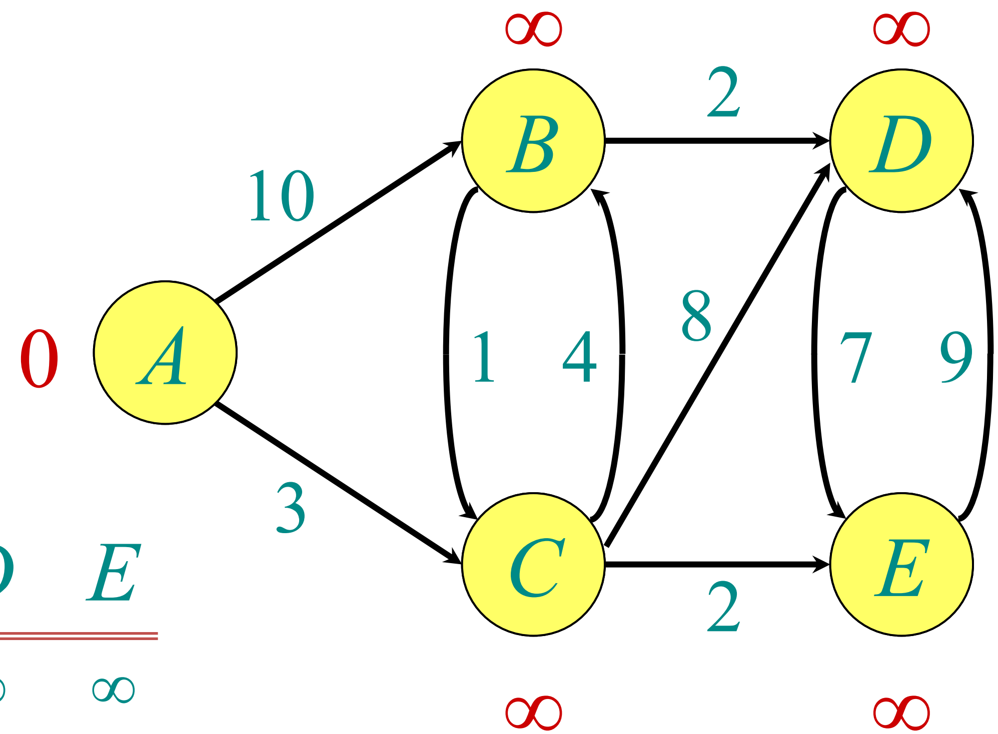
Example of Dijkstra's algorithm

Graph with nonnegative edge weights:



Example of Dijkstra's algorithm

Initialize:

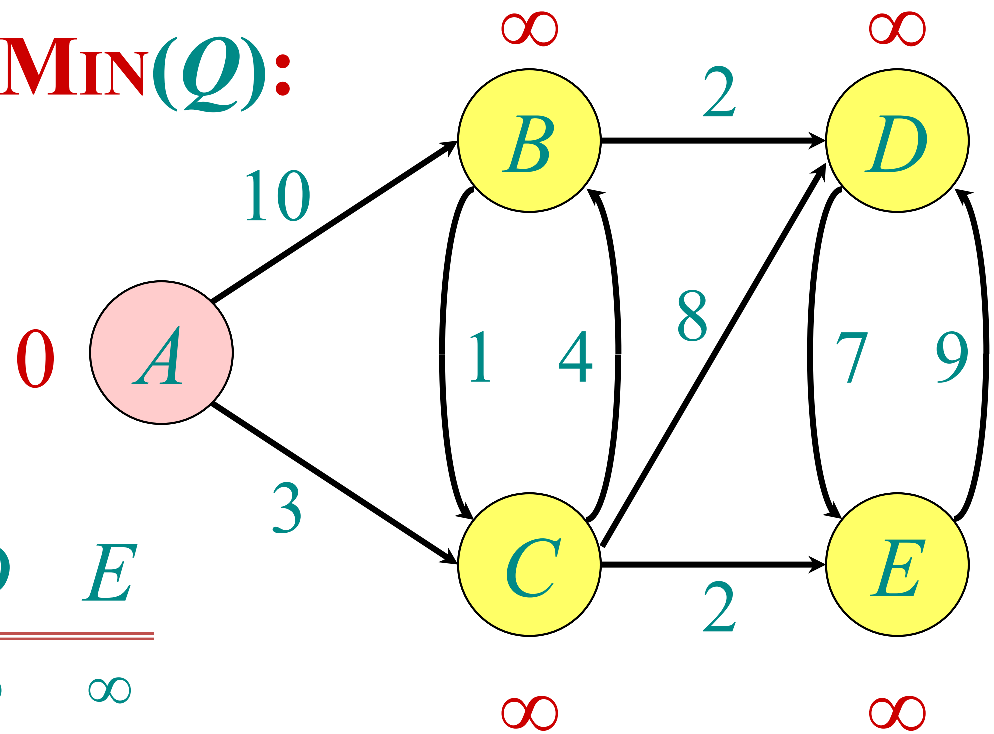


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: {}

Example of Dijkstra's algorithm

"A" ← **EXTRACT-MIN(Q):**

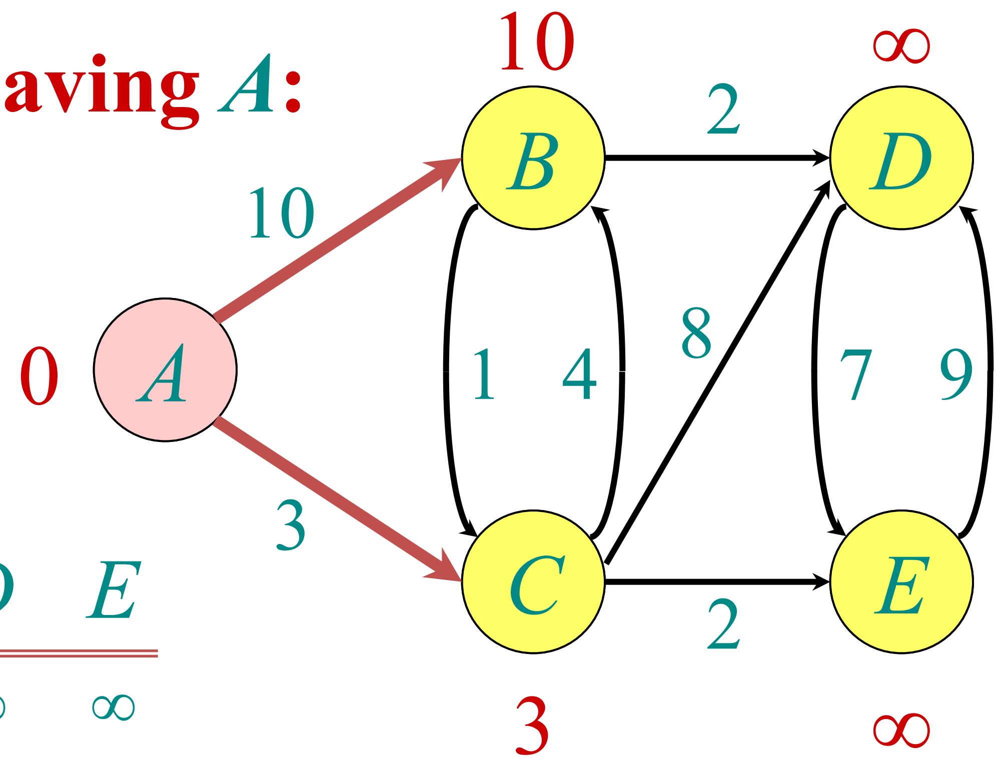


Q:	A	B	C	D	E
	0	∞	∞	∞	∞

S: { A }

Example of Dijkstra's algorithm

Relax all edges leaving *A*:

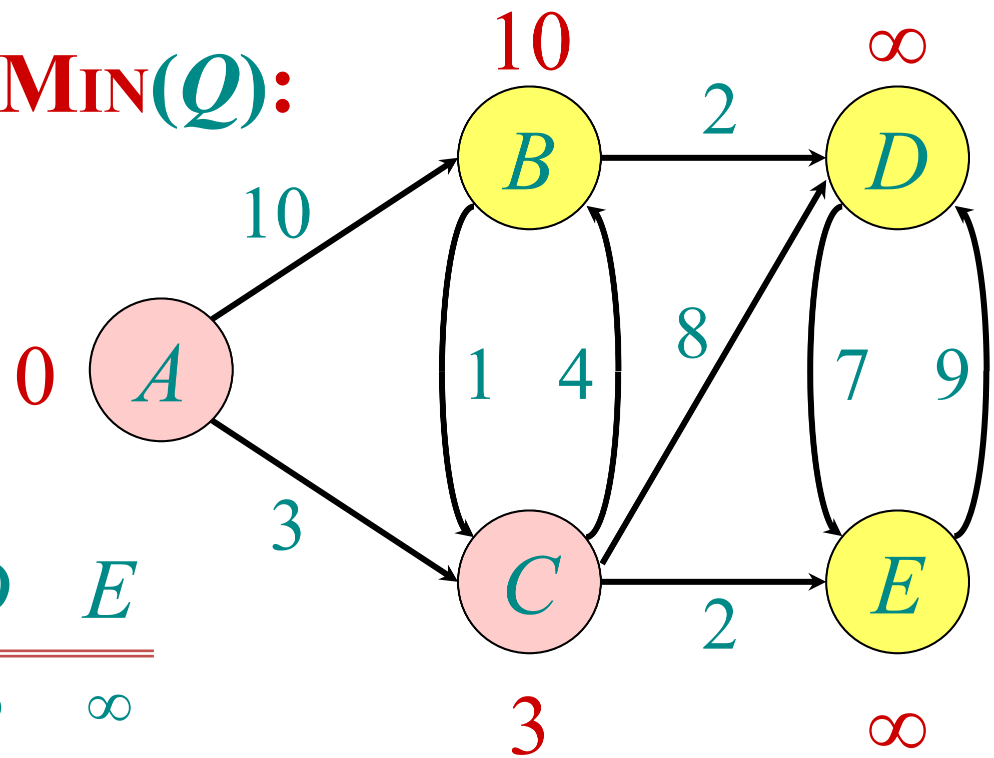


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A }

Example of Dijkstra's algorithm

"C" ← **EXTRACT-MIN(Q):**

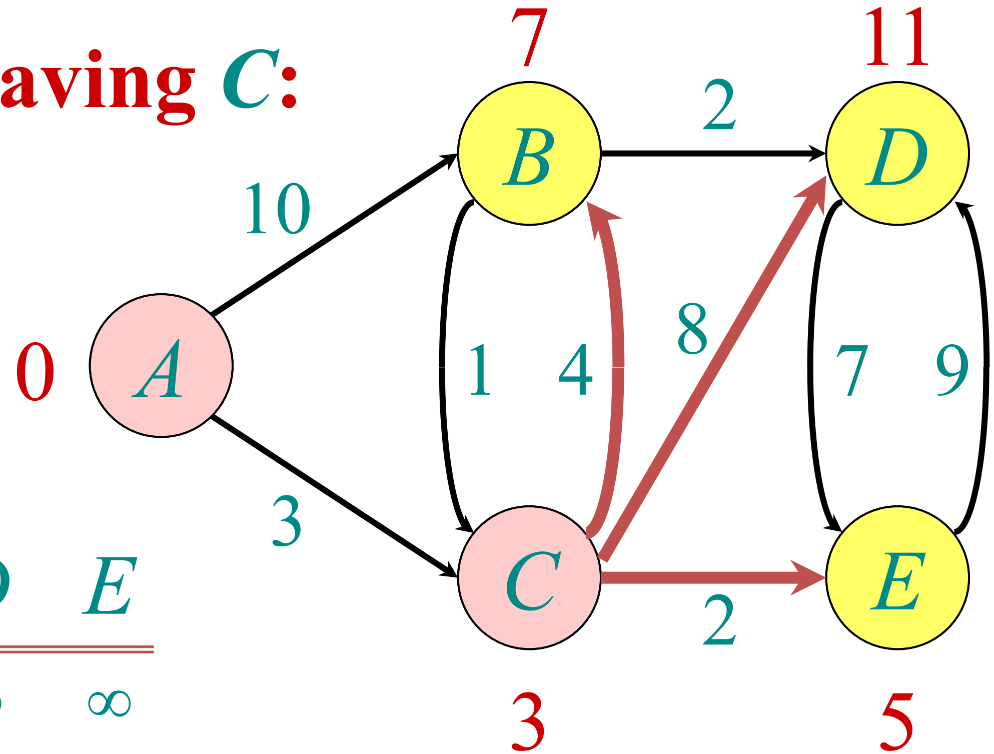


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞

S: { A, C }

Example of Dijkstra's algorithm

Relax all edges leaving *C*:

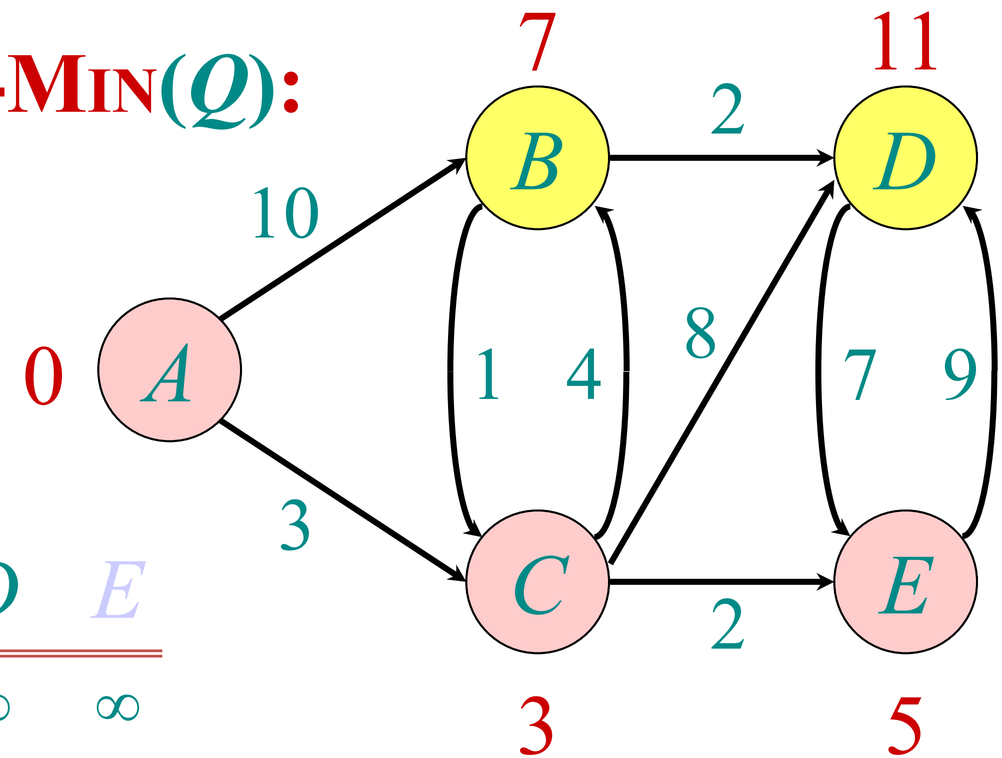


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5

S: { A, C }

Example of Dijkstra's algorithm

"*E*" ← EXTRACT-MIN(*Q*):

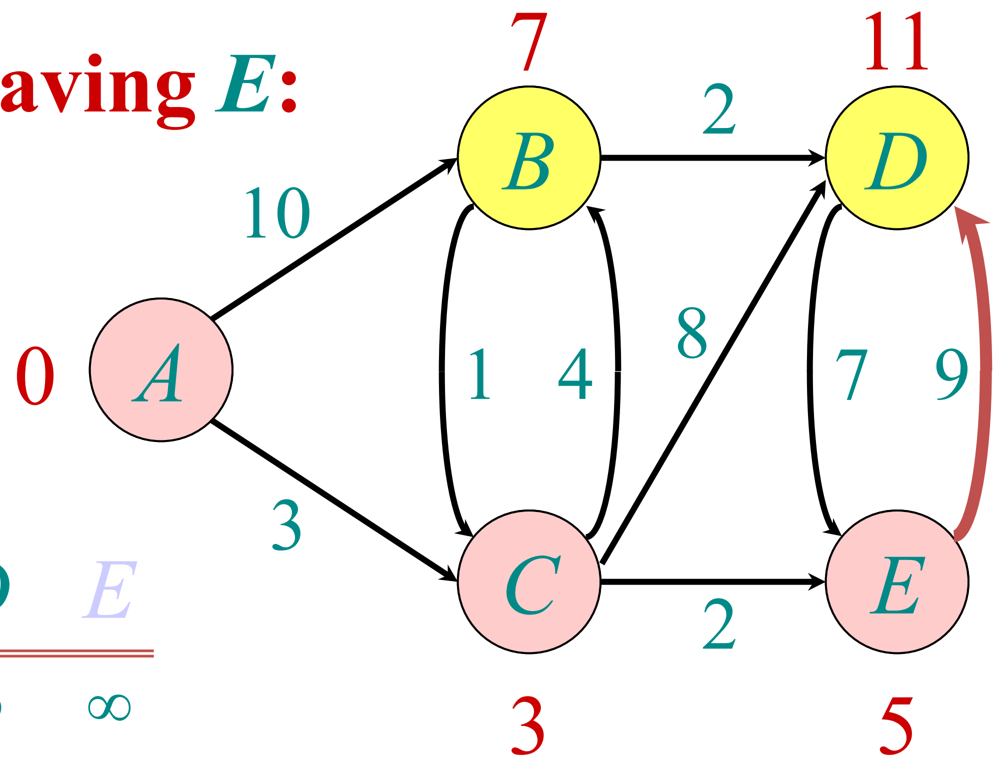


Q:	A	B	C	D	E
	$\boxed{0}$	∞	∞	∞	∞
		10	$\boxed{3}$	∞	∞
		7		11	$\boxed{5}$

S: { A, C, E }

Example of Dijkstra's algorithm

Relax all edges leaving *E*:

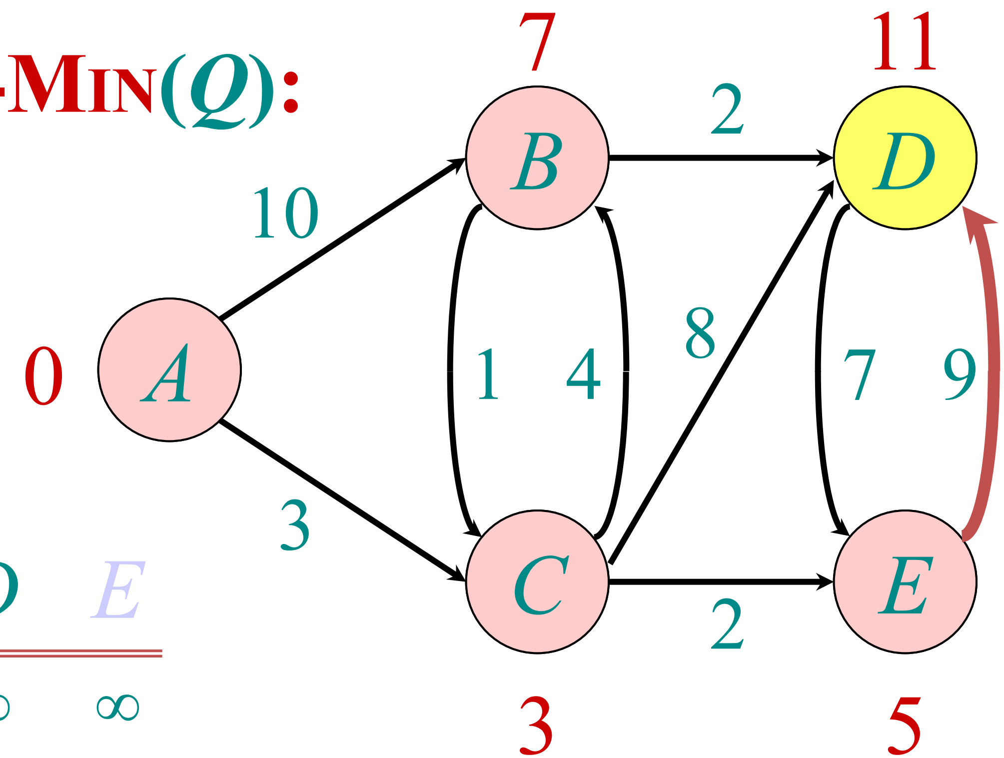


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	

S: { A, C, E }

Example of Dijkstra's algorithm

"B" ← **EXTRACT-MIN(Q):**



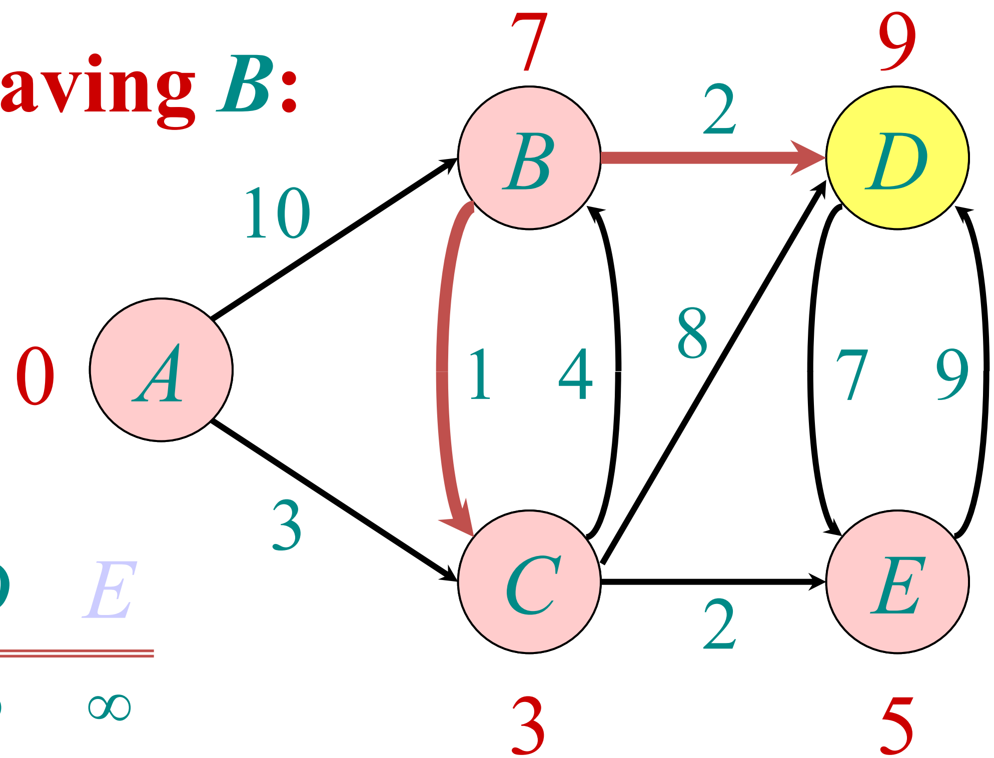
Q:

A	B	C	D	E
0	∞	∞	∞	∞
	10	3	∞	∞
	7		11	5
	7		11	

S: { A, C, E, B }

Example of Dijkstra's algorithm

Relax all edges leaving *B*:

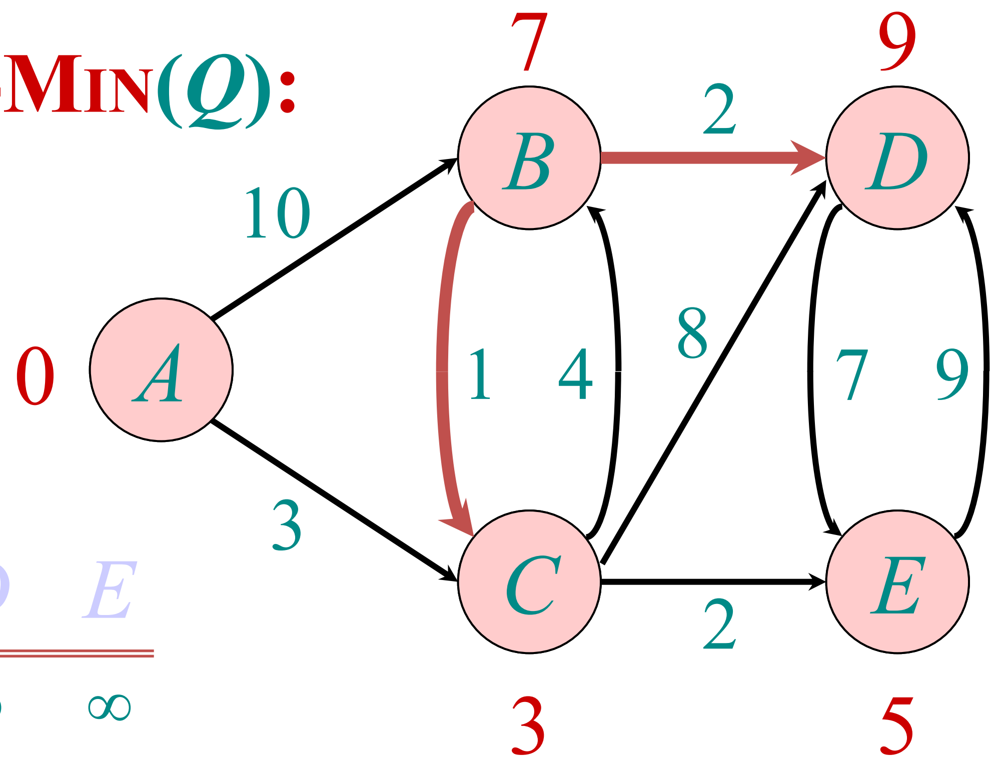


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { A, C, E, B }

Example of Dijkstra's algorithm

"*D*" ← EXTRACT-MIN(*Q*):

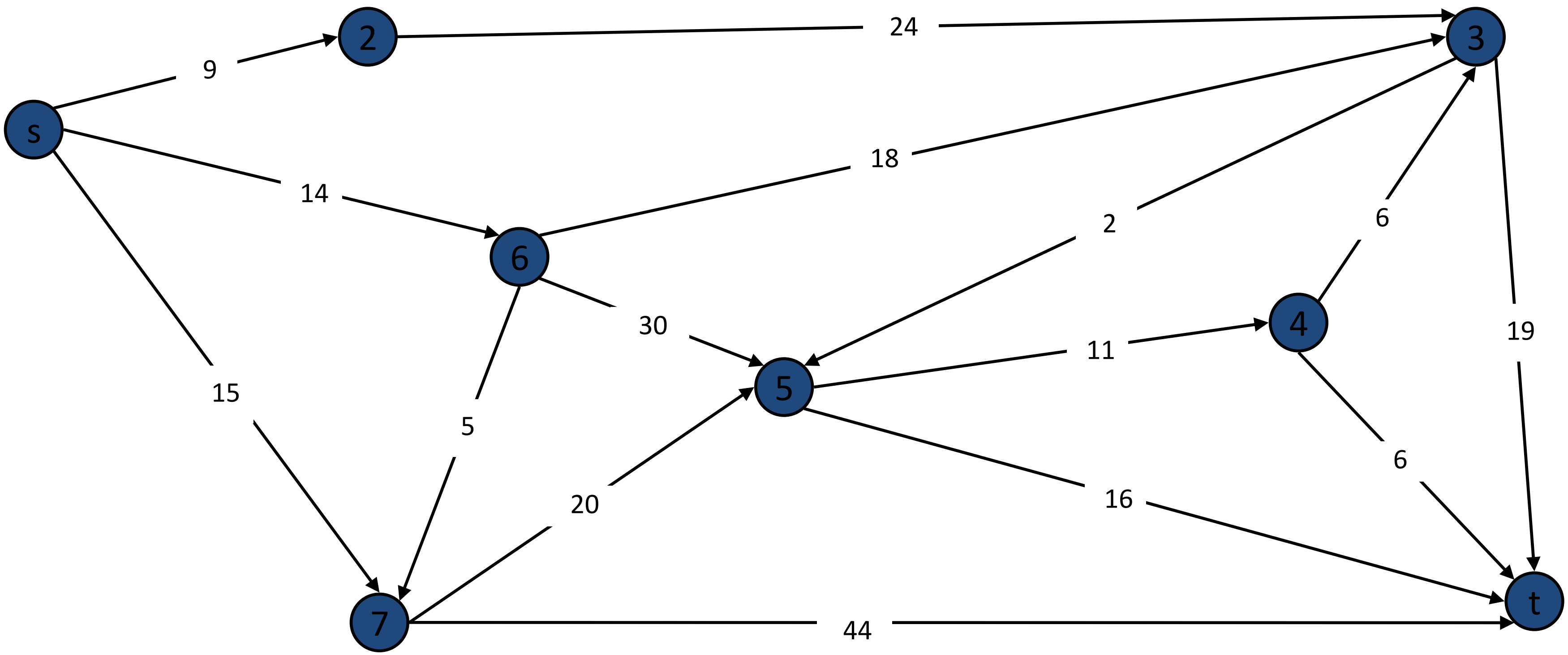


Q:	A	B	C	D	E
	0	∞	∞	∞	∞
		10	3	∞	∞
		7		11	5
		7		11	
				9	

S: { A, C, E, B, D }

Dijkstra's Shortest Path Algorithm

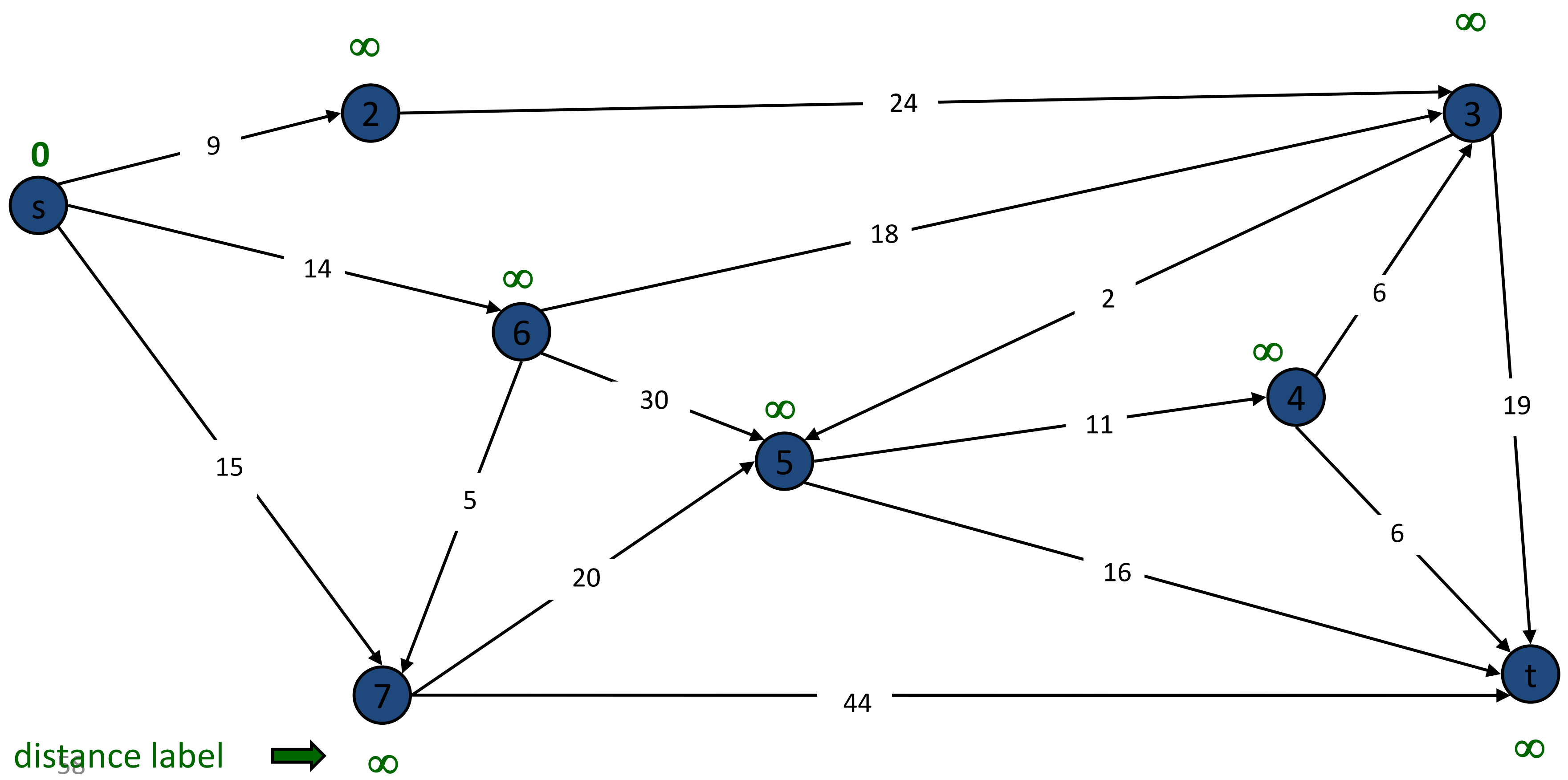
- Find shortest path from s to t.



Dijkstra's Shortest Path Algorithm

S = { }

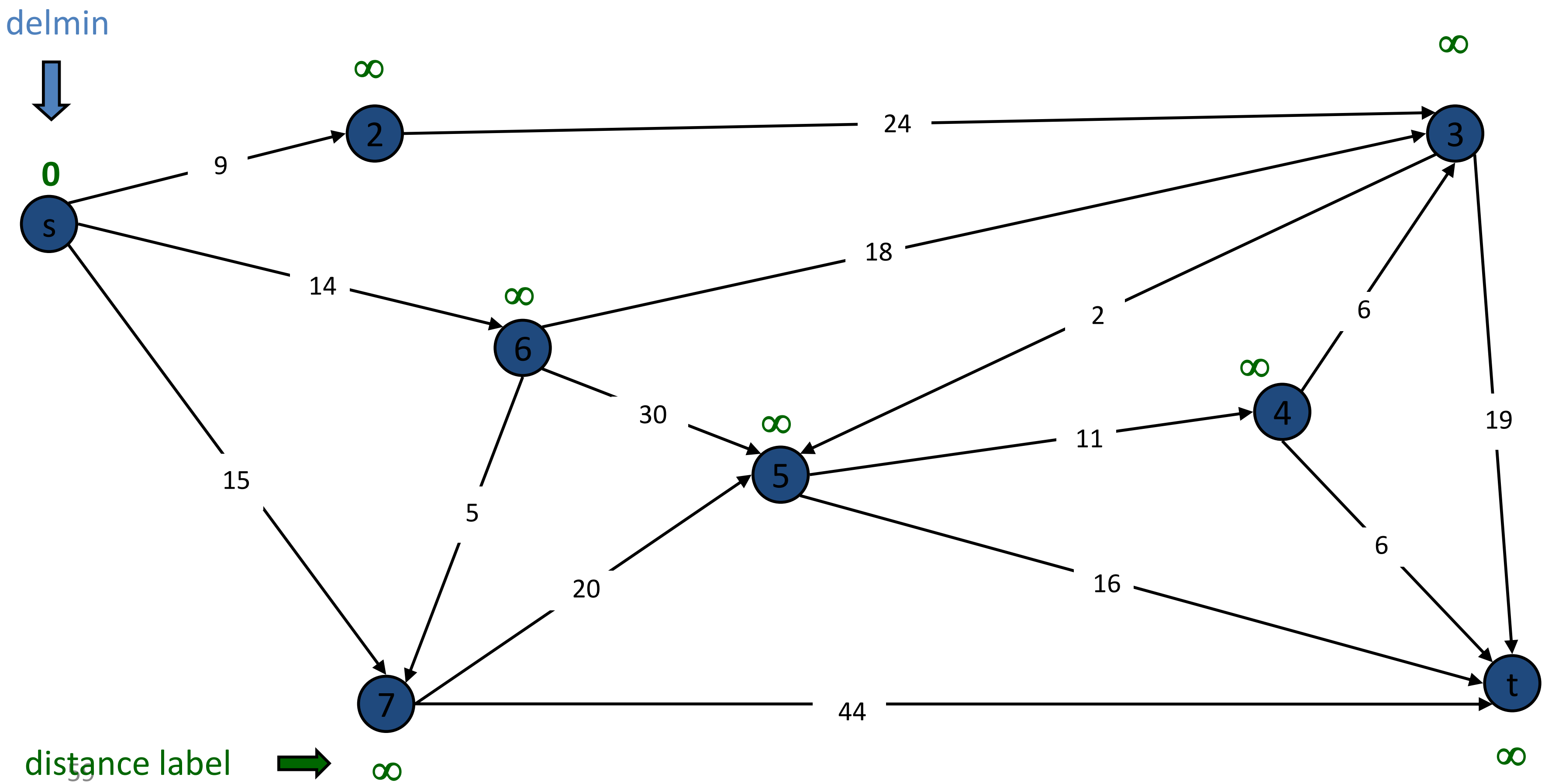
PQ = { s, 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { }

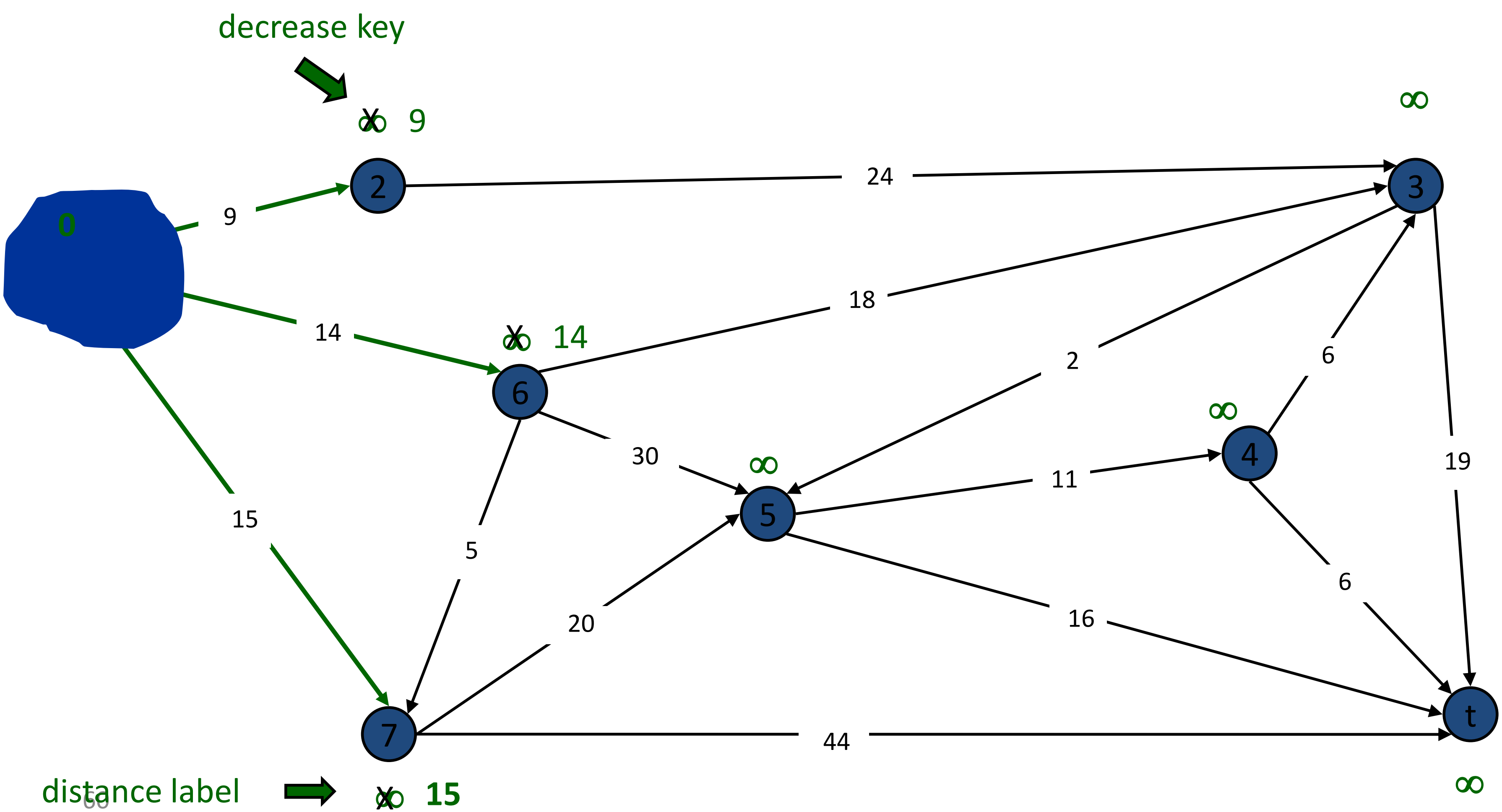
PQ = { s, 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s }

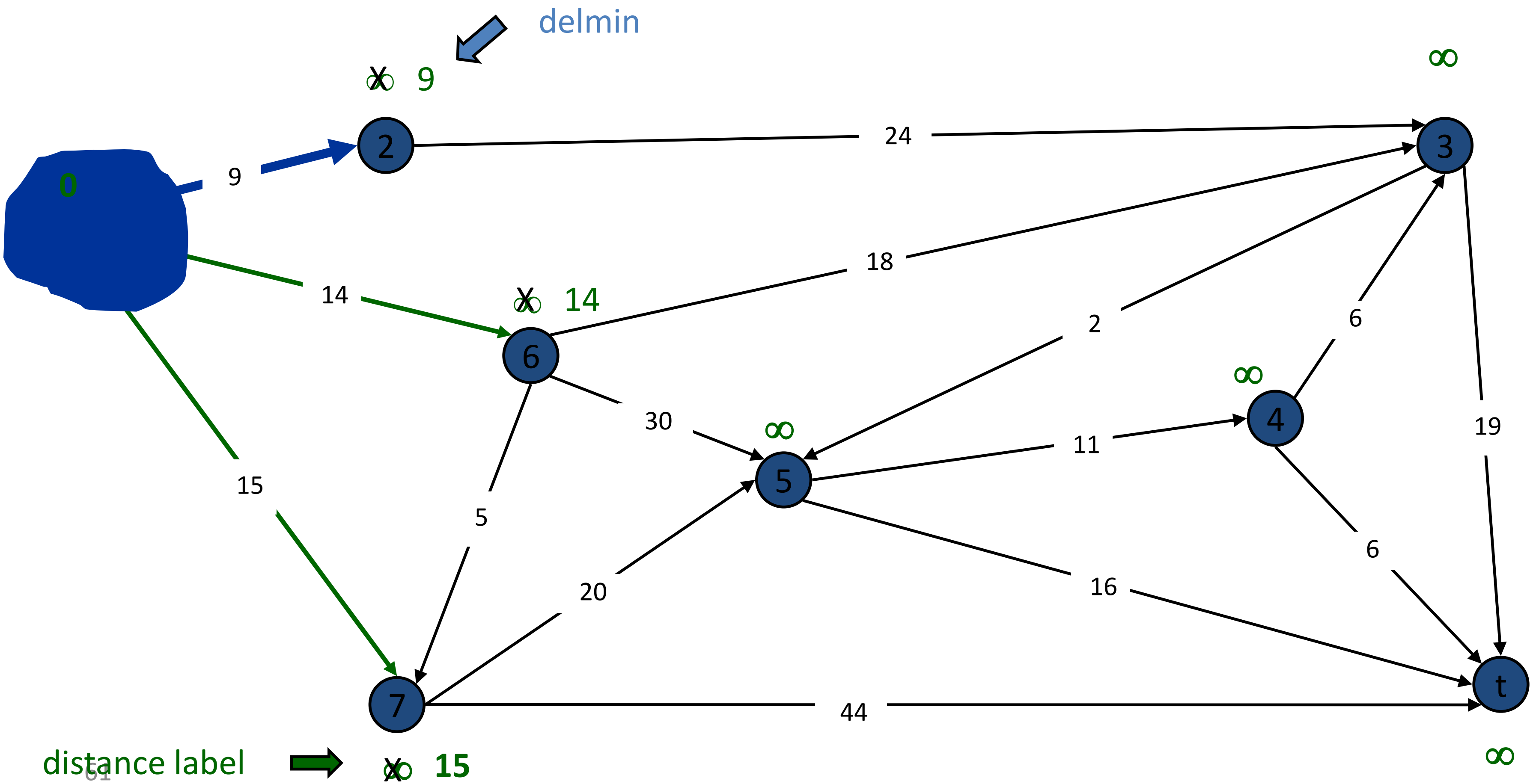
PQ = { 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s }

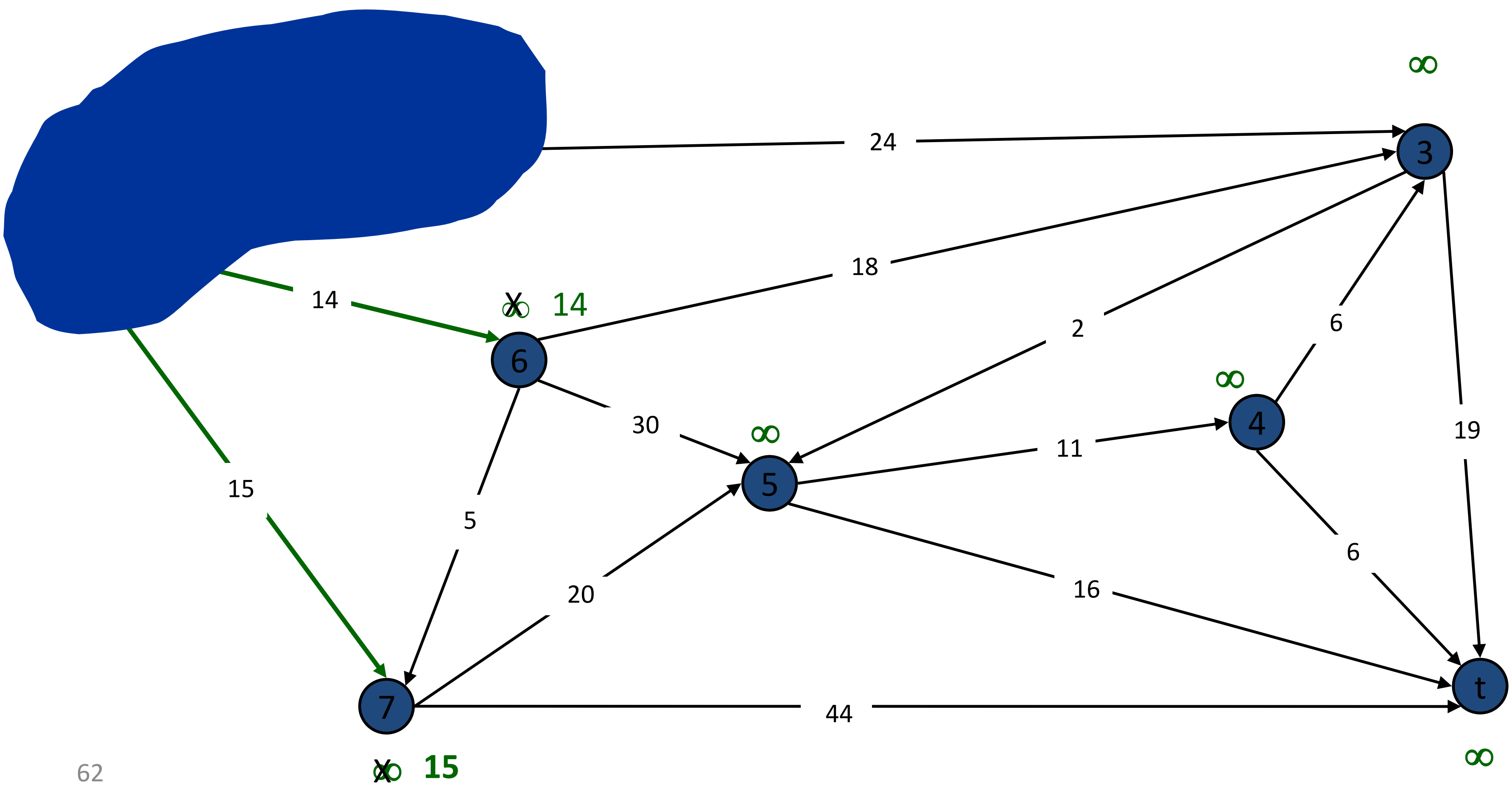
PQ = { 2, 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2 }

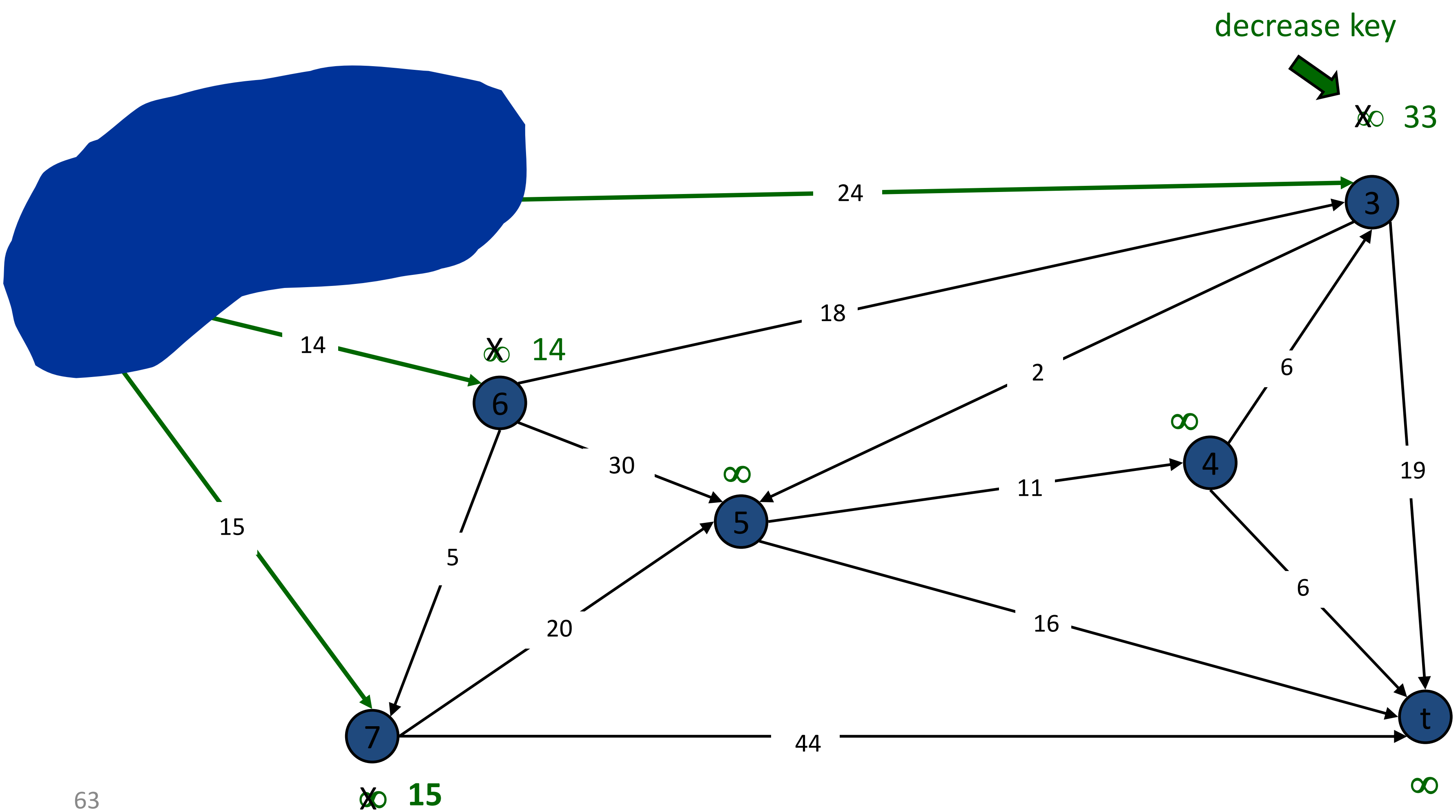
PQ = { 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2 }

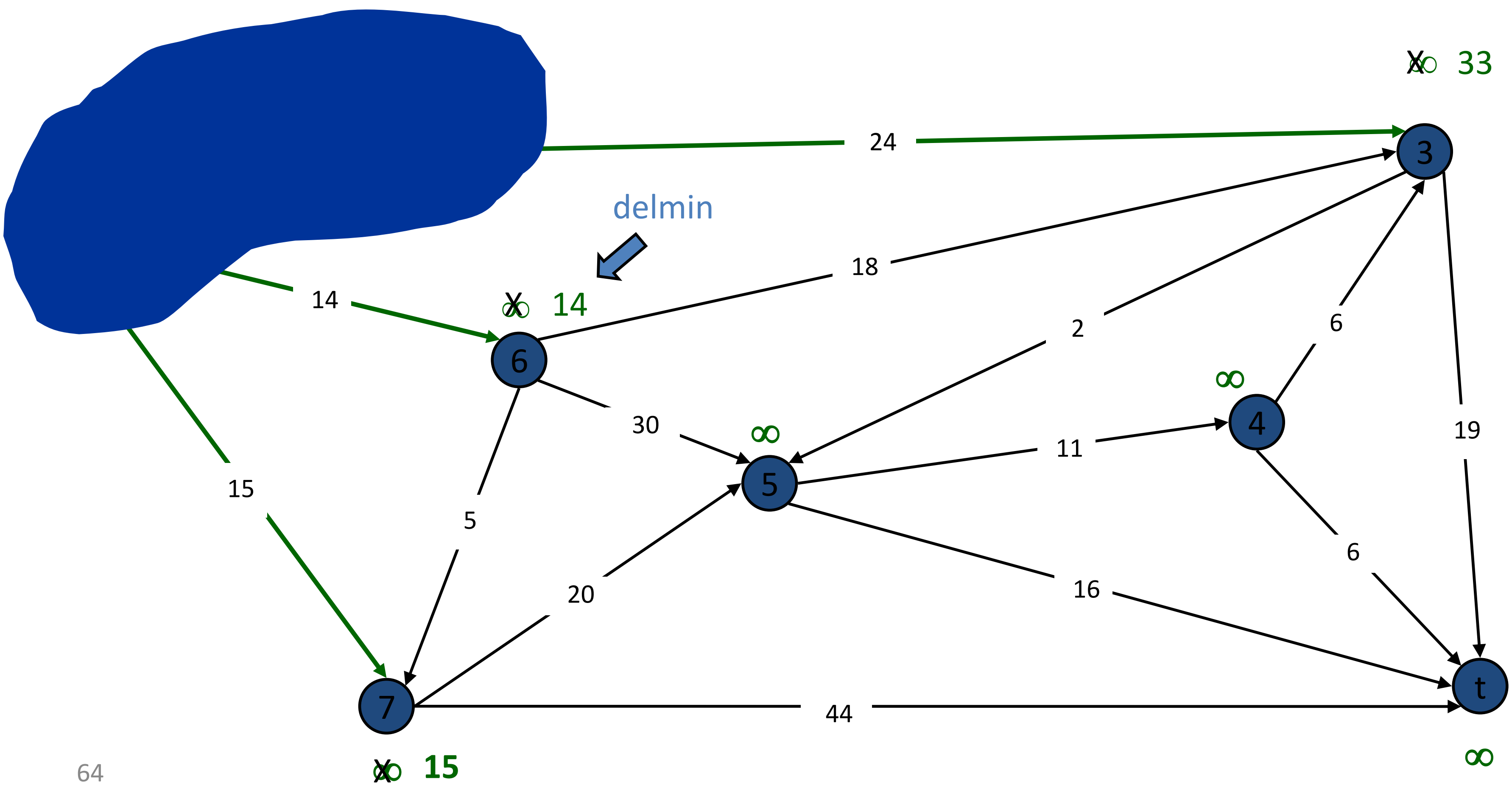
PQ = { 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2 }

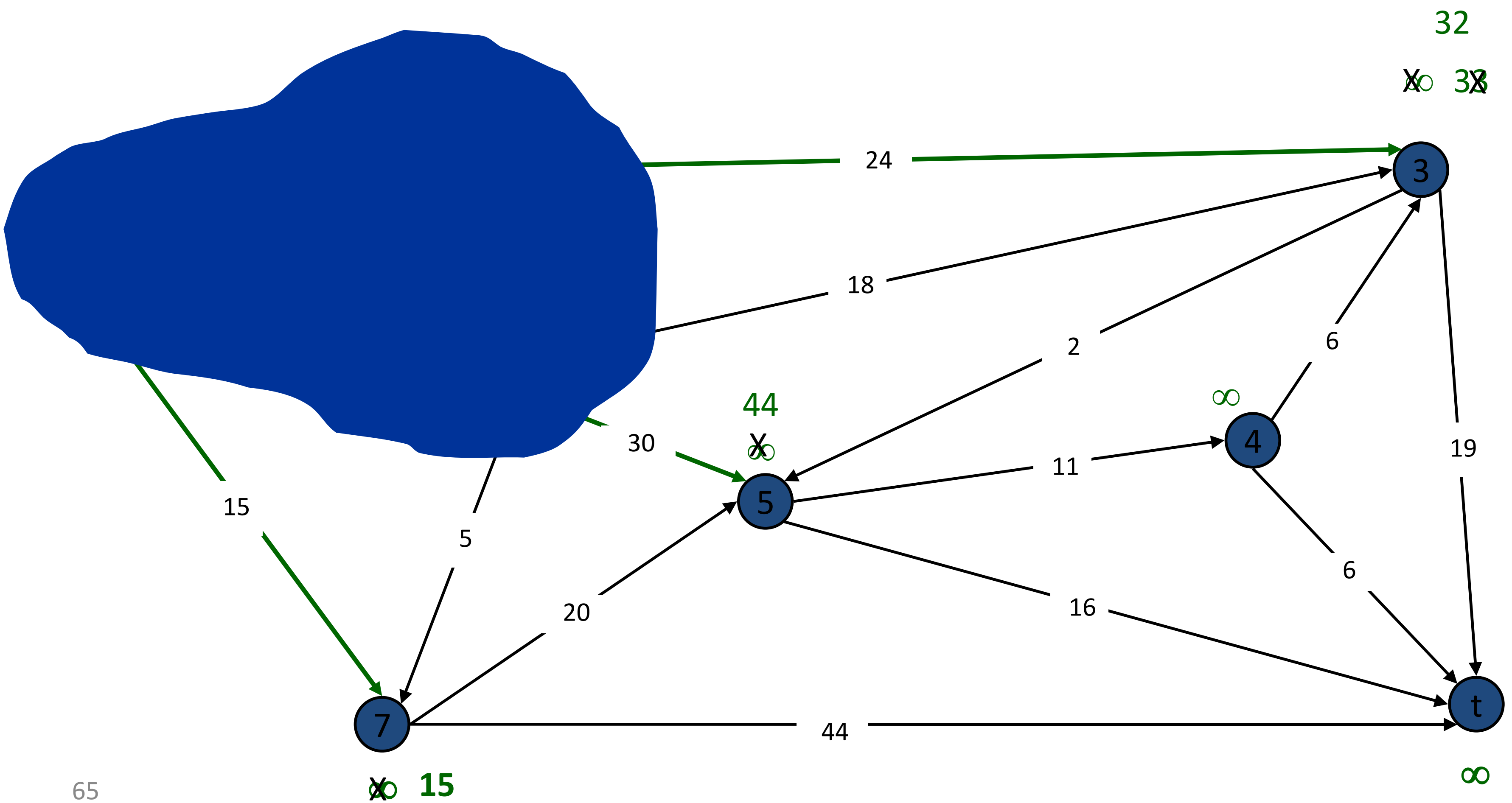
PQ = { 3, 4, 5, 6, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2, 6 }

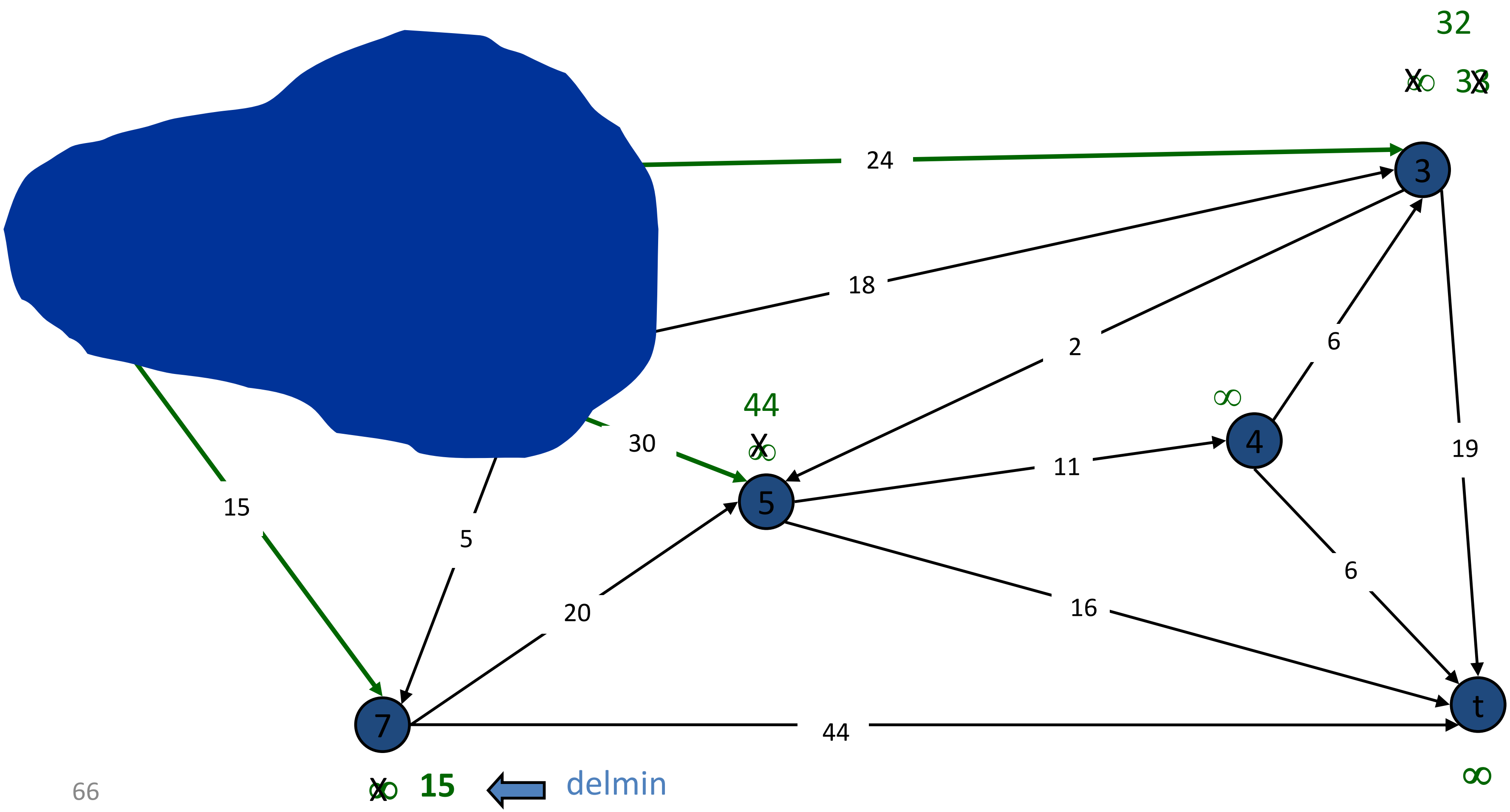
PQ = { 3, 4, 5, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2, 6 }

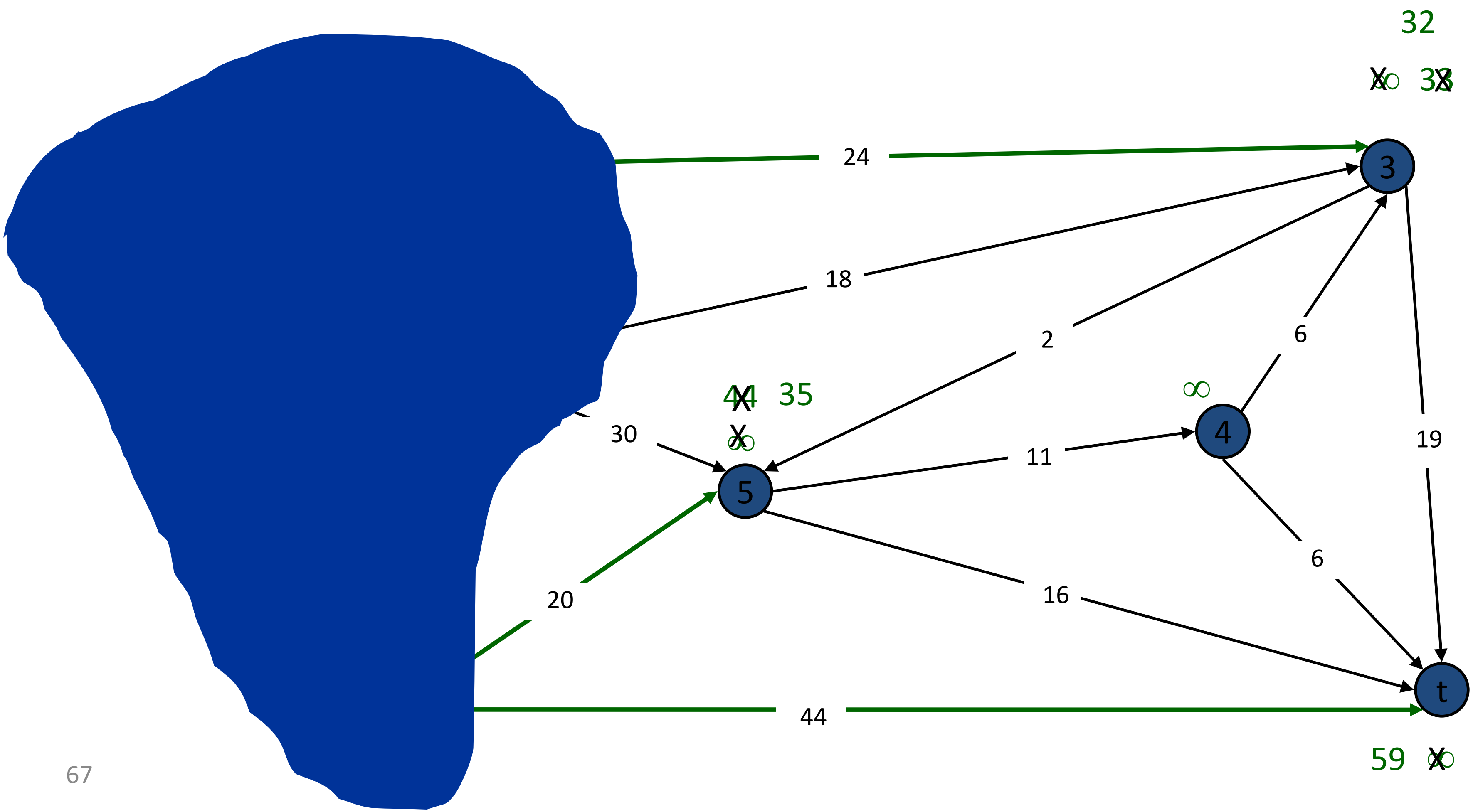
PQ = { 3, 4, 5, 7, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7 }

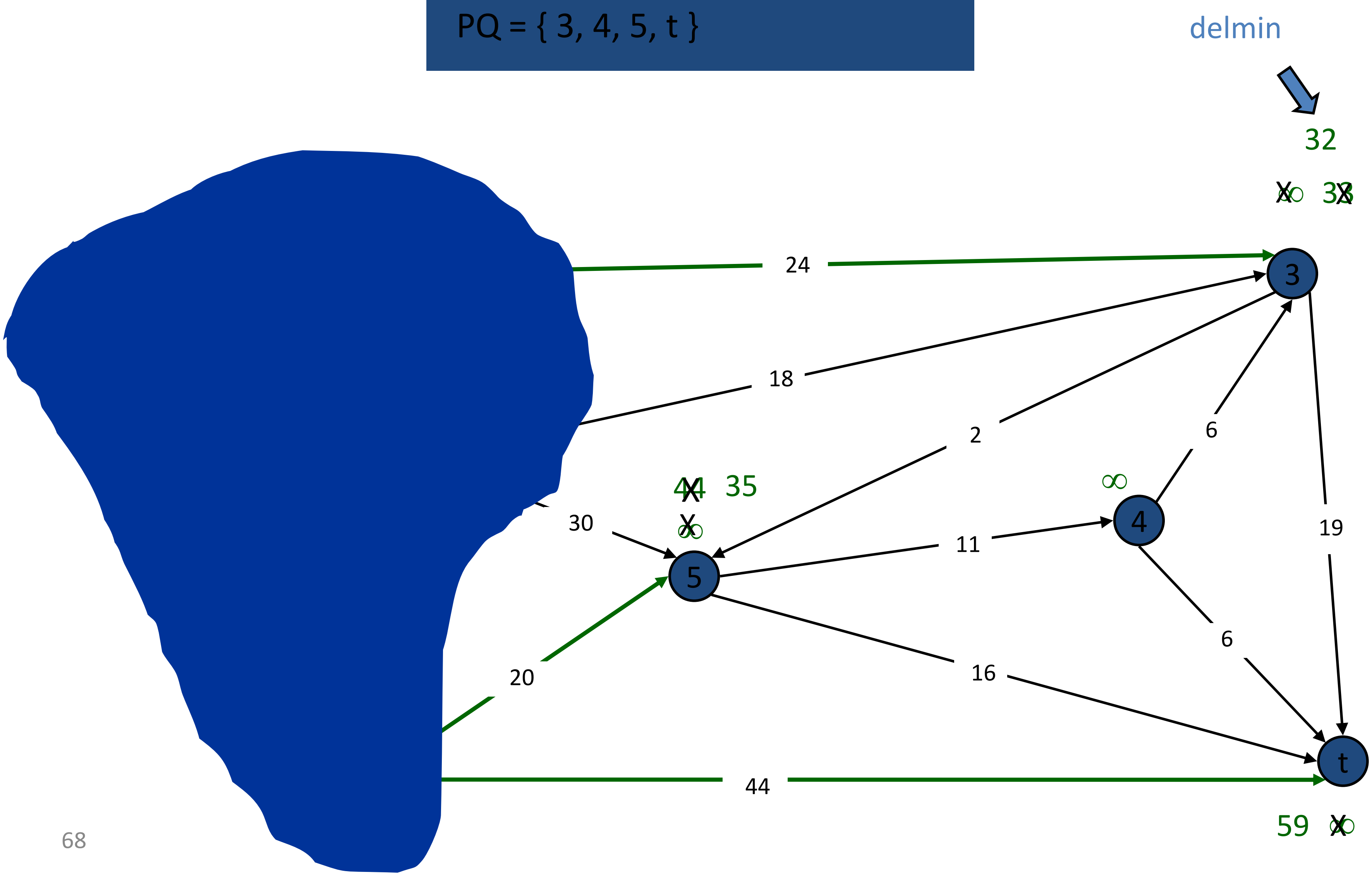
PQ = { 3, 4, 5, t }



Dijkstra's Shortest Path Algorithm

S = { s, 2, 6, 7 }

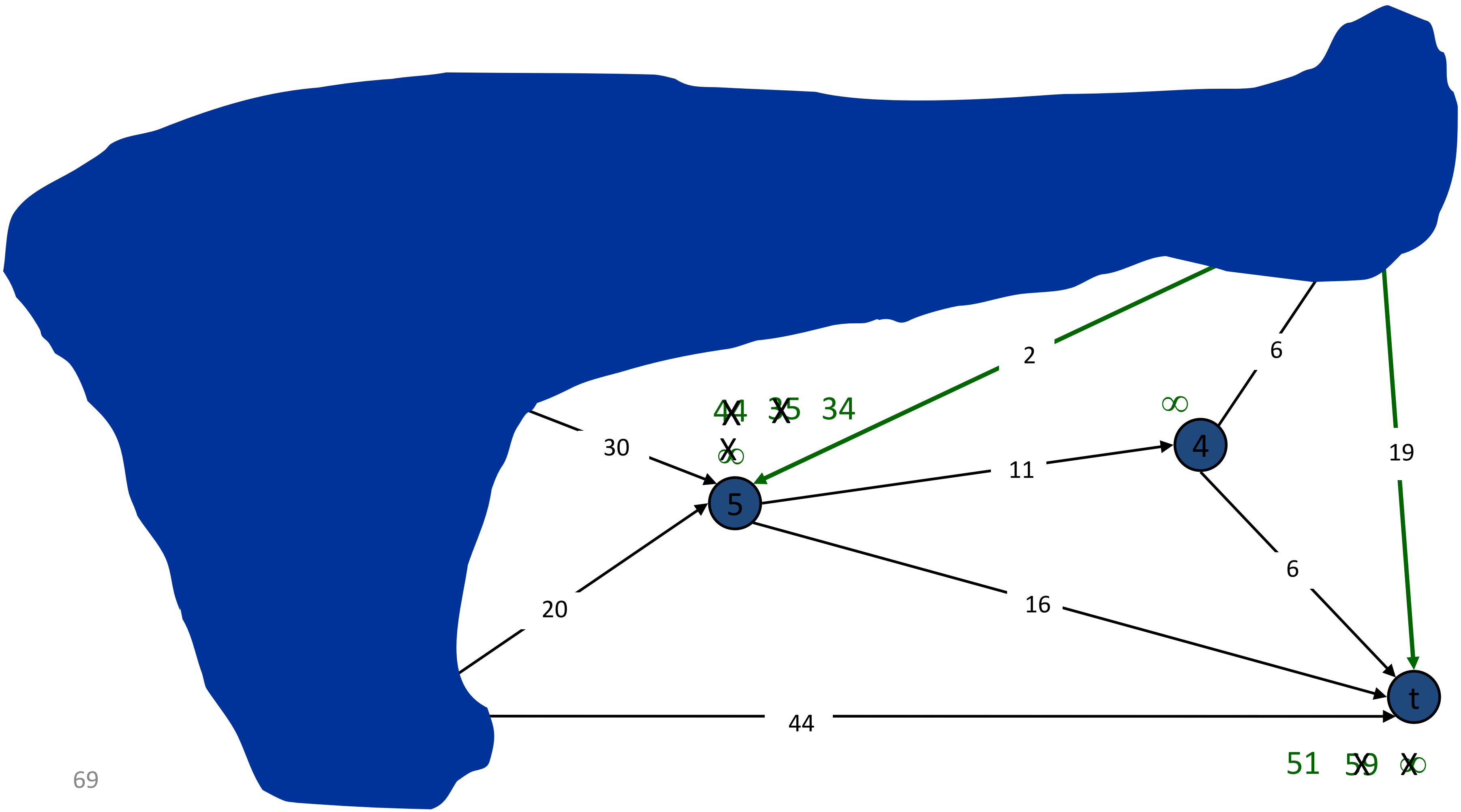
PQ = { 3, 4, 5, t }



Dijkstra's Shortest Path Algorithm

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

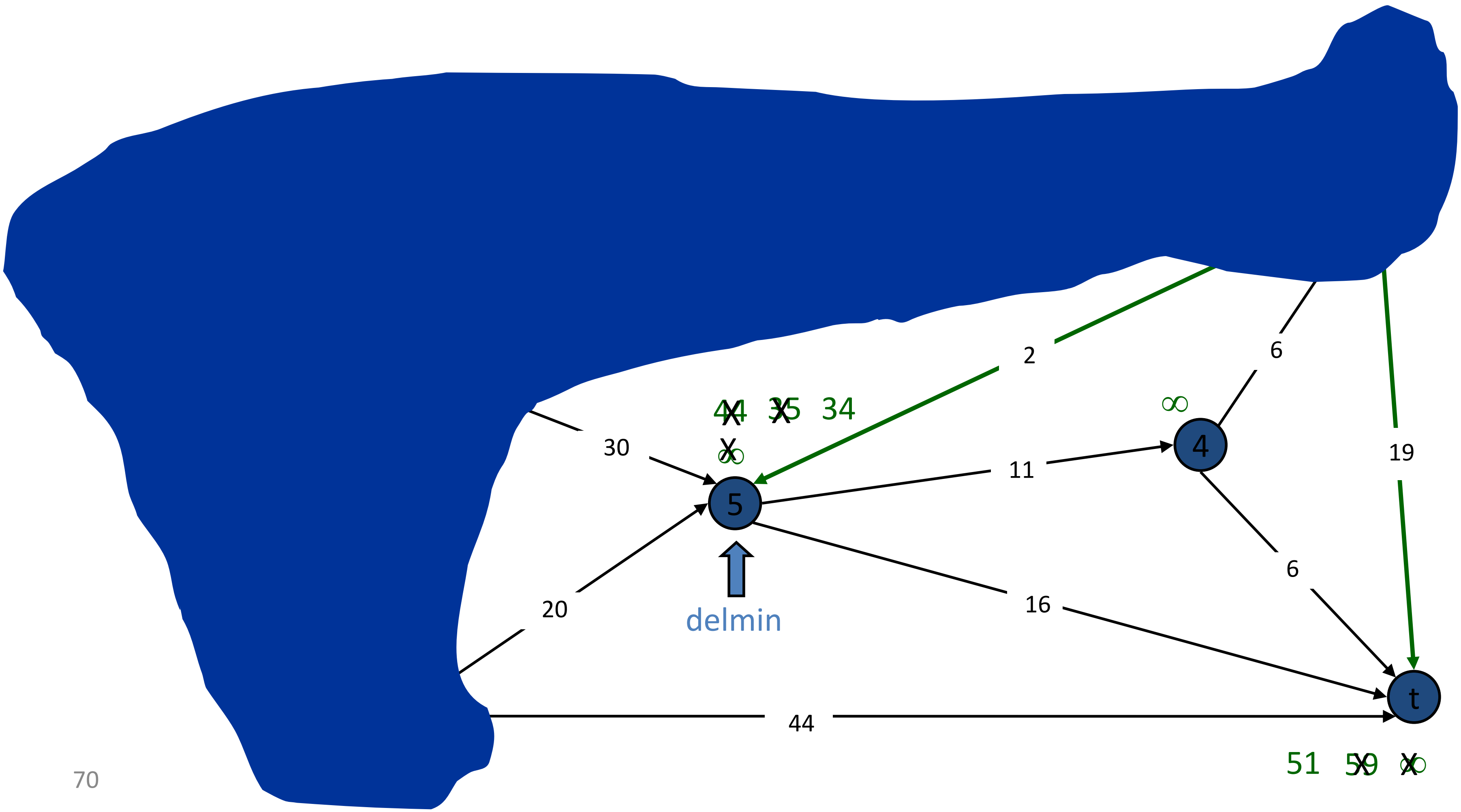
PQ = { 4, 5, t }



Dijkstra's Shortest Path Algorithm

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

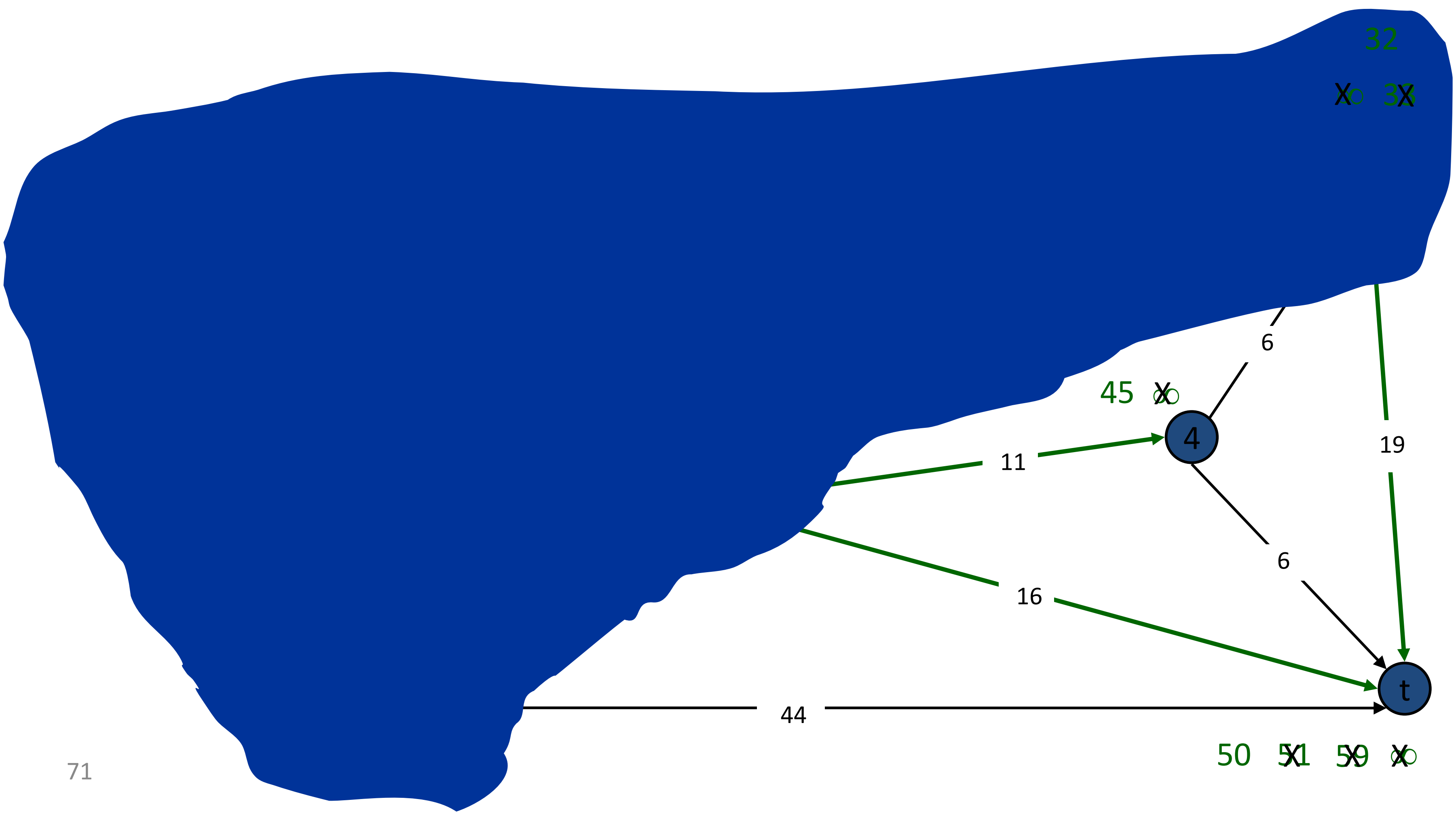
PQ = { 4, 5, t }



Dijkstra's Shortest Path Algorithm

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

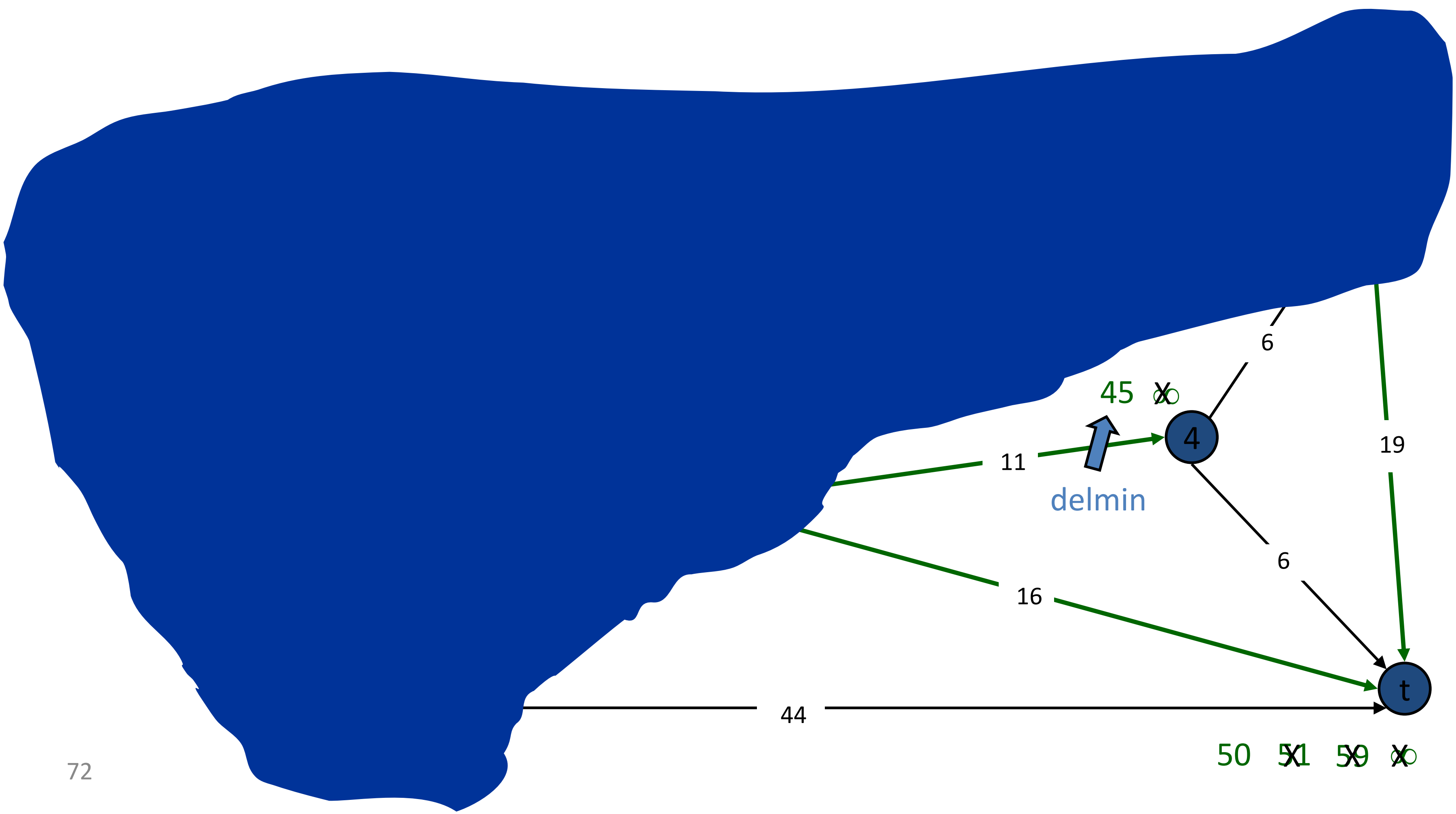
PQ = { 4, t }



Dijkstra's Shortest Path Algorithm

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

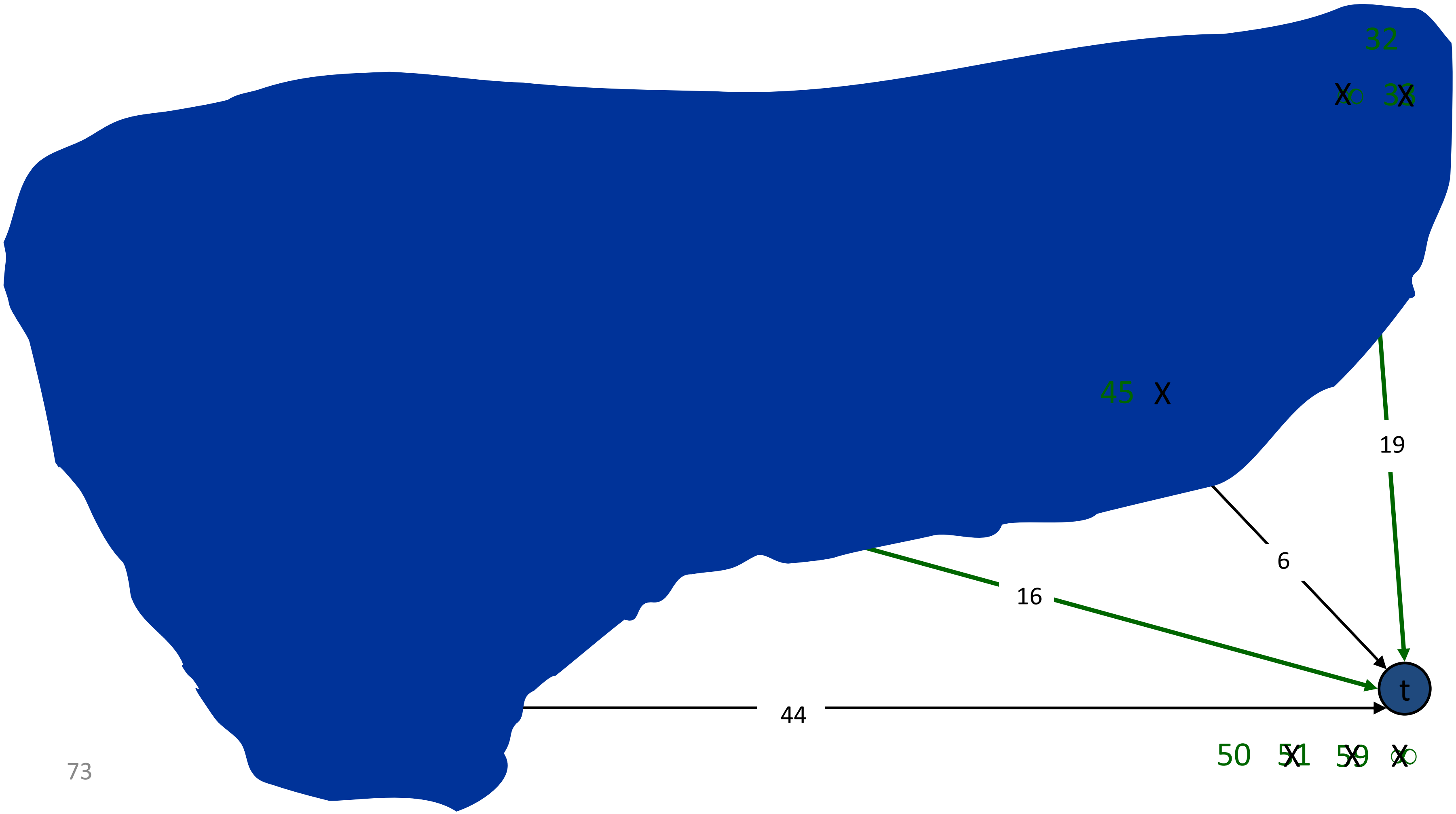
PQ = { 4, t }



Dijkstra's Shortest Path Algorithm

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

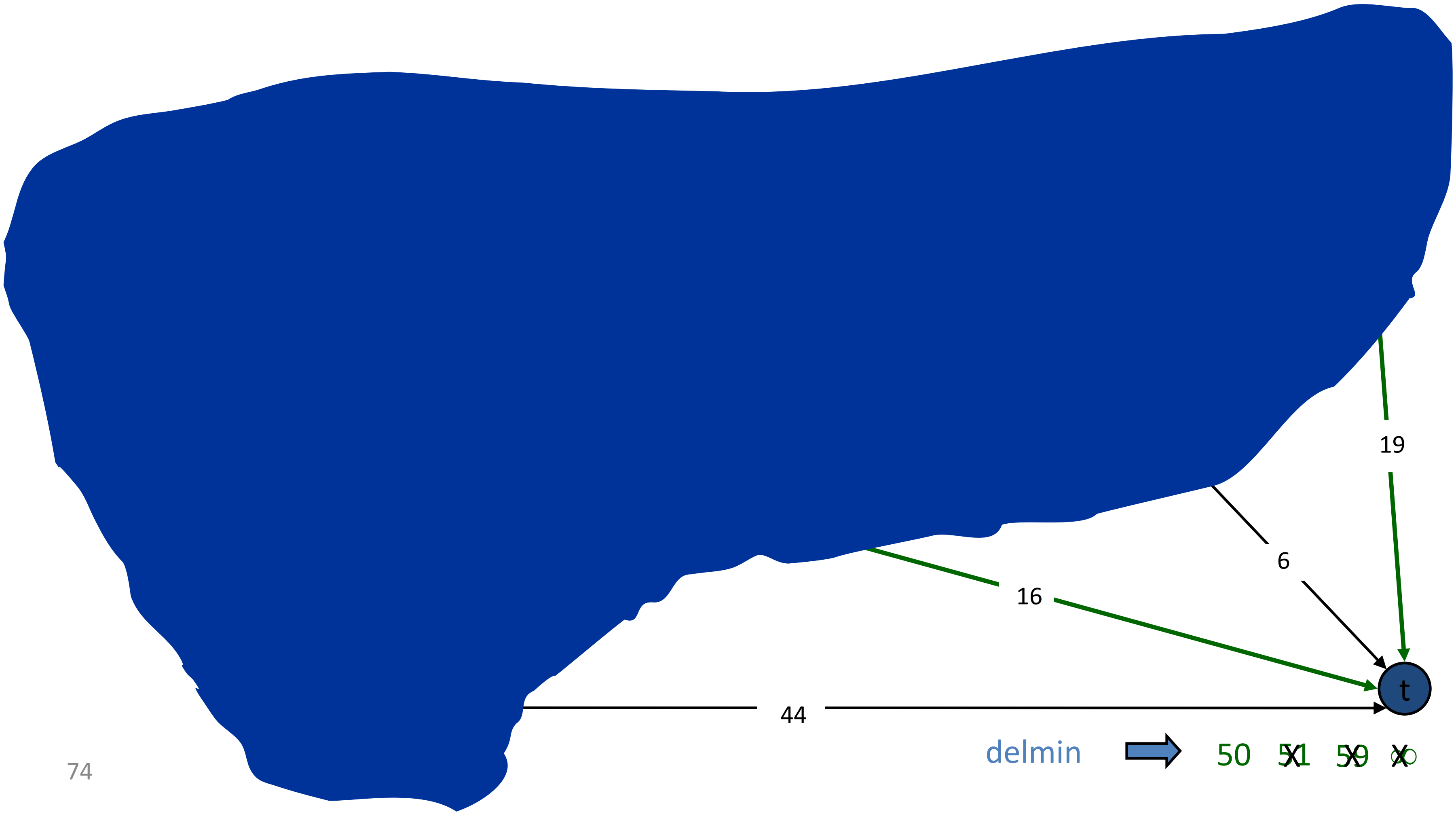
PQ = { t }



Dijkstra's Shortest Path Algorithm

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

PQ = { t }



Dijkstra's Shortest Path Algorithm

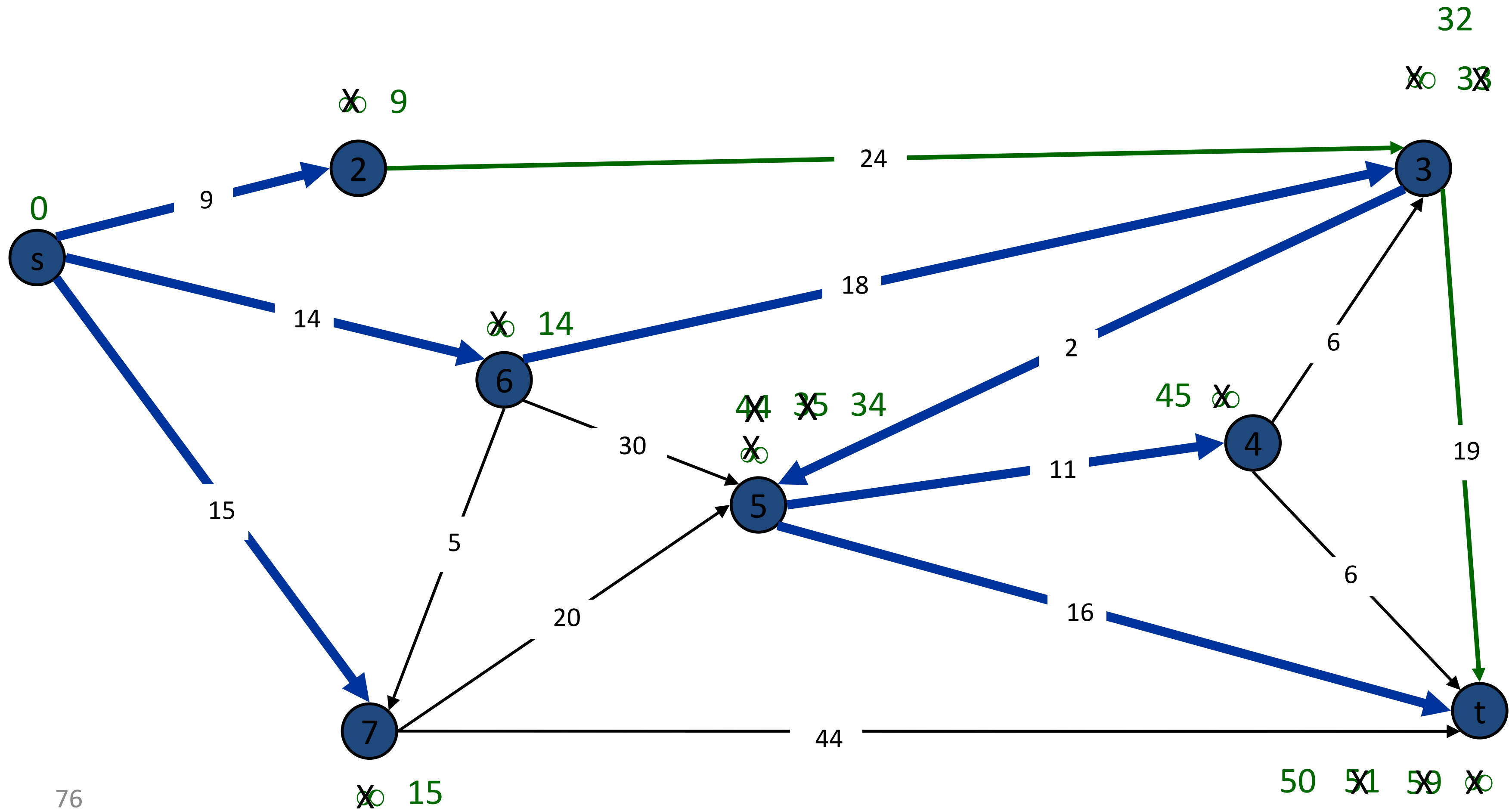
S = { s, 2, 3, 4, 5, 6, 7, t }

PQ = { }

Dijkstra's Shortest Path Algorithm

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

PQ = { }



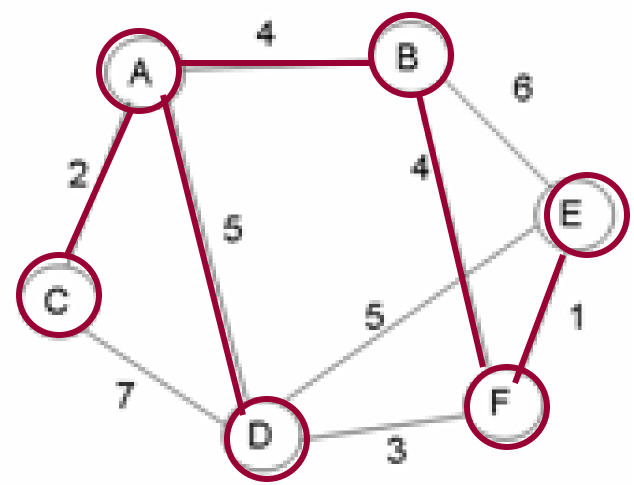
Summary

- Given a weighted directed graph, we can find the shortest distance between two vertices by:
  - starting with a trivial path containing the initial vertex
  - growing this path by always going to the next vertex which has the shortest current path

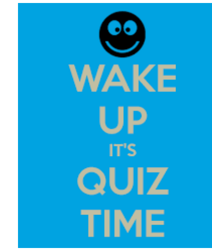
Practice

Node	Included	Distance	Path
A	t	-	-
B	~~f~~ t	4	A
C	~~f~~ t	2	A
D	~~f~~ t	5	A
E	~~f~~ t	~~∞~~ ~~10~~ 9	~~-~~ ~~B~~ F
F	~~f~~ t	~~∞~~ 8	~~-~~ B

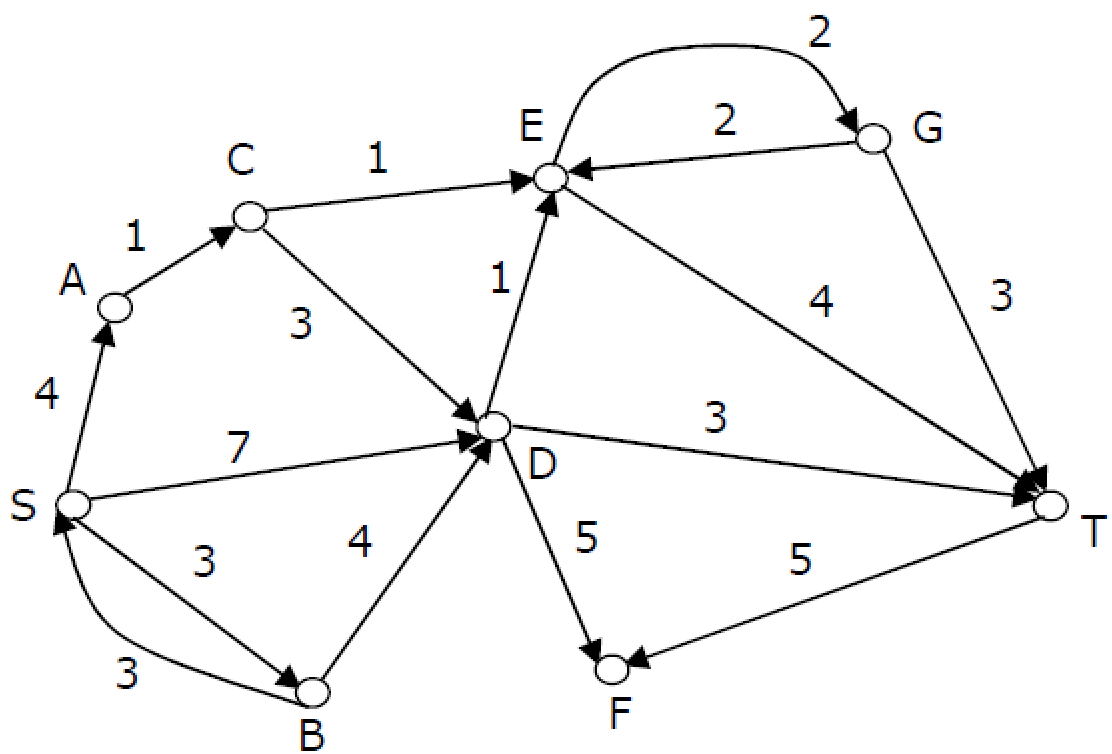
Give the shortest path tree for node A for this graph using Dijkstra's shortest path algorithm. Show your work with the 3 arrays given and draw the resultant shortest path tree with edge weights included.



Quiz 1

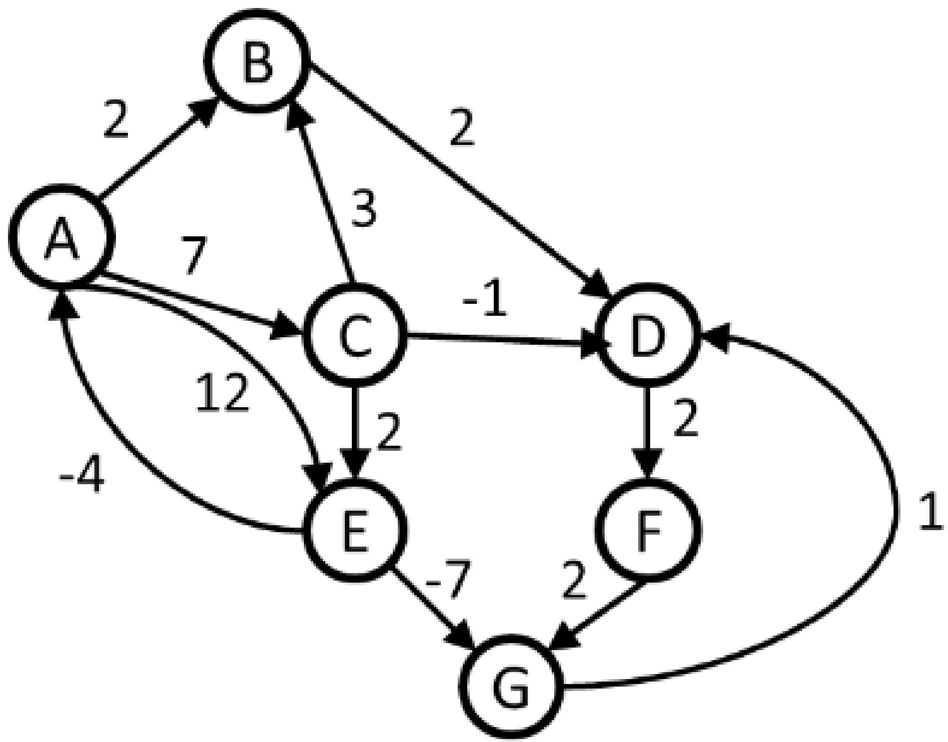


- Consider the directed graph shown in the figure below. There are multiple shortest paths between vertices S and T. Which one will |



Quiz 2

- Calculate shortest paths from A to every other vertex using dijkstra algorithm



Quiz 3

- Show the result of Dijkstra's algorithm from F to D