COMP S265F Unit 5: Graph Algorithms: Shortest Path Problem

Dr. Keith Lee

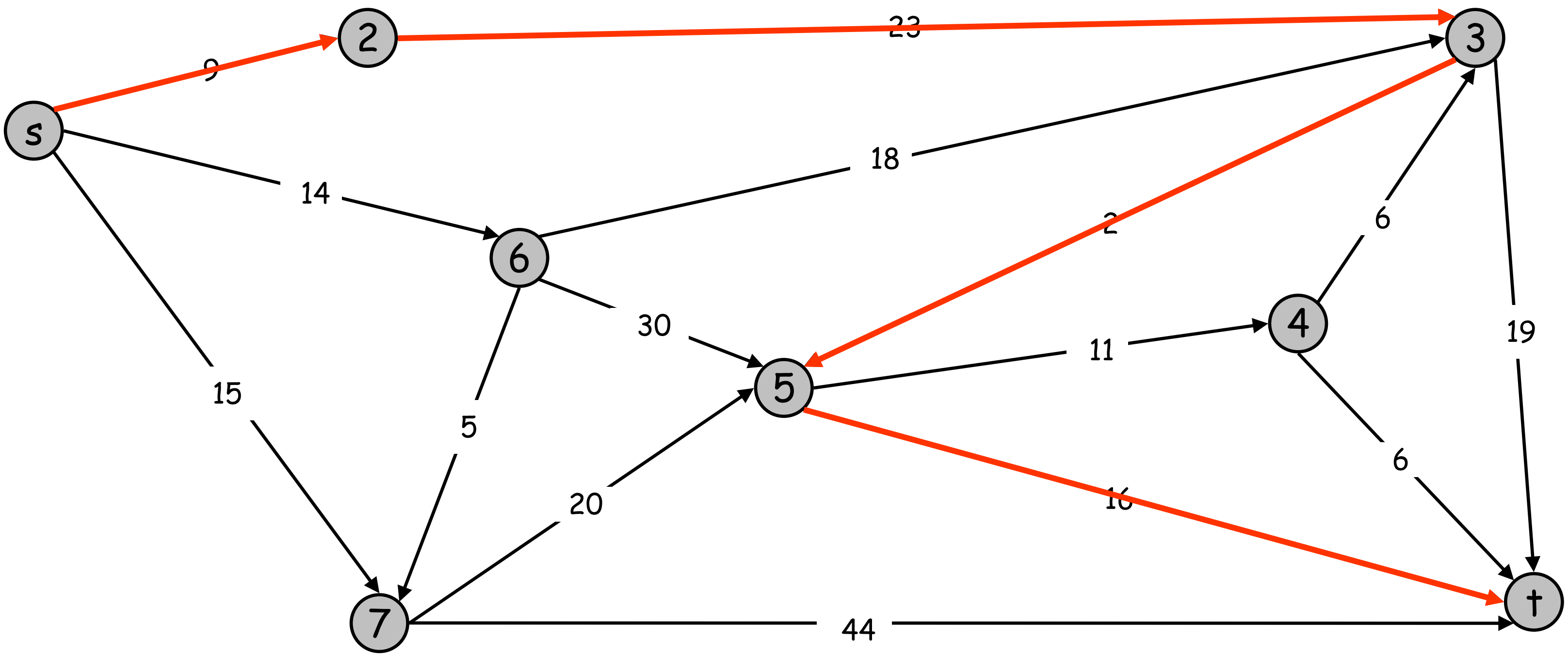
School of Science and Technology

The Open University of Hong Kong

Overview

- Shortest Path Problem
  - Weighted directed graph, Distance of a path
  - Shortest Path Tree
- Dijkstra's algorithm
  - $\delta(s,u)$: length of shortest path from the source s to vertex u
  - **d**(u,v): distance (not weight) of an edge (u,v)
  - Minimum distance outgoing edge
  - Proof of correctness
- Implementing Dijkstra's algorithm
  - Array value d[i] for each vertex I
  - Time complexity analysis
  - How to update d[i] faster using a *Relax* operation?
  - Dijkstra's algorithm & Time complexity

Shortest path problem



Shortest path problem: Definition

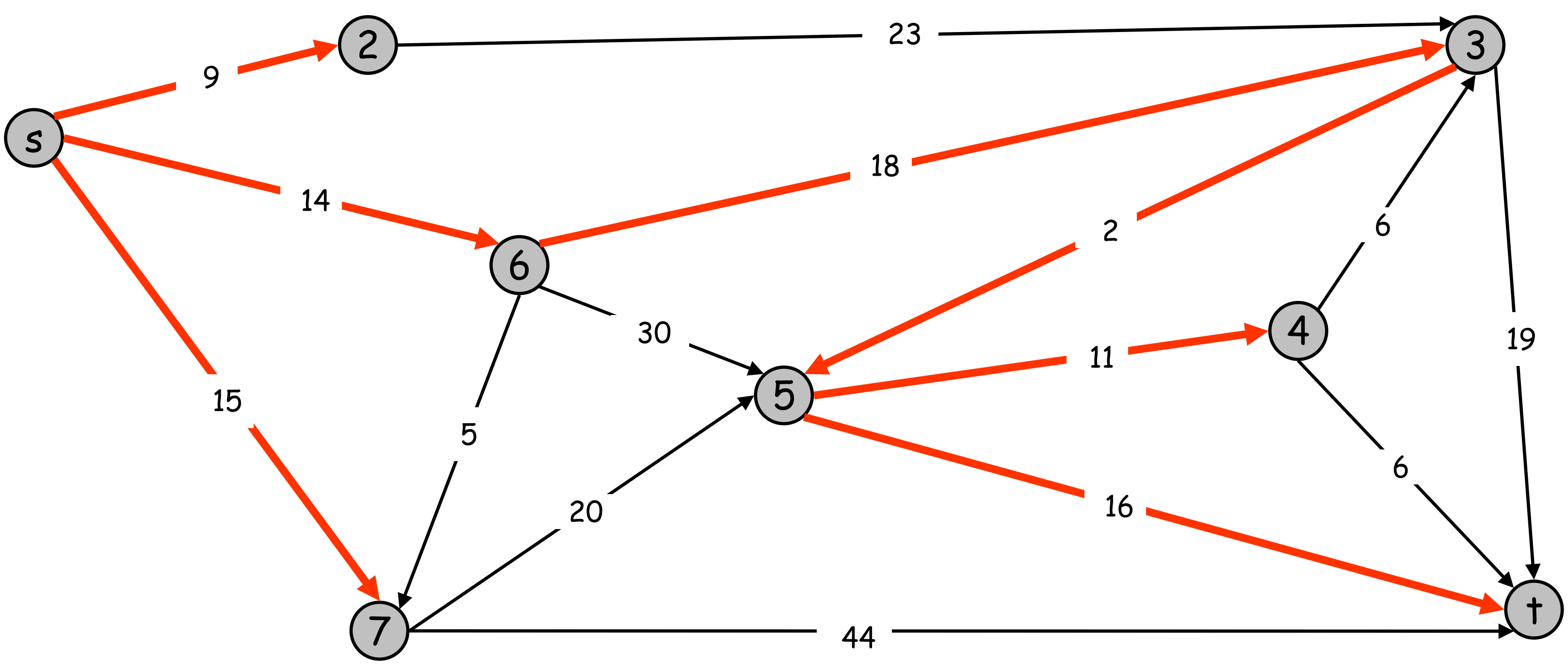
- **Input:**
  - a weighted directed graph $G = (V, E)$;
  - a source vertex $s \in V$, and a destination vertex $t \in V$;
  - every edge (a,b) has a weight $w(a,b) > 0$.
- **Definitions:**
  - A (directed) path **P** from **x** to **y** is a sequence of directed edges $(x,u_1), (u_1,u_2), (u_2,u_3),\ldots, (u_k, y)$.
  - distance of path **P** = sum of weight of the edges in **P**.

- **Shortest path problem**
  - Find a path from the source **s** to the destination **t** with the ***minimum*** distance.
  - ***Note:*** If all edge has unit weight (i.e., unweighted graph), then BFS can find the shortest path.

Shortest Path Tree (SPT)

- A ***shortest path tree (SPT)*** is a rooted tree (with root **s**) of G that satisfies the following property:

> For every vertex **u** ∈ **V**, the tree path from **s** to **u** is a shortest path from **s** to **u**.



- Now, we show how to construct such tree.

The problem

The ***single-source, all-destinations*** shortest path problem:

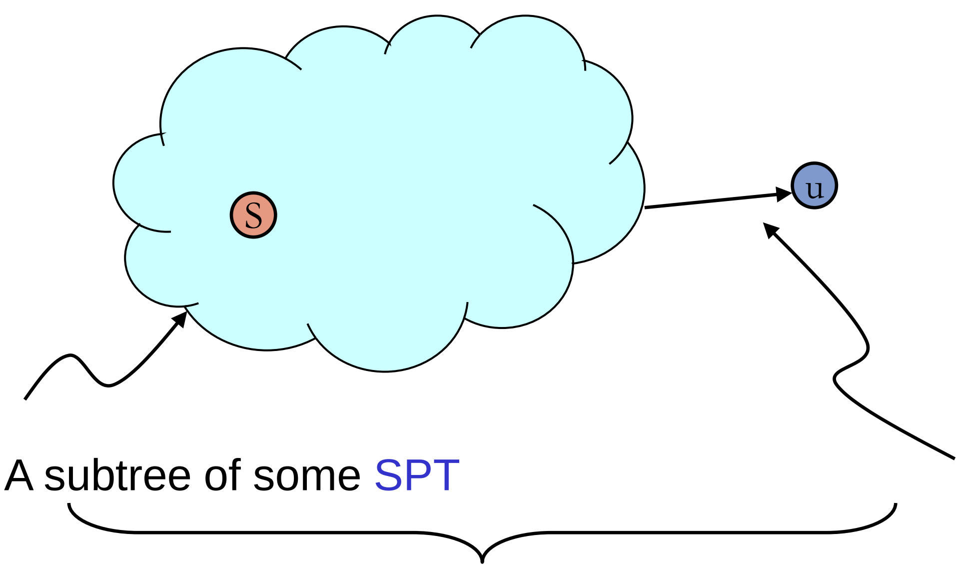
- **Input:**
  A weighted directed graph $G = (V, E)$, and a source $\mathbf{s} \in V$.
- **Output:**
  Build a shortest path tree (SPT) of G rooted at $\mathbf{s}$.

- We introduce the **Dijkstra's algorithm** that builds such SPT.

The Dijkstra's algorithm

- Dijkstra is a greedy algorithm.
- A general picture:

An edge is an **outgoing edge** of P if it goes from an endpoint in P to another endpoint not in P.



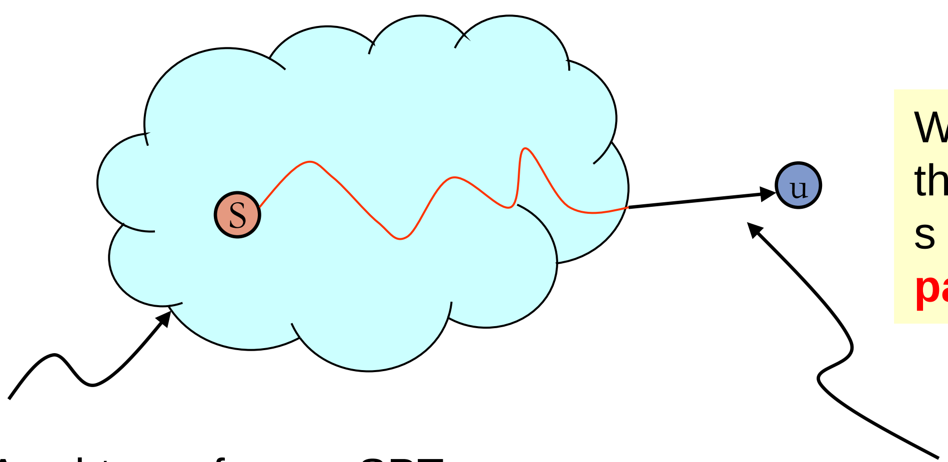
P: A subtree of some SPT

Add a correct outgoing edge.

A larger P which is still a subtree of some SPT

The Dijkstra's algorithm

- Dijkstra is a greedy algorithm.
- A general picture:



We need to ensure that the tree path from s to u is indeed a **shortest path** from s to u in G.

P: A subtree of some SPT

Add a correct outgoing edge.

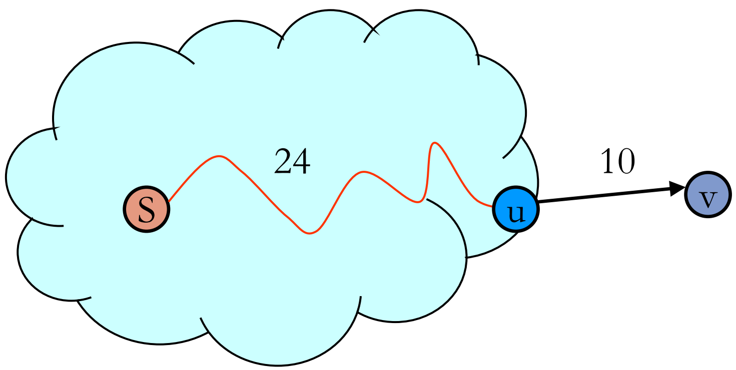
A larger P which is still a subtree of some SPT

What kind of outgoing edge do we need this time?

- For every vertex **u** $\in$ **V**, let δ(**s**,**u**) denotes the length of the shortest path from the source **s** to **u**.

- Given any edge **e** = (u,v) of G, define the (edge) distance of **e** to be

$$\mathbf{d}(u,v) = \delta(s,u) + \mathbf{w}(u,v).$$

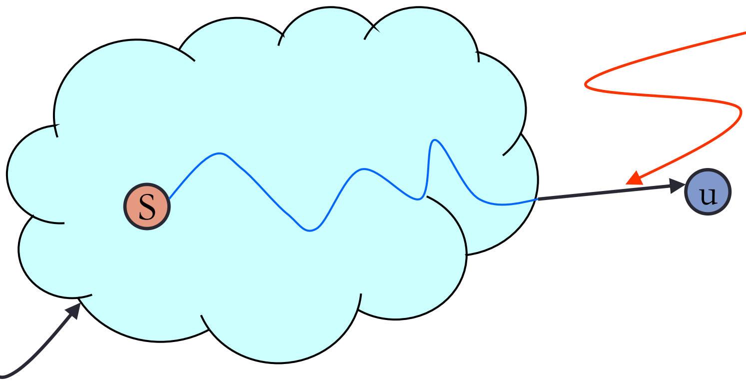


d(u,v) = 34

What kind of outgoing edge do we need this time?

Algorithm: Starting with P={s}, repeatedly add to P the minimum distance outgoing edge of P until P includes all the vertices.

We add the outgoing edge of P with minimum distance.

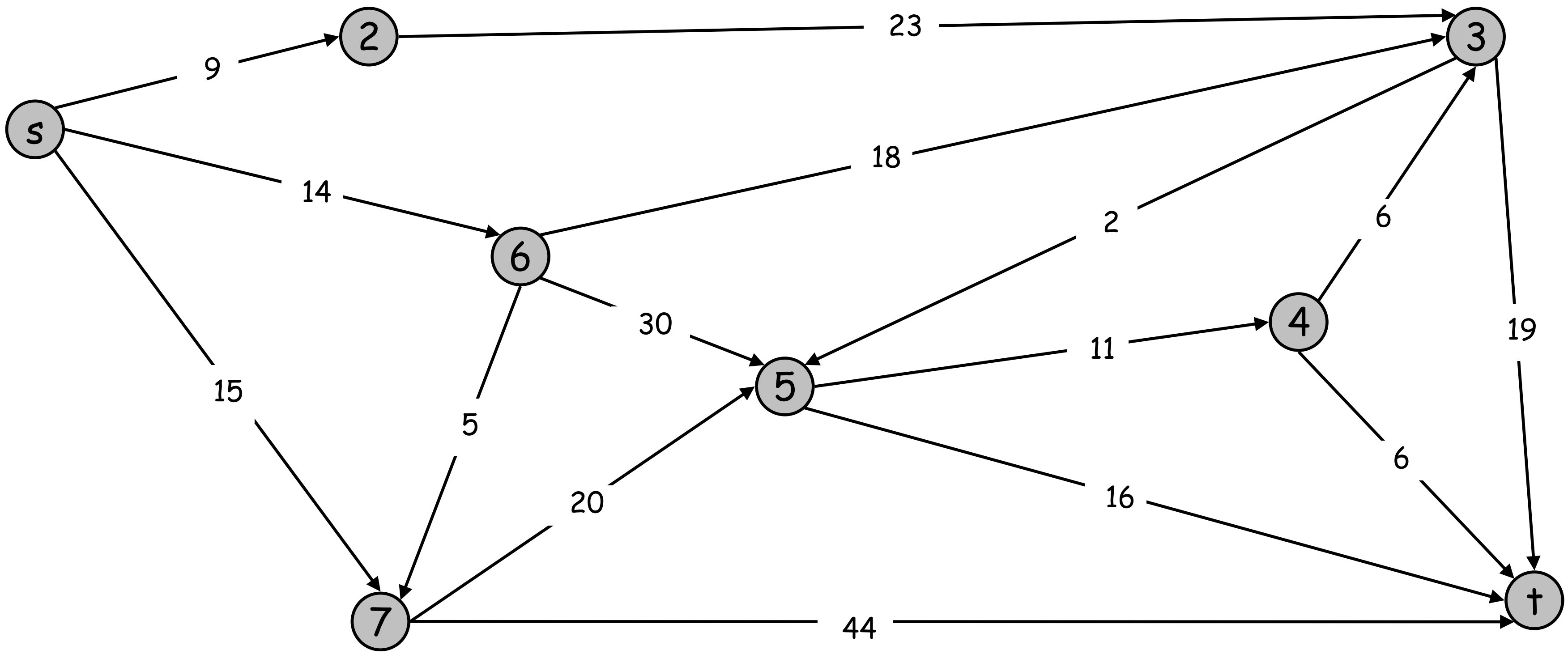


P: A subtree of some SPT.

Add the right edge e that ensures P + e is a subtree of some SPT.

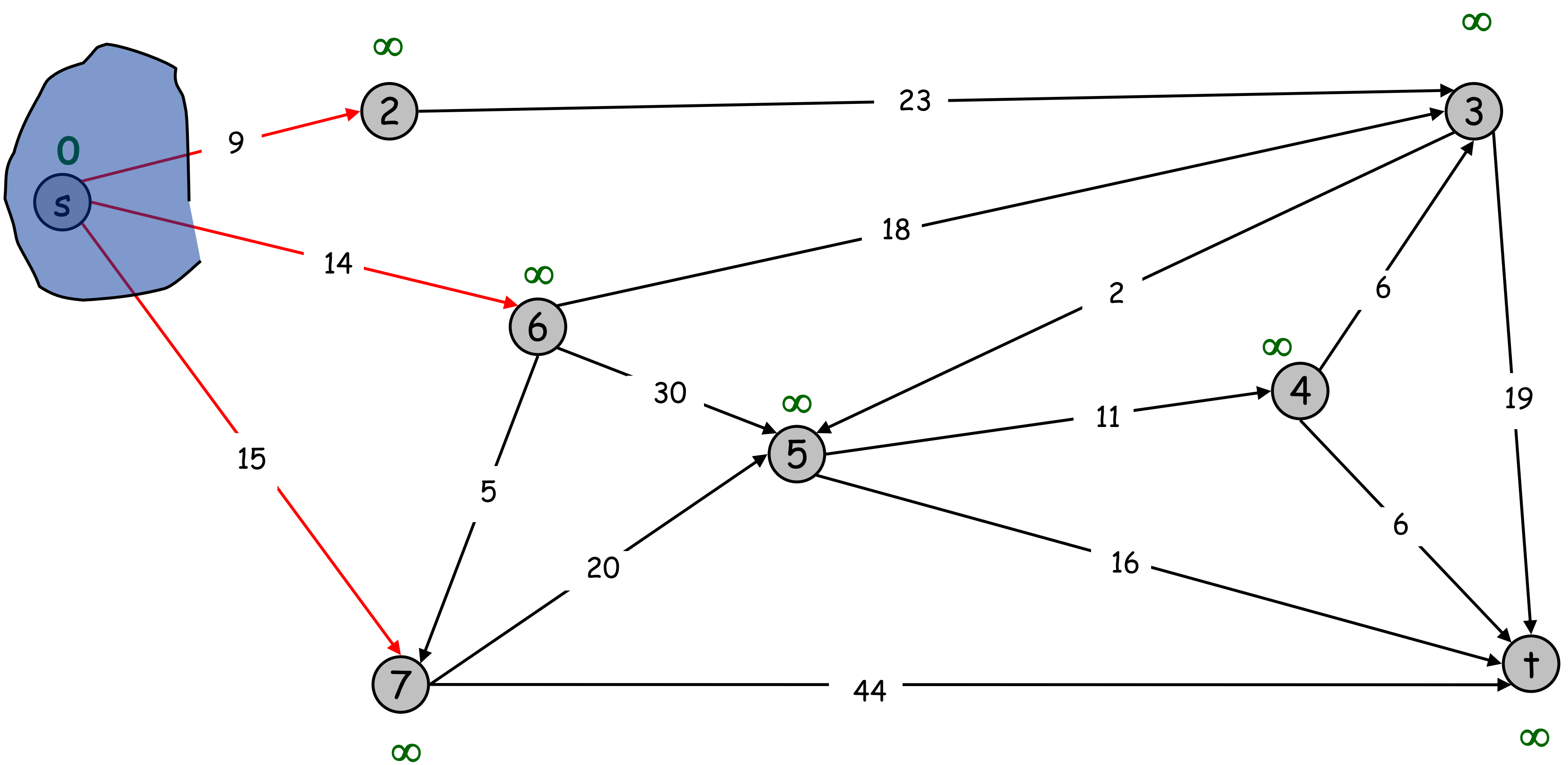
The Dijkstra's algorithm: Sample run

- Find the shortest path from **s** to **t**.
- We will record δ(**s**,**u**) for every vertex **u** $\in$ **V**.



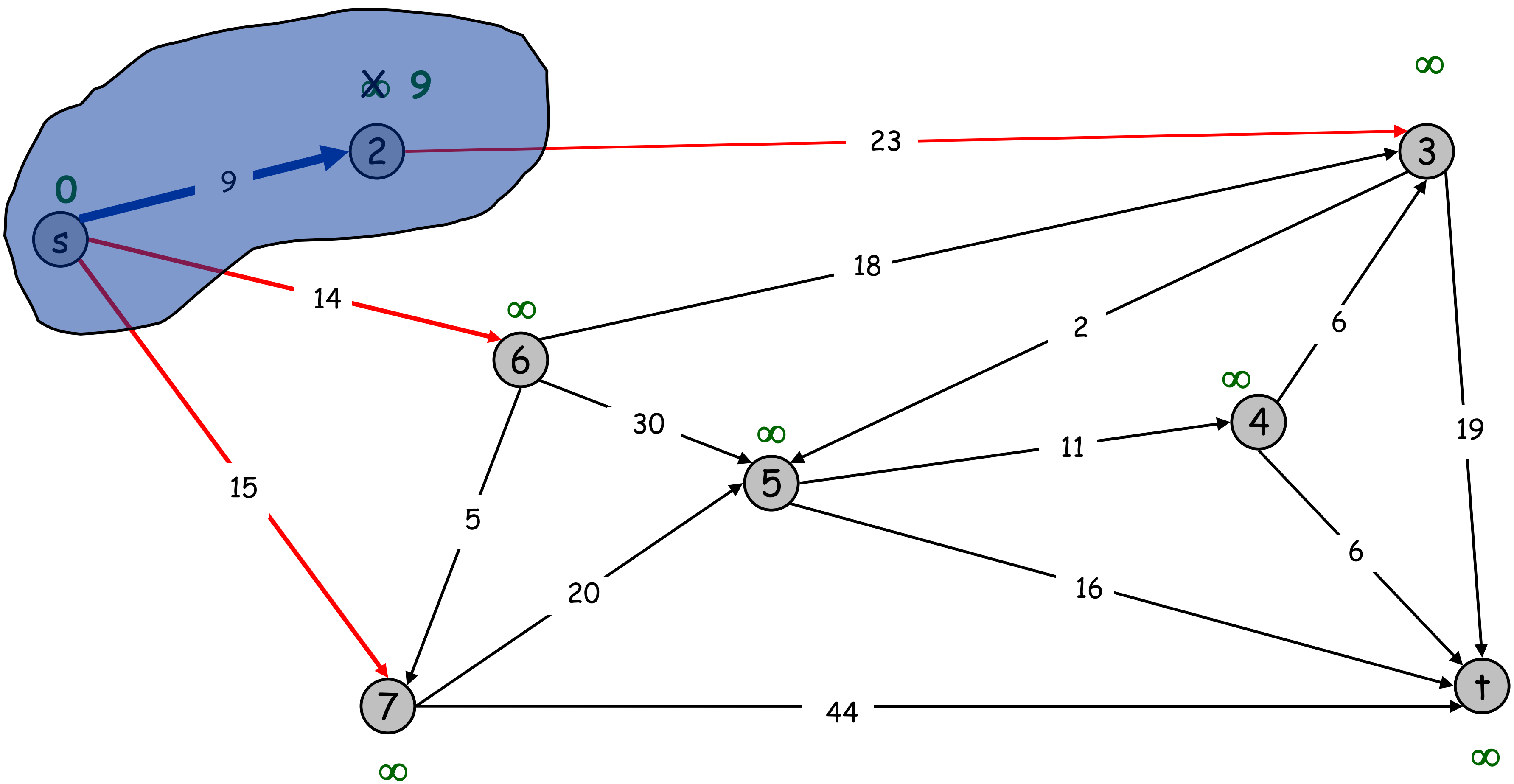
Sample run (Step 1)

P= { s }



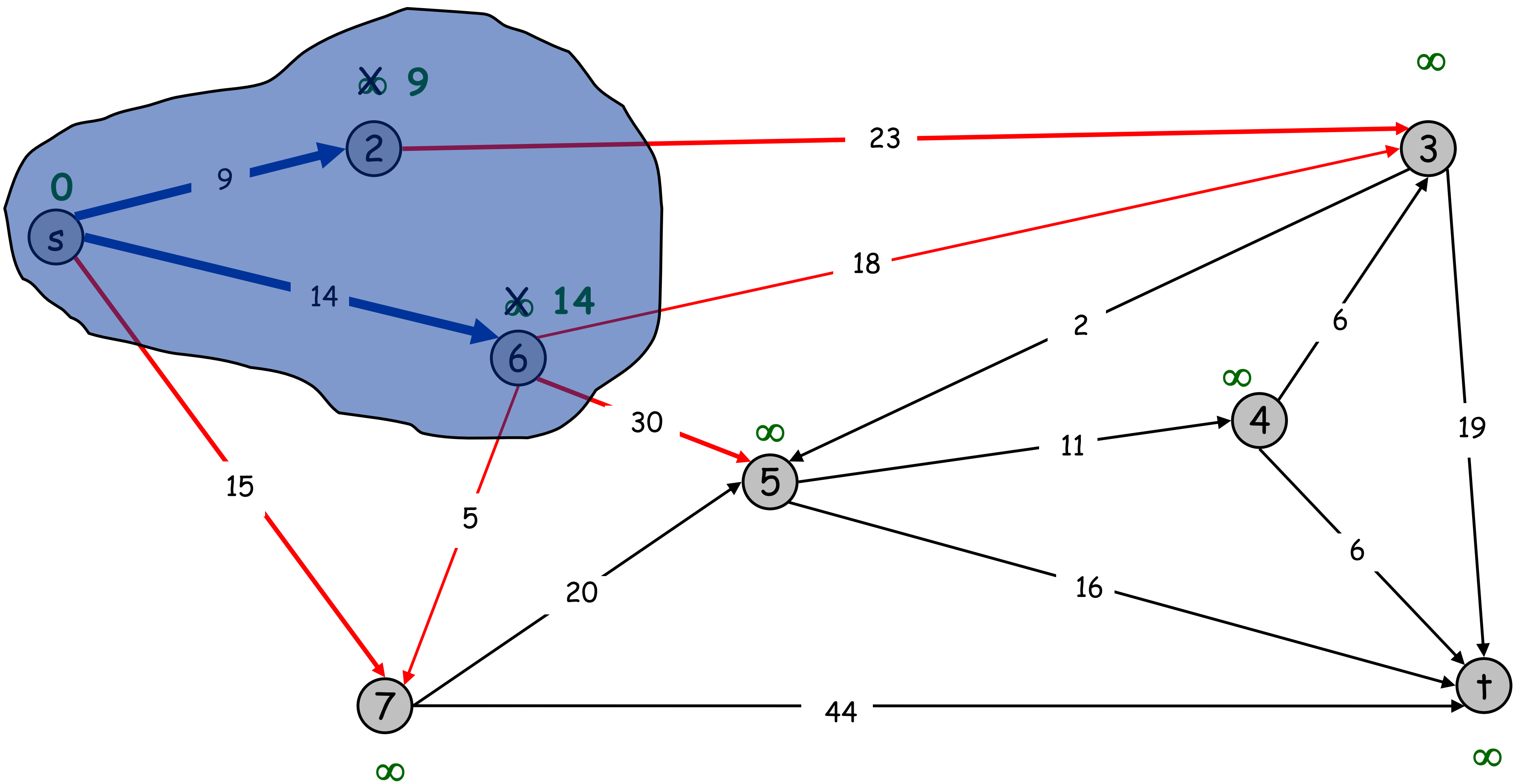
Sample run (Step 2)

P = { s, 2 }



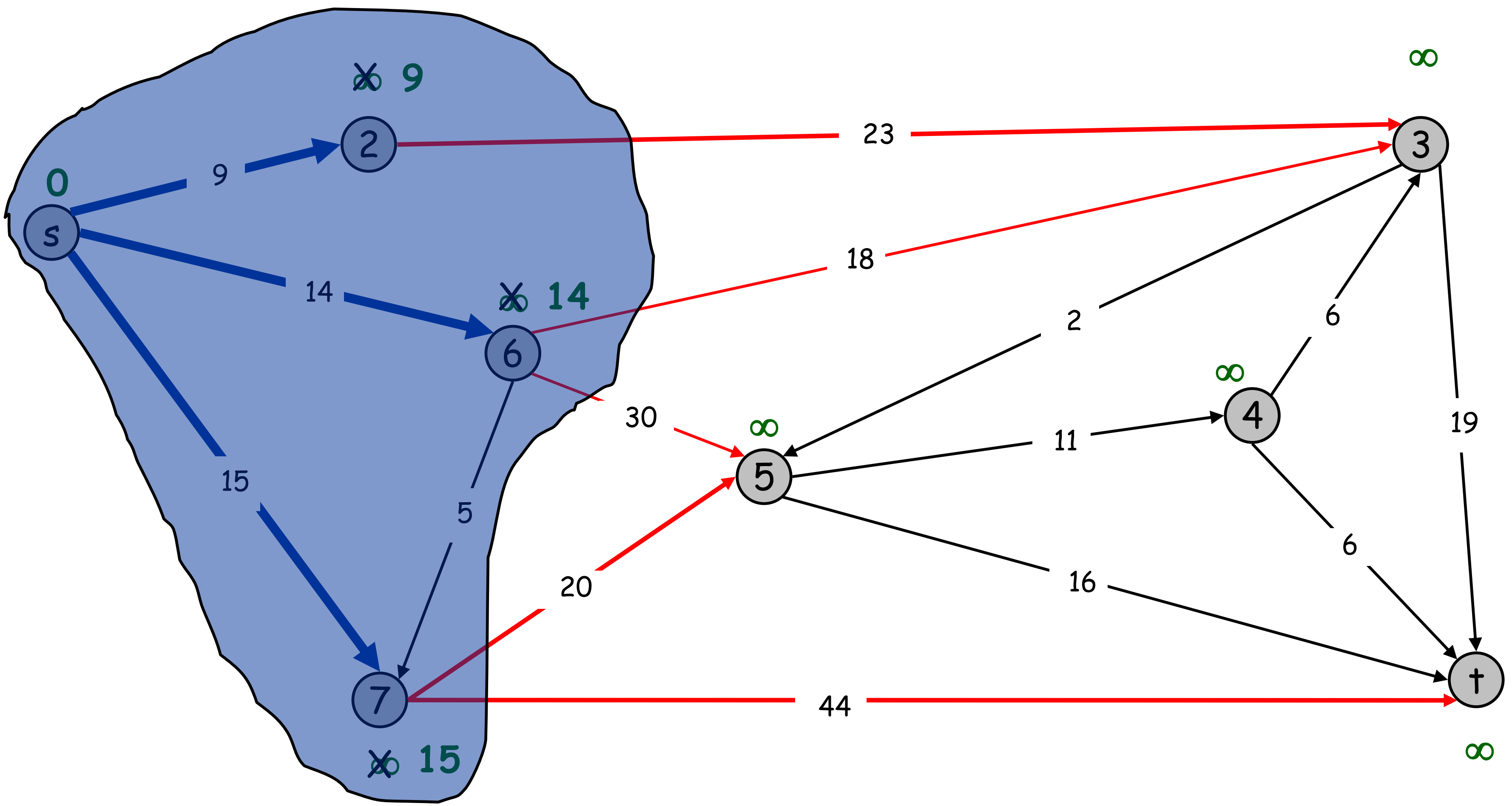
Sample run (Step 3)

P = { s, 2, 6 }



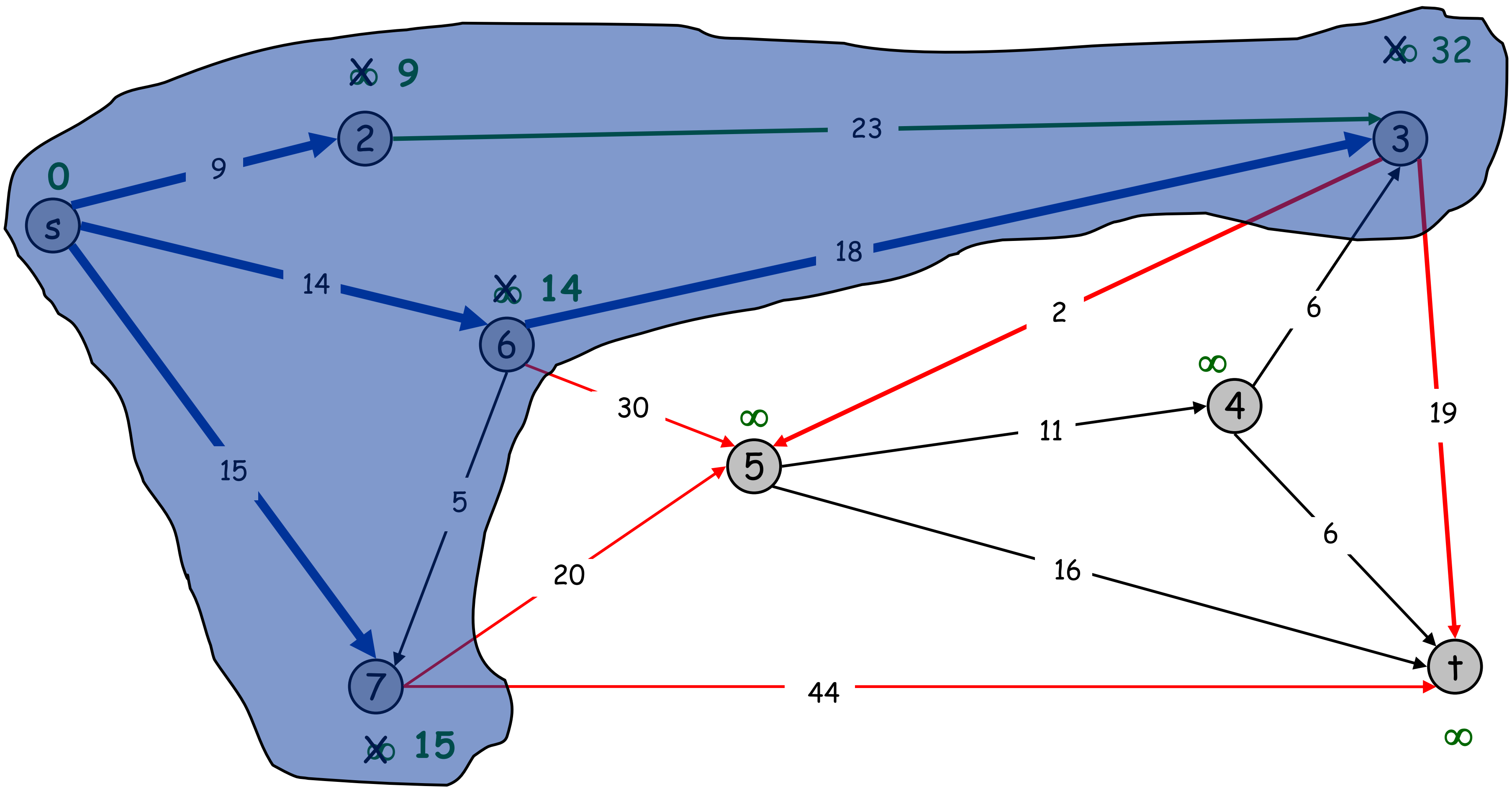
Sample run (Step 4)

P = { s, 2, 6, 7 }



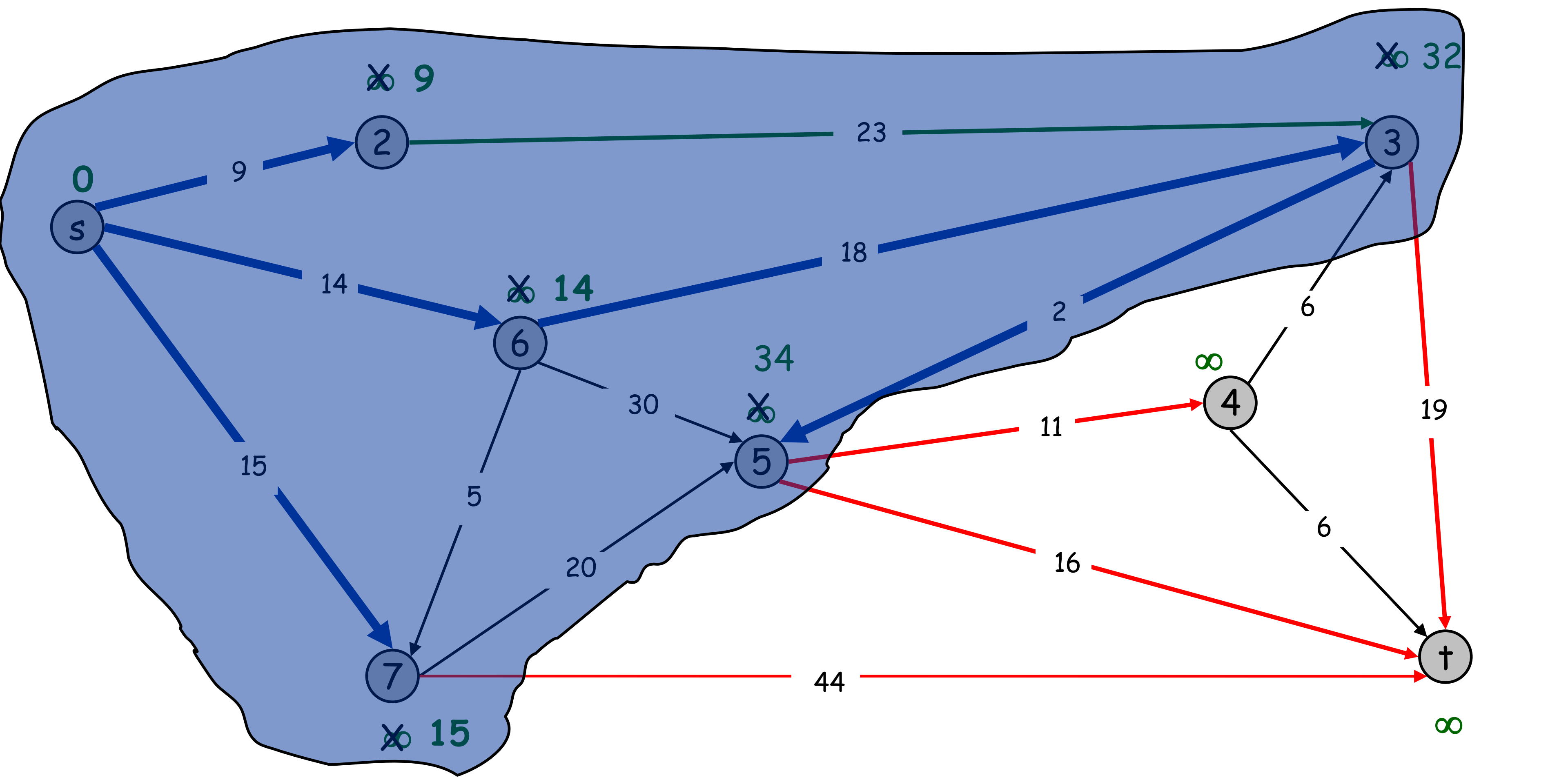
Sample run (Step 5)

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



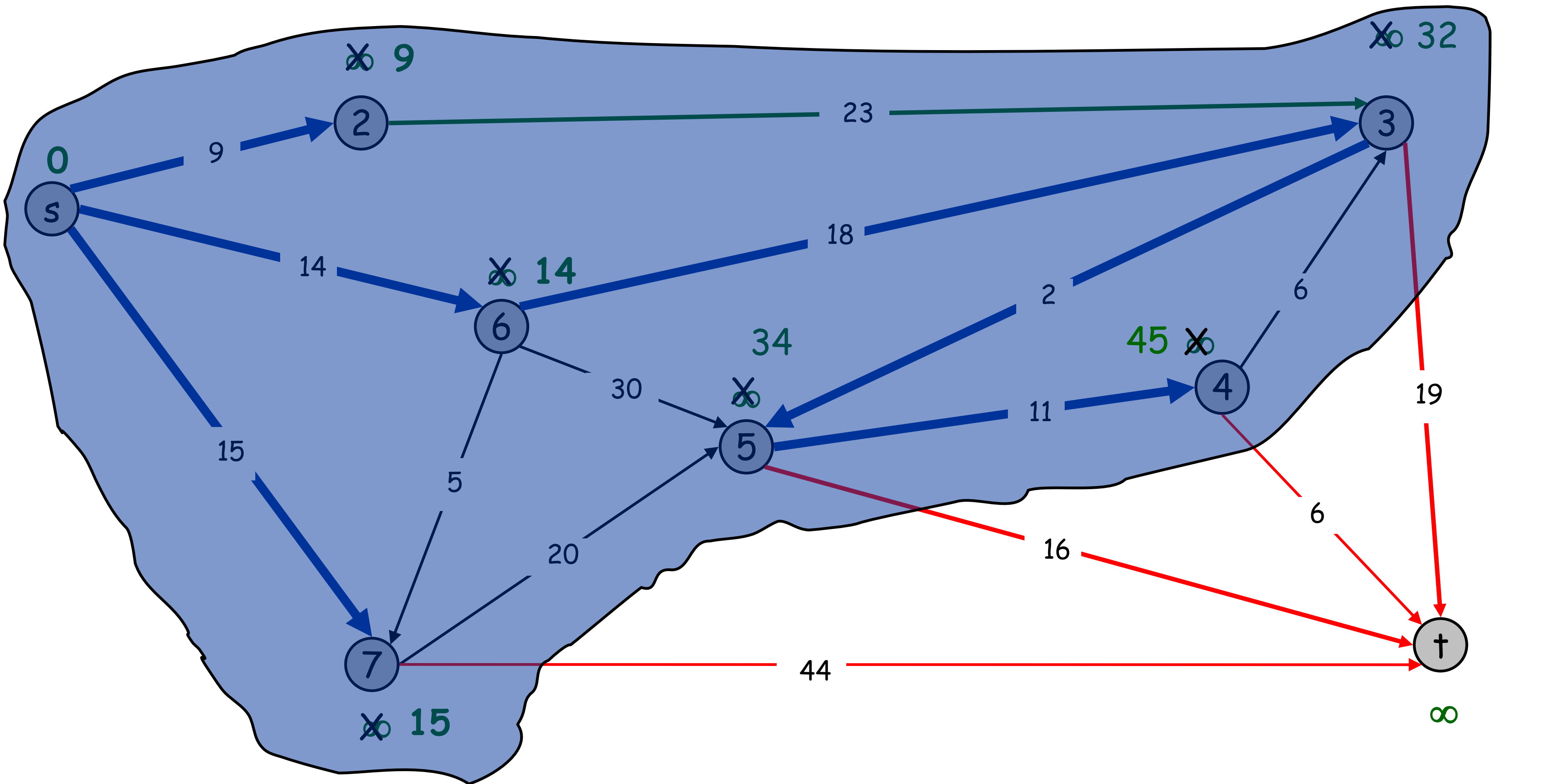
Sample run (Step 6)

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



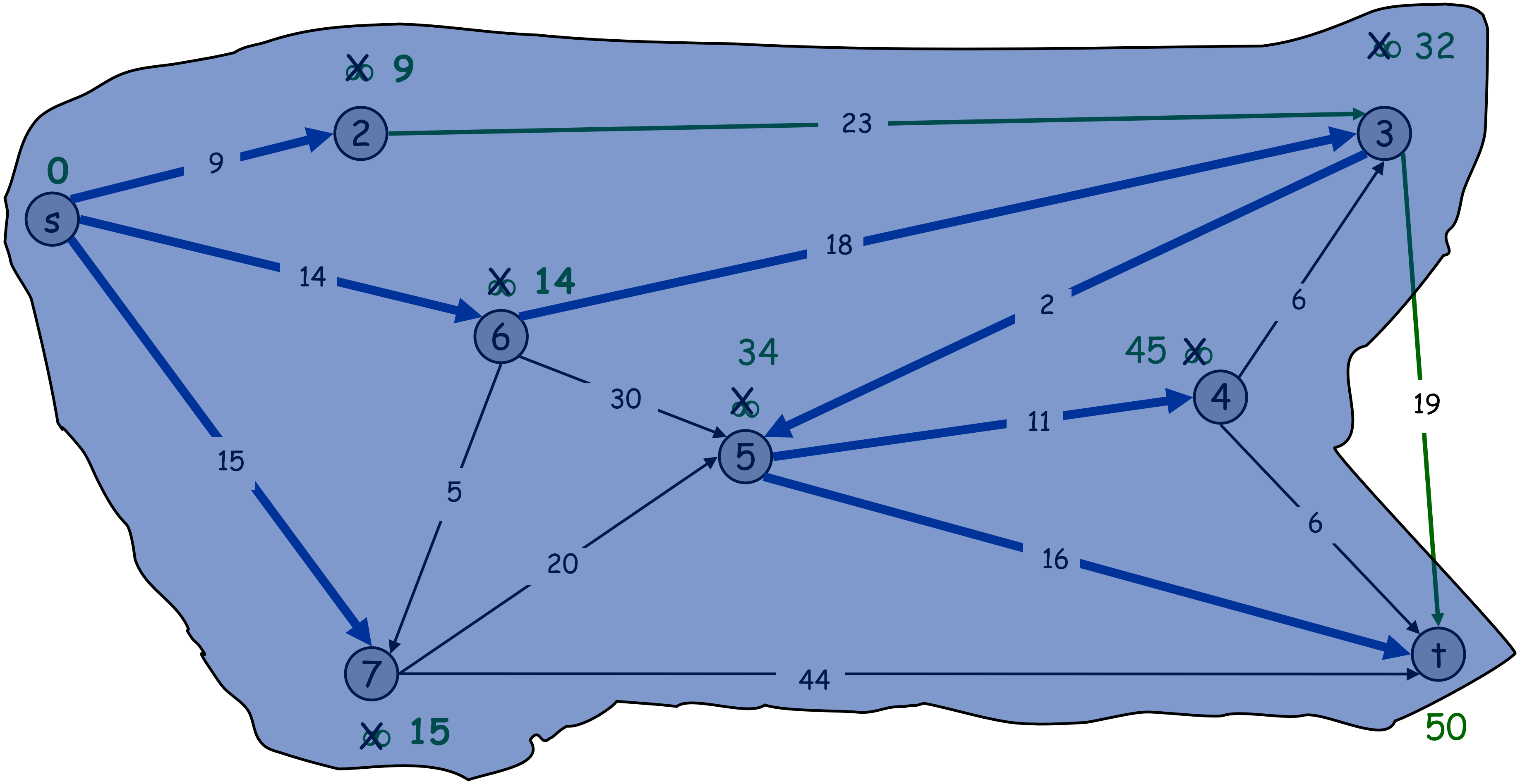
Sample run (Step 7)

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



Sample run (Step 8)

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



Correctness of Dijkstra’s algorithm

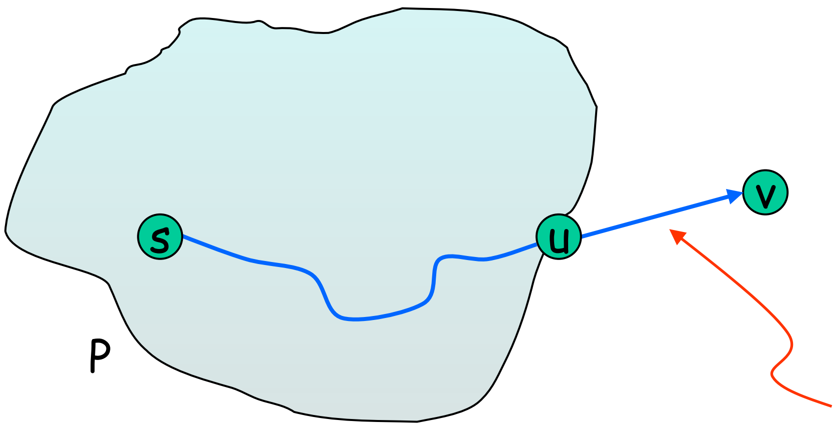
Lemma. After every iteration, the following is true: for every vertex **u** in P, the tree path from **s** to **u** is indeed a shortest path.

Proof. Again, we prove the lemma by induction on the iteration.

- **Base case:** The 1st iteration is obviously true because in this case, P contains only the source **s**, and the tree path from **s** to **s** in P is the path with no edge, which is obviously a shortest path.
- **Induction hypothesis:** Suppose that the lemma is true for the (i-1)st iteration. We consider the i-th iteration.

Correctness of Dijkstra's algorithm (cont')

We consider the i-th iteration.



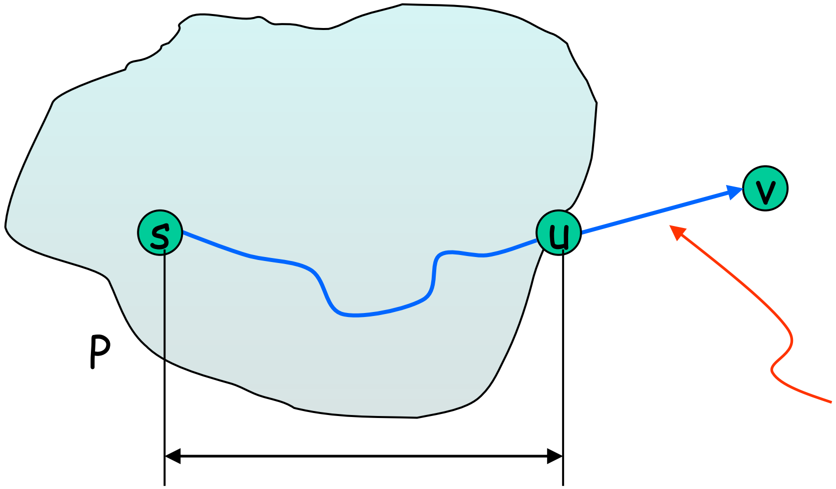
the outgoing edge of P with minimum distance.

The P in the (i-1)st iteration

The P in the ith iteration

Correctness of Dijkstra’s algorithm (cont’)

We consider the i-th iteration.



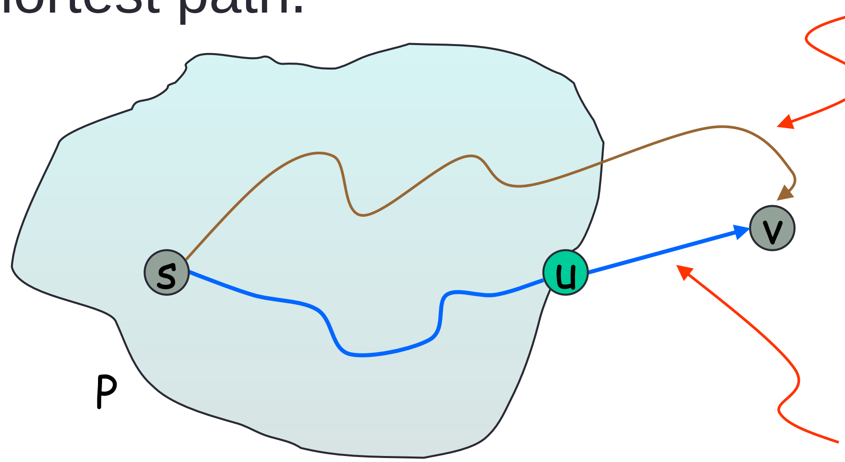
length of this path
= δ(s,u) + w(u,v)
= d(u,v).

By the induction hypothesis,
this tree path is a shortest path,
and its length is δ(s,u).

Correctness of Dijkstra's algorithm (cont')

We now prove that the tree path from s to v (i.e., the blue path) is indeed a shortest path.

σ: any path from s to v.



length of this path
= $\delta(s,u) + w(u,v)$
= $d(u,v)$.

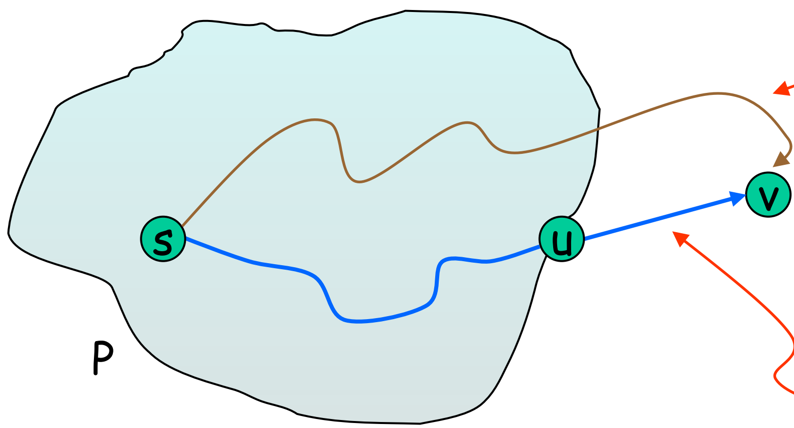
$$\sigma = \langle \underbrace{s\ \ p_0\ \ p_1\ \dots\ p_m}_{\text{all in P}}\ \ q_1\ \dots\ \ q_k\ v \rangle$$

not in P

Correctness of Dijkstra's algorithm (cont')

We now prove that the tree path from s to v (i.e., the blue path) is indeed a shortest path.

σ: any path from s to v.



length of this path
= δ(s,u) + w(u,v)
= d(u,v).

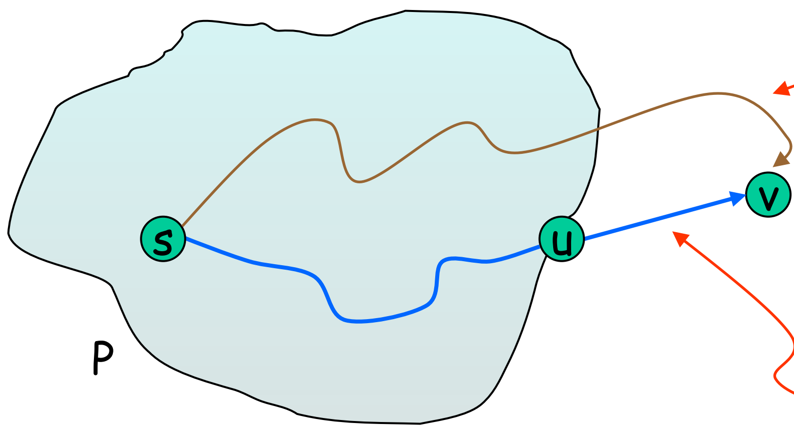
$w(\sigma) = w(s\ \ p_0\ \ p_1\ \ldots\ \ p_m\ \ q_1\ \ldots\ \ q_k\ v) \geq w(s\ \ p_0 \ldots\ \ p_m\ \ q_1)$

Because the new shorter path has few edges, and **all edges have positive weight**

Correctness of Dijkstra’s algorithm (cont’)

We now prove that the tree path from s to v (i.e., the blue path) is indeed a shortest path.

σ: any path from s to v.



length of this path
$= \delta(s,u) + w(u,v)$
$= d(u,v)$.

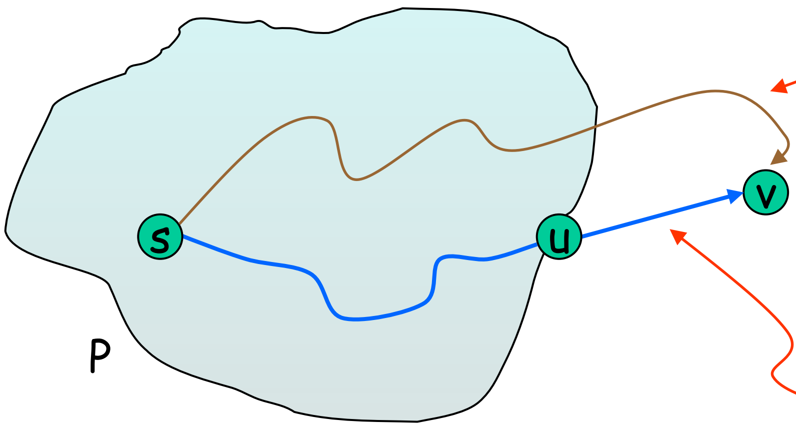
$$w(\sigma) = w(s\ \ p_0\ \ p_1\ \ldots\ \ p_m\ \ q_1\ \ \ldots\ \ q_k\ v) \geq w(\underbrace{s\ \ p_0 \ldots\ \ p_m}_{}\ \ q_1)$$

A path from s to p_m, and (p_m,q_1) is an outgoing edge of P.

Correctness of Dijkstra's algorithm (cont')

We now prove that the tree path from s to v (i.e., the blue path) is indeed a shortest path.

σ: any path from s to v.



length of this path
= δ(s,u) + w(u,v)
= d(u,v).

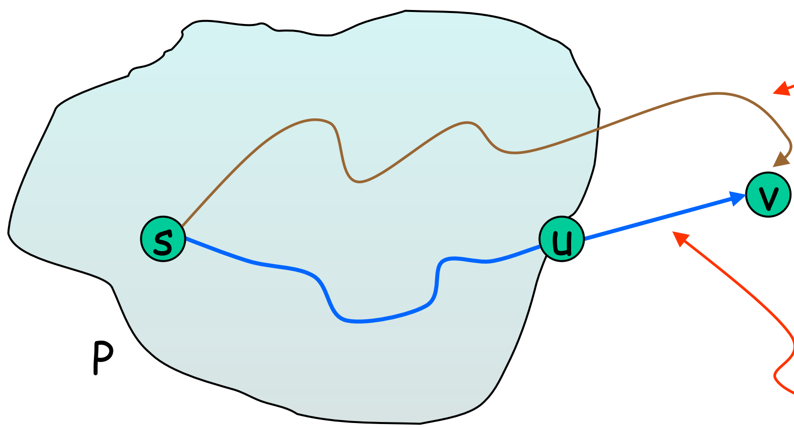
$$w(\sigma) = w(s\ \ p_0\ p_1 \ldots\ p_m\ q_1\ \ldots\ \ q_k\ v) \geq w(s\ \ p_0 \ldots\ p_m\ q_1) \geq \delta(s, p_m) + w(p_m, q_1) = d(p_m, q_1)$$

The distance of the path (s $p_0 \ldots\ p_m$) is at least the length of the shortest path from s to p_m.

Correctness of Dijkstra's algorithm (cont')

We now prove that the tree path from s to v (i.e., the blue path) is indeed a shortest path.

σ: any path from s to v.



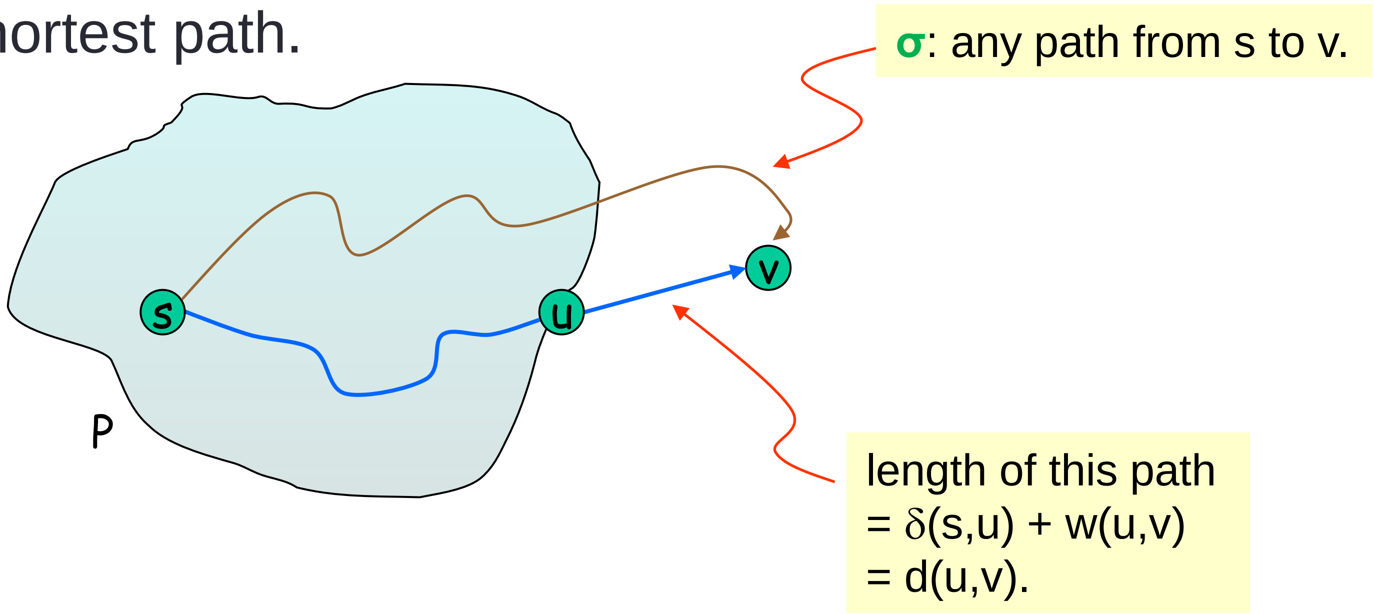
length of this path
$= \delta(s,u) + w(u,v)$
$= d(u,v)$.

$$w(\sigma) = w(s\ \ p_0\ p_1\ \dots\ p_m\ q_1\ \dots\ q_k\ v) \geq w(s\ \ p_0 \dots\ p_m\ q_1) \geq \delta(s, p_m) + w(p_m, q_1)$$
$$= d(p_m, q_1) \geq d(u,v)$$

Because (u,v) is the outgoing edge with minimum distance.

Correctness of Dijkstra's algorithm (cont')

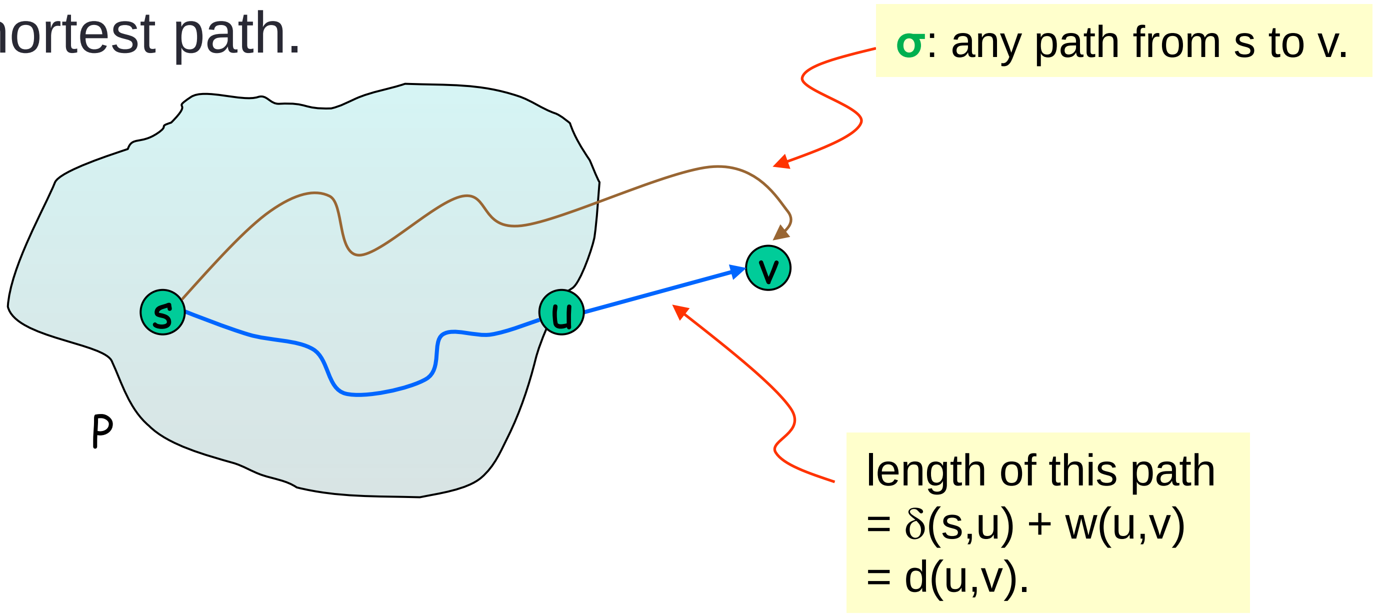
We now prove that the tree path from s to v (i.e., the blue path) is indeed a shortest path.



Thus, length of the blue path ≤ length of **σ**. Since **σ** can be any path from s to v, the blue path is indeed a shortest path.

Correctness of Dijkstra's algorithm (cont')

We now prove that the tree path from s to v (i.e., the blue path) is indeed a shortest path.



Moreover, d(u,v) is the shortest path distance from s to v.

Implementing the Dijkstra's algorithm

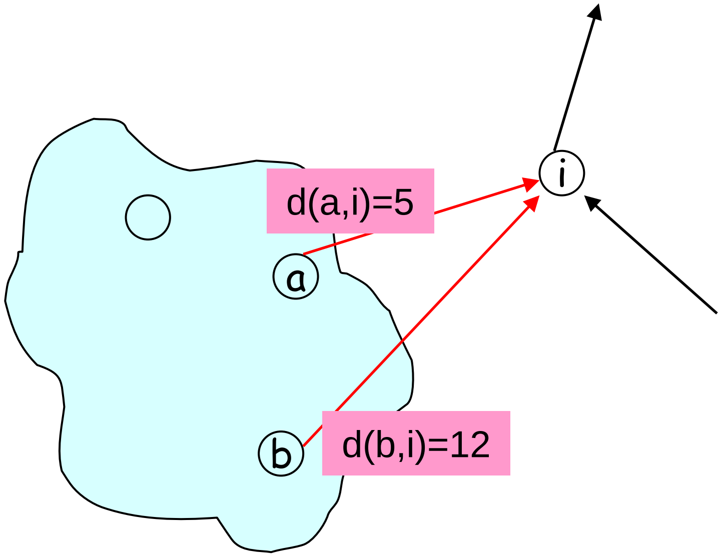
- Suppose that the n vertices are labeled by a unique integer in {0, 1, 2, …, n-1}, and vertex 0 is the source.

- The main data structure used in our implementation is an array **d[0..n-1]**. During each iteration, the array **d** will maintain the following invariant:
  - for any vertex **i** in the shortest path subtree P, we have

    $$d[i] = \delta(0, i).$$

  - for any vertex **i** not in P, **d[i]** is equal to the distance of the minimum outgoing edge of P that is incident to **i**.

Value of d[i] for vertex i $\notin$ P



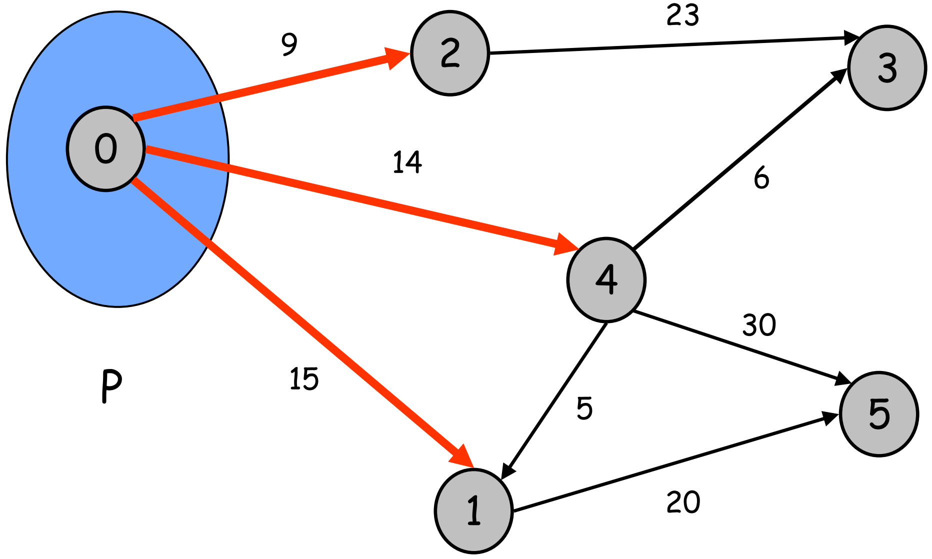
- Vertex i has two outgoing edges incident into it, namely (a,i) and (b,i).

- Since (a,i) has the smaller distance 5, we have d[i] = 5.

Value of d[i]: Example

Initially, P={0}, d=

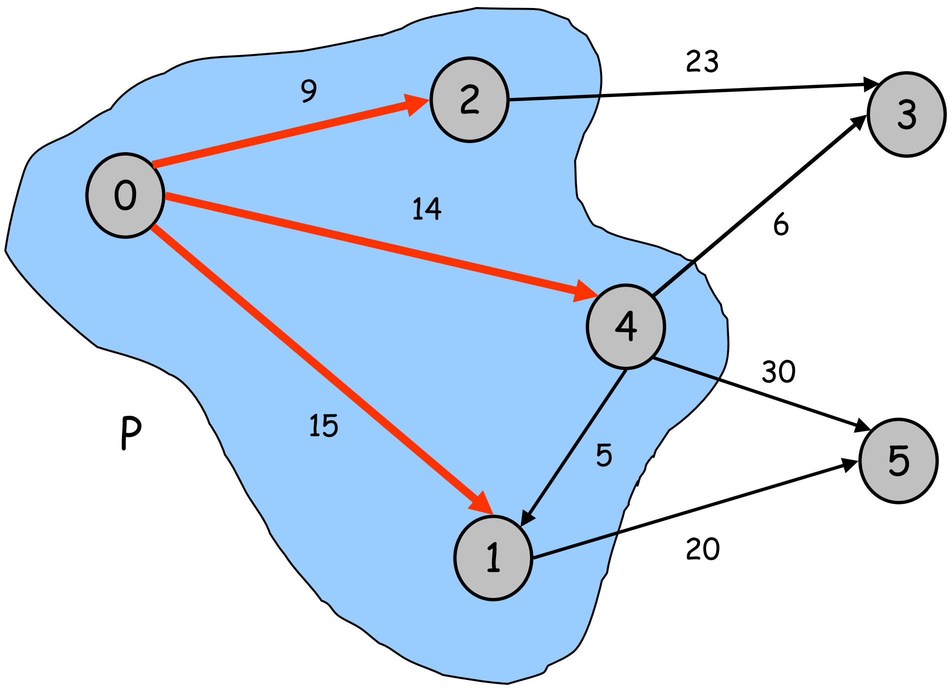
0	1	2	3	4	5
0	15	9	∞	14	∞



Value of d[i]: Example (cont')

In general (say, after 3 iterations), d=

0	1	2	3	4	5
0	15	9	20	14	35



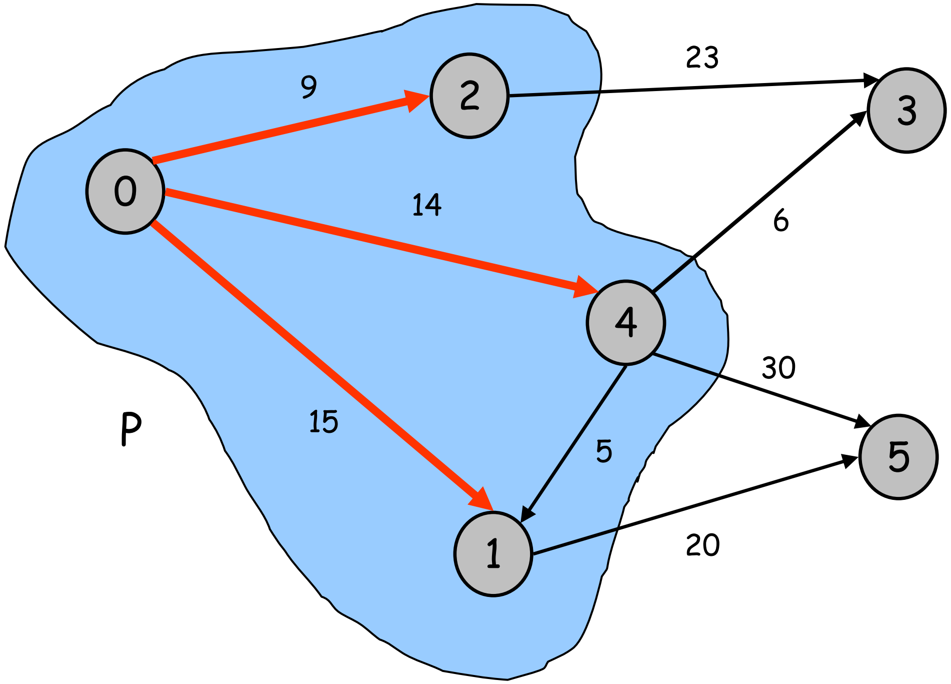
Question: How to find the minimum distance outgoing edge of P?

Ans: Scan the array d to find the vertex v such that
(i) v is not in P, and
(ii) d[v] has the smallest value.

Value of d[i]: Example (cont')

In general (say, after 3 iterations), d=

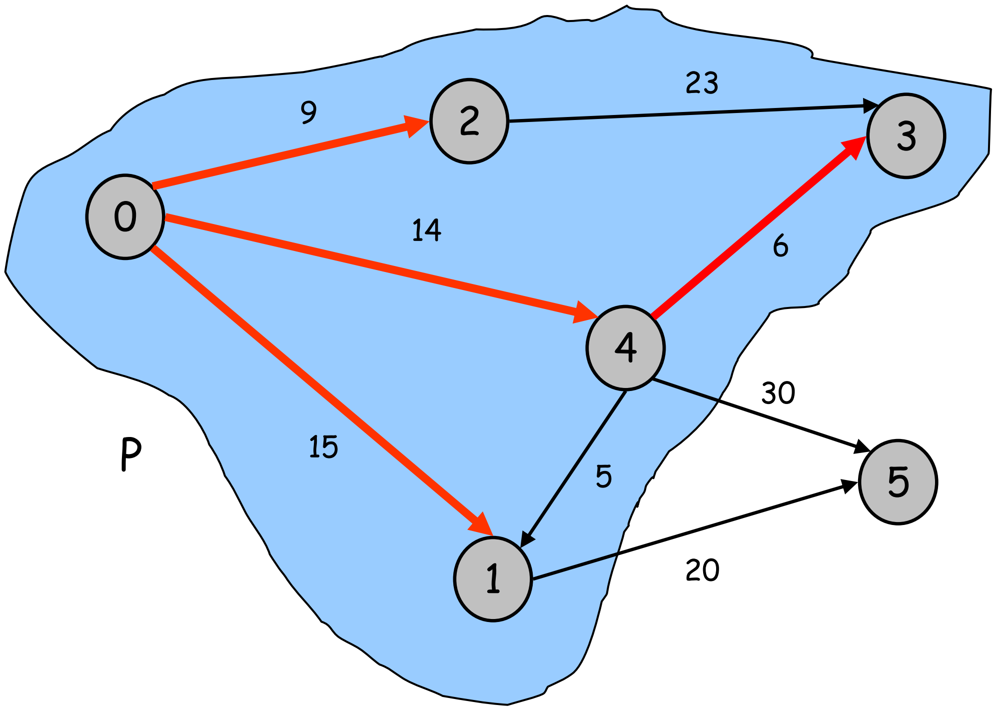
0	1	2	3	4	5
0	15	9	20	14	35



Value of d[i]: Example (cont')

In general (say, after 3 iterations), d=

0	1	2	3	4	5
0	15	9	20	14	35

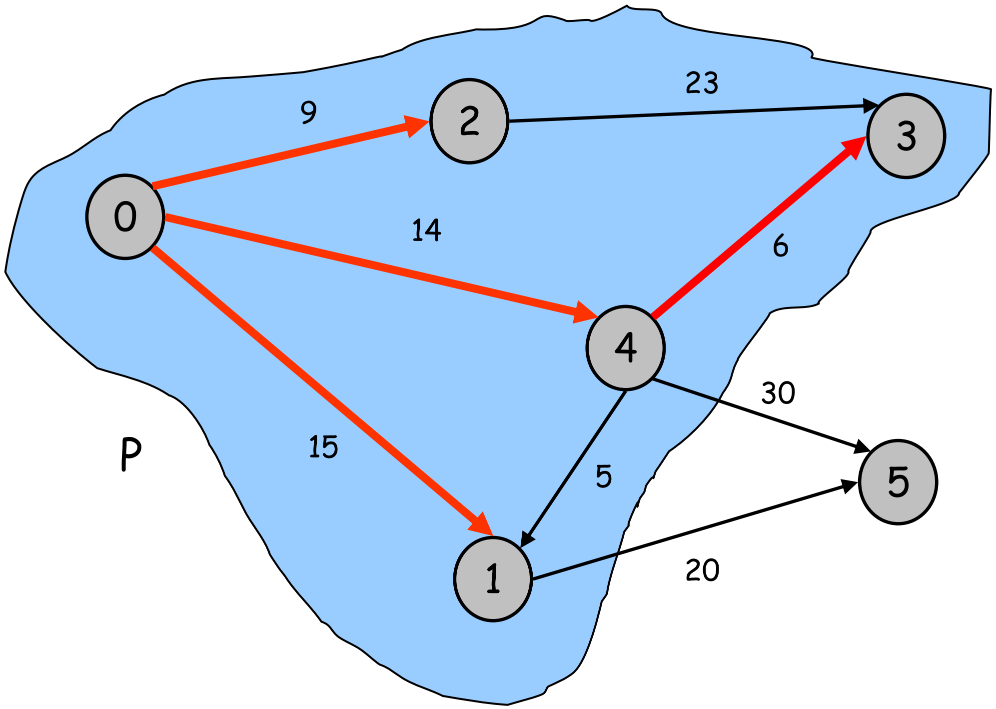


From our correctness proof, we can conclude that d(4,3) = δ(0,3), the distance of the shortest path from the source 0 to 3.

Value of d[i]: Example (cont')

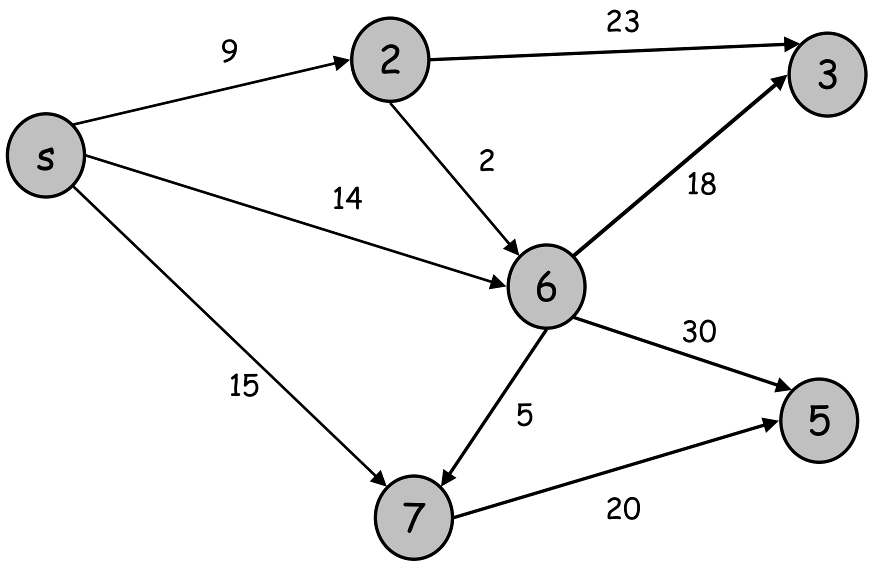
In general (say, after 3 iterations), d=

0	1	2	3	4	5
0	15	9	20	14	35



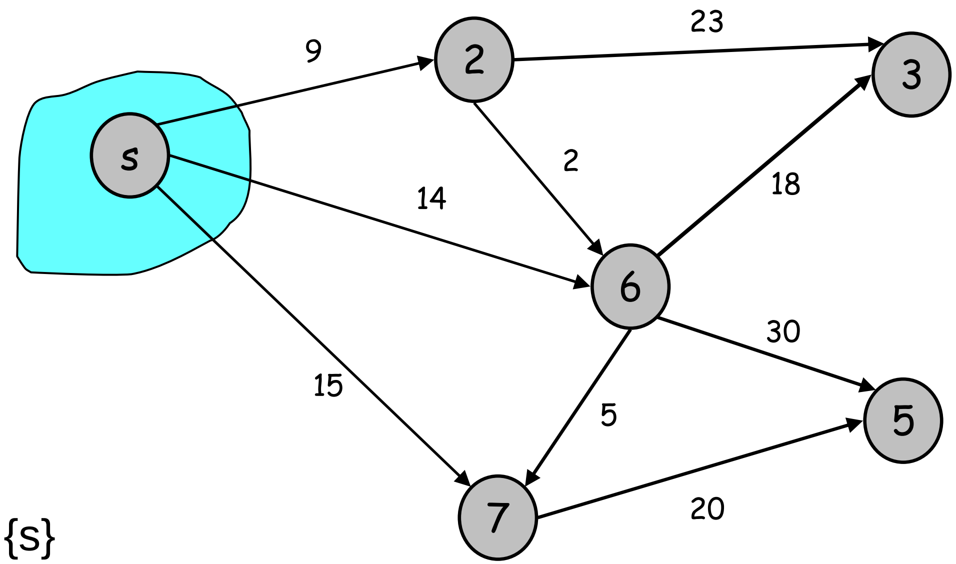
Thus, when we add a vertex v into P, we have d[v] = $\delta(0,v)$; this maintains the first requirement for d.

Dijkstra's algorithm: Sample run



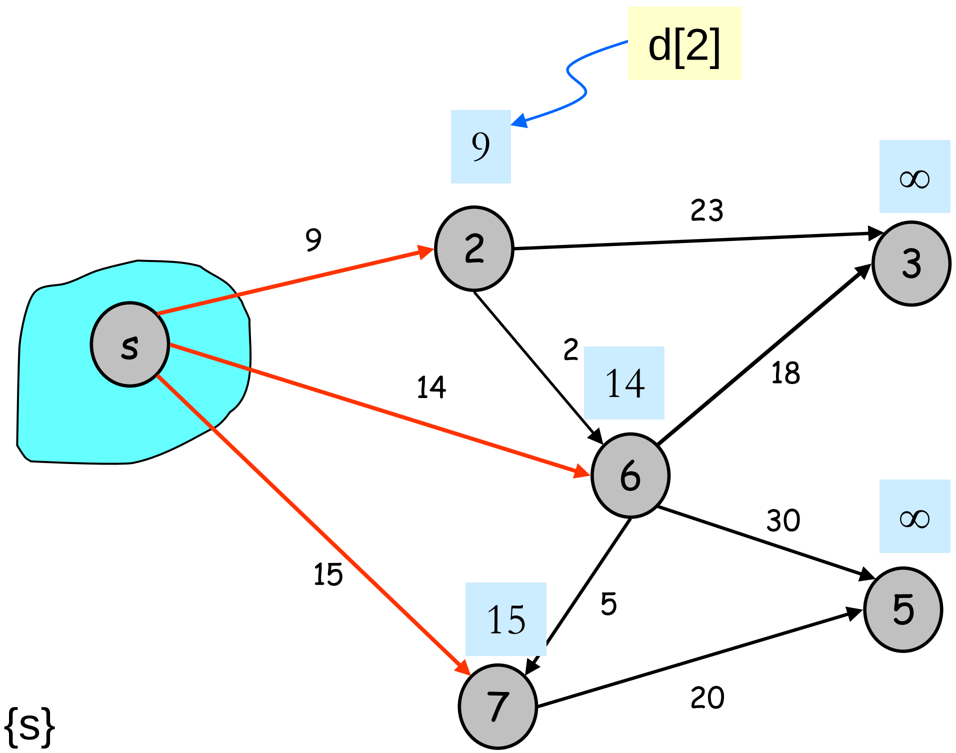
P = {}

Dijkstra's algorithm: Sample run (Iteration 1)



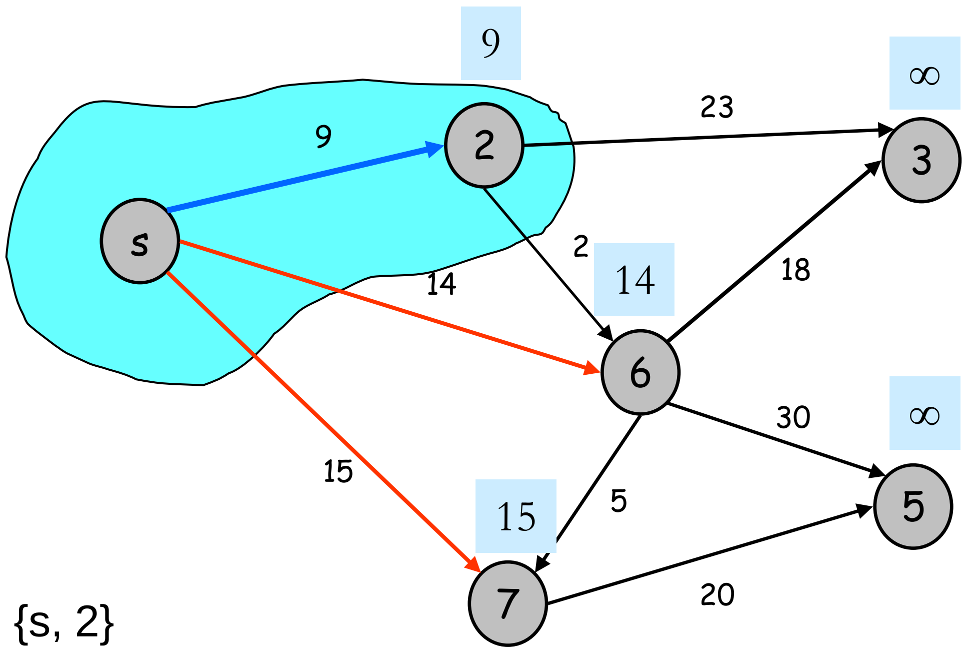
P = {s}

Dijkstra's algorithm: Sample run (Iteration 1)



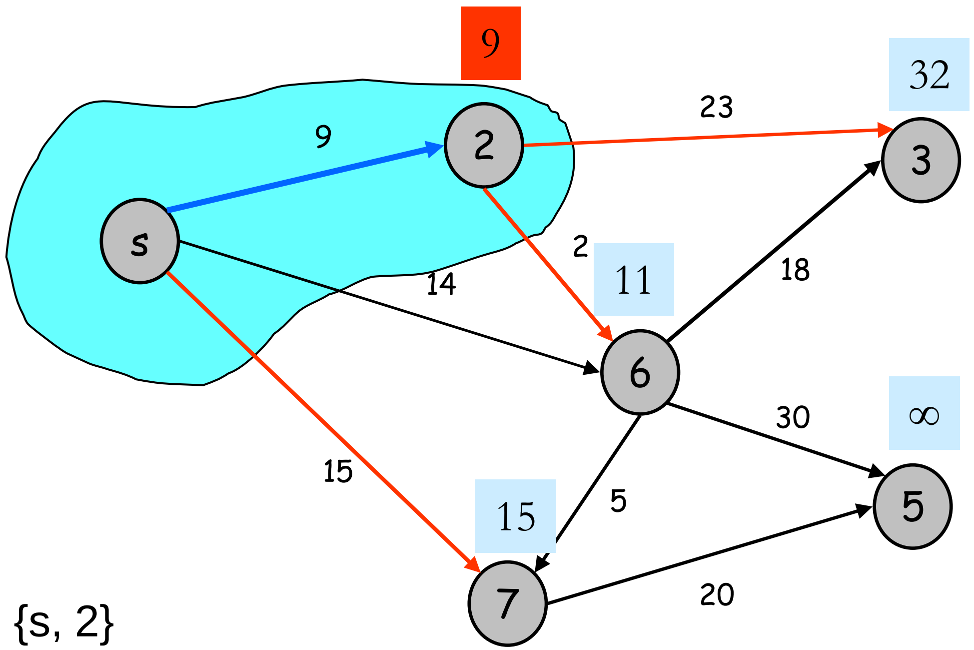
P = {s}

Dijkstra’s algorithm: Sample run (Iteration 2)



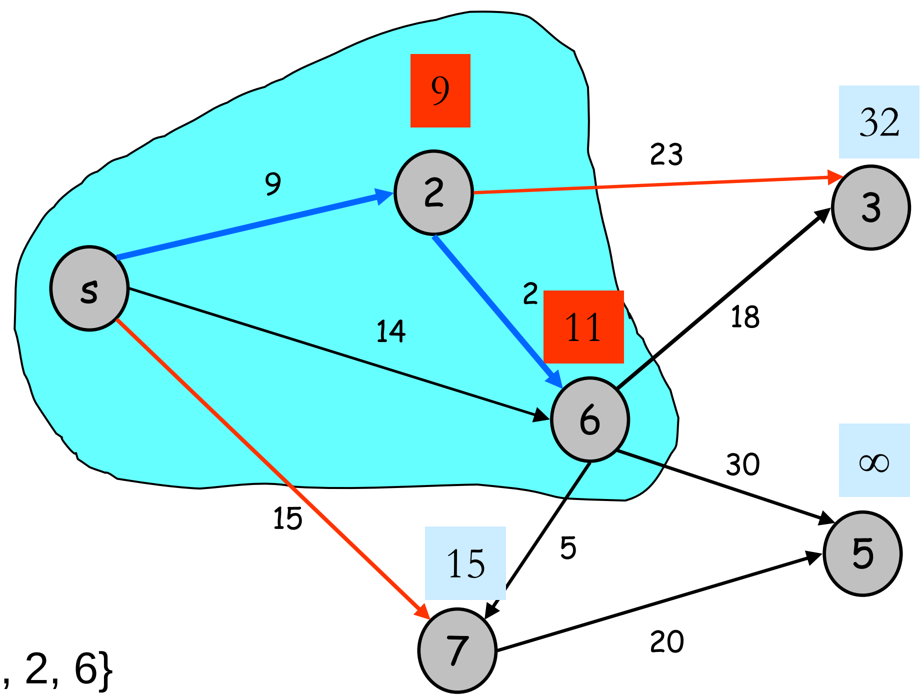
P = {s, 2}

Dijkstra's algorithm: Sample run (Iteration 2)



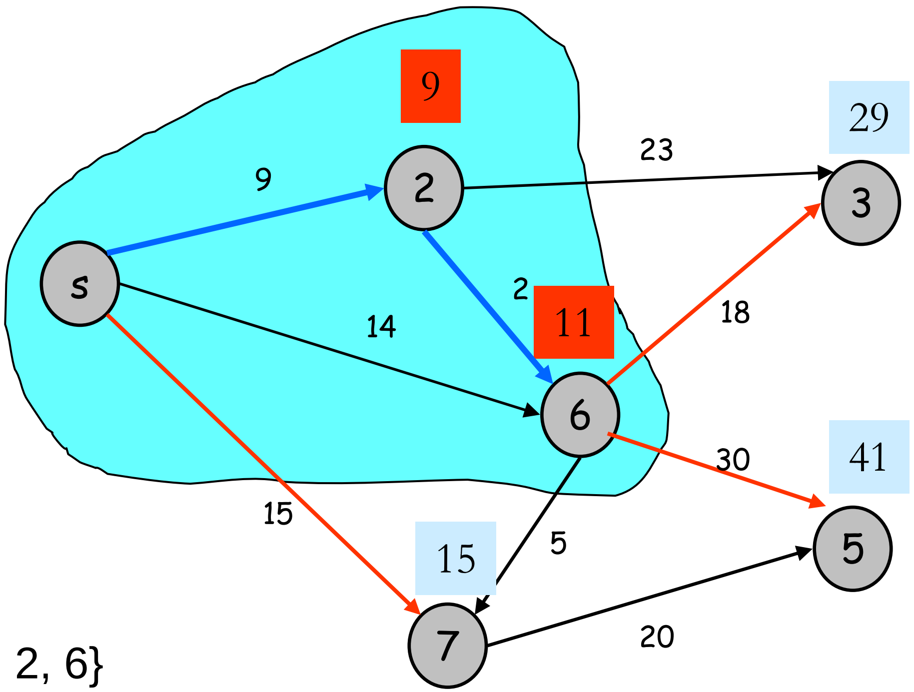
P = {s, 2}

Dijkstra’s algorithm: Sample run (Iteration 3)



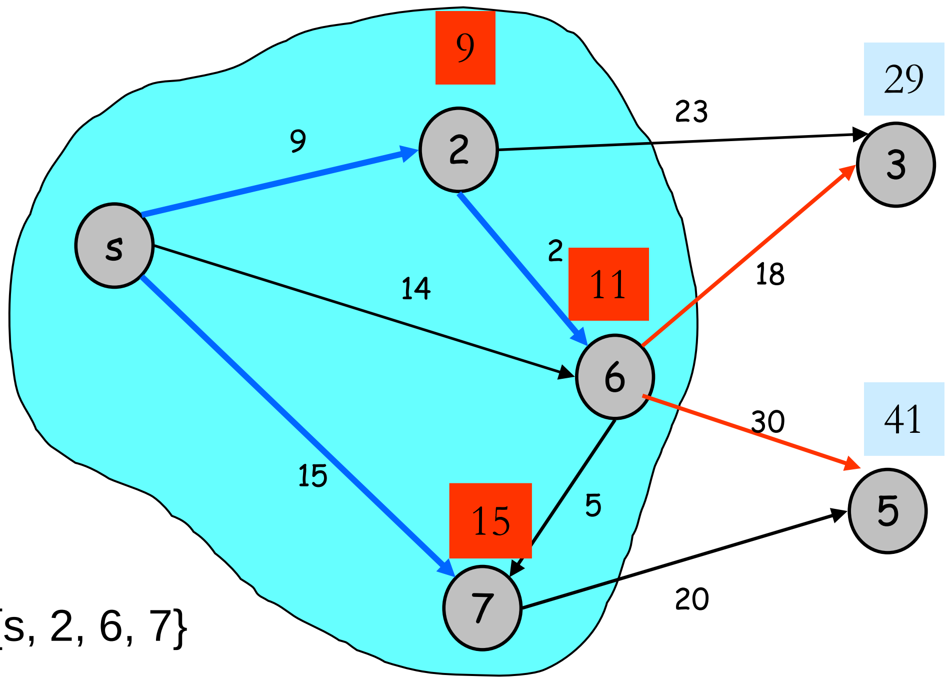
P = {s, 2, 6}

Dijkstra's algorithm: Sample run (Iteration 3)



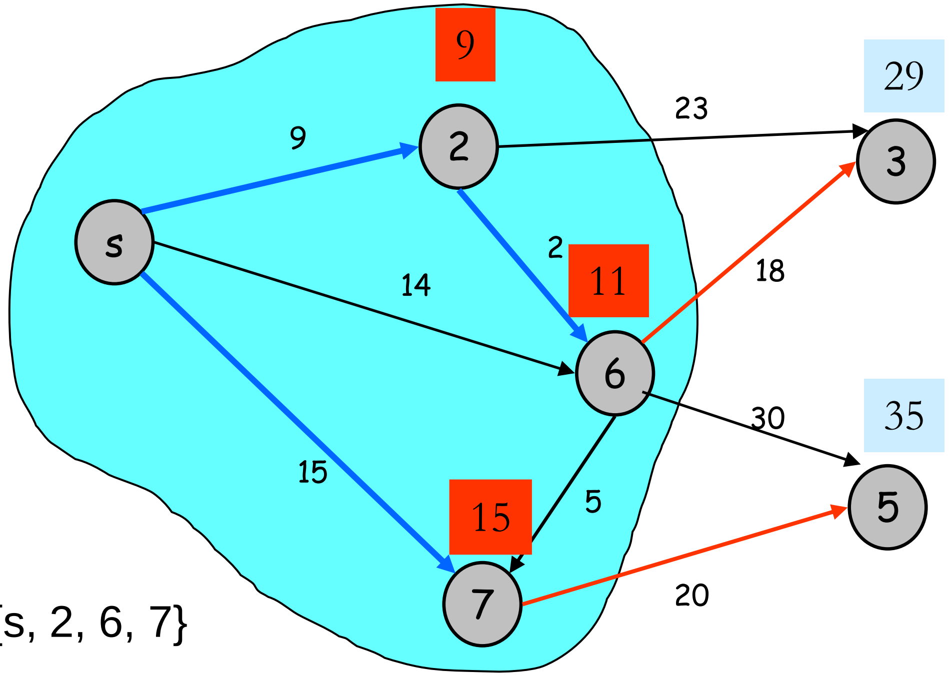
P = {s, 2, 6}

Dijkstra's algorithm: Sample run (Iteration 4)



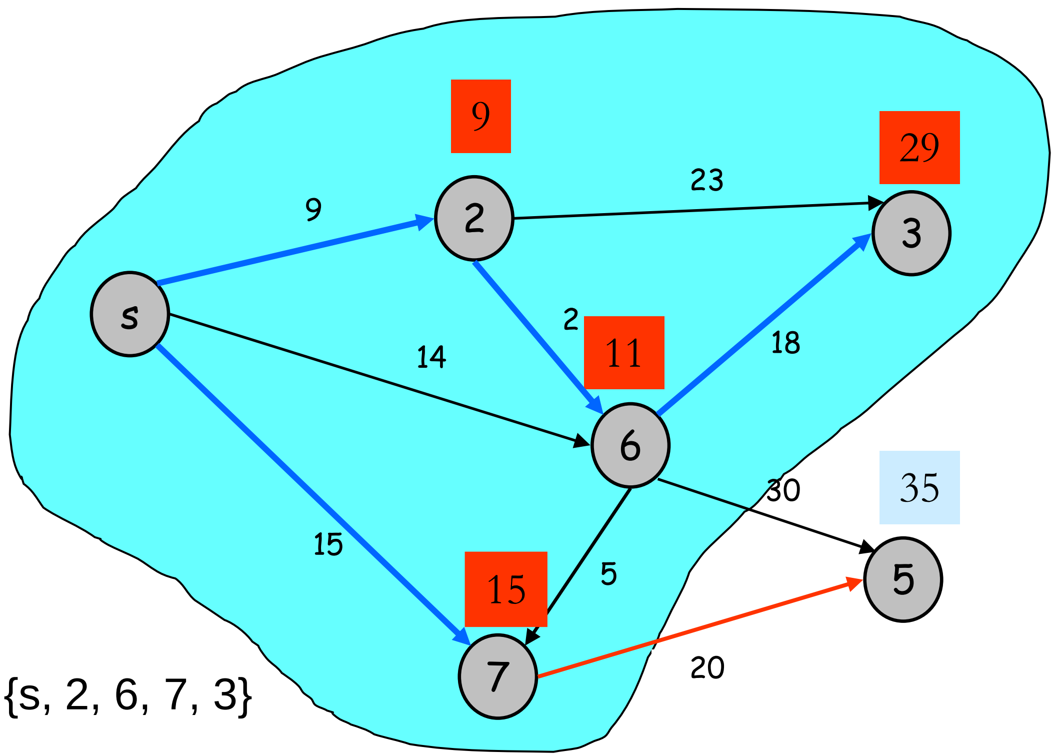
P = {s, 2, 6, 7}

Dijkstra's algorithm: Sample run (Iteration 4)



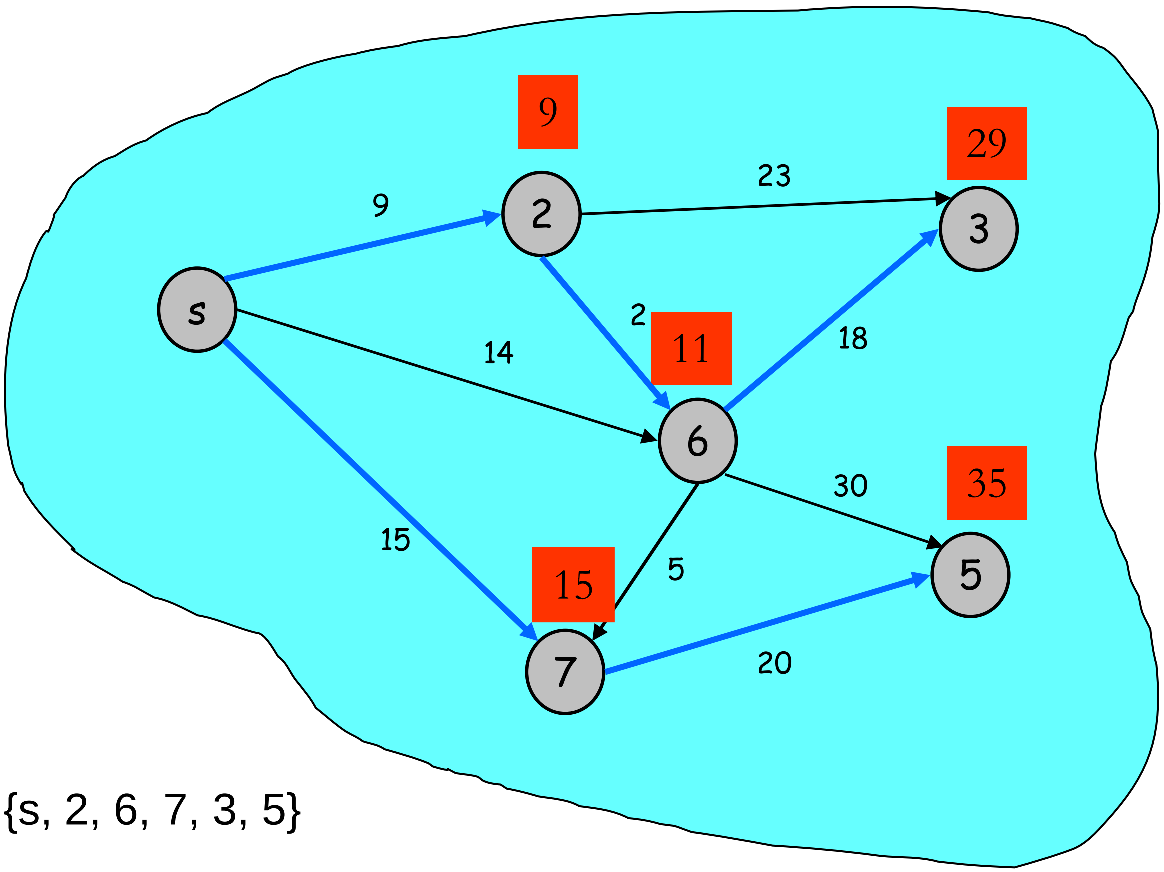
P = {s, 2, 6, 7}

Dijkstra's algorithm: Sample run (Iteration 5)



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

Dijkstra’s algorithm: Sample run (Iteration 6)



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

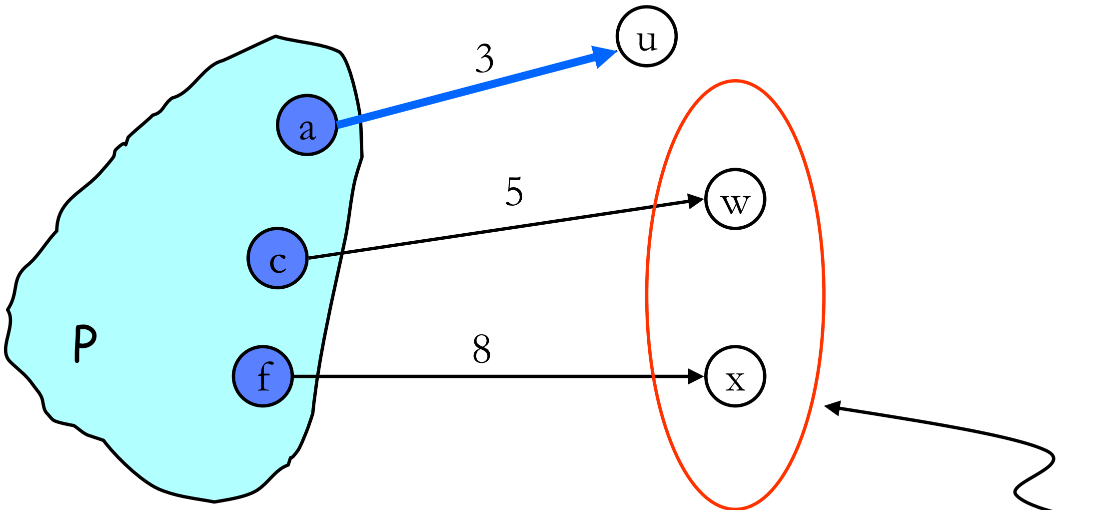
Dijkstra's algorithm: Time complexity

- Note that if we update the array in brute-force (i.e., scan the adjacency list of every vertex not in P once to find the minimum outgoing edge incident into it), it takes O(E) time.

- Then, the total time complexity is O(VE) because we have |V|-1 iterations, and for each iteration,
  - it takes O(V) time to scan the array d for finding the minimum outgoing edge, and
  - it takes O(E) time to update the array.

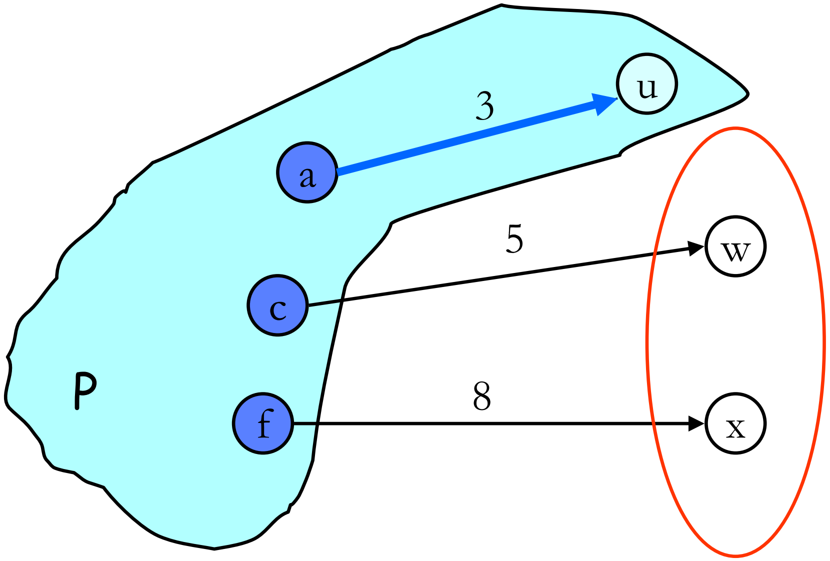
- Hence, the total time is O(V(V+E))=O(VE) time (because we assume the graph is connected and this implies |E| ≥ |V|-1).

How to do the update faster?



These vertices are not neighbors of u.

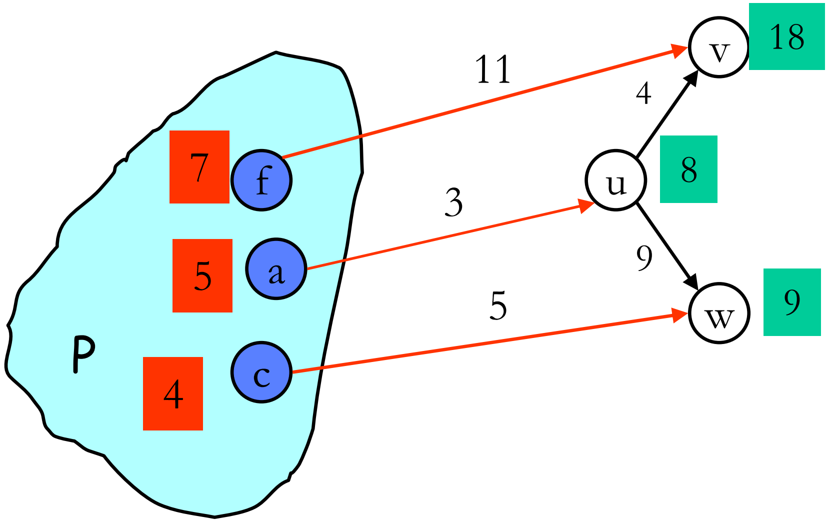
How to do the update faster? (cont’)



Hence, no change after adding u to P

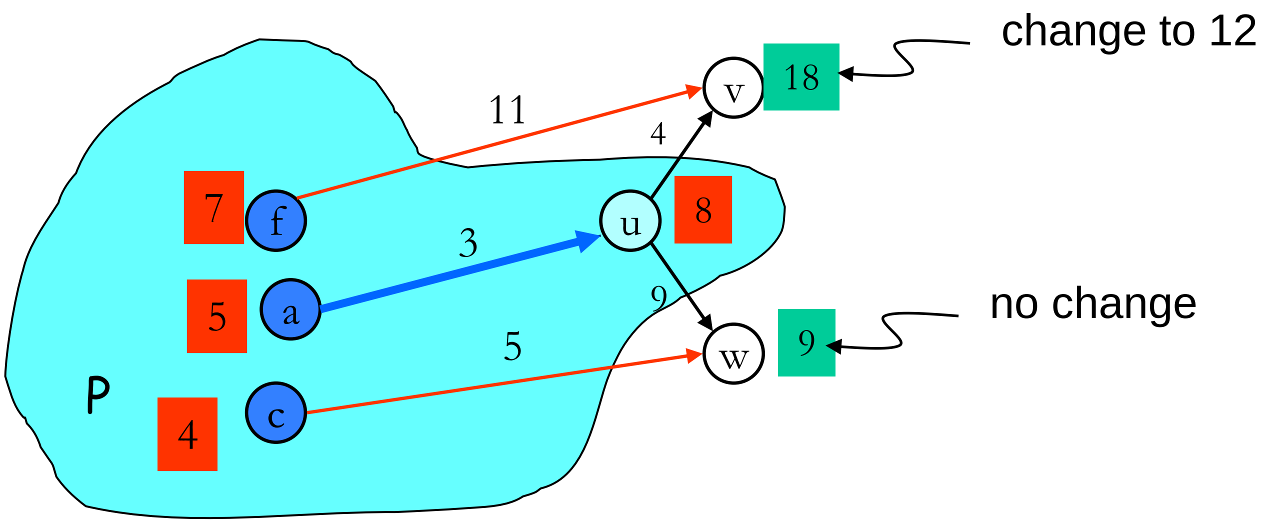
How to do the update faster? (cont’)

Thus, we only need to consider the neighbors of u.



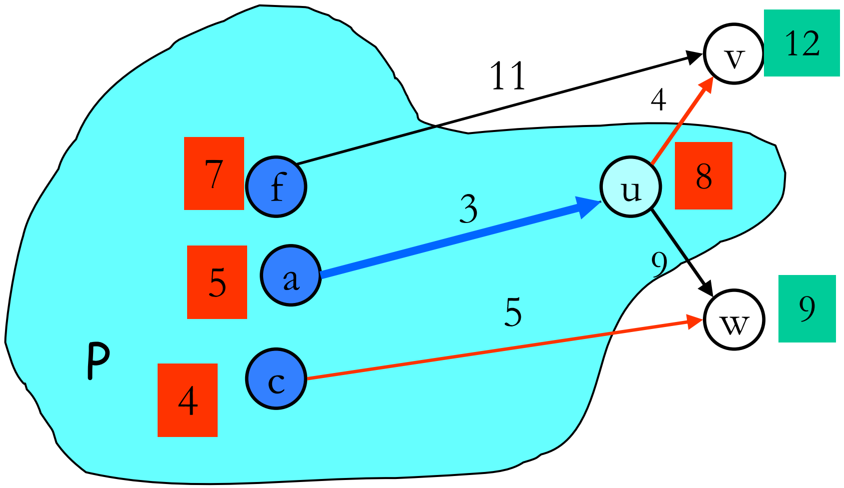
How to do the update faster? (cont')

Thus, we only need to consider the neighbors of u.



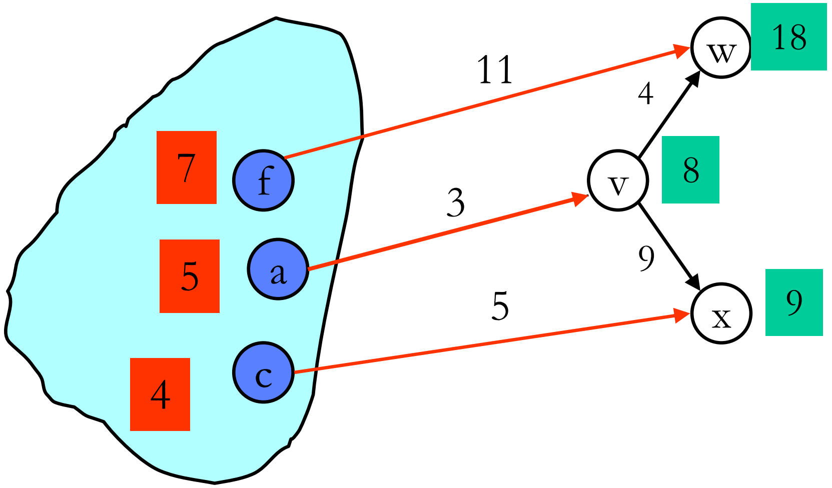
How to do the update faster? (cont’)

Thus, we only need to consider the neighbors of u.



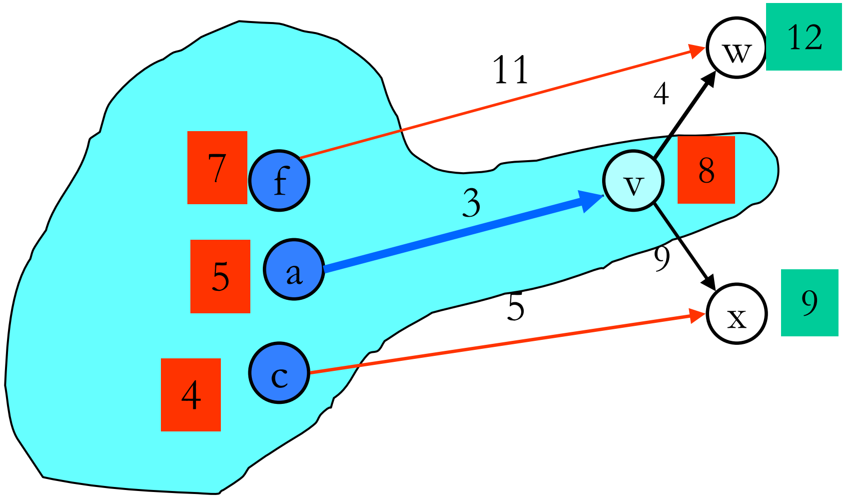
A useful procedure: Relax

```
Relax(v,w): # for each (v,w)
  if d[w]>d[v]+w(v,w):
    d[w]=d[v]+w(v,w)
```



A useful procedure: Relax (cont')

```
Relax(v,w): # for each (v,w)
  if d[w]>d[v]+w(v,w):
    d[w]=d[v]+w(v,w)
```



`Relax(v,w)`

`Relax(v,x)` ➔ no change

Dijkstra’s algorithm & Time complexity

```
Relax(v,w): # for each (v,w)
  if d[w]>d[v]+w(v,w):
    d[w]=d[v]+w(v,w)
```

```
Dijkstra(G=(V,E,w), s):
  P = {}
  Initialize array d[i] = ∞ for all vertices i
  d[s] = 0
  while |P| < |V|:
    find the vertex v ∈ V-P with the smallest d[v]
    P = P ∪ {v}
    for every neighbor w of vertex v:
      if w ∉ P:
        Relax(v,w)
```

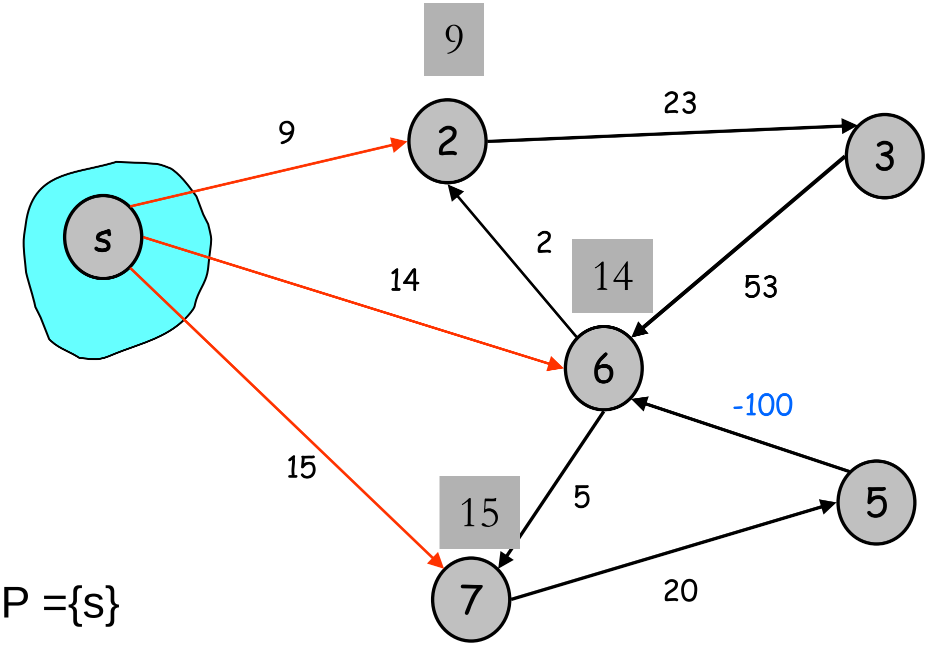
1. Finding v with smallest d[v]: O(V) time

   $\Rightarrow$Total time complexity of finding v in all |V| iterations: $O(V^2)$ time
2. Total time complexity of performing Relax: O(E) time

- **Time complexity: $O(E + V^2)$ time**
- ***Note:*** If you know *Fibonacci Heap*, the time complexity can be reduced to ***O(E + V log V)***.

Remark: No negative weight is allowed

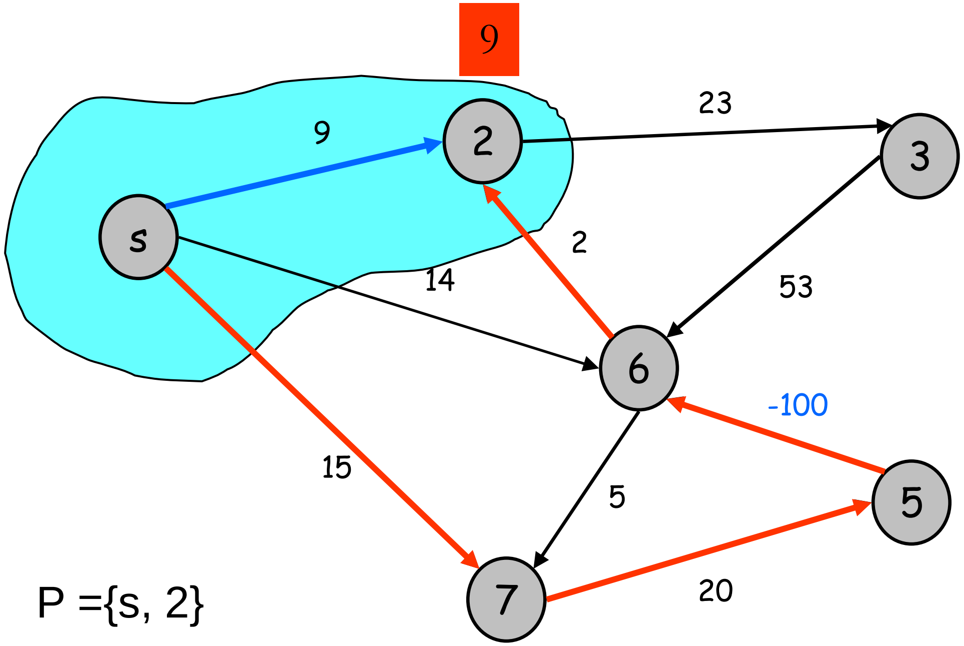
- Note that the correctness of Dijkstra's depends on the fact that no edge has negative weight.



P ={s}

Remark: No negative weight is allowed (cont')

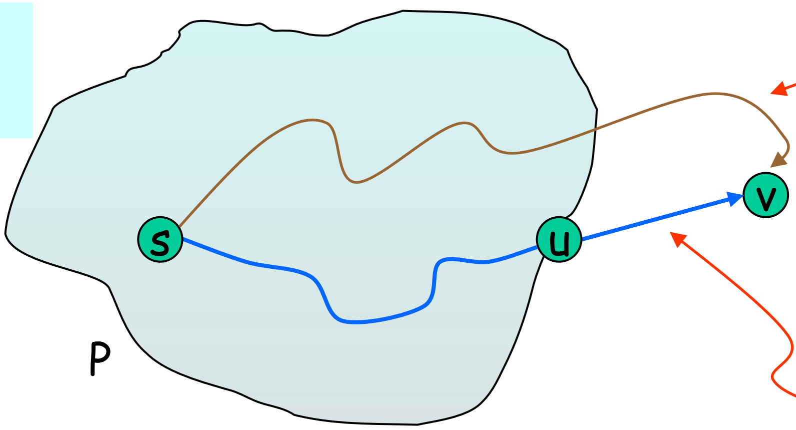
- Note that the correctness of Dijkstra's depends on the fact that no edge has negative weight.



Remark: No negative weight is allowed (cont')

- Note that the correctness of Dijkstra's depends on the fact that no edge has negative weight.

Proof of correctness given in Slide 24



σ: any path from s to v.

length of the path $= \delta(s,u) + w(u,v) = d(u,v)$.

$$w(\sigma) = \underbrace{w(s\ p_0\ p_1 \ldots p_m\ q_1 \ldots q_k\ v) \geq w(s\ p_0 \ldots p_m\ q_1)}_{}$$

No longer true if the thrown edges have –ve weight