Data Structures & Algorithms

Adil M. Khan

Professor of Computer Science

Innopolis University
a.khan@innopolis.ru

Recap

1. Optimization Problems
2. Greedy Algorithms
3. Minimum Spanning Tree

- MST in Unweighted Graphs
- MST in Weighted Graphs (Prim’s Algorithm, and Krushkal’s Algorithm)

Objectives

1. Shortest Path Applications
2. Formalize the problem
3. Variants of Shortest Path Problems
4. Dijkstra’s Algorithm
5. Bellman Ford and Floyd Warshall Algorithms

Shortest Path

- Given a weighted graph G and two vertices u and v, we want to find a path of minimum total weight between u and v.

- Applications

  - Internet packet routing

  - Flight reservations

  - Driving directions

Shortest Path Problem

- In shortest path problems, we are given a weighted graph $G = (V, E)$, with weight function $w\colon E \; -> \; R$

- The weight $w(p)$ of a path $p = (v_0, v_1, \ldots, v_k)$ is

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

Shortest Path Problem

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

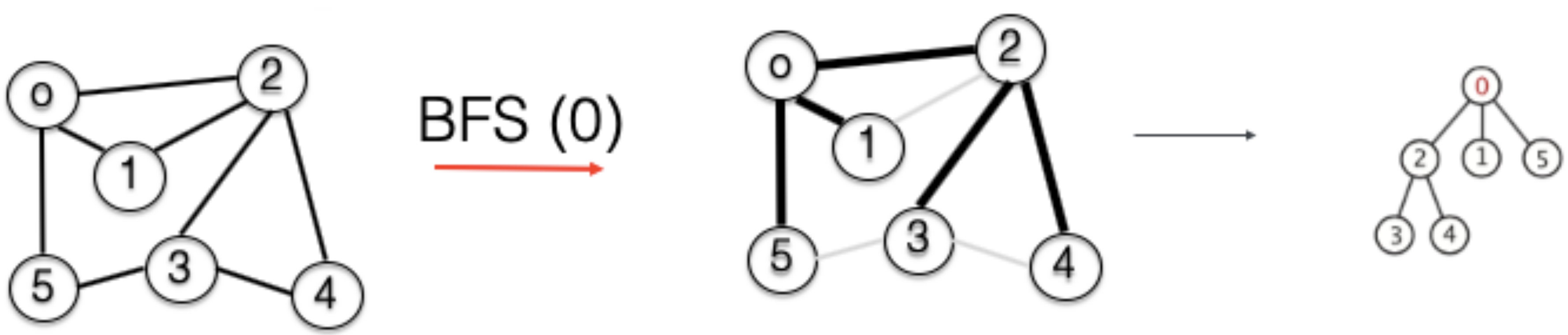
- **Shortest path weight** from u to v is then given as

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

- A shortest path from u to v is then defined as any path p with weight $w(p) = \delta(u,v)$

Shortest Path Algorithms
Unweighted Graphs

- You have already learned an algorithm which can find such a path
- Breadth First Search



Shortest Path Algorithms
Weighted Graphs

- Single-source shortest path
- All-pair shortest path

Shortest Path Algorithms
Weighted Graphs

- Single-source shortest path problems:
  - "Given a graph $G = (V, E)$, we want to find a shortest path from a given source vertex $s \in V$ to each vertex $v \in V$"
- Dijkstra's Algorithm
- Bellman-Ford Algorithm

Shortest Path Algorithms
Weighted Graphs

- All-pairs shortest path problems:
  - "Given a graph $G = (V, E)$, we want to find shortest path between all pairs of vertices in G"
- Floyd-Warshall

Dijkstra's Algorithm

- Finds the shortest path from a given a vertex s to every other vertex in G

- Works on the same idea as the Prim's algorithm, with a small difference

Recall: Prim's Algorithm

Algorithm PrimJarnik(G):

Input: An undirected, weighted, connected graph G with n vertices and m edges

Output: A minimum spanning tree T for G

Pick any vertex s of G
$D[s] = 0$
for each vertex $v \neq s$ **do**
  $D[v] = \infty$
Initialize $T = \emptyset$.
Initialize a priority queue Q with an entry $(D[v], (v, \mathsf{None}))$ for each vertex v, where $D[v]$ is the key in the priority queue, and (v, None) is the associated value.
while Q is not empty **do**
  (u, e) = value returned by Q.remove_min()
  Connect vertex u to T using edge e.
  for each edge $e' = (u, v)$ such that v is in Q **do**
    {check if edge (u, v) better connects v to T}
    if $w(u, v) < D[v]$ **then**
      $D[v] = w(u, v)$
      Change the key of vertex v in Q to $D[v]$.
      Change the value of vertex v in Q to (v, e').
return the tree T

What's the Similarity with Prim's Algo?

- We grow a "tree" of vertices, beginning with $\boldsymbol{s}$ and eventually covering all the vertices

- We store (in a PQ) with each vertex $\boldsymbol{v}$ a key $\boldsymbol{d(v)}$ representing the distance of $\boldsymbol{v}$ from $\boldsymbol{s}$

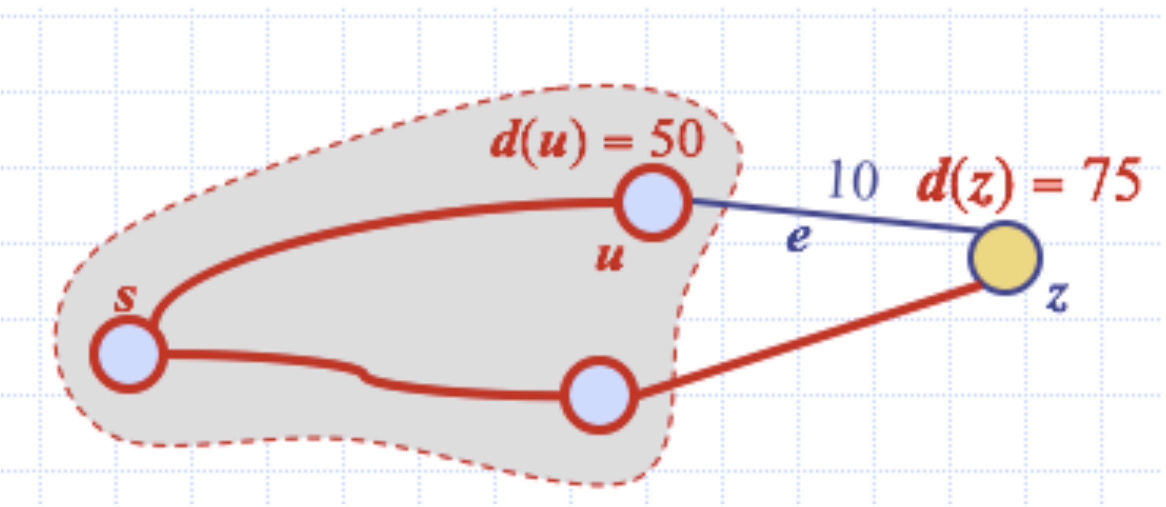
- At each step

  - We add to the tree the vertex $\boldsymbol{u}$ outside the tree with the smallest distance key, $\boldsymbol{d(u)}$

  - We update the keys of the vertices adjacent to $\boldsymbol{u}$

What's the difference?

- Consider an edge $e = (u, z)$ such that
  - u is the vertex most recently added to the tree
  - z is not in the tree



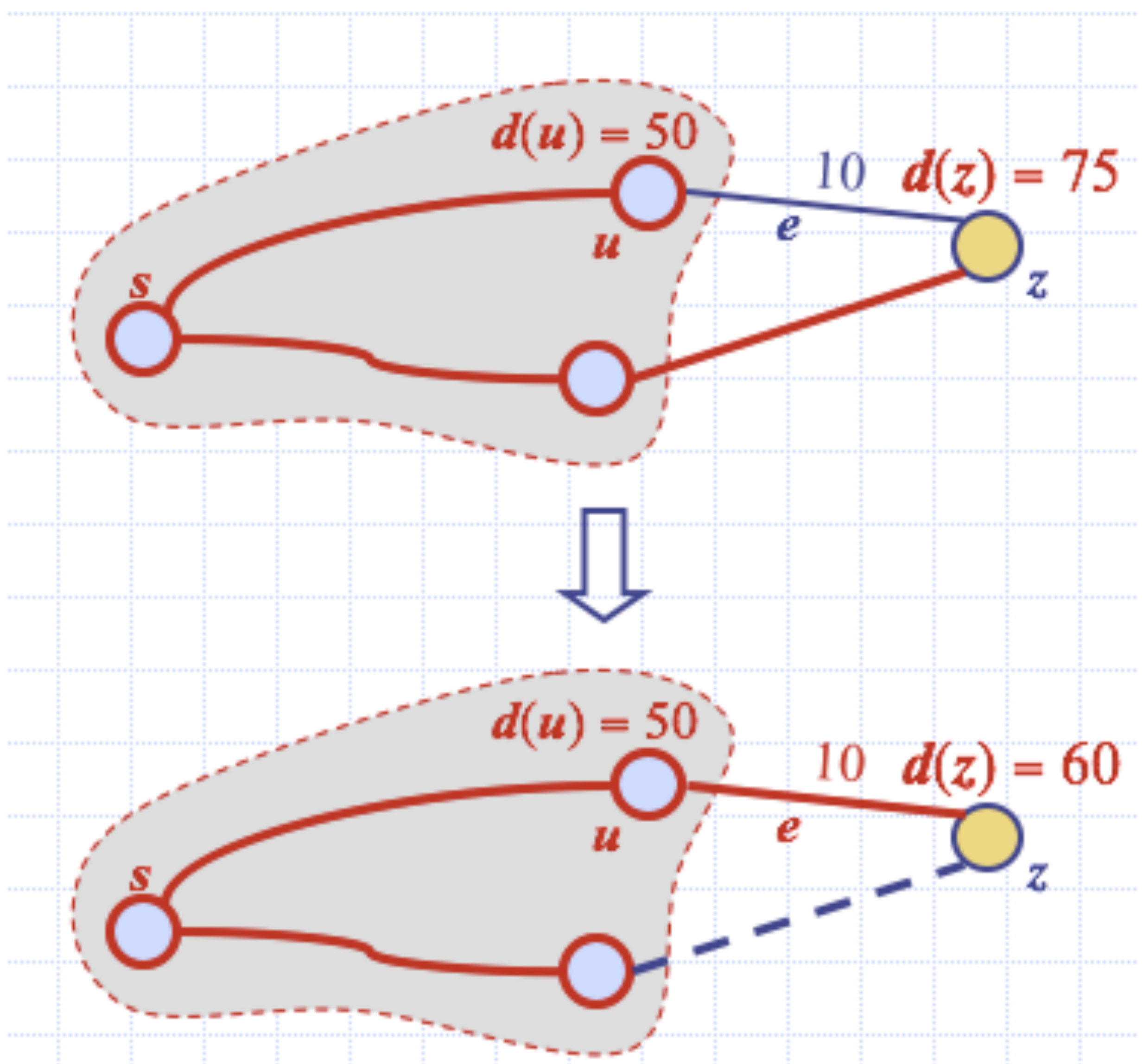
- The relaxation of edge e updates distance $d(z)$ as follows:

$$d(z) = \min\{d(z), d(u) + weight(e)\}$$

What's the difference?

- Consider an edge $\boldsymbol{e} = (\boldsymbol{u}, \boldsymbol{z})$ such that
  - $\boldsymbol{u}$ is the vertex most recently added to the tree
  - $\boldsymbol{z}$ is not in the tree
- The relaxation of edge $\boldsymbol{e}$ updates distance $\boldsymbol{d}(\boldsymbol{z})$ as follows:

$$d(z) = \min\{d(z), d(u) + weight(e)\}$$



Pseudocode

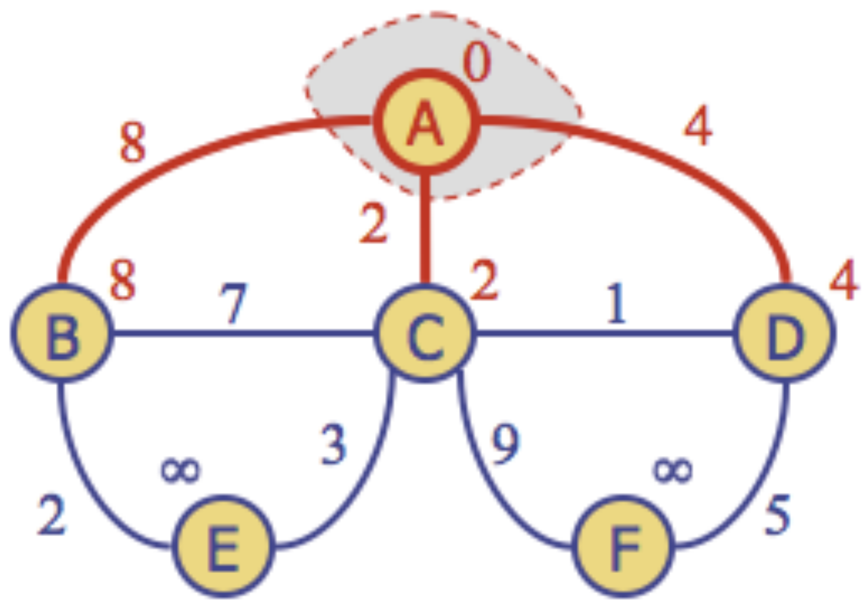
Algorithm $\mathsf{ShortestPath}(G,s)$:

Input: A weighted graph G with nonnegative edge weights, and a distinguished vertex s of G.

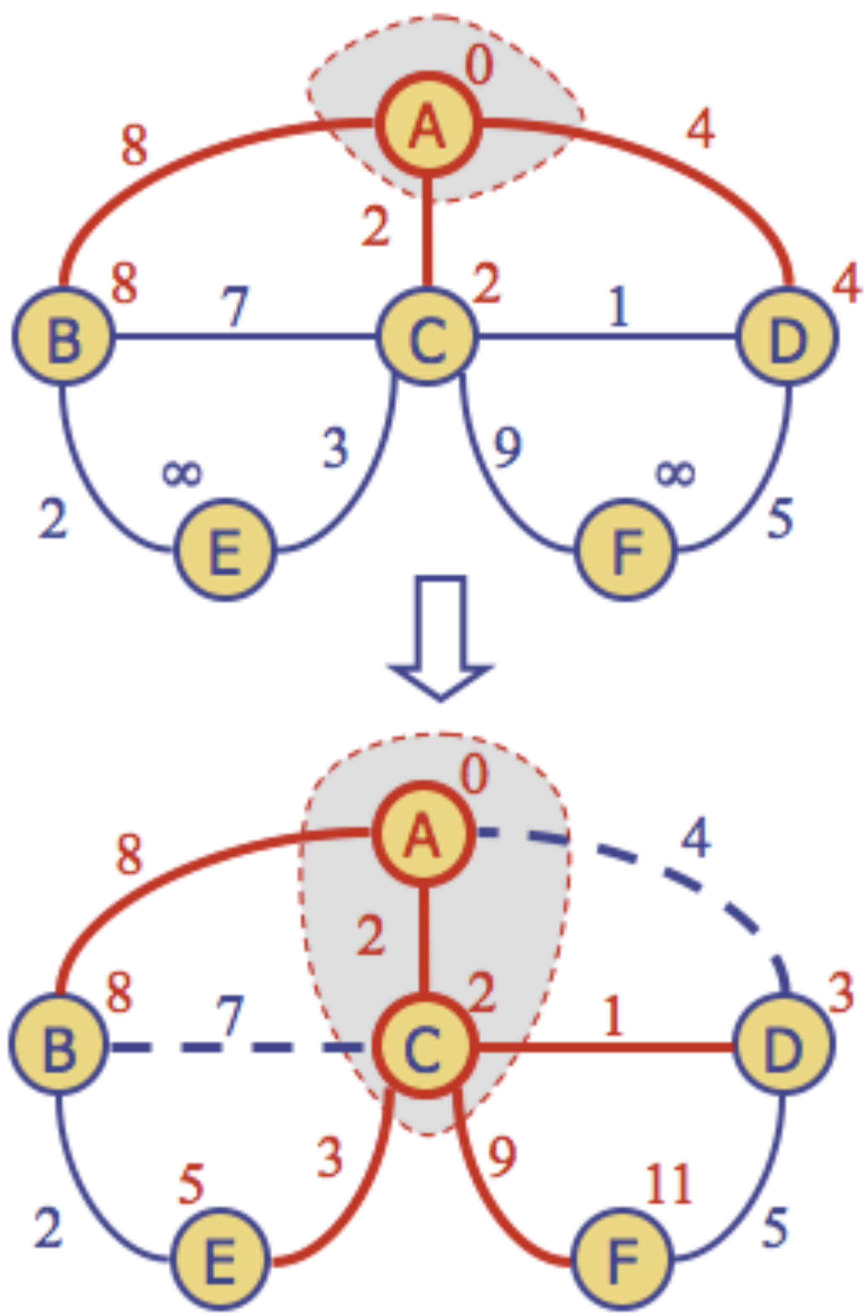
Output: The length of a shortest path from s to v for each vertex v of G.

Initialize $D[s] = 0$ and $D[v] = \infty$ for each vertex $v \neq s$.
Let a priority queue Q contain all the vertices of G using the D labels as keys.
while Q is not empty **do**
  {pull a new vertex u into the cloud}
  u = value returned by Q.remove_min()
  for each vertex v adjacent to u such that v is in Q **do**
    {perform the ***relaxation*** procedure on edge (u,v)}
    if $D[u] + w(u,v) < D[v]$ **then**
      $D[v] = D[u] + w(u,v)$
      Change to $D[v]$ the key of vertex v in Q.
return the label $D[v]$ of each vertex v

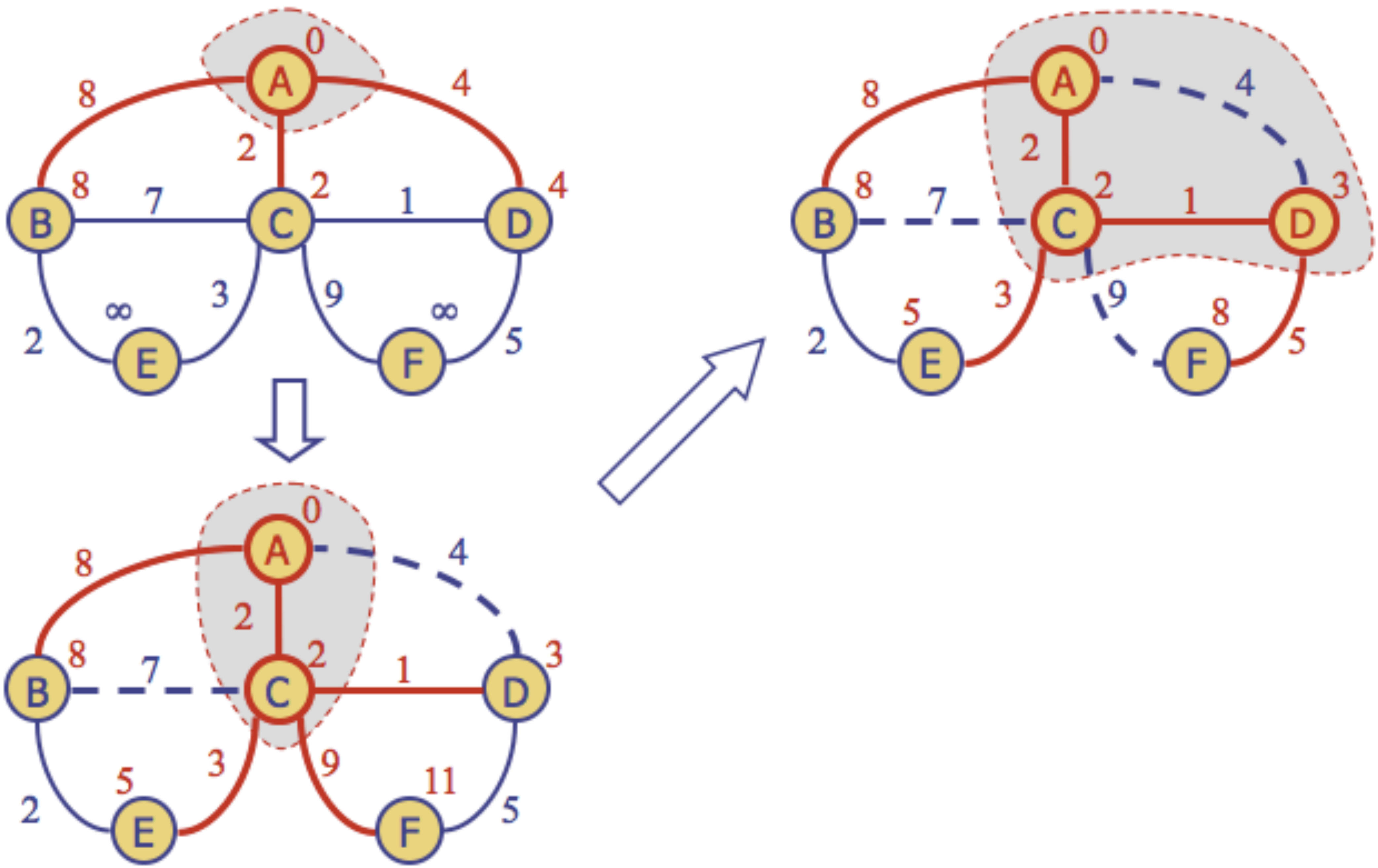
Example



Example

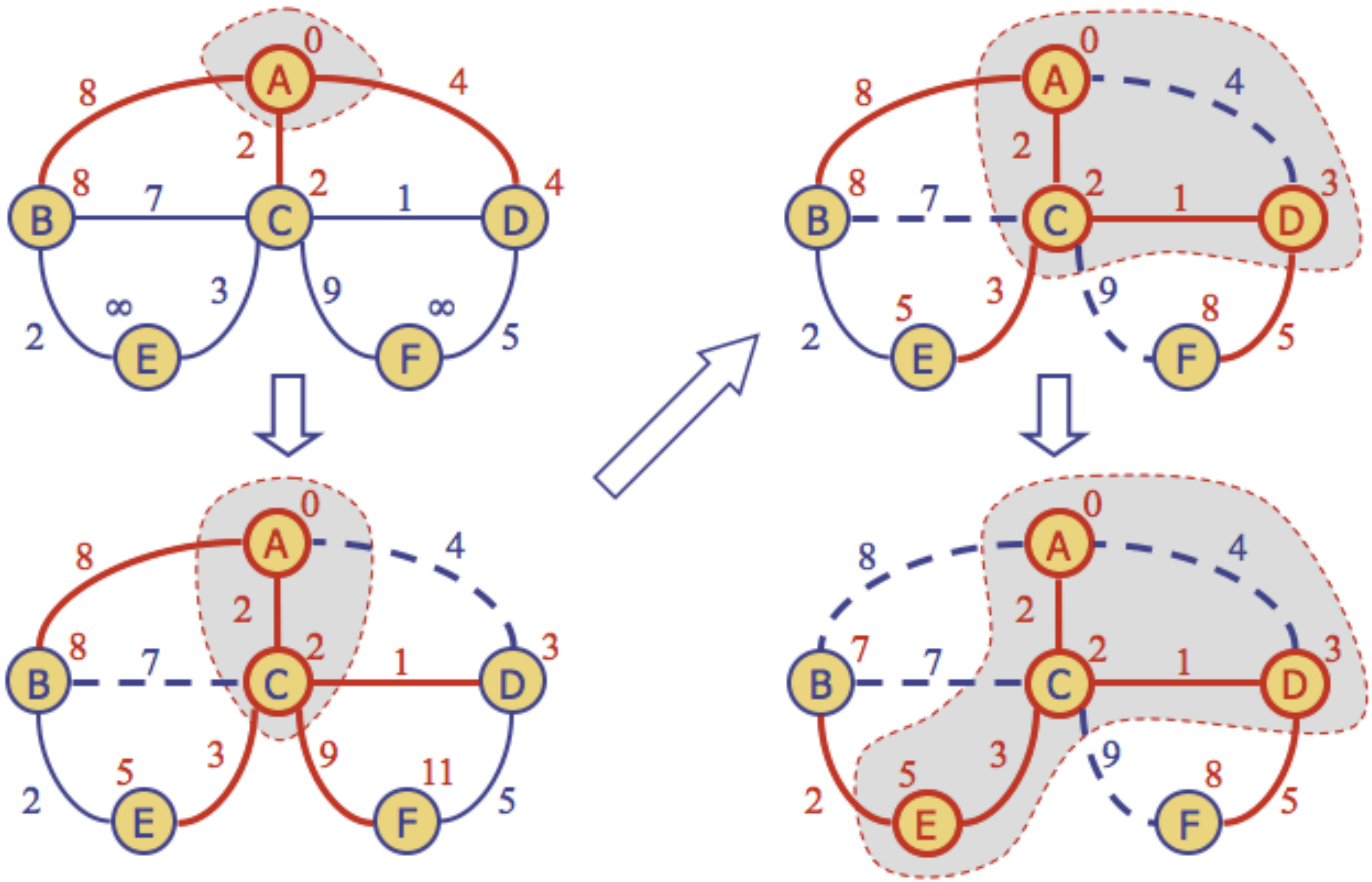


Example



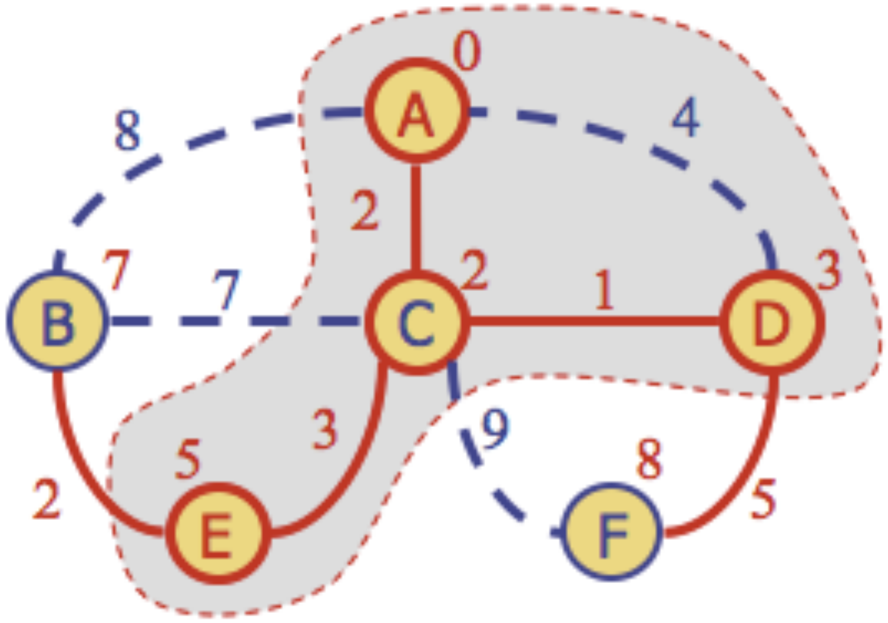
0
A
8
4
2
B
8
7
C
2
1
D
4
∞
3
9
∞
2
5
E
F
0
A
8
4
2
B
8
7
C
2
1
D
3
5
3
9
11
2
5
E
F
0
A
8
4
2
B
8
7
C
2
1
D
3
5
3
9
8
2
5
E
F

Example

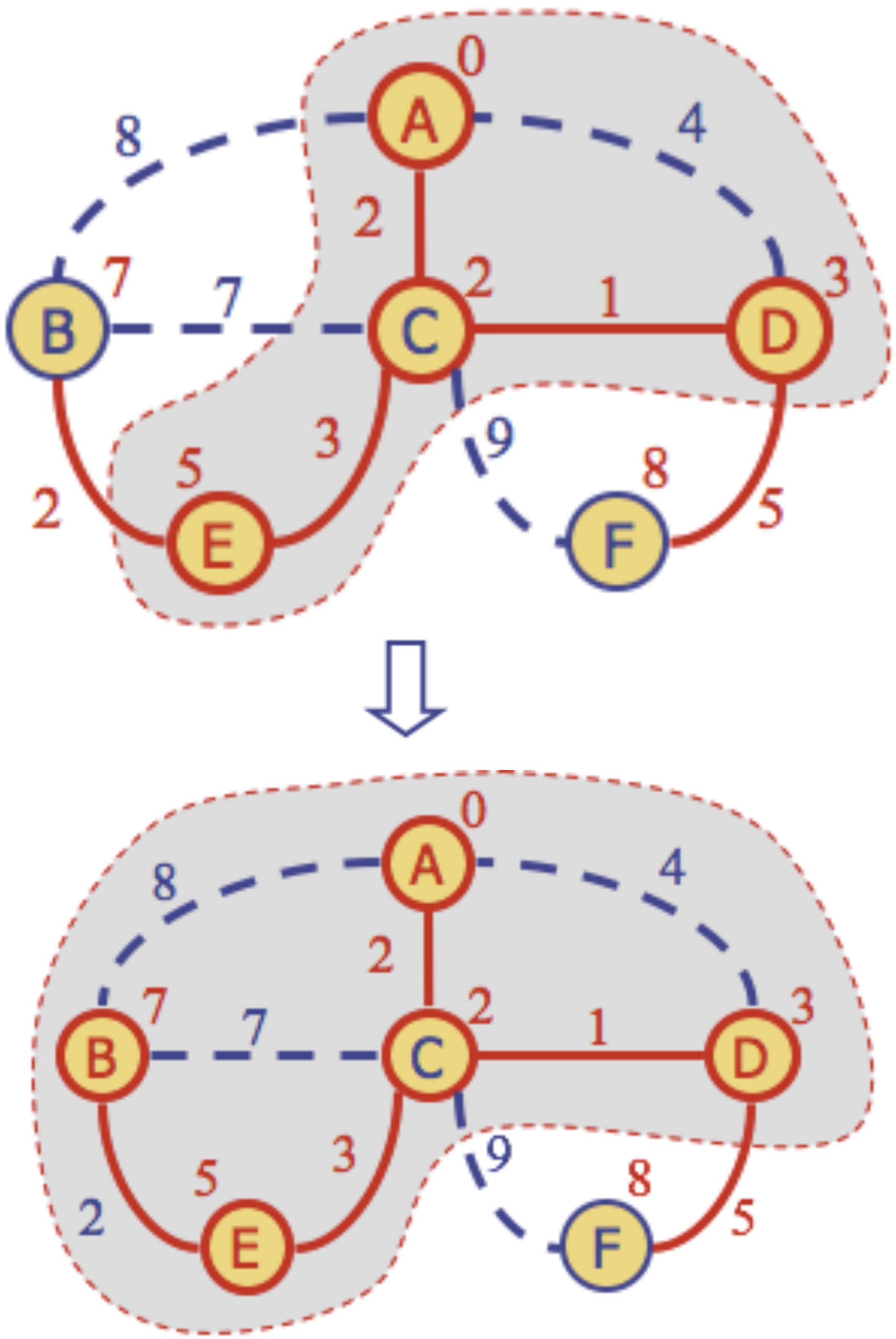


A
B
C
D
E
F
0
8
4
2
8
7
2
1
4
∞
3
9
∞
2
5
0
8
4
2
8
7
2
1
3
5
3
9
11
2
5
0
8
4
2
8
7
2
1
3
5
3
9
8
2
5
0
8
4
2
7
7
2
1
3
5
3
9
8
2
5

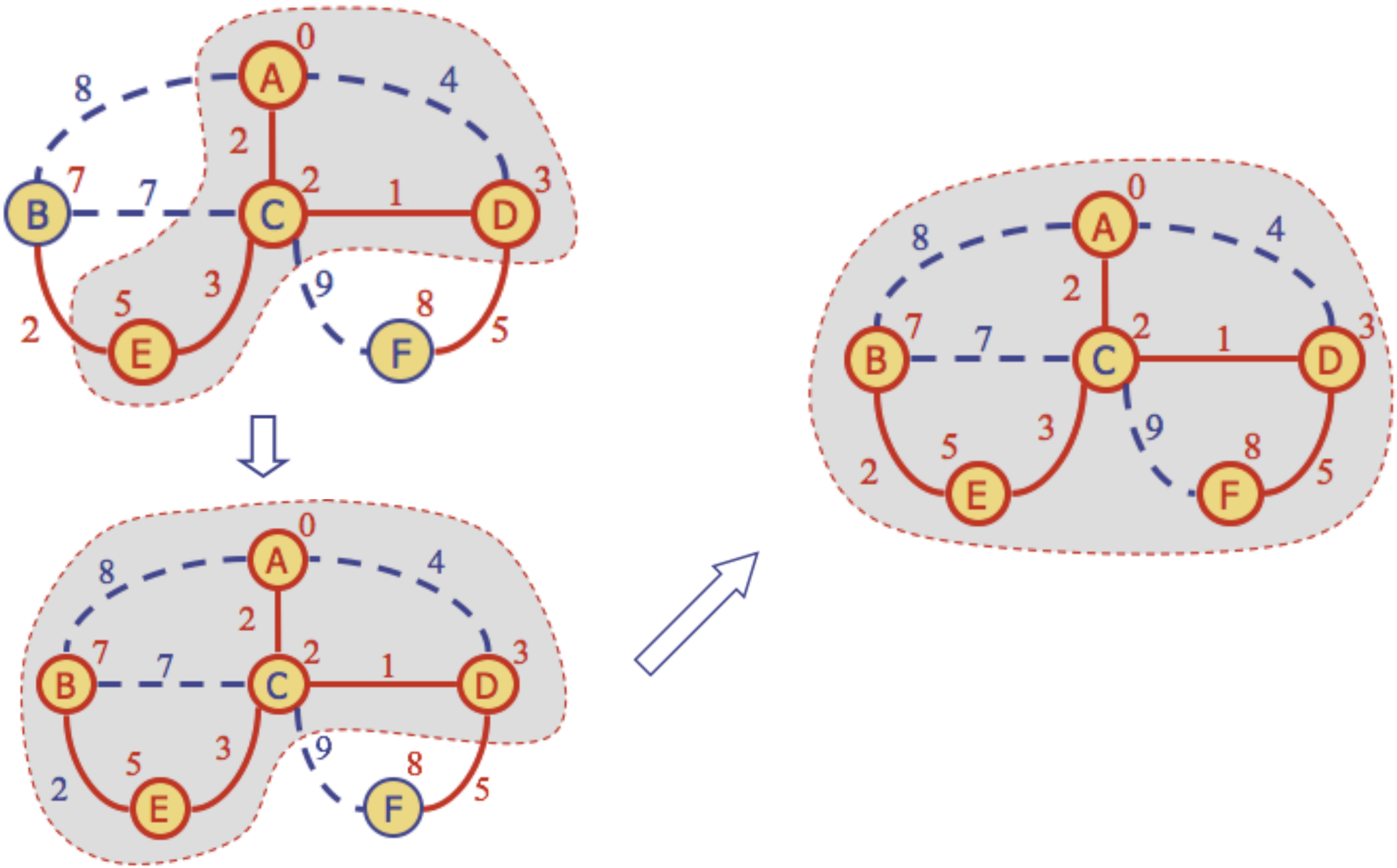
Example



Example



Example


0
A
8
4
2
7
7
2
1
3
B
C
D
9
8
5
3
5
2
E
F
0
A
8
4
2
7
7
2
1
3
B
C
D
9
8
5
3
5
2
E
F
0
A
8
4
2
7
7
2
1
3
B
C
D
9
8
5
3
5
2
E
F

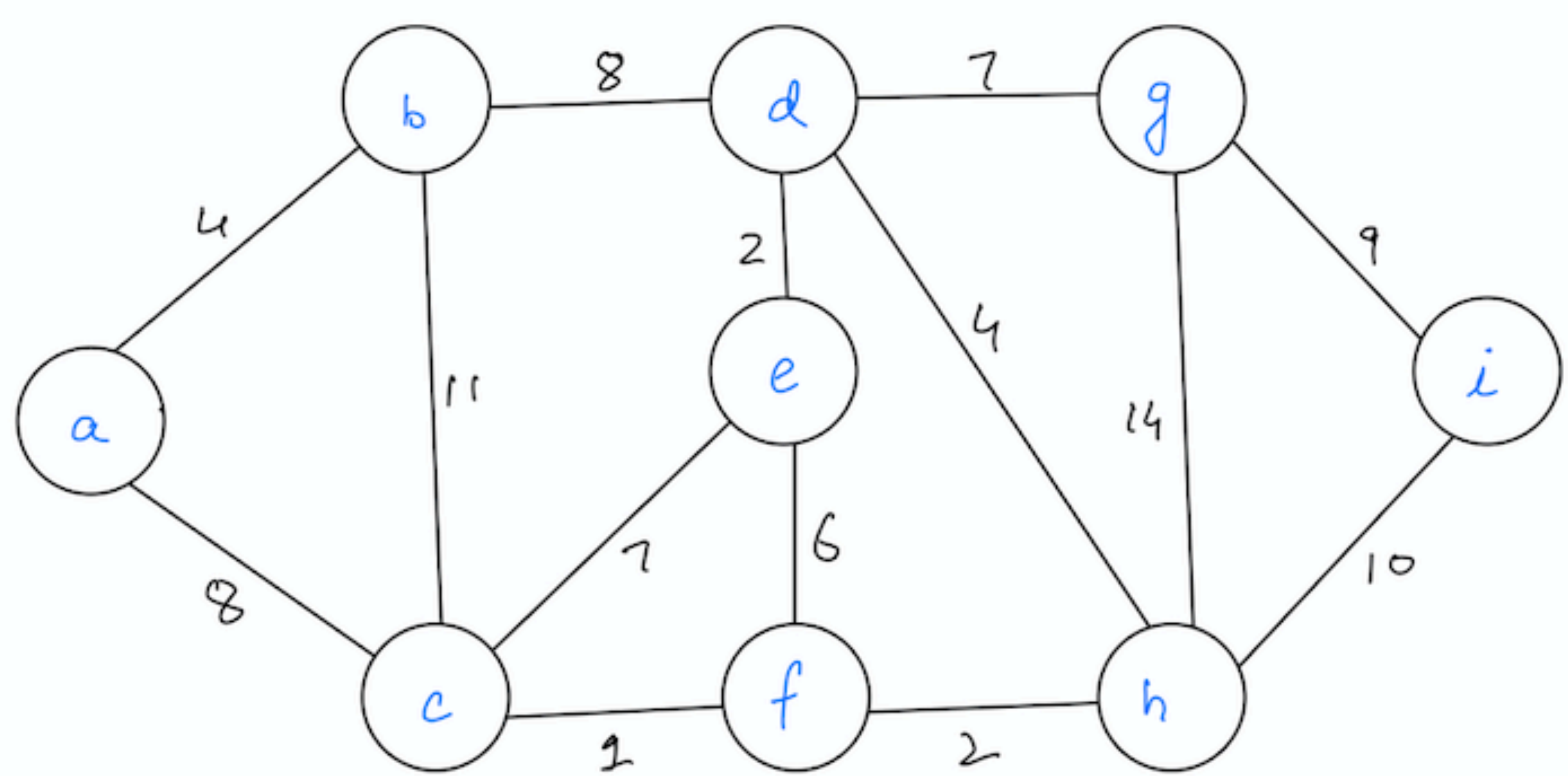
Pseudocode From Cormen's Book

```
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

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

Example on the Whiteboard



Time Complexity

- Three main tasks

1. Creation of PQ – $O(|V| \log(|V|))$

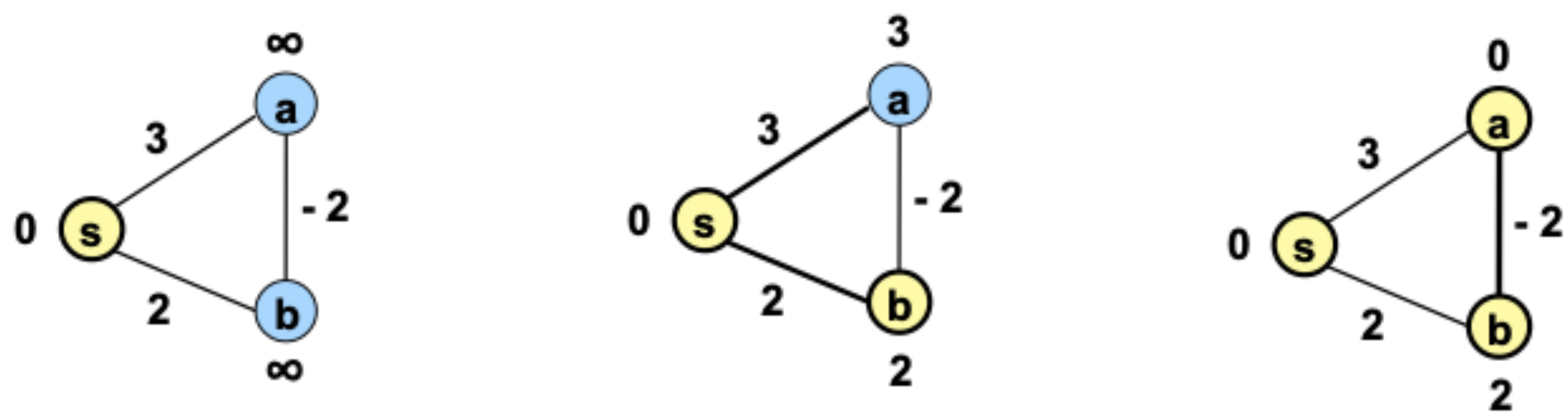
2. Emptying the PQ – $O(|V| \log(|V|))$

3. Updating the PQ – $O(|E| \log(|V|))$

- Thus, T: $O(|E| \log(|V|))$

Dijkstra’s Algorithm

- Does not work with negative edges



Thus, Dijkstra’s algorithm would visit *b* then *a* and leave *b* with a distance of 2 instead of the correct distance 1

Summary

1. Shortest Path
2. Single-source shortest path
3. All-pairs shortest path

- Dijkstra’s Algorithm
- When does Dijkstra fail?
- Bellman Ford and All-to-All (Floyd-Warshall’s Algorithm) will be covered in the tutorial