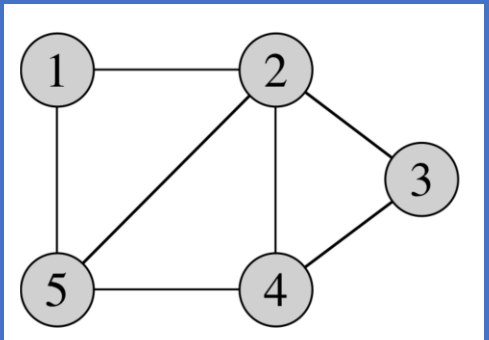
CS141: Intermediate Data Structures and Algorithms



Single-Source Shortest-Paths (SSSP)

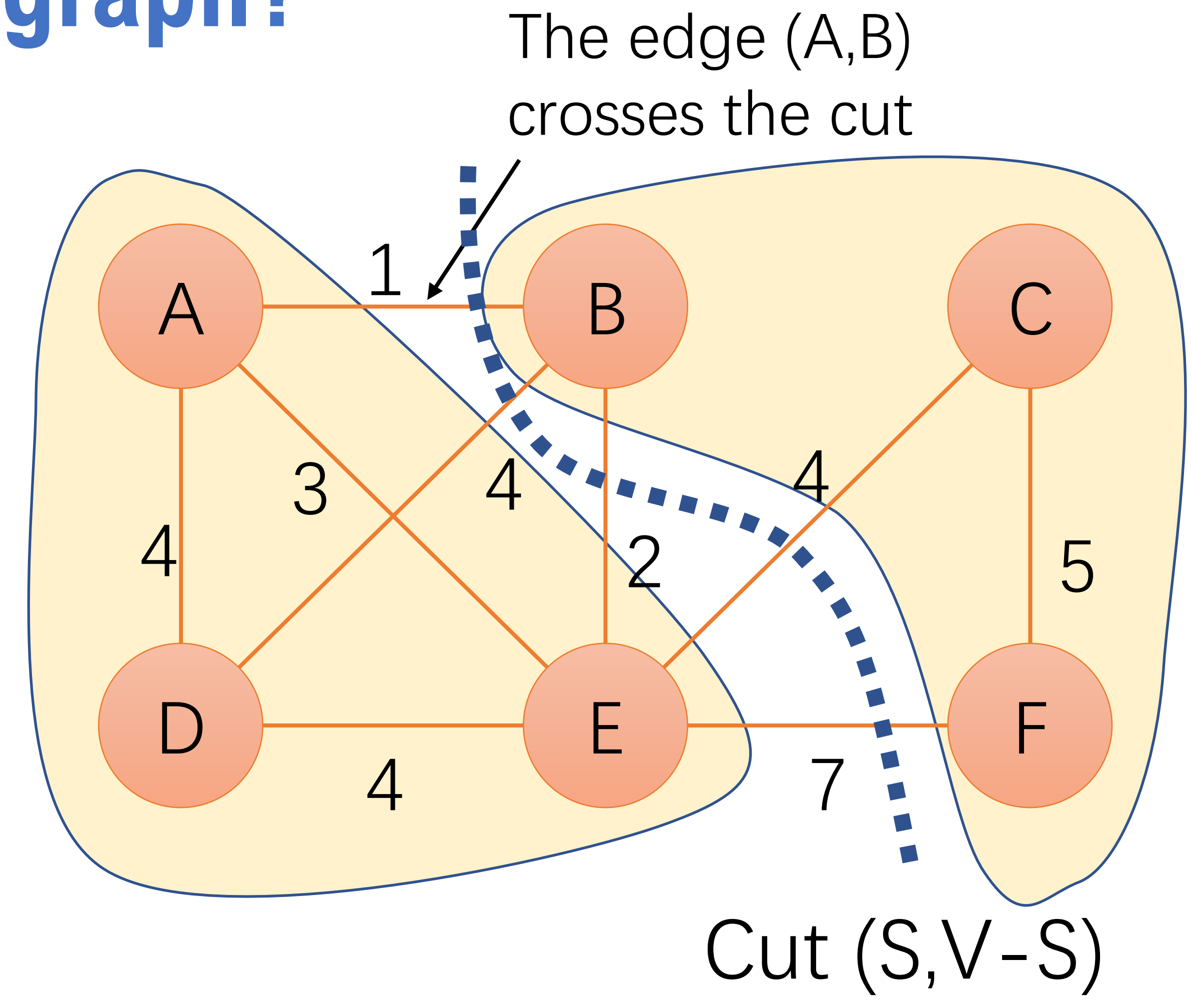
Yihan Sun

This lecture covers Section 24.1-24.3 of CLRS

Optimality Proof for Greedy MST Algorithms

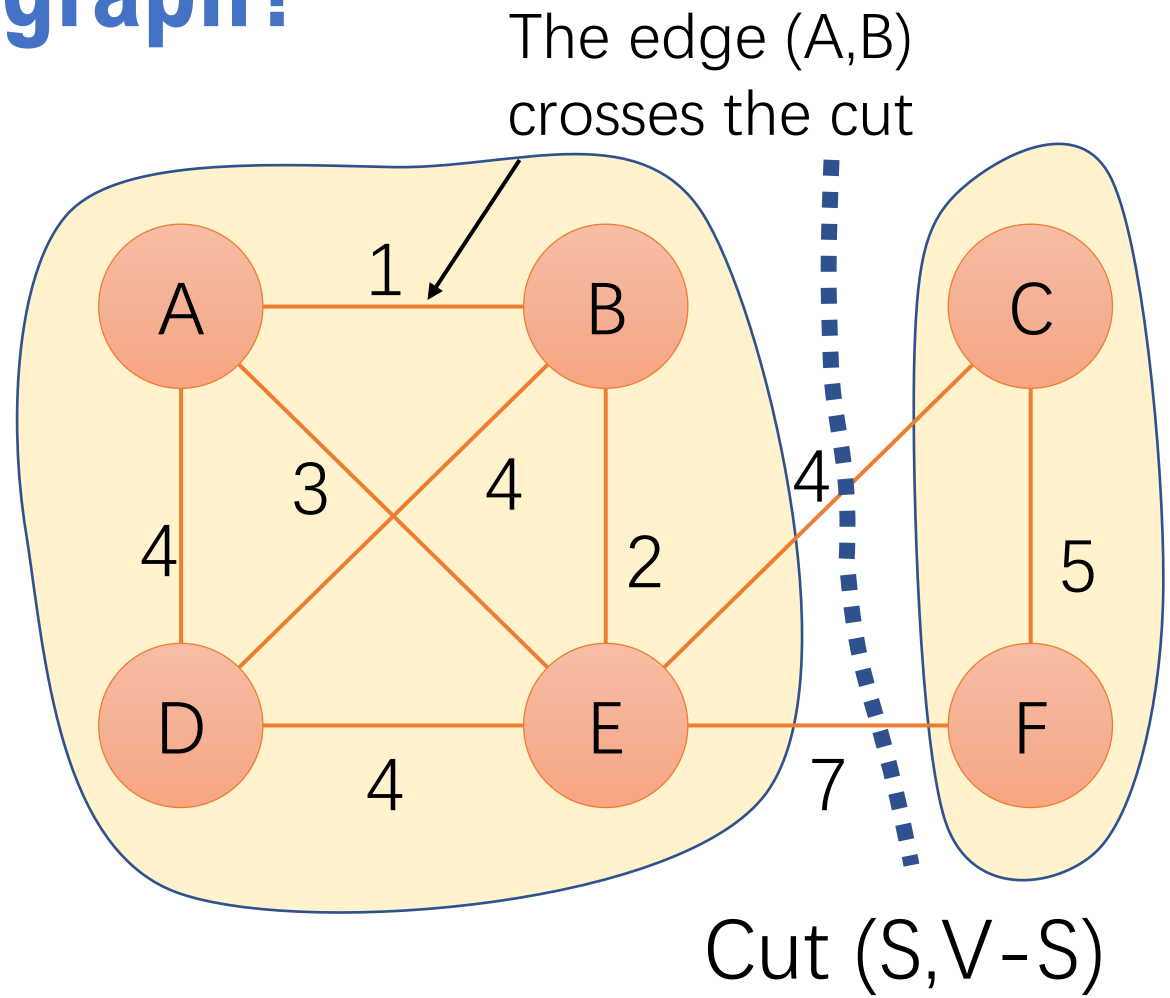
What is a cut of a graph?

$$S = \{A, D, E\}$$
$$V - S = \{B, C, F\}$$



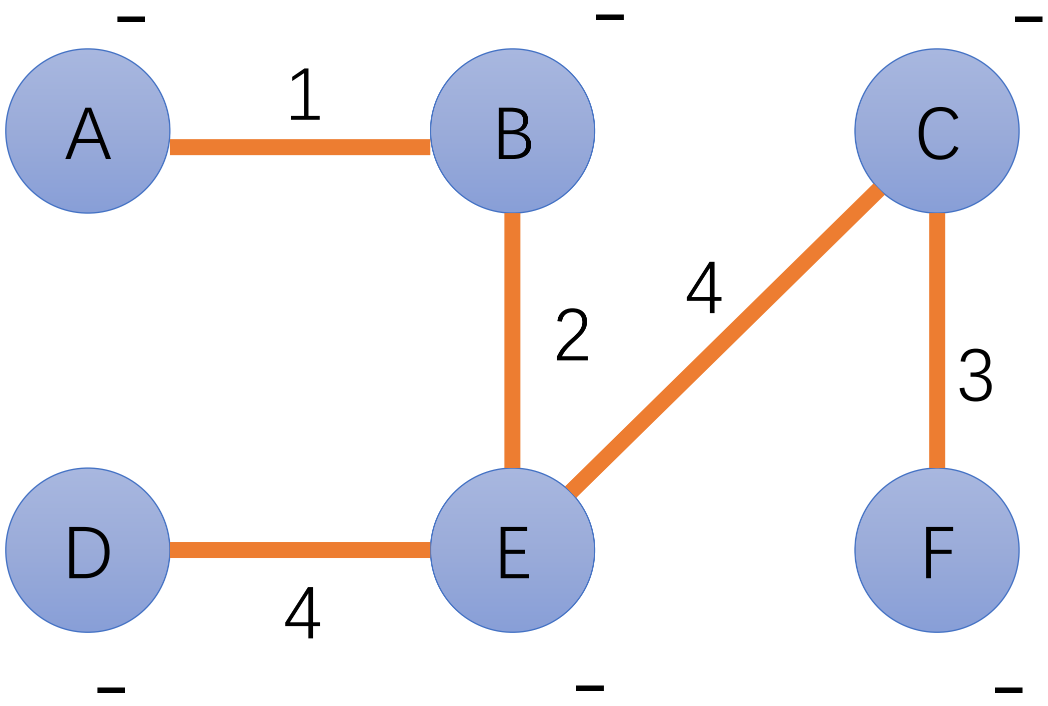
What is a cut of a graph?

$$S = \{A, B, D, E\}$$
$$V - S = \{C, F\}$$



The lightest edge in a cut must be in the MST

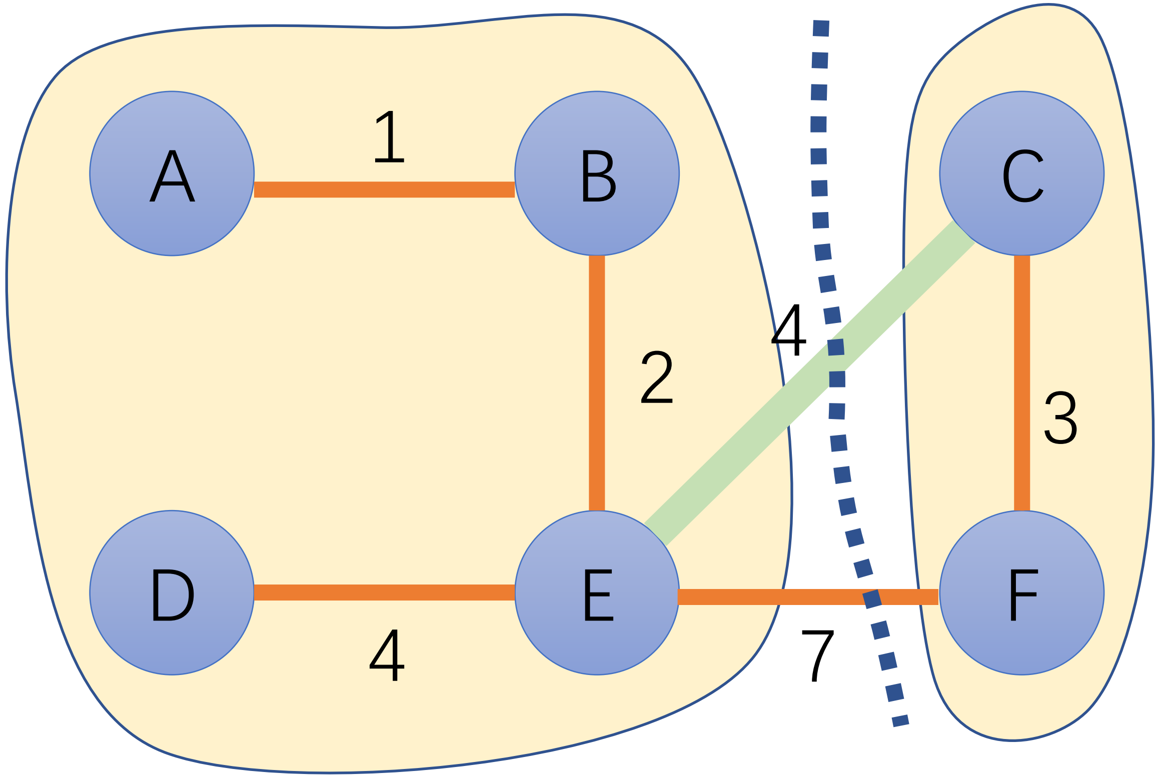
The MST of this graph



The lightest edge in a cut must be in the MST

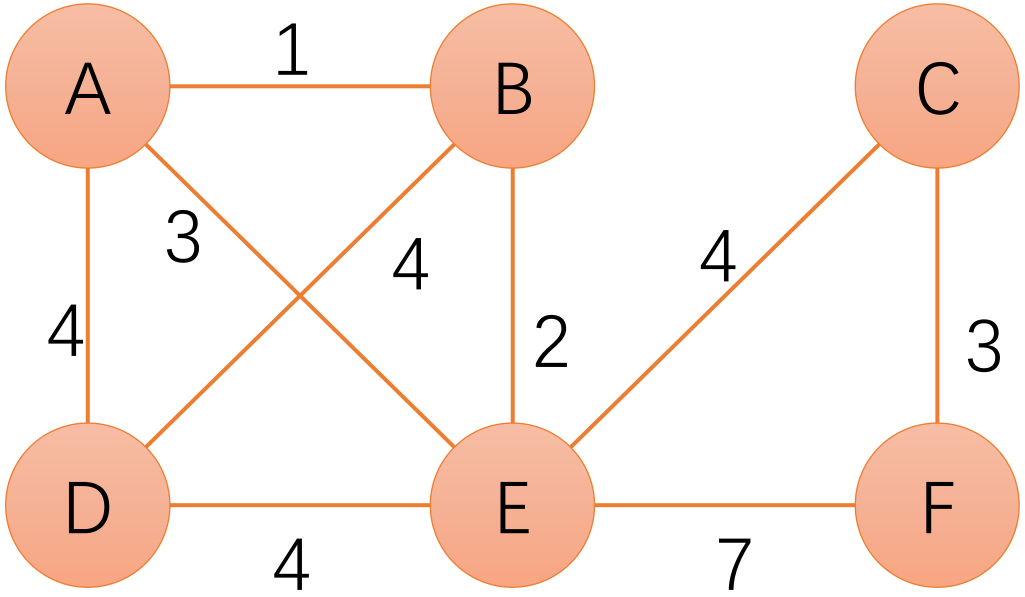
A simple proof for greedy choice

$$S = \{A, B, D, E\}$$
$$V - S = \{C, F\}$$



The lightest edge in a cut must be in the MST

Kruskal’s MST in Action



(u,v,w):

(A,B,1)

(B,E,2)

(A,E,3)

(C,F,3)

(A,D,4)

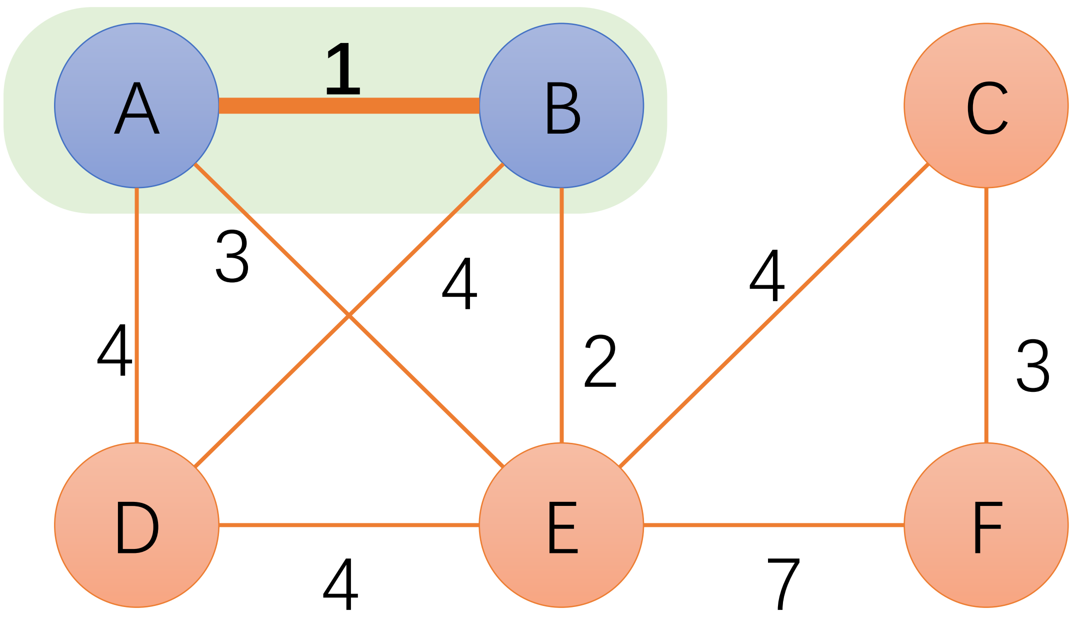
(B,D,4)

(D,E,4)

(C,E,4)

(E,F,7)

Kruskal's MST in Action



(u,v,w):

(A,B,1)

(B,E,2)

(A,E,3)

(C,F,3)

(A,D,4)

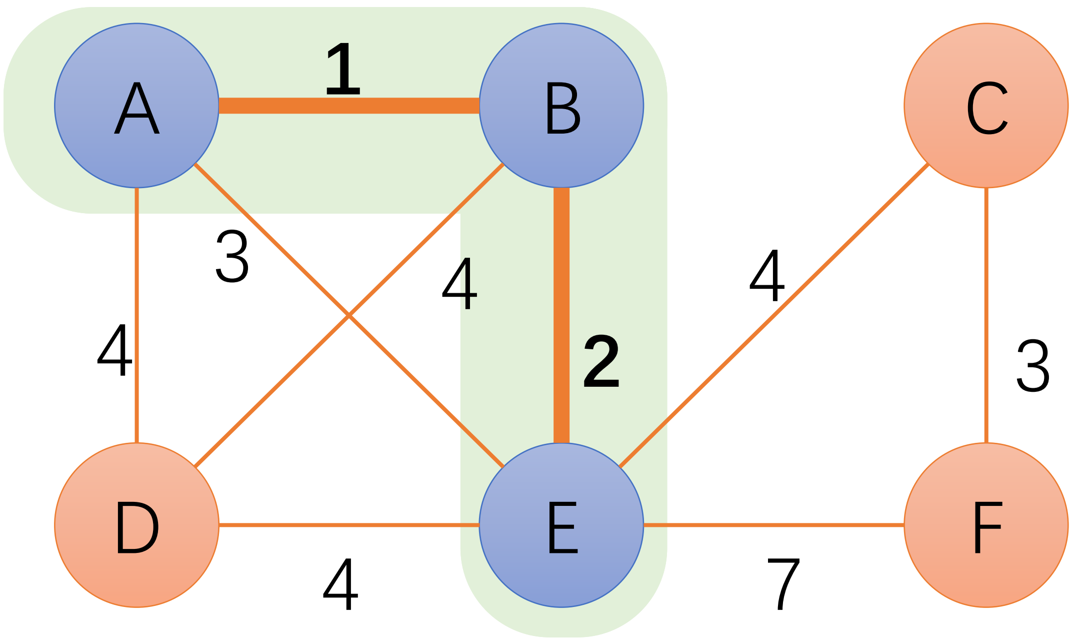
(B,D,4)

(D,E,4)

(C,E,4)

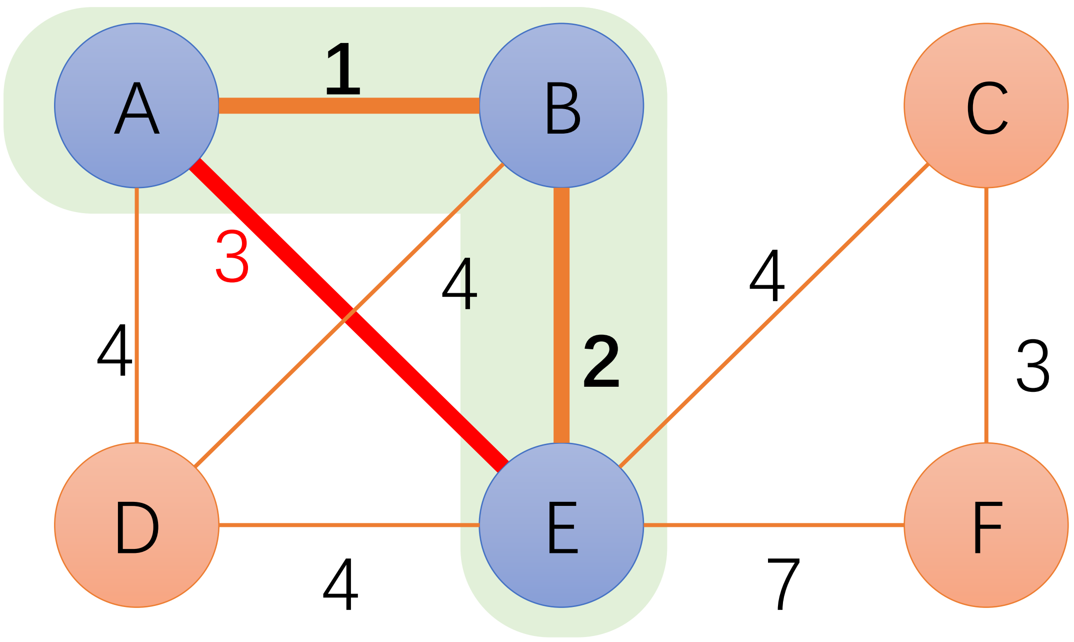
(E,F,7)

Kruskal's MST in Action



(u,v,w):
(A,B,1)
(B,E,2)
(A,E,3)
(C,F,3)
(A,D,4)
(B,D,4)
(D,E,4)
(C,E,4)
(E,F,7)

Kruskal's MST in Action



(u,v,w):

(A,B,1)

(B,E,2)

(A,E,3)

(C,F,3)

(A,D,4)

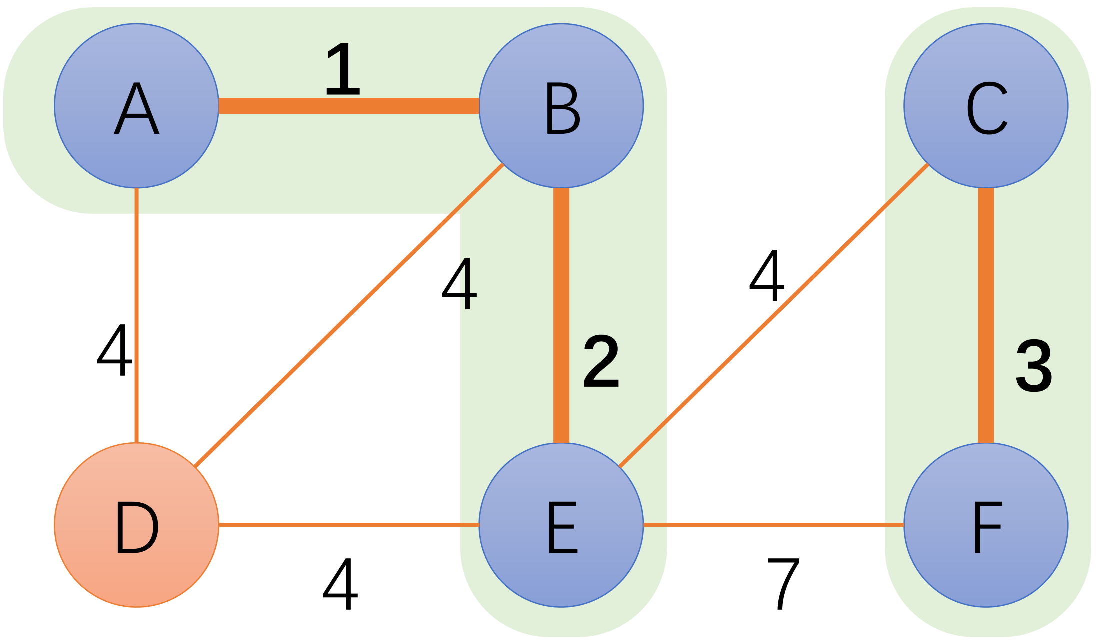
(B,D,4)

(D,E,4)

(C,E,4)

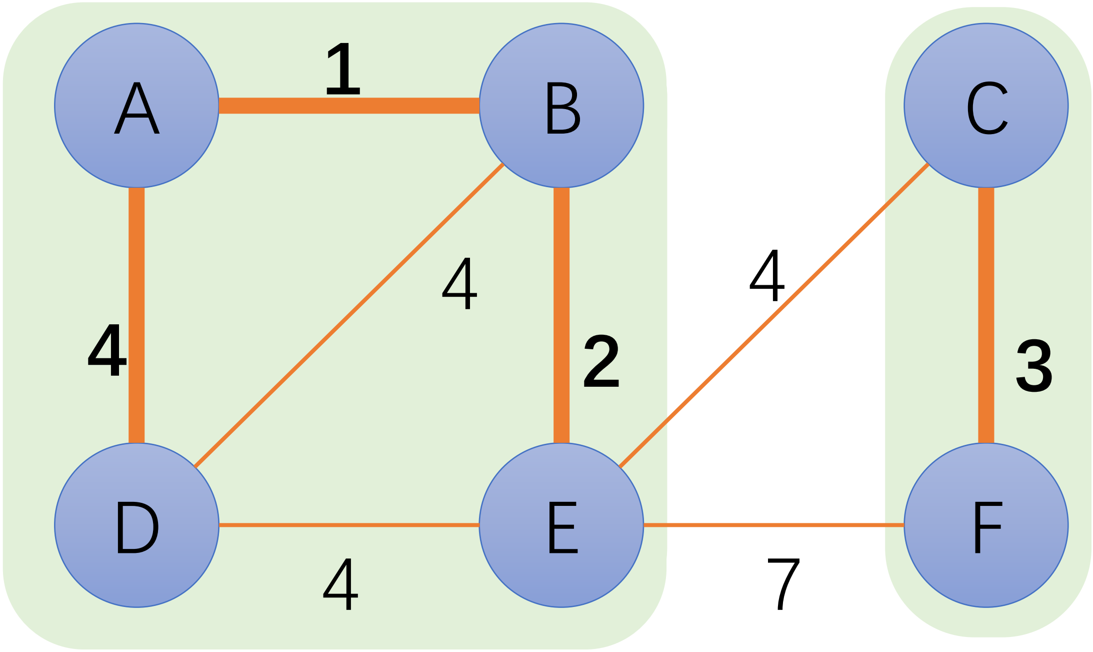
(E,F,7)

Kruskal's MST in Action



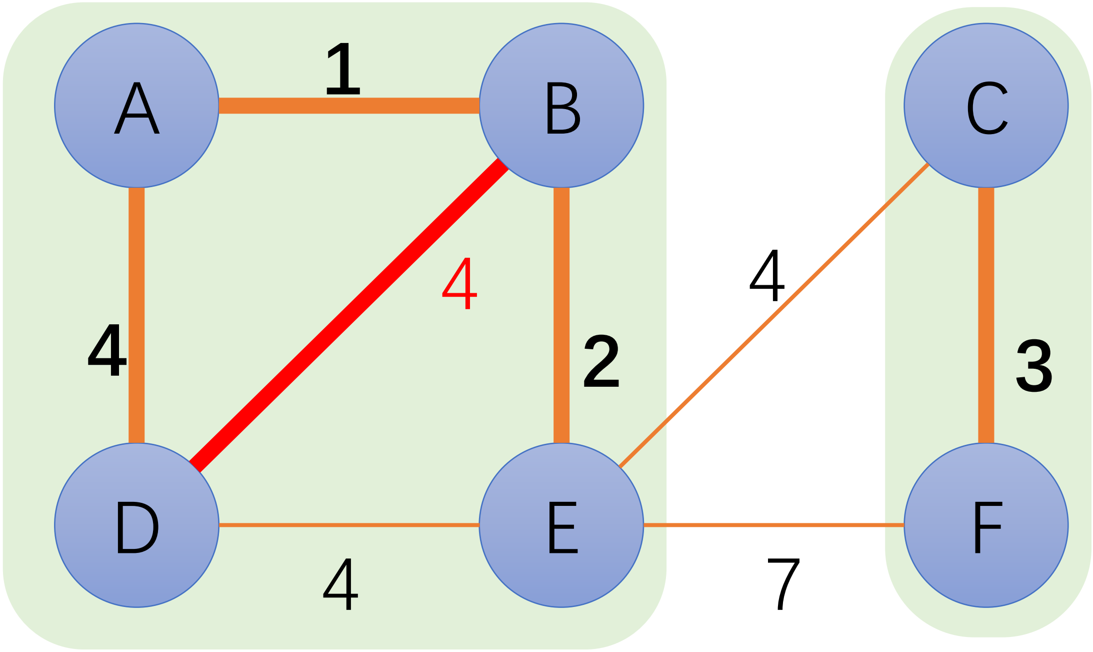
(u,v,w):
(A,B,1)
(B,E,2)
(A,E,3)
(C,F,3)
(A,D,4)
(B,D,4)
(D,E,4)
(C,E,4)
(E,F,7)

Kruskal's MST in Action



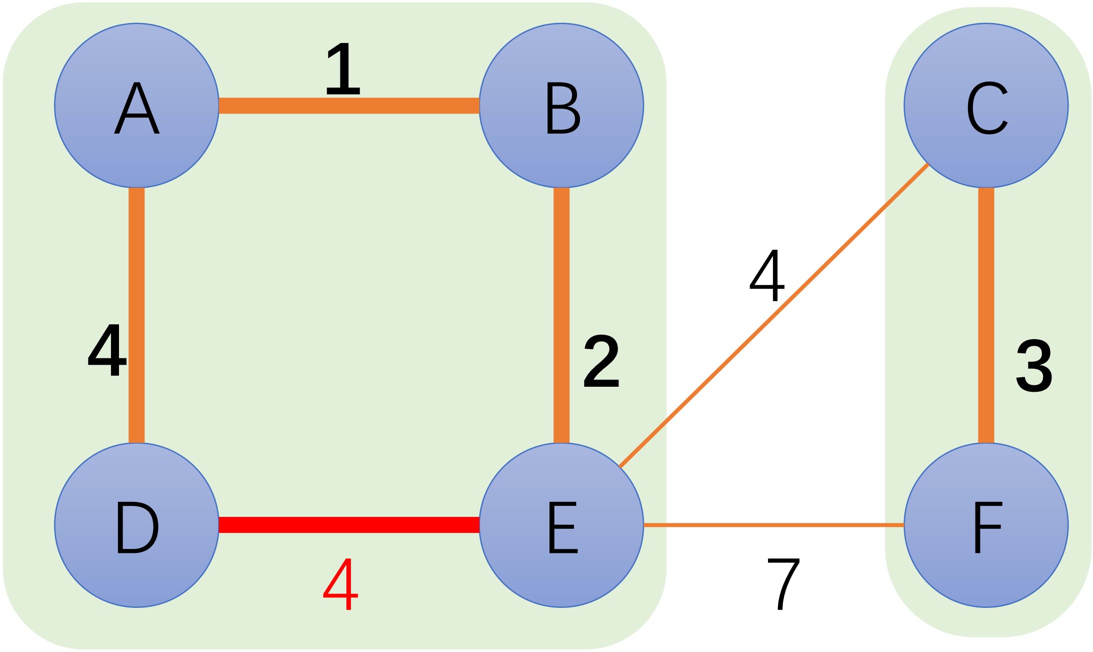

```
(u,v,w):
(A,B,1)
(B,E,2)
(A,E,3)
(C,F,3)
(A,D,4)
(B,D,4)
(D,E,4)
(C,E,4)
(E,F,7)
```

Kruskal's MST in Action



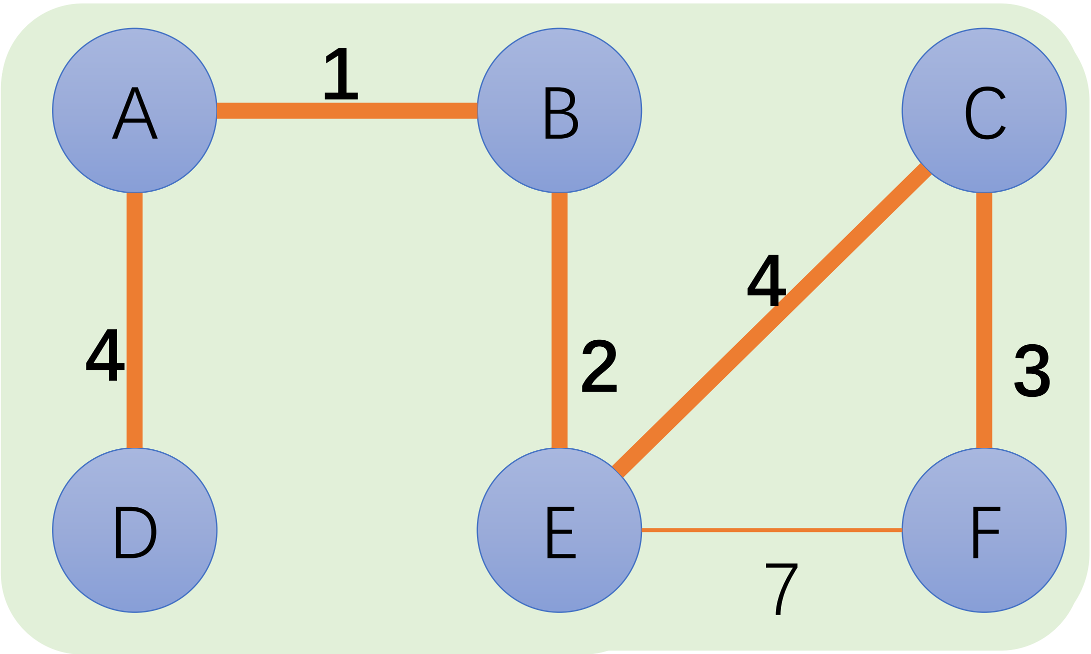
(u,v,w):
(A,B,1)
(B,E,2)
(A,E,3)
(C,F,3)
(A,D,4)
(B,D,4)
(D,E,4)
(C,E,4)
(E,F,7)

Kruskal's MST in Action




```
(u,v,w):
(A,B,1)
(B,E,2)
(A,E,3)
(C,F,3)
(A,D,4)
(B,D,4)
(D,E,4)
(C,E,4)
(E,F,7)
```

Kruskal's MST in Action



(u,v,w):
(A,B,1)
(B,E,2)
(A,E,3)
(C,F,3)
(A,D,4)
(B,D,4)
(D,E,4)
(C,E,4)
(E,F,7)

Correction:
Kruskal's Algorithm (simple implementation)

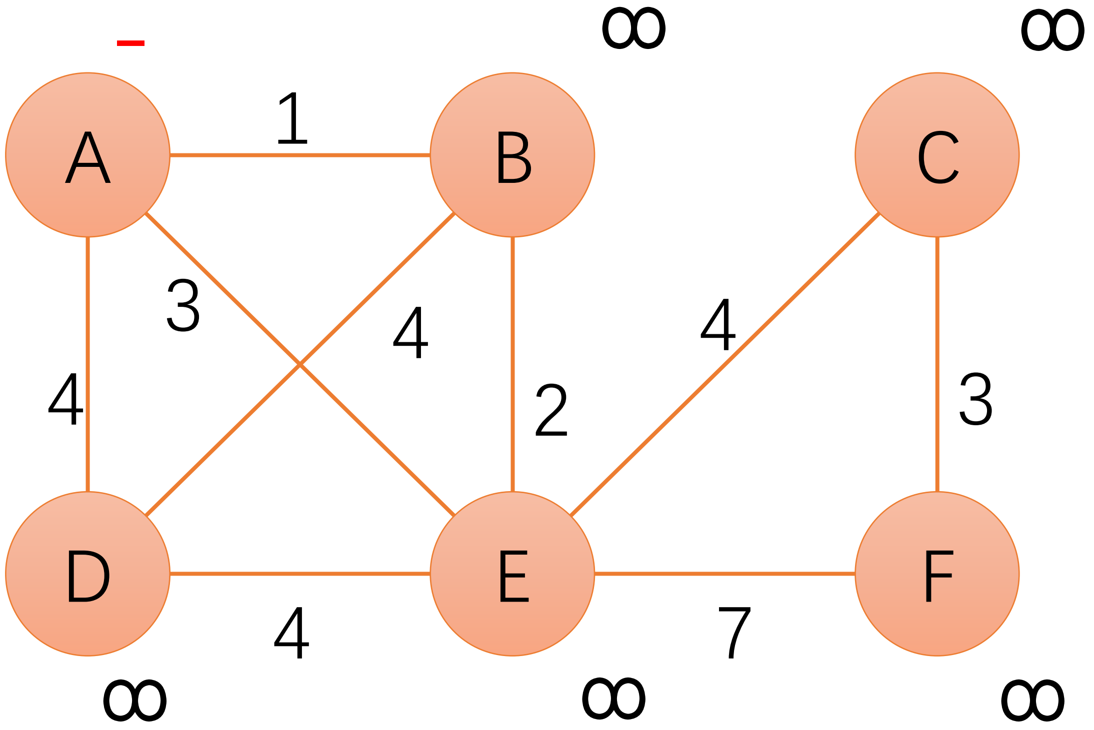
- Use an array to check the connectivity

$$O(n^2 + m \log n)$$

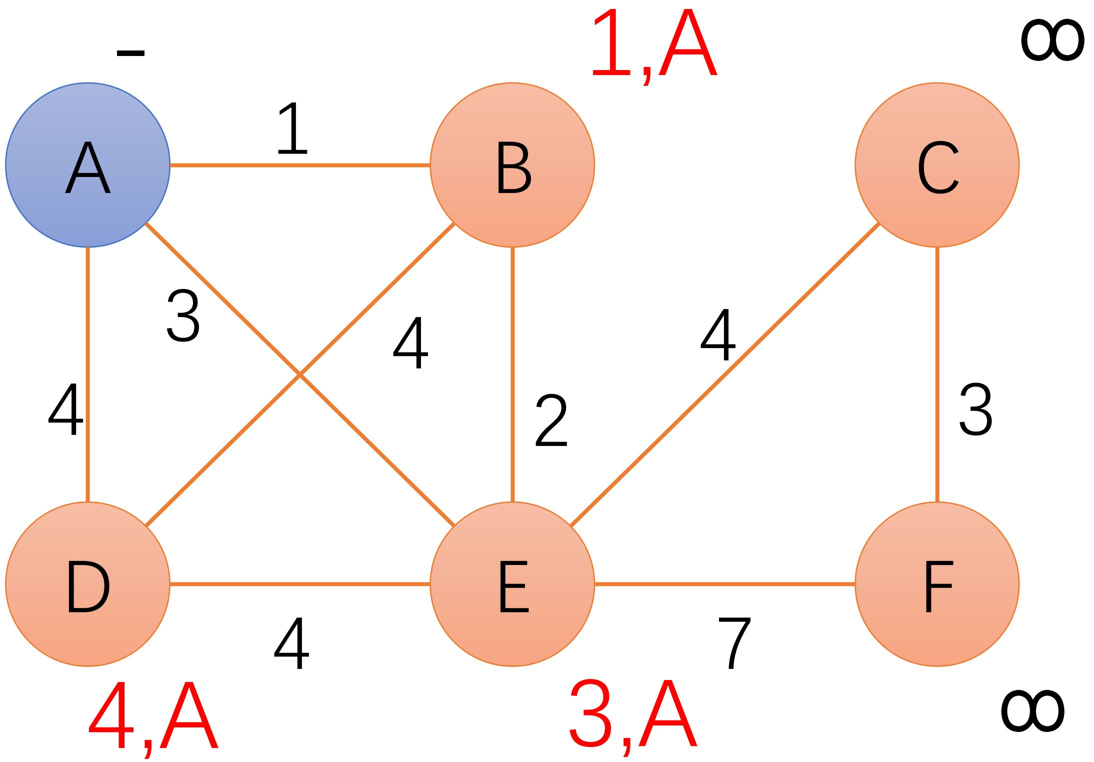
$\text{KRUSKAL}(G, w)$

$O(1)$	$A = \emptyset$
	for each vertex $v \in G.V$ // n iterations
$O(1)$	$\quad \text{MAKE-SET}(v)$
$O(m \log m)$	sort the edges of $G.E$ into nondecreasing order by weight w
	for each (u, v) taken from the sorted list // m iterations
$O(1)$	$\quad$ **if** $\text{FIND-SET}(u) \neq \text{FIND-SET}(v)$ // m iterations
$O(1)$	$\quad\quad A = A \cup \{(u, v)\}$ // n iterations
$O(n)$	$\quad\quad \text{UNION}(u, v)$
	return A

Prim's MST in Action



Prim's MST in Action



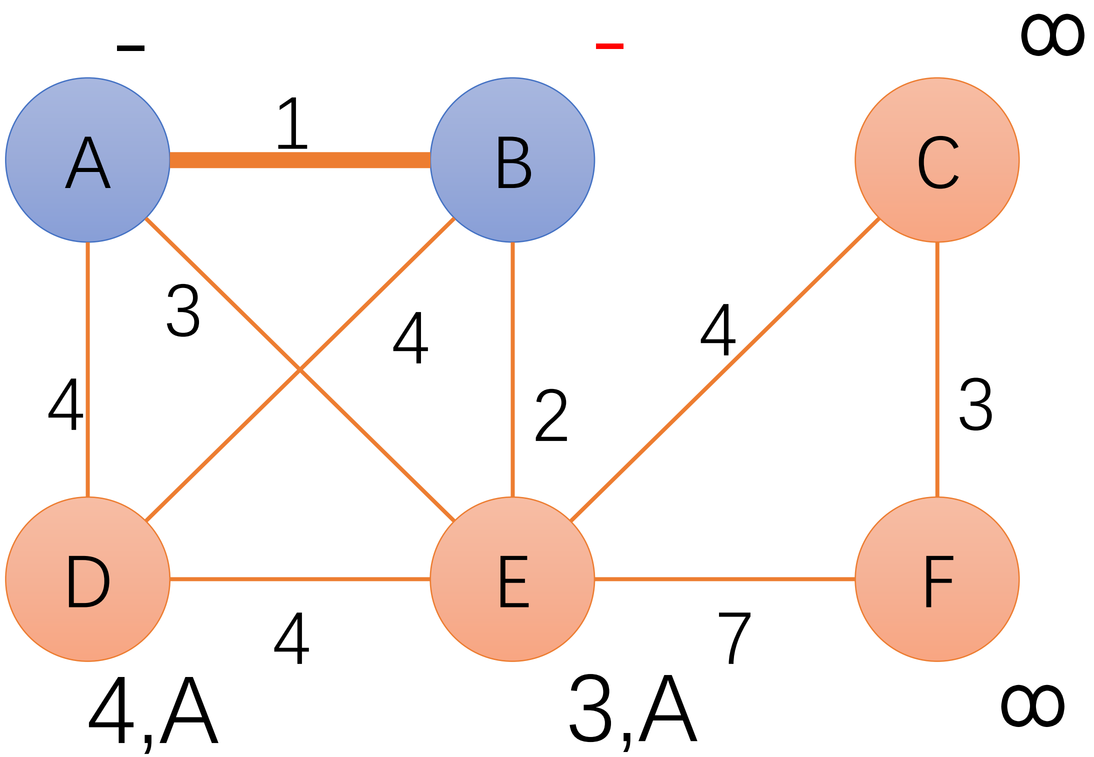
Active Set:

(B, 1)

(E, 3)

(D, 4)

Prim's MST in Action

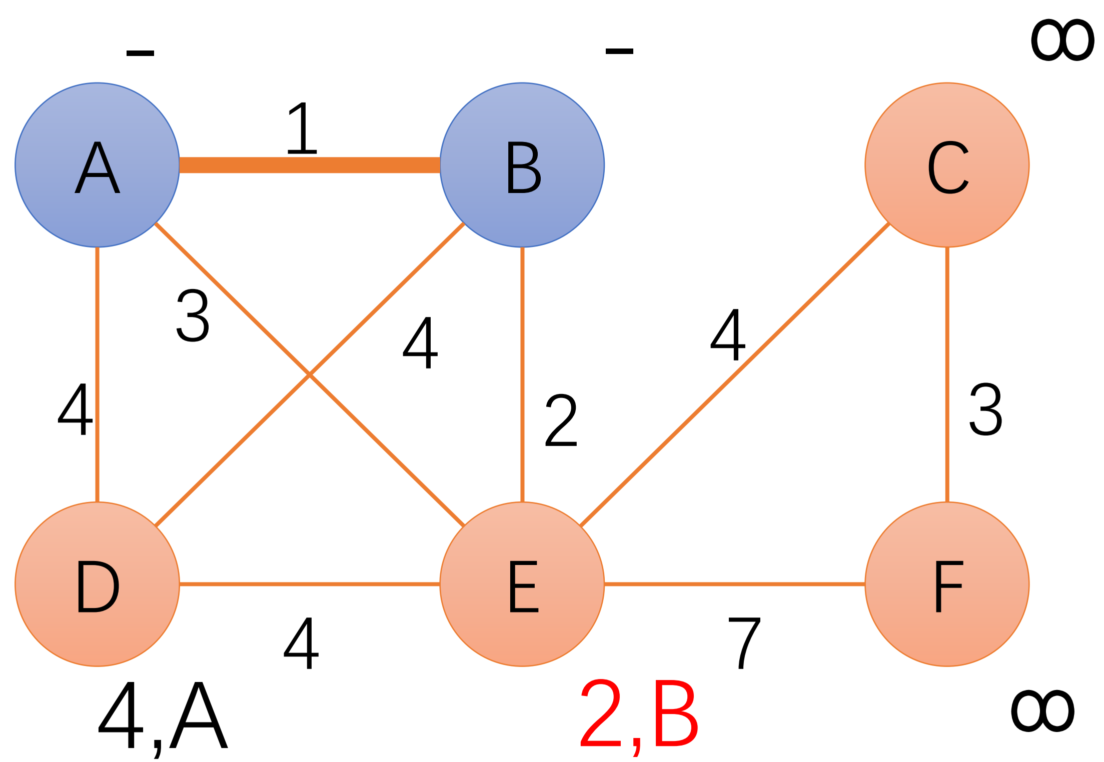


Active Set:

(E, 3)

(D, 4)

Prim’s MST in Action

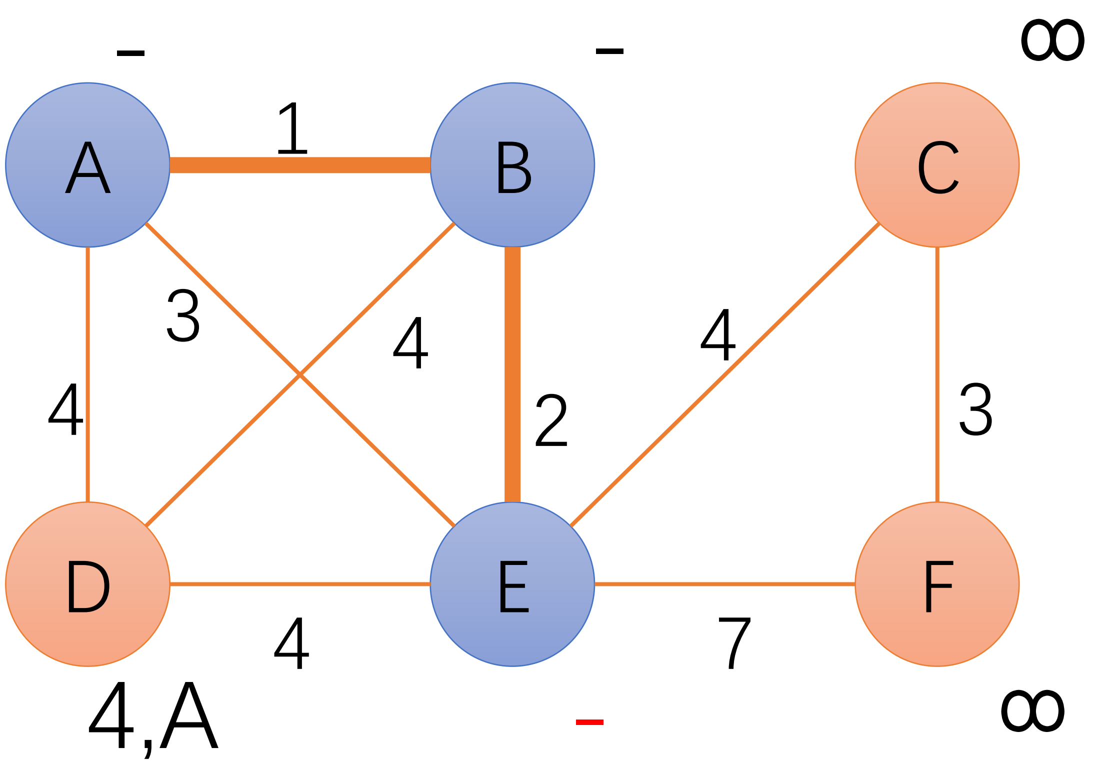


Active Set:

(E, 2)

(D, 4)

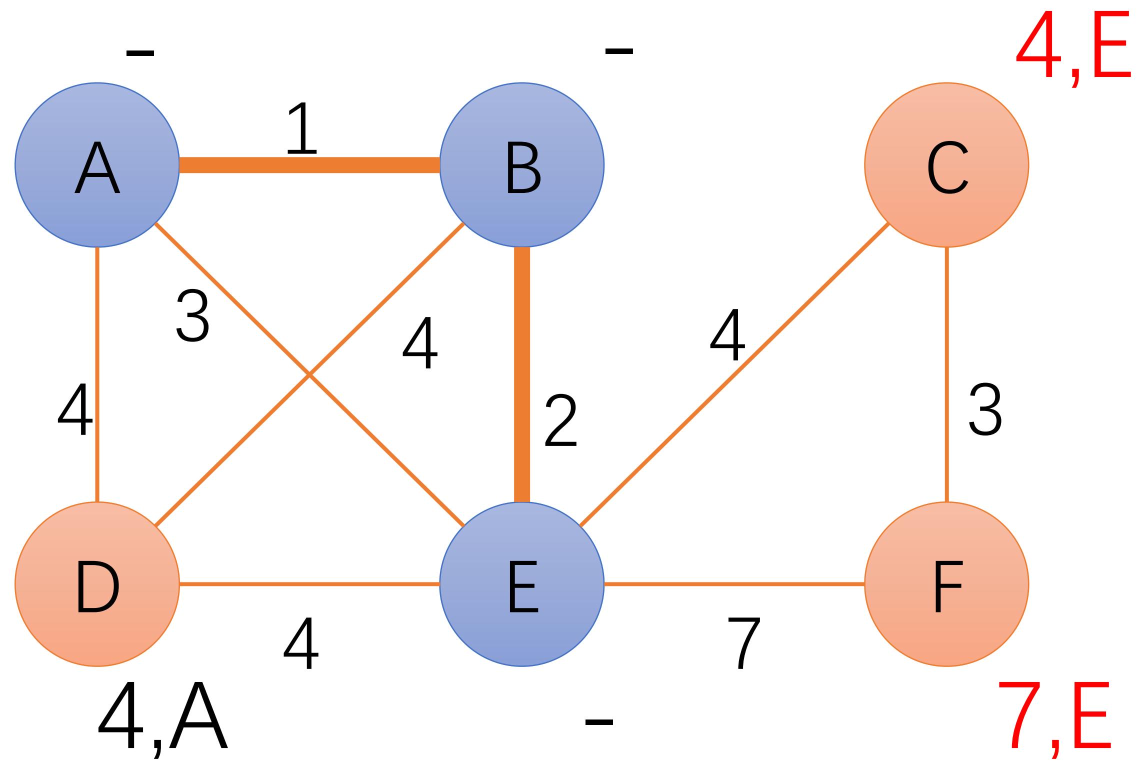
Prim's MST in Action



Active Set:

(D, 4)

Prim's MST in Action

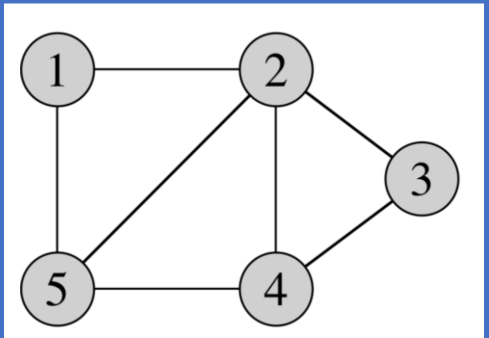


Active Set:

(C, 4)

(D, 4)

(F, 7)

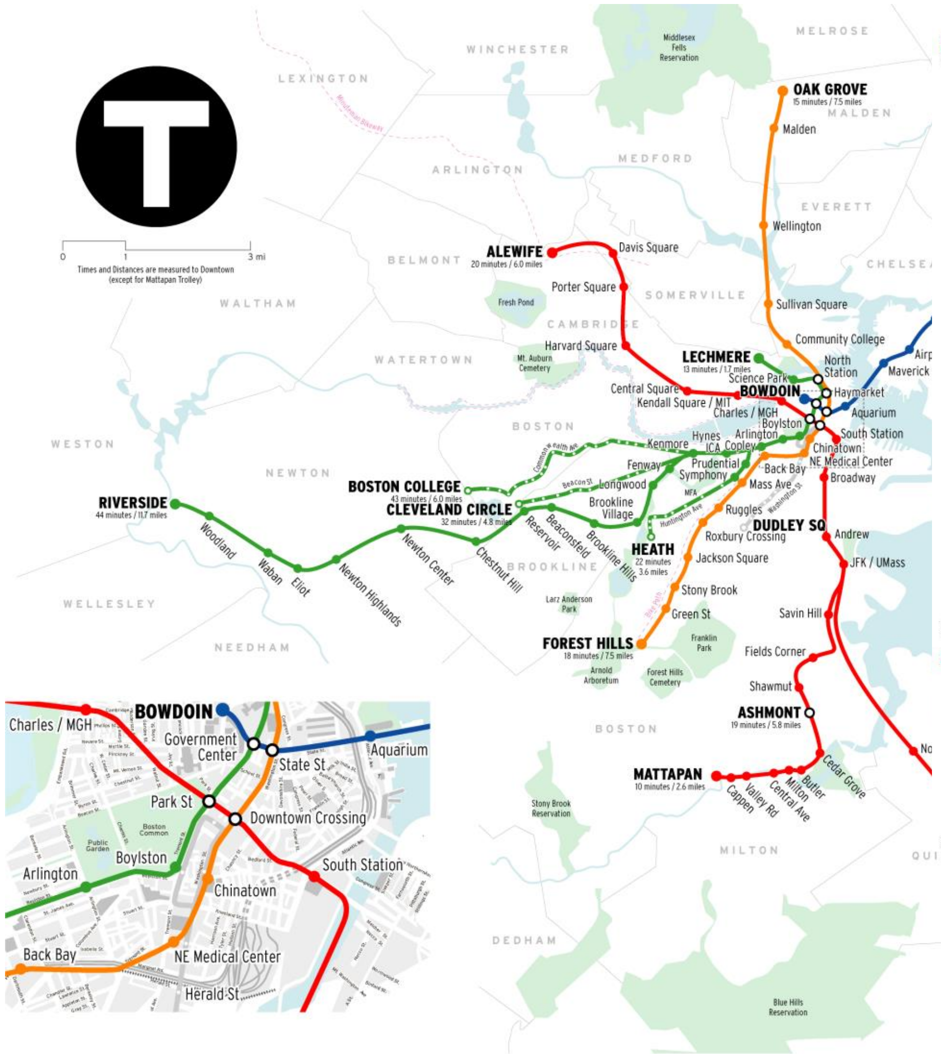


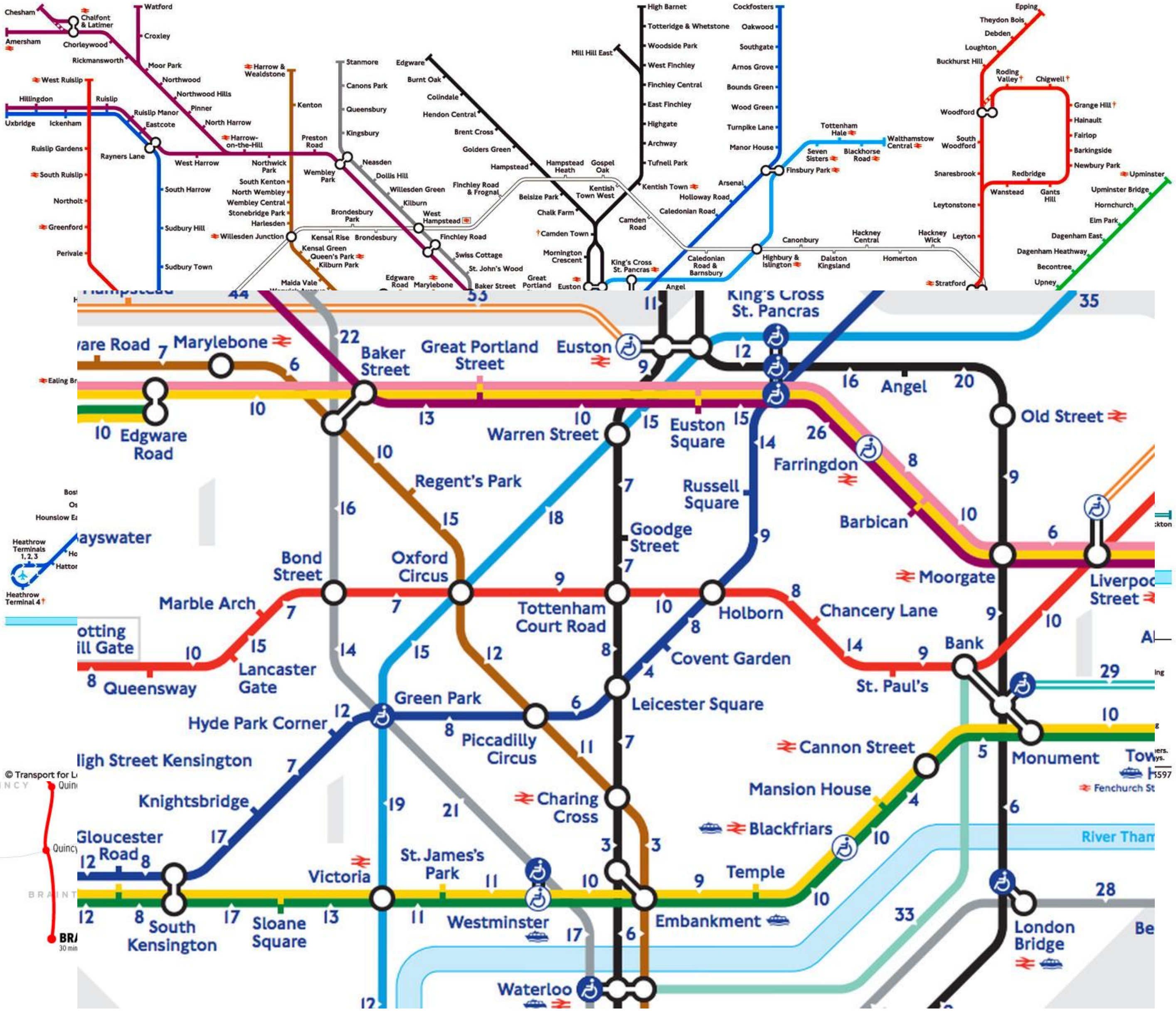
Single-Source Shortest-Paths (SSSP)

Yihan Sun

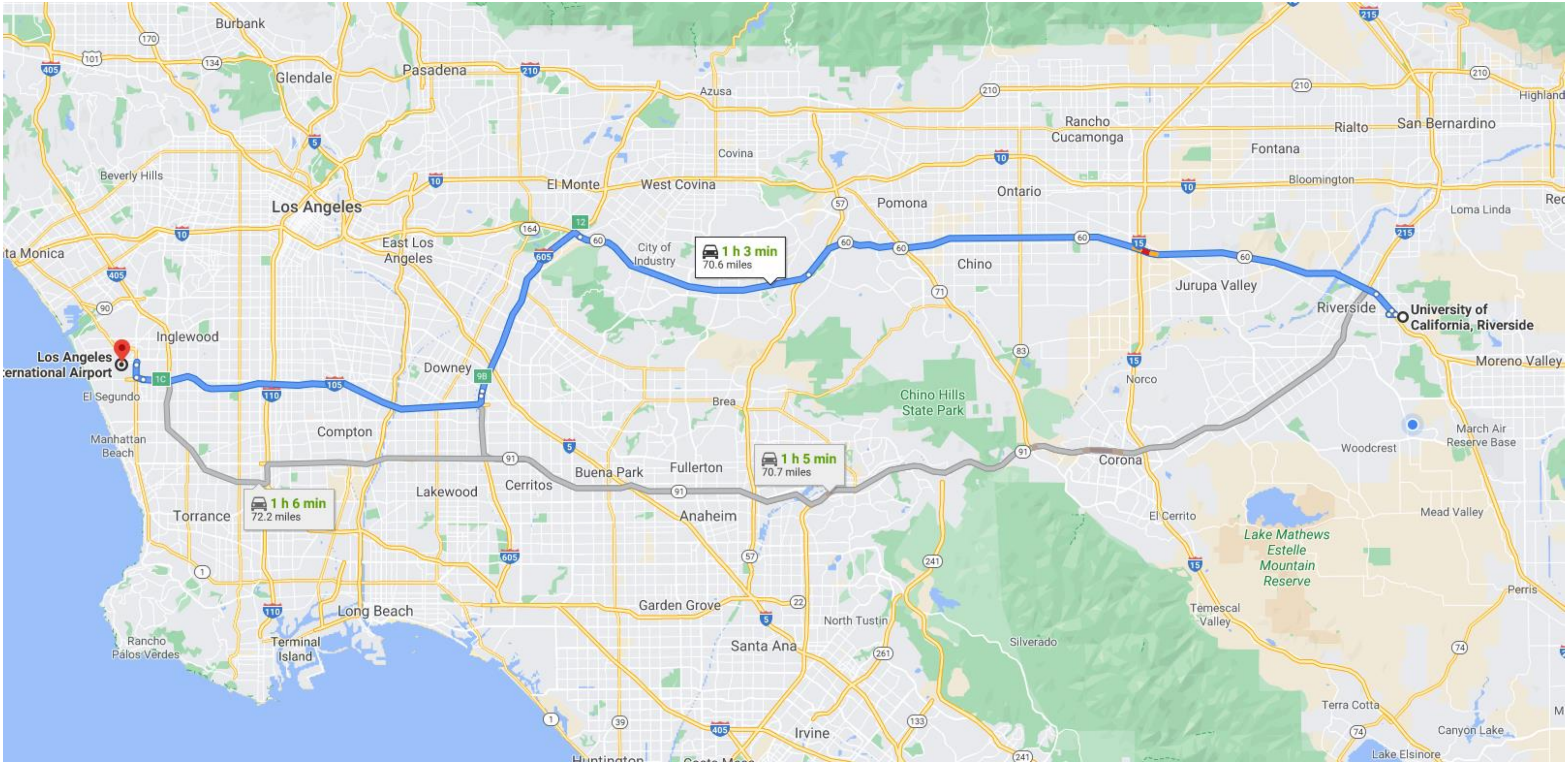
This lecture covers Section 24.1-24.3 of CLRS

Taking subways when travelling

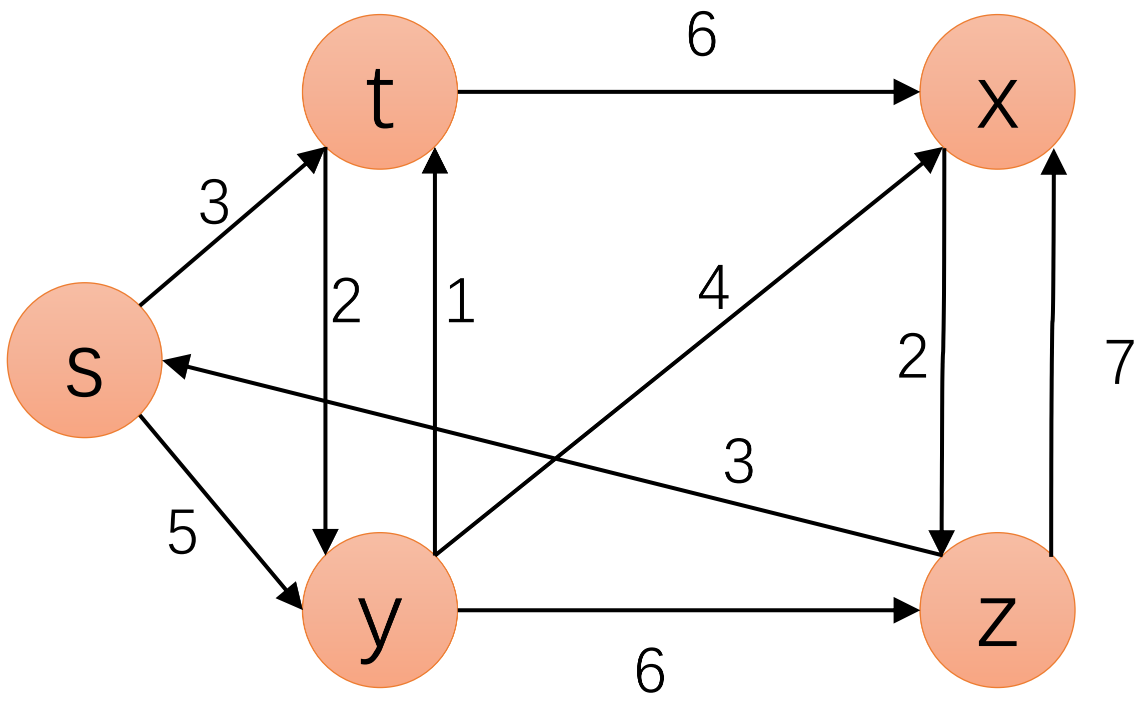




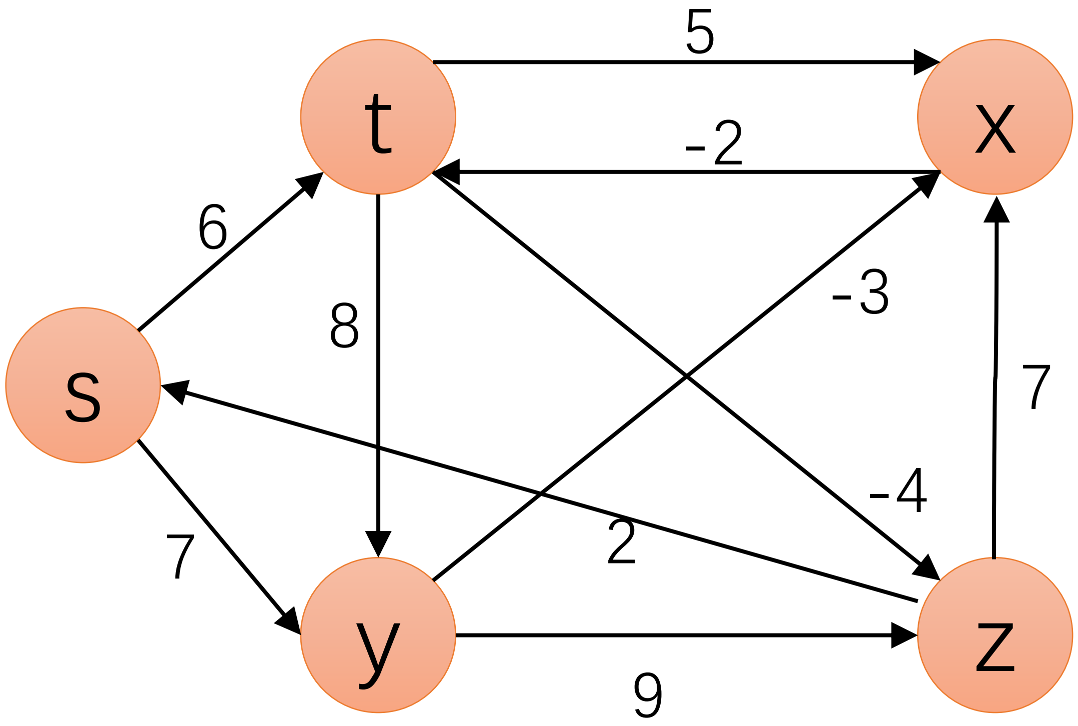
Navigation using Google Maps



Example: Positive Weights



Example: Negative Weights



- **Example: Difference constraints (CLRS Section 24.4)**

The shortest-path problem

- Given a weighted graph $G = (V, E)$, with weight function $w: E \to \mathbb{R}$
- The weight $w(p)$ of a path $p = \langle v_0, v_1, \ldots, v_k \rangle$ is the sum of the weights of its constituent edges:

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

- Shortest-path weight $\delta(u, v)$ from u to v is:

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

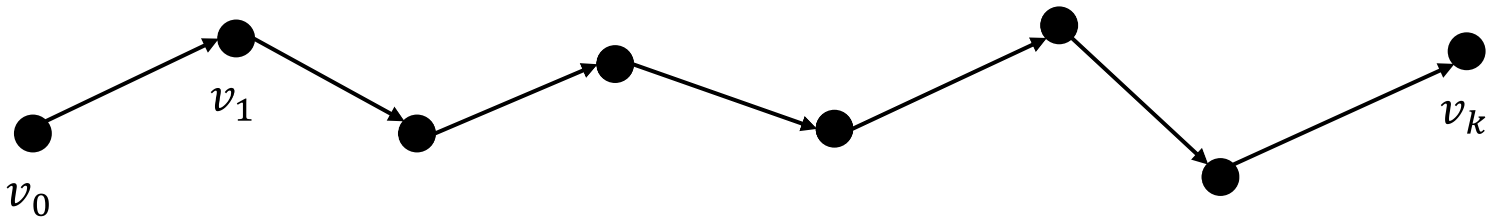
- The shortest path is any path with smallest path weight
- (Directed/undirected, positive/negative, weighted/unweighted)

Problem Variants

- **Single-source single-destination shortest path (s-t shortest path)**
- **Single-source all-destinations shortest paths (SSSP)**
- **All-pairs shortest paths (APSP)**
- **Number of paths**
  - Social network analysis

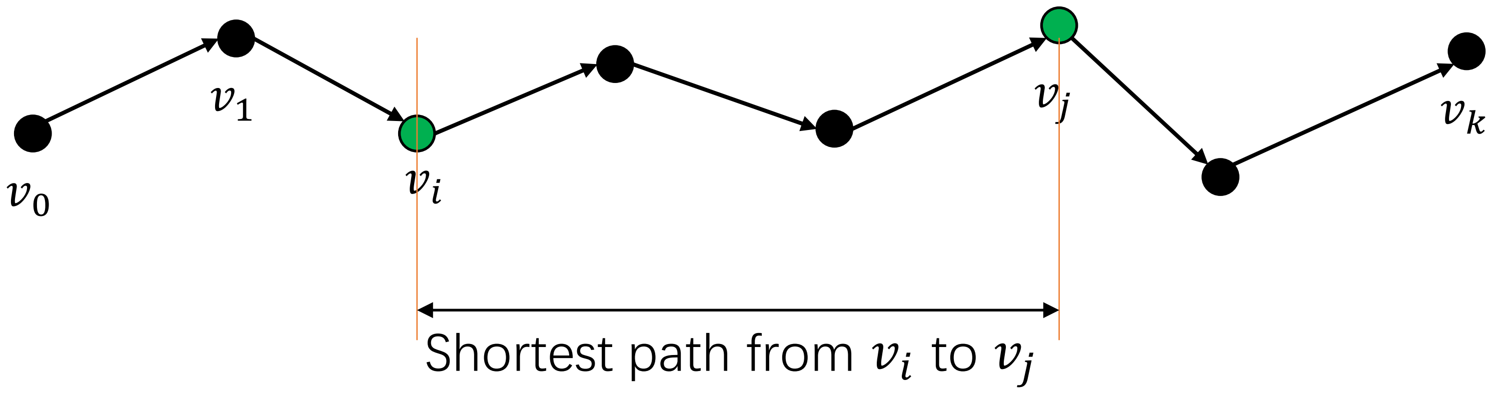
Optimal Substructure

Shortest path
from v_0 to v_k



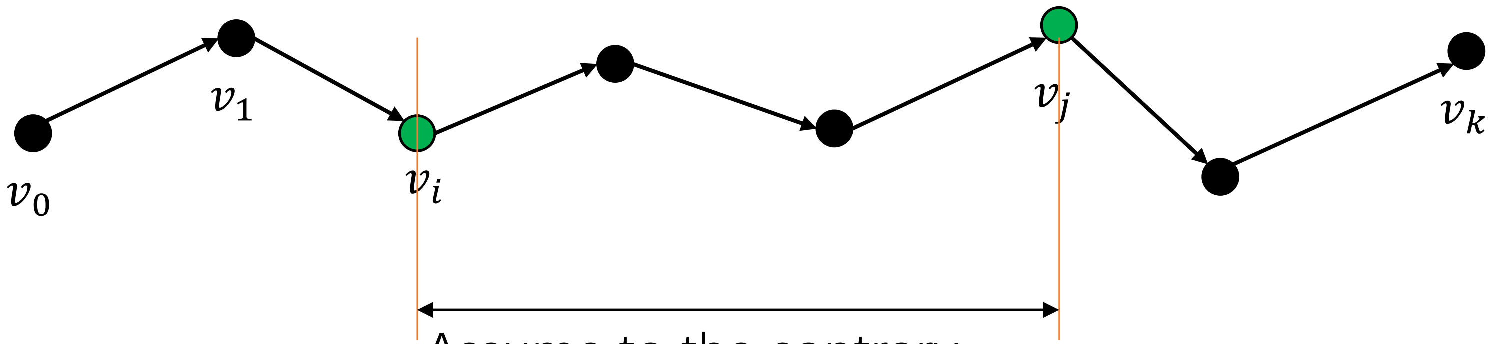
Optimal Substructure

Shortest path from v_0 to v_k

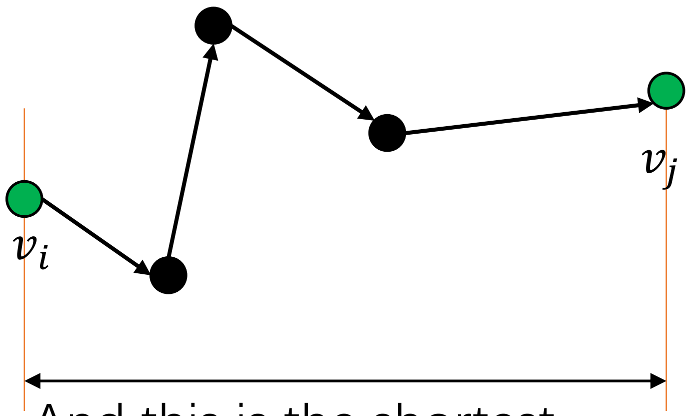


Optimal Substructure

Shortest path from v_0 to v_k



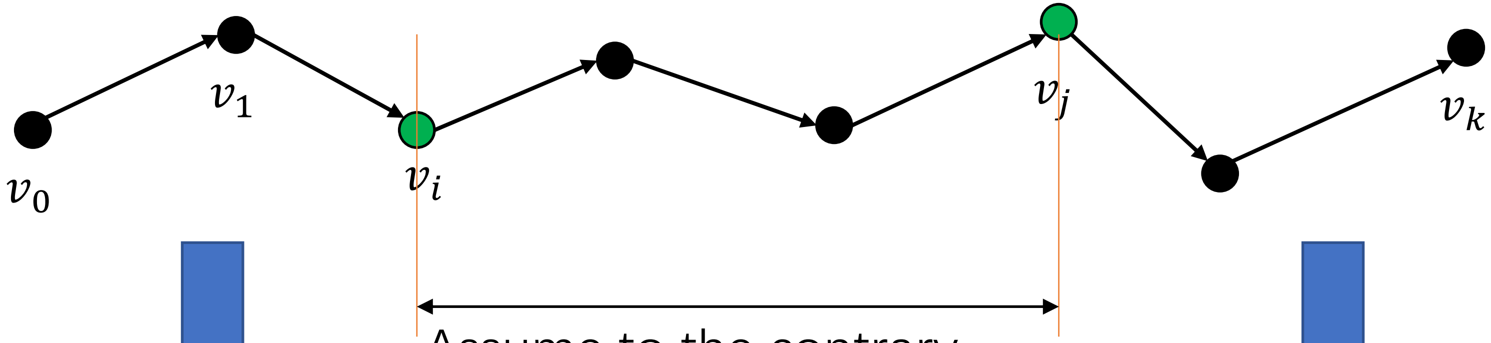
Assume to the contrary that this is not the shortest path from v_i to v_j



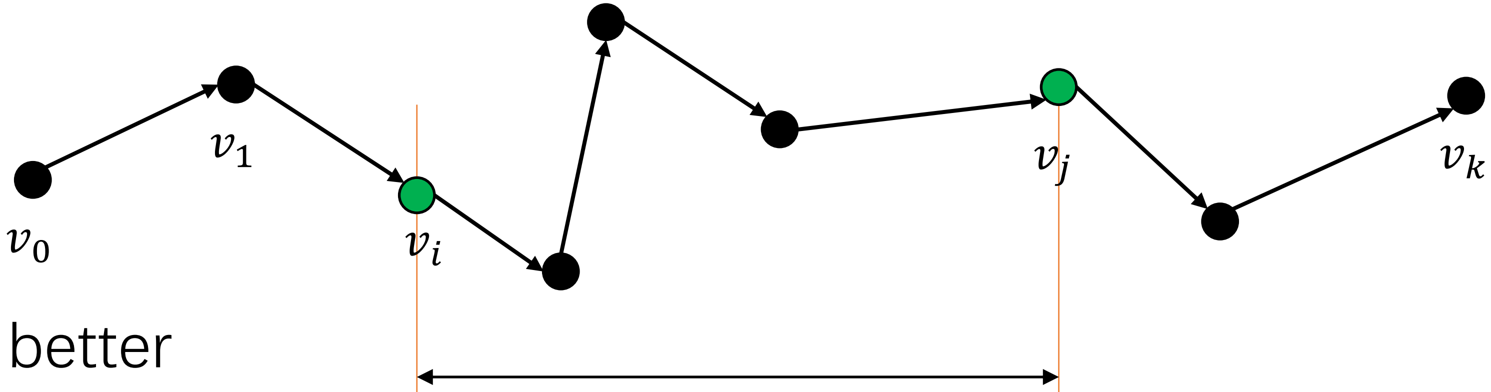
And this is the shortest path from v_i to v_j

Optimal Substructure

Shortest path from v_0 to v_k



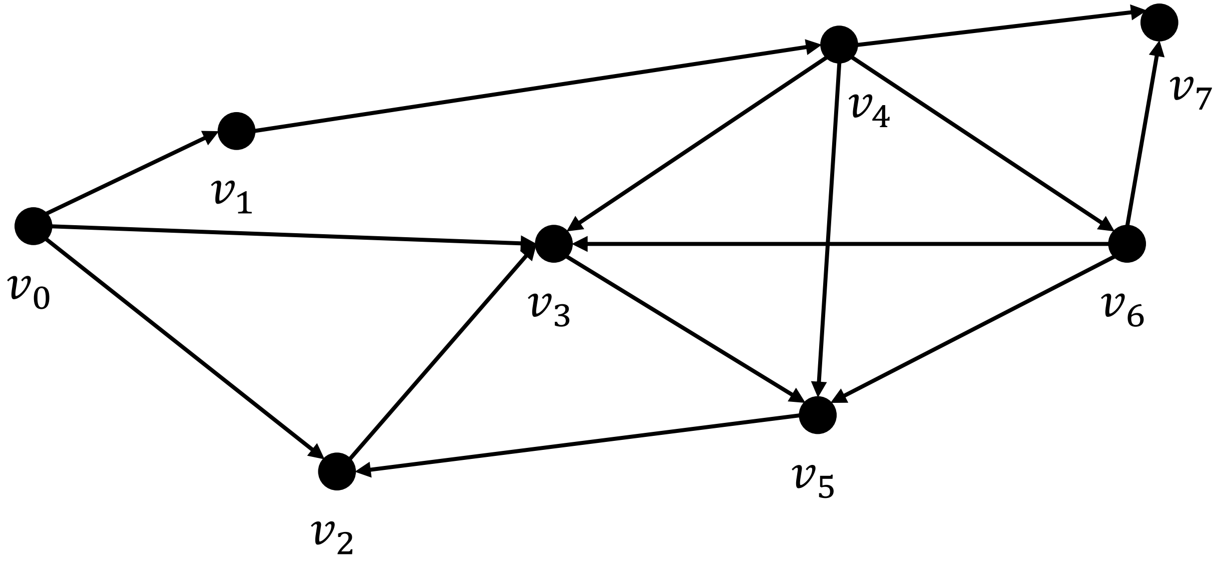
Assume to the contrary that this is not the shortest path from v_i to v_j



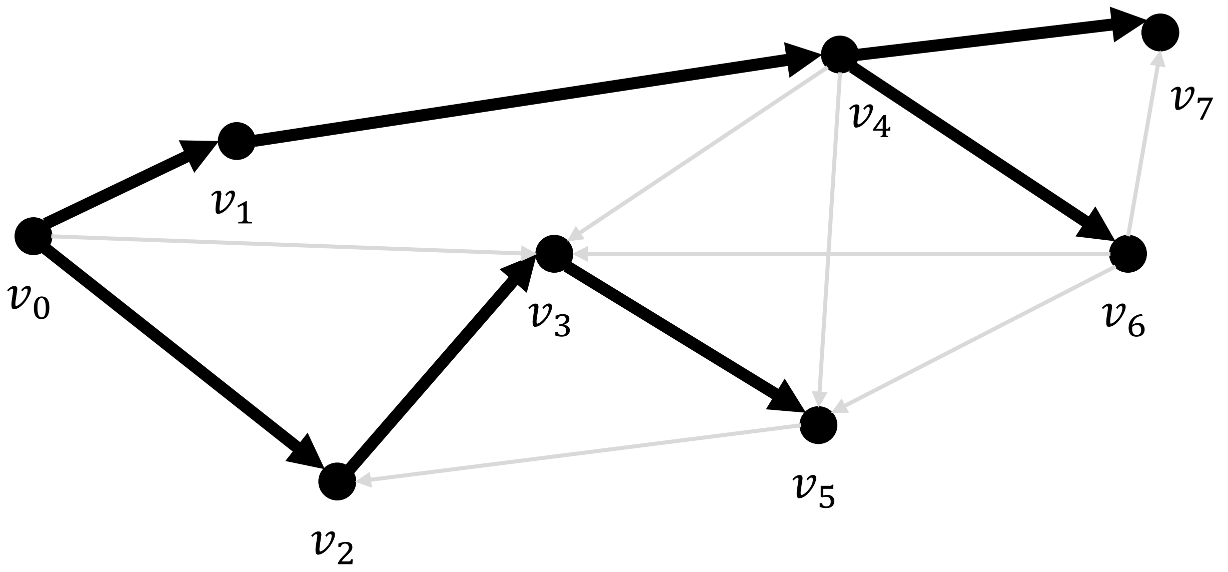
And this is the shortest path from v_i to v_j

Then this new path is better than the optimal answer from v_0 to v_k which is a contradiction

The “shortest-path tree” for SSSP



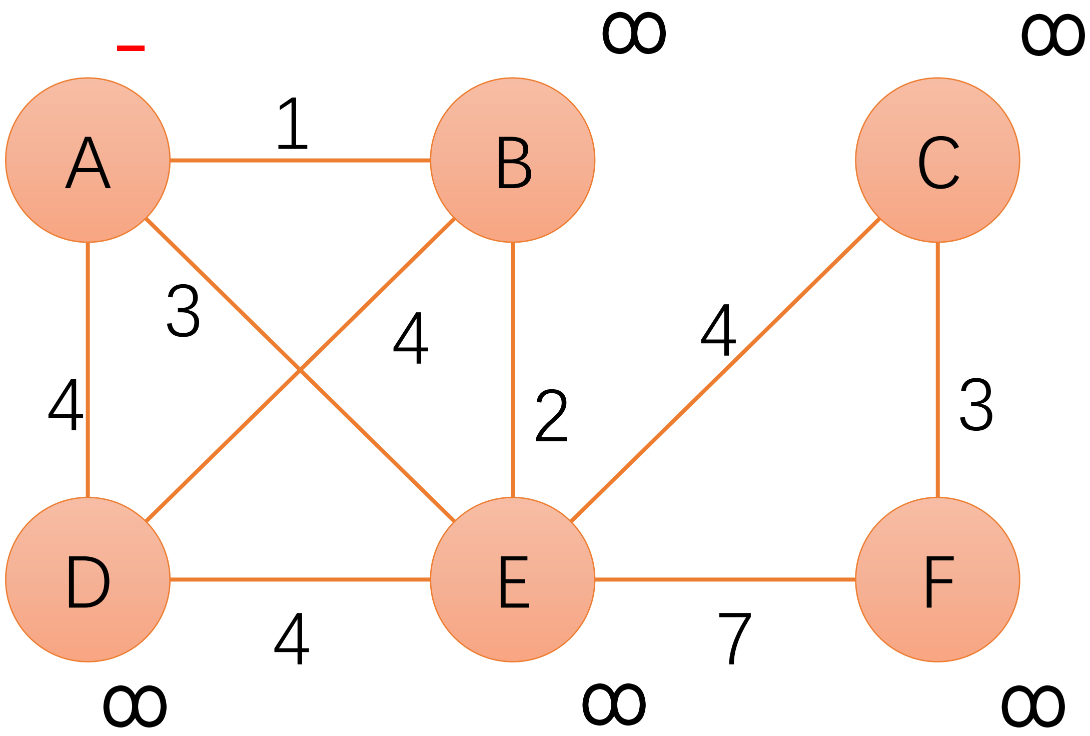
The “shortest-path tree” for SSSP



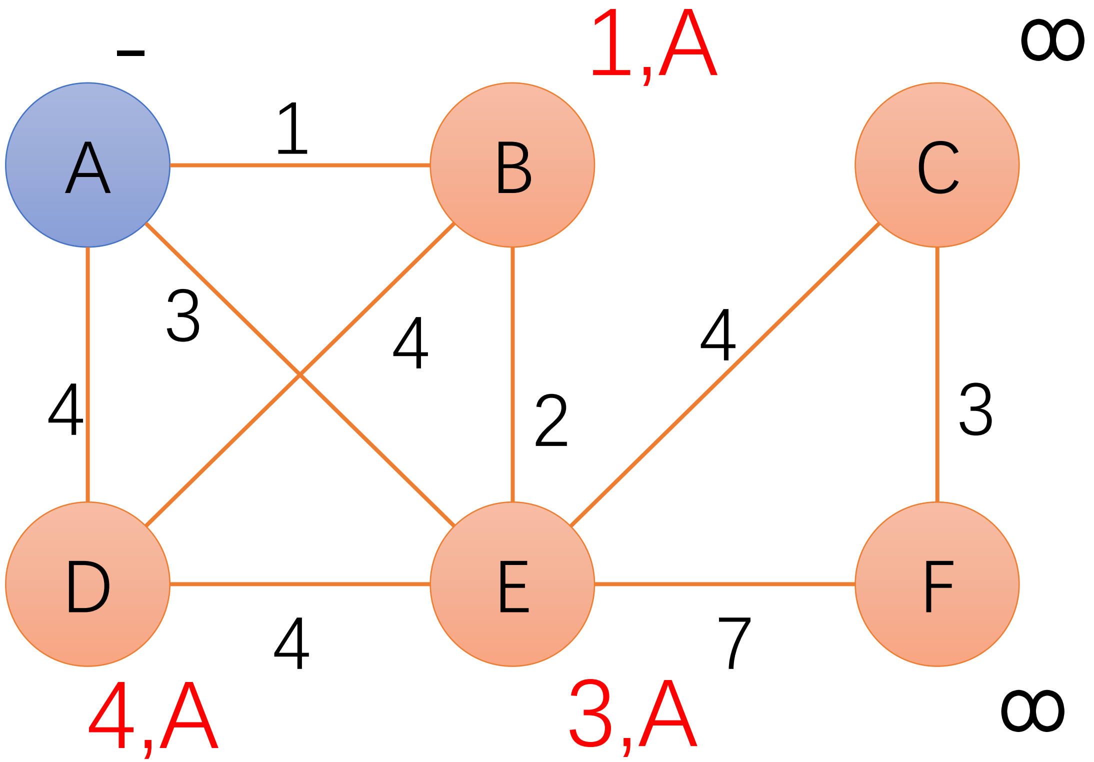
- The shortest-paths form a tree from the source
- Each tree node stores the distance (the SSSP distance to vertex), and optionally the predecessor that can be used to retrieve the path

Dijkstra's algorithm

Prim’s MST



Prim’s MST



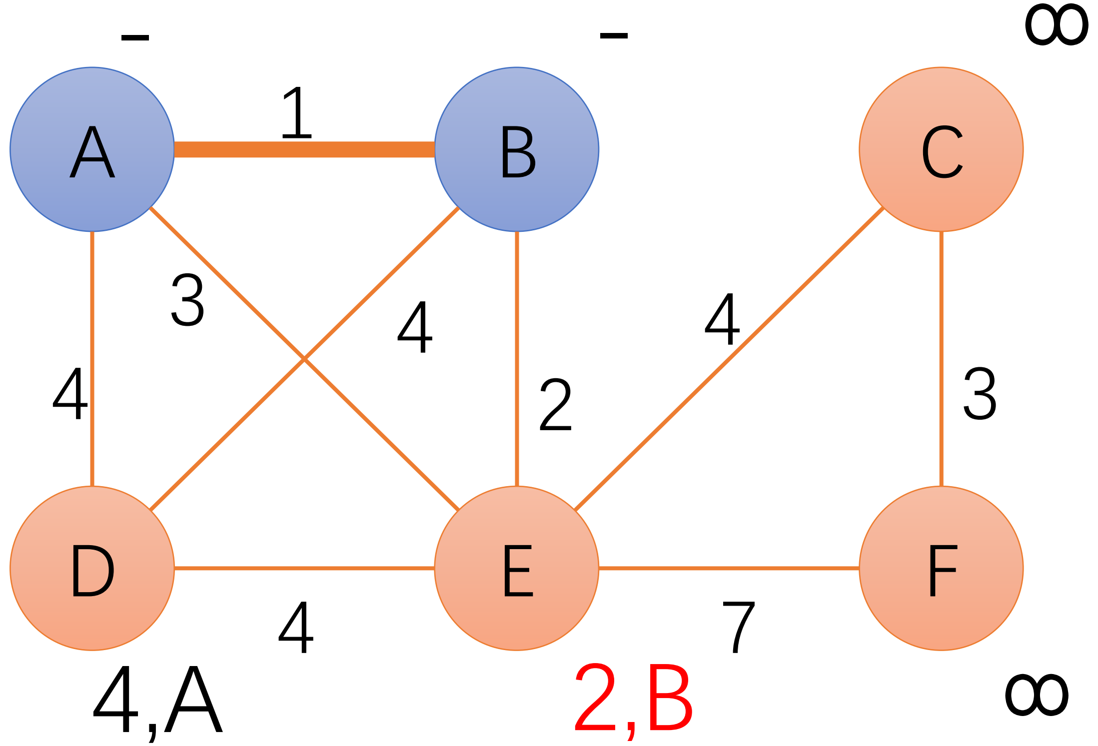
Active Set:

(B, 1)

(E, 3)

(D, 4)

Prim's MST

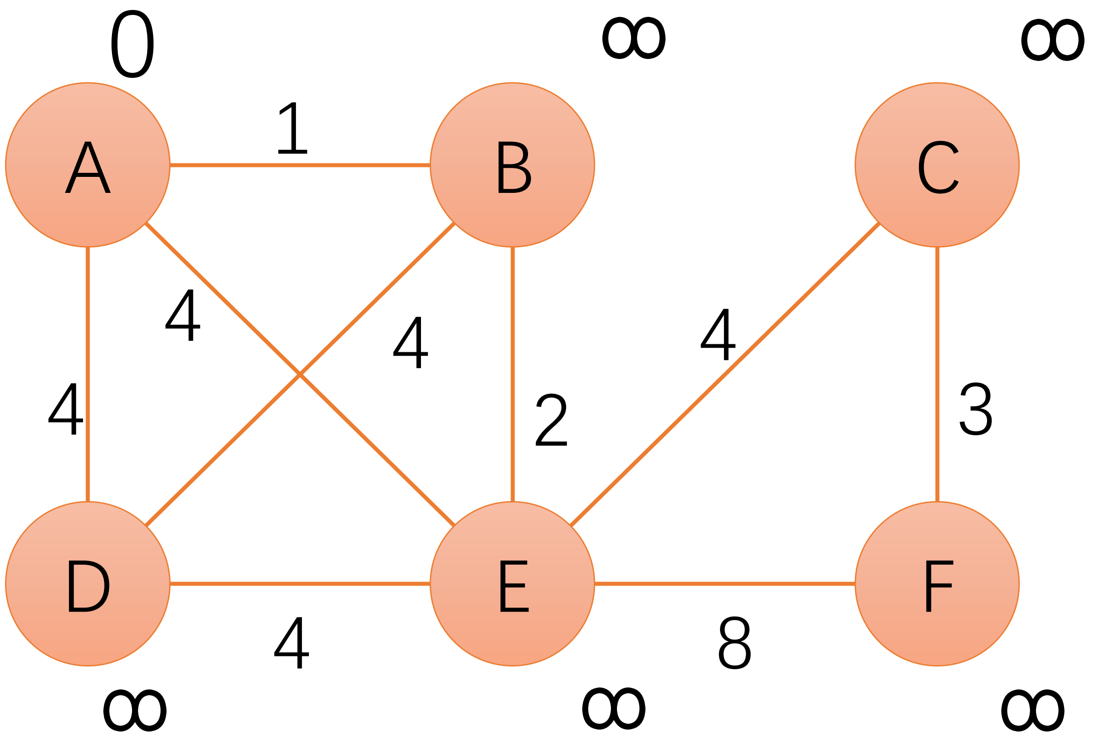


Active Set:

(E, 2)

(D, 4)

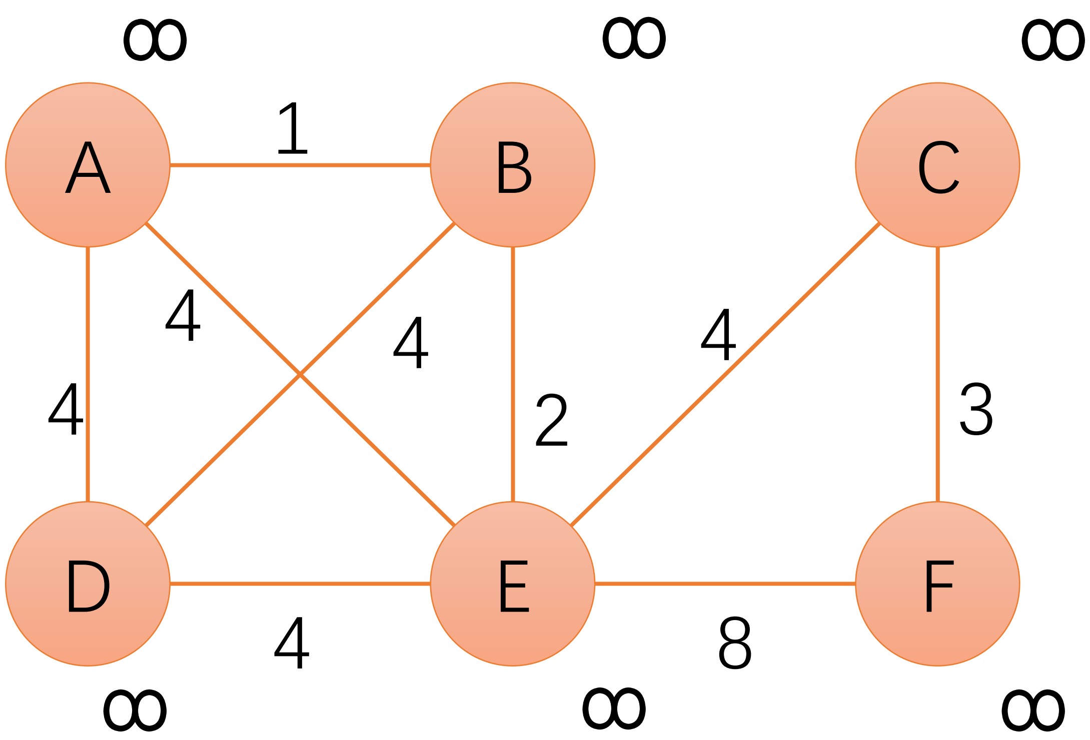
Which one must be the shortest distance?



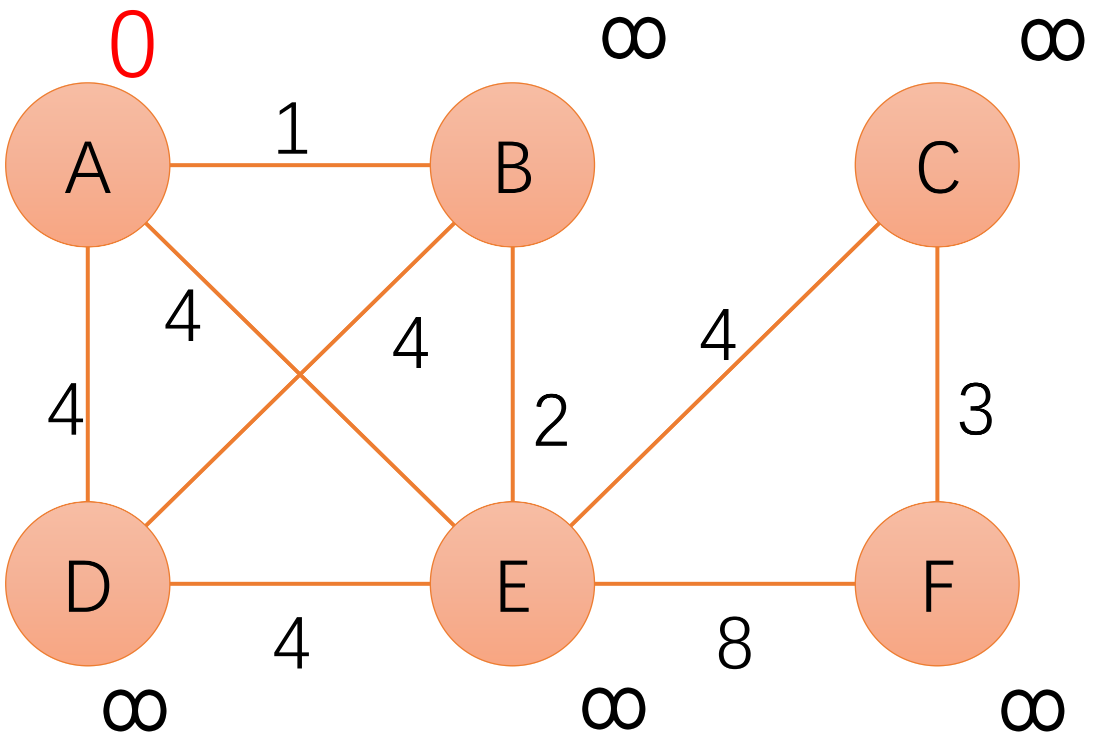
Dijkstra's Algorithm (greedy, similar to Prim's)

- Start from the source node and mark it as visited, and relax the neighbors using cur_dis+edge_dis
- Find the node with the lowest weight, marked it as settled and relax the neighbors
- Repeat until all the n nodes are settled
- (Only work for positive-weight graphs)

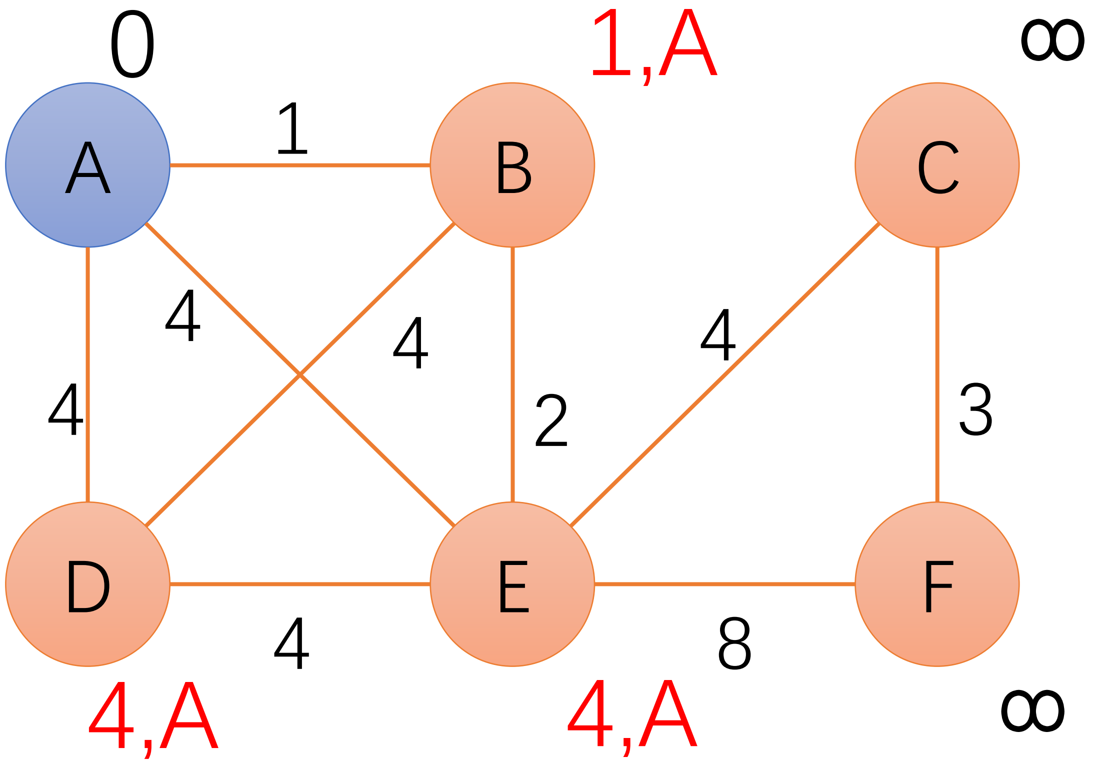
Dijkstra’s algorithm in Action



Dijkstra's algorithm in Action



Dijkstra’s algorithm in Action



Active Set:

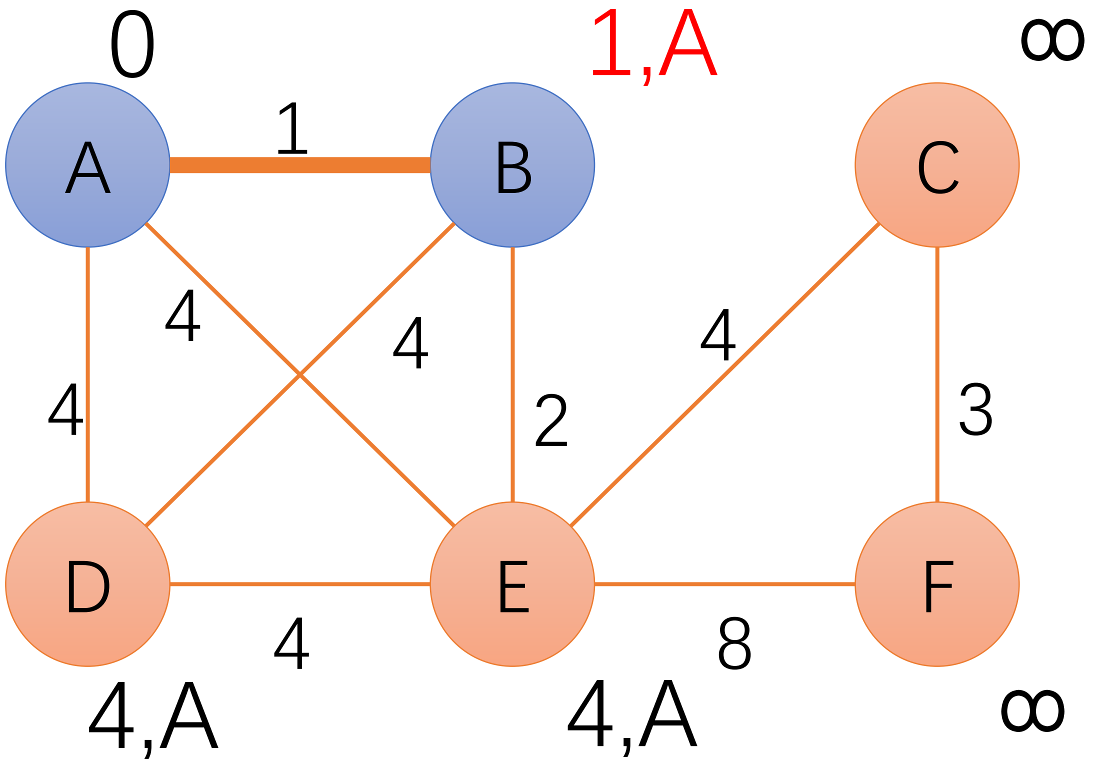
(B, 1)

(E, 4)

(D, 4)

Relax: if u is settled and v is u’s neighbor, then $\delta(v) = \min\{\delta(v), \delta(u) + w(u,v)\}$

Dijkstra’s algorithm in Action

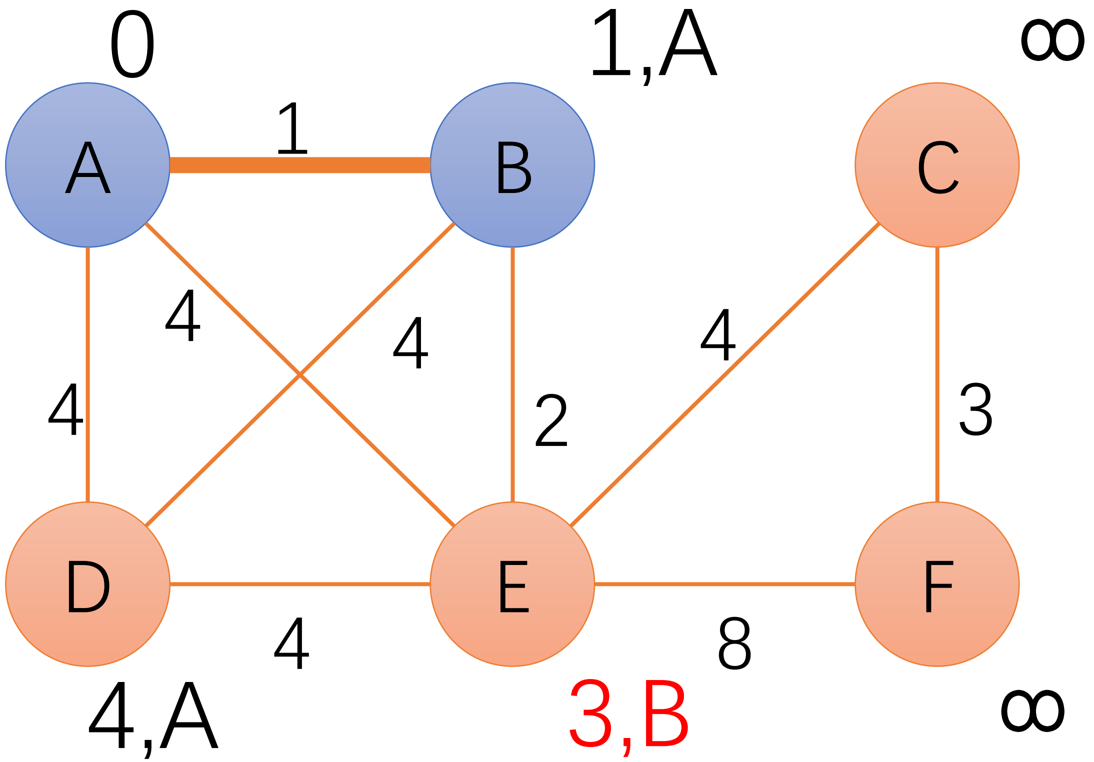


Active Set:

(E, 4)

(D, 4)

Dijkstra's algorithm in Action

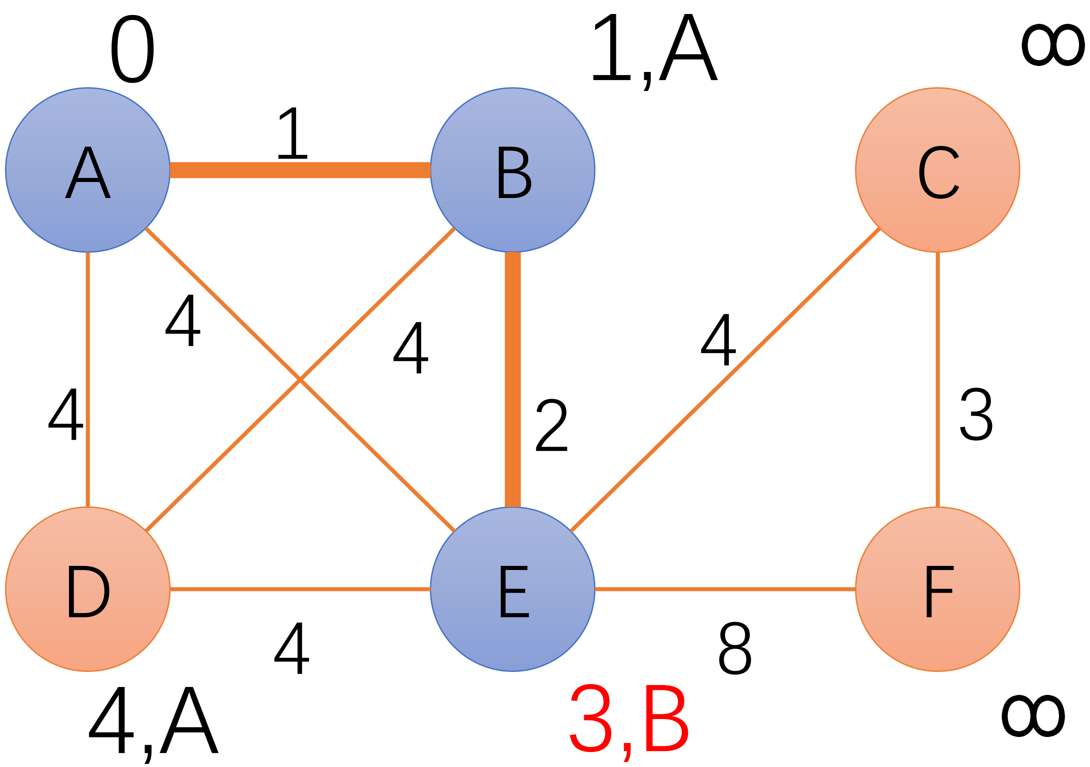


Active Set:

(E, 3)

(D, 4)

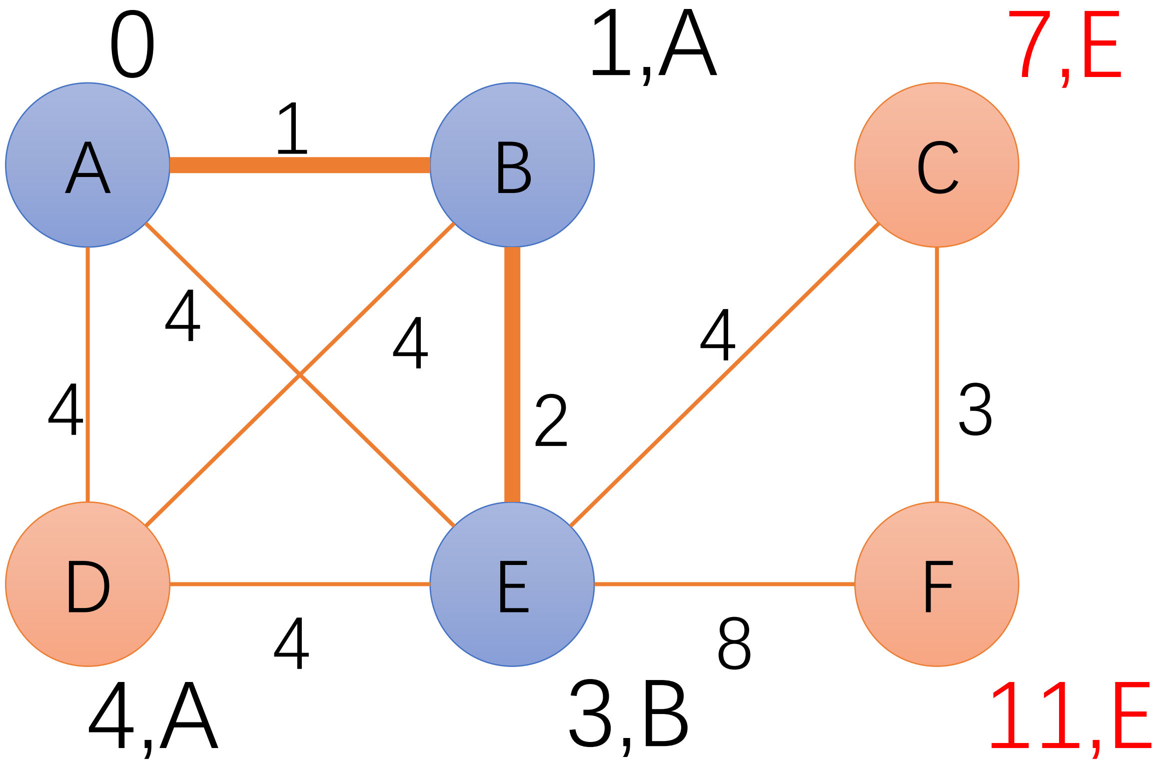
Dijkstra’s algorithm in Action



Active Set:

(D, 4)

Dijkstra’s algorithm in Action



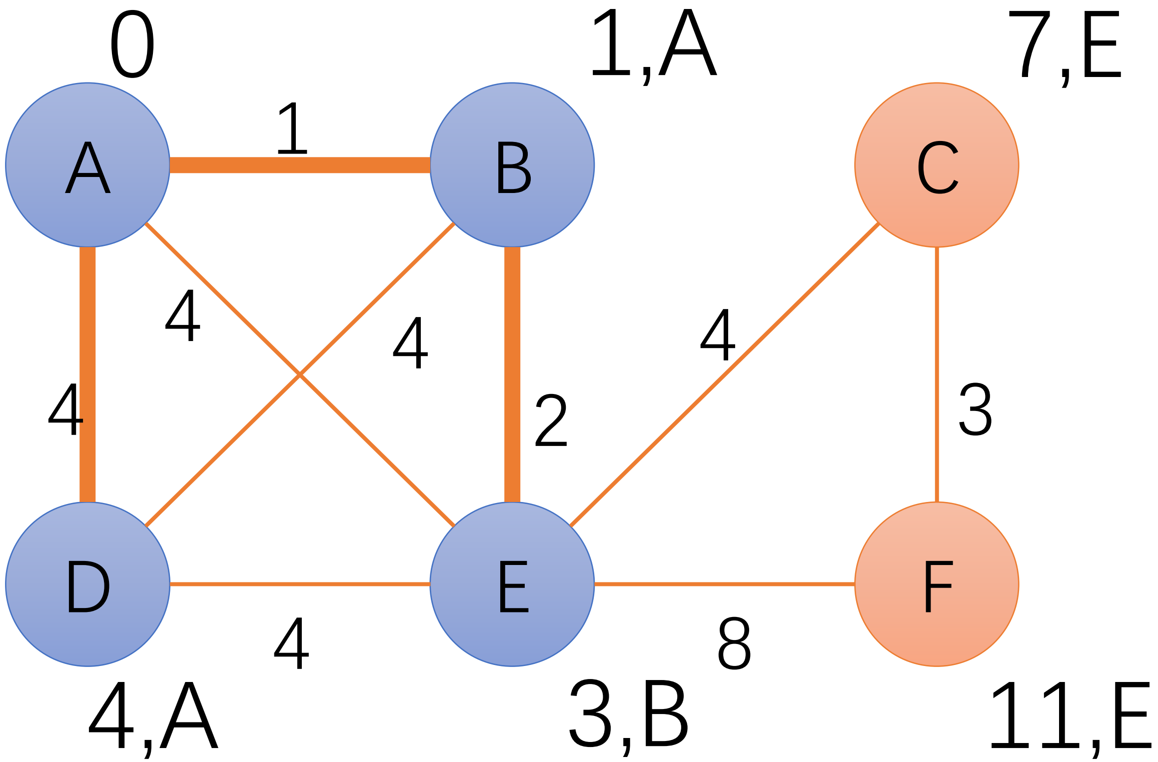
Active Set:

(D, 4)

(C, 7)

(F, 11)

Dijkstra's algorithm in Action

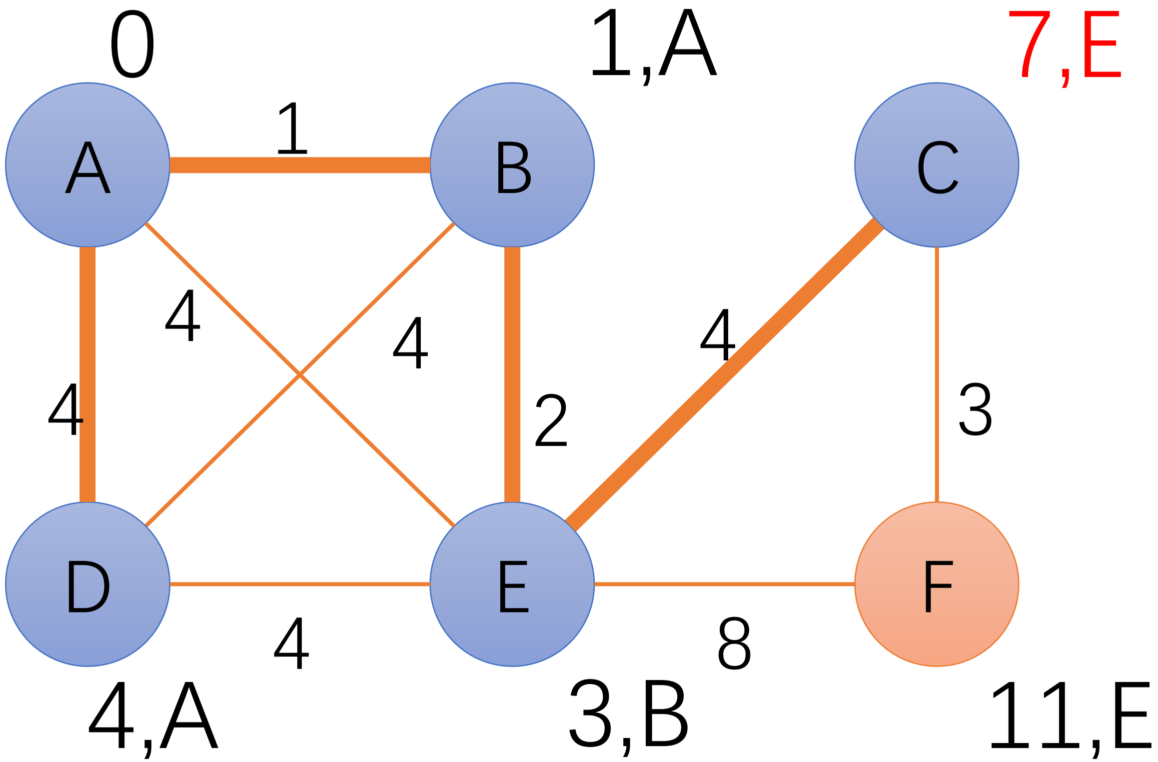


Active Set:

(C, 7)

(F, 11)

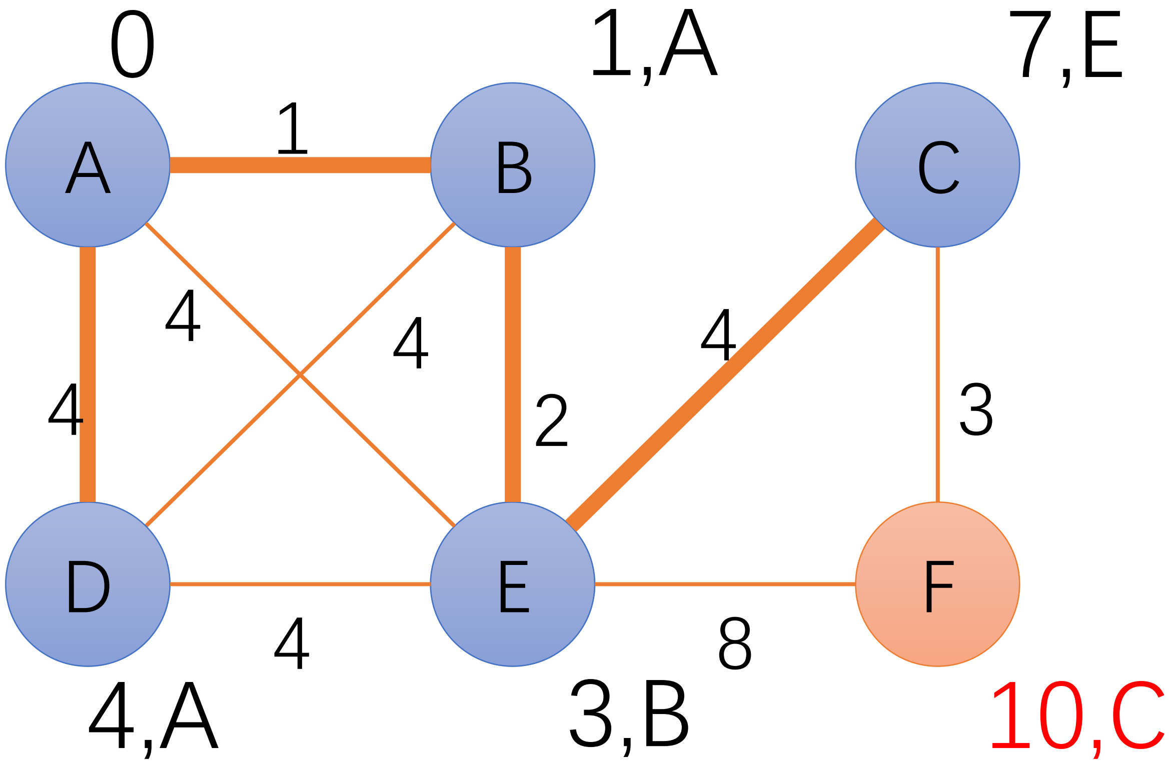
Dijkstra's algorithm in Action



Active Set:

(F, 11)

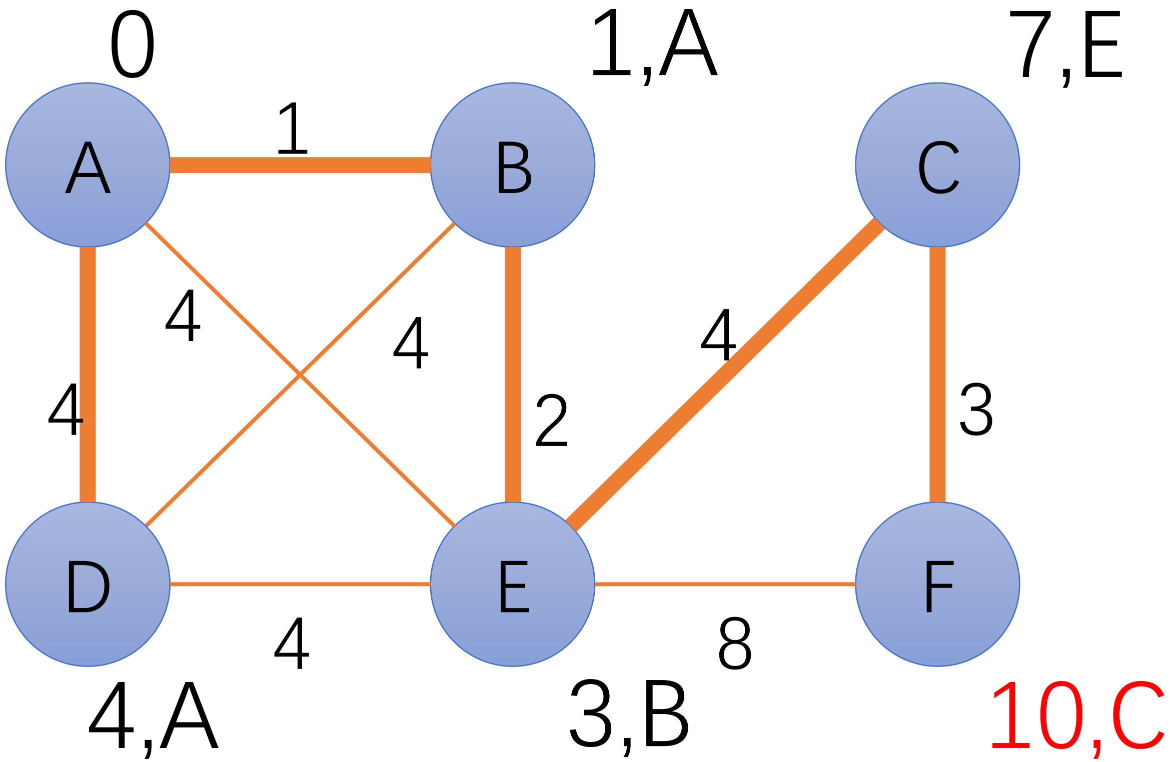
Dijkstra's algorithm in Action



Active Set:

(F, 10)

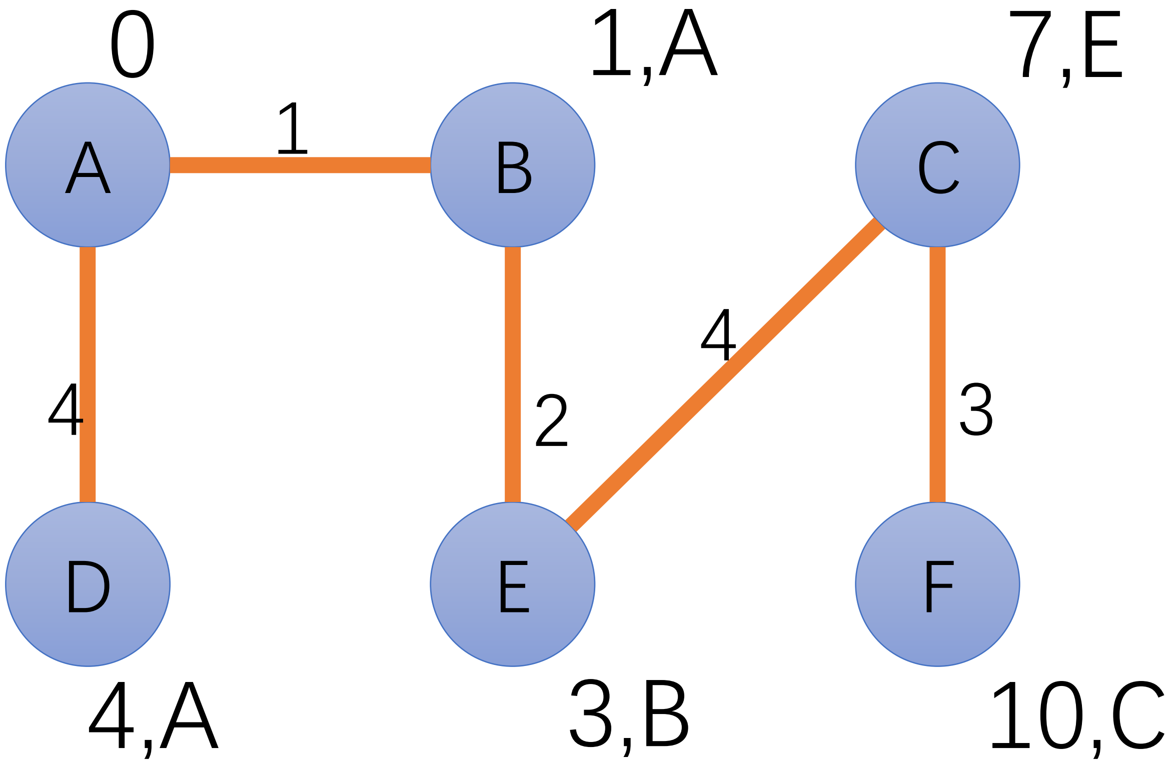
Dijkstra's algorithm in Action



Active Set:

–

The final shortest-path tree from A



Dijkstra(G, w, s)

- $\delta(u) = \infty$ for all $u \in V$, 0 for $\delta(s)$
- $S = \emptyset$
- $Q = \{s\}$
- **while** $Q \neq \emptyset$
  - $u =$ Extract-Min(Q)
  - $S = S \cup \{u\}$
  - **for each** $\boldsymbol{v} \in N(u)$
    - $\delta(v) = \min\{\delta(v), \delta(u) + w(s, v)\}$
    - (update in $\boldsymbol{Q}$)

- $\boldsymbol{\delta(u)}$: tentative distance
- $\boldsymbol{S}$: settled set
- $\boldsymbol{Q}$: priority queue
- $\boldsymbol{w(u, v)}$: weight of edge from $\boldsymbol{u}$ to $\boldsymbol{v}$
- $\boldsymbol{N(u)}$: neighbor set of $\boldsymbol{u}$

Dijkstra(G, w, s): using a flat array

- $\delta(u) = \infty$ for all $u \in V$, 0 for $\delta(s)$
- $S = \emptyset$
- $Q = \{s\}$
- **while** $Q \neq \emptyset$
  - $u =$ Extract-Min(Q)
  - $S = S \cup \{u\}$
  - **for each** $\boldsymbol{v} \in N(u)$
    - $\delta(v) = \min\{\delta(v), \delta(u) + w(s, v)\}$
    - (update in $\boldsymbol{Q}$)

$\boldsymbol{O}(\boldsymbol{n})$ per operation

$\boldsymbol{O}(1)$ per operation

- $\boldsymbol{\delta}(\boldsymbol{u})$: tentative distance
- $\boldsymbol{S}$: settled set
- $\boldsymbol{Q}$: priority queue
- $\boldsymbol{w}(\boldsymbol{u}, \boldsymbol{v})$: weight of edge from $\boldsymbol{u}$ to $\boldsymbol{v}$
- $\boldsymbol{N}(\boldsymbol{u})$: neighbor set of $\boldsymbol{u}$

$\boldsymbol{O}(\boldsymbol{m} + \boldsymbol{n}^2) = \boldsymbol{O}(\boldsymbol{n}^2)$ cost in total

Dijkstra(G, w, s): using binary heap

- $\delta(u) = \infty$ for all $u \in V$, 0 for $\delta(s)$
- $S = \emptyset$
- $Q = \{s\}$
- while $Q \neq \emptyset$
  - $u =$ Extract-Min(Q)
  - $S = S \cup \{u\}$
  - for each $v \in N(u)$
    - $\delta(v) = \min\{\delta(v), \delta(u) + w(s, v)\}$
    - (update in Q)

$\boldsymbol{O}(\log \boldsymbol{n})$ per operation

$\boldsymbol{O}(\log \boldsymbol{n})$ per operation

- $\boldsymbol{\delta}(\boldsymbol{u})$: tentative distance
- $\boldsymbol{S}$: settled set
- $\boldsymbol{Q}$: priority queue
- $\boldsymbol{w}(\boldsymbol{u}, \boldsymbol{v})$: weight of edge from $\boldsymbol{u}$ to $\boldsymbol{v}$
- $\boldsymbol{N}(\boldsymbol{u})$: neighbor set of $\boldsymbol{u}$

Use binary heap to implement the priority queue (Chapter 6 in CLRS, taught in CS 14)

$\boldsymbol{O}\big((\boldsymbol{m} + \boldsymbol{n}) \log \boldsymbol{n}\big)$ cost in total

Dijkstra(G, w, s): using Fibonacci heap

- $\delta(u) = \infty$ for all $u \in V$, 0 for $\delta(s)$
- $S = \emptyset$
- $Q = \{s\}$
- **while** $Q \neq \emptyset$
  - $u =$ Extract-Min(Q)
  - $S = S \cup \{u\}$
  - **for each** $\boldsymbol{v} \in N(u)$
    - $\delta(v) = \min\{\delta(v), \delta(u) + w(s, v)\}$
    - (update in $\boldsymbol{Q}$)

$\boldsymbol{O}(\log \boldsymbol{n})$ per operation

$\boldsymbol{O}(1)$ per operation

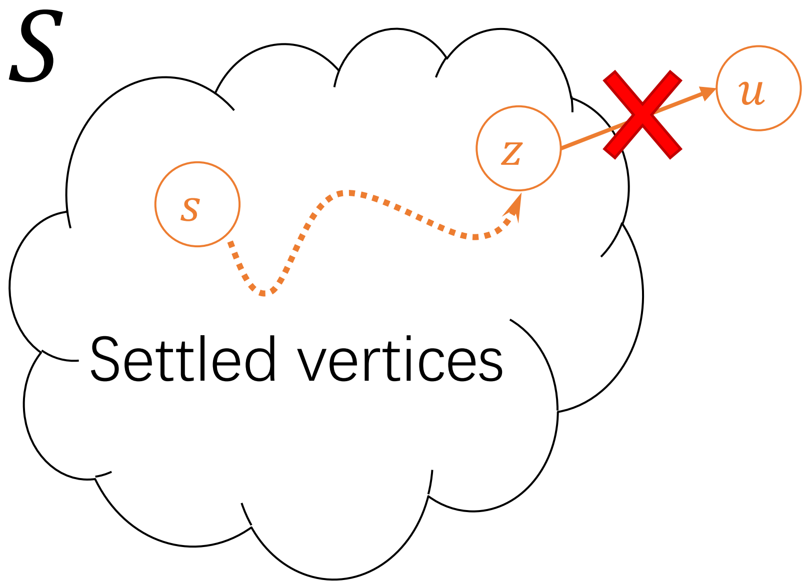
- $\boldsymbol{\delta}(\boldsymbol{u})$: tentative distance
- $\boldsymbol{S}$: settled set
- $\boldsymbol{Q}$: priority queue
- $\boldsymbol{w}(\boldsymbol{u}, \boldsymbol{v})$: weight of edge from $\boldsymbol{u}$ to $\boldsymbol{v}$
- $\boldsymbol{N}(\boldsymbol{u})$: neighbor set of $\boldsymbol{u}$

Use Fibonacci heap to implement the priority queue (Chapter 19 in CLRS)

$\boldsymbol{O}(\boldsymbol{m} + \boldsymbol{n} \log \boldsymbol{n})$ cost in total

Why is Dijkstra correct?

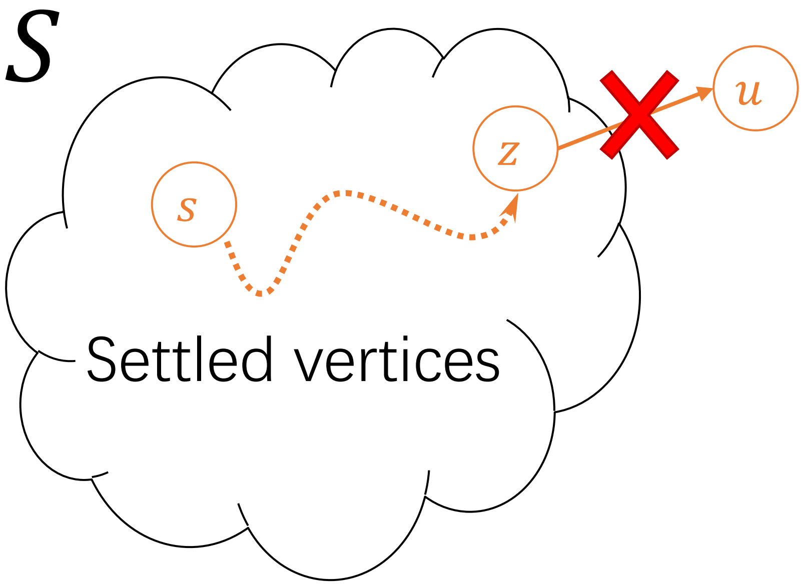
- Invariant: the closest vertex find in each round is finalized (settled)
- The current tentative distance is already correct! – no need to wait



What Dijkstra's find

Consider the second last vertex on the shortest path to u

- **If it is a settled vertex z (via an optimal path $s, \ldots, \boldsymbol{z}, \boldsymbol{u}$)**



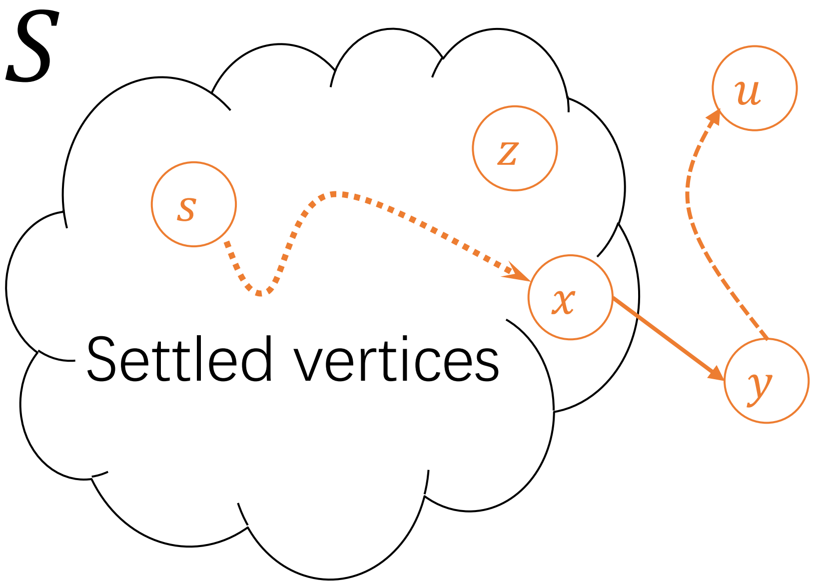
What Dijkstra's find

Based on optimal substructure,

$$\delta(z) = \text{SP}(z)$$
$$\Downarrow$$
$$\delta(u) = \delta(z) + w(z,u)$$
$$= \text{SP}(z) + w(z,u) = \text{SP}(u)$$

The second last vertex to u is not settled

- **Optimal shortest path is via $s\ \ldots x, y, \ldots, u$ that y is the first vertex not in S**



Real shortest path

x was settled and relaxed y

$$\Downarrow$$

$$\delta(y) = \mathrm{SP}(y)$$

y is closer than u

$$\Downarrow$$

$$\delta(y) = \mathrm{SP}(y) < \delta(u)$$

Dijkstra should extract y instead of u, which leads to a contradiction

Bellman-Ford Algorithm

Why Bellman-Ford Algorithm?

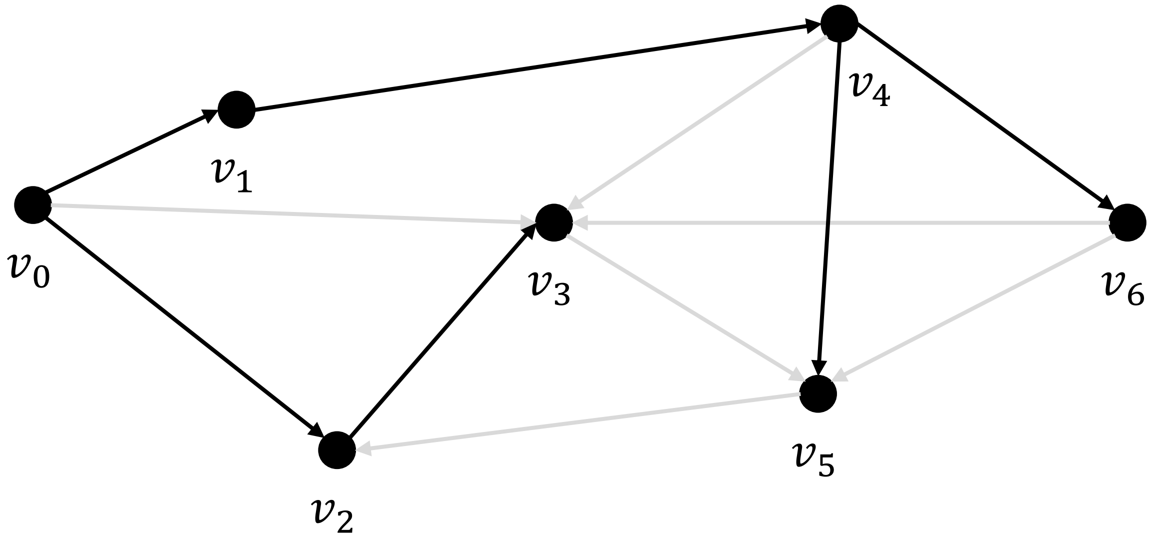
- Bellman-Ford Algorithm is a dynamic programming algorithm and is extremely simple (even simpler than Dijkstra)

- It has a higher cost than Dijkstra, but can handle graphs with negative edge weights

- It can be parallelized

Bellman-Ford is a dynamic programming algorithm

- Let $D_{i,k}$ indicate the shortest distance from source s to vertex i using no more than k hops (number of edges)



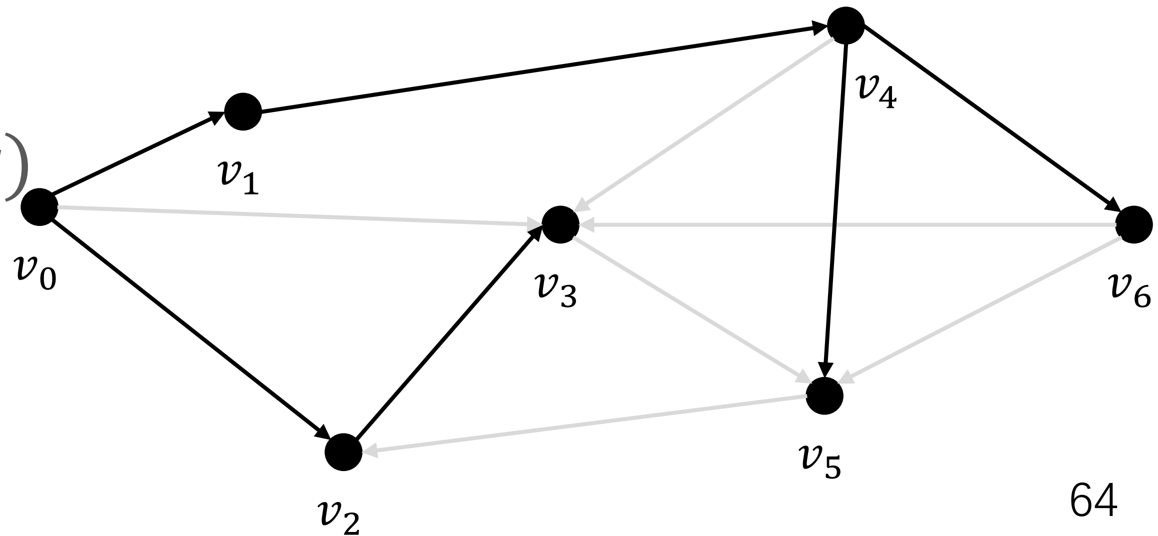
Bellman-Ford is a dynamic programming algorithm

- Let $D_{i,k}$ indicate the shortest distance from source s to vertex i using no more than k hops (number of edges)

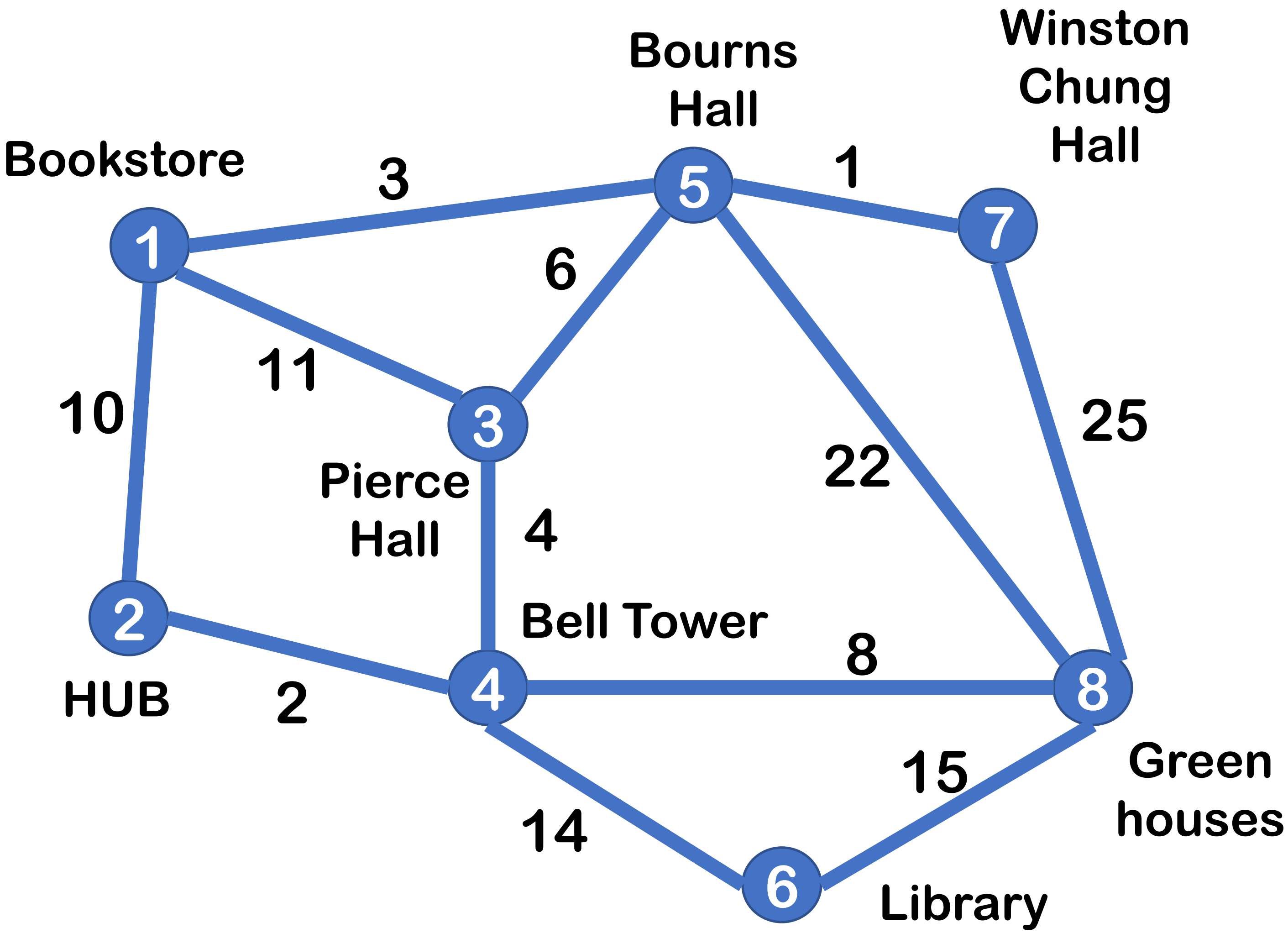
- Consider the last edge:

$$D_{i,k} = \min \begin{cases} D_{i,k-1} \\ \min_{(j,i)\in E} \{D_{j,k-1} + w(j,i)\} \end{cases}$$

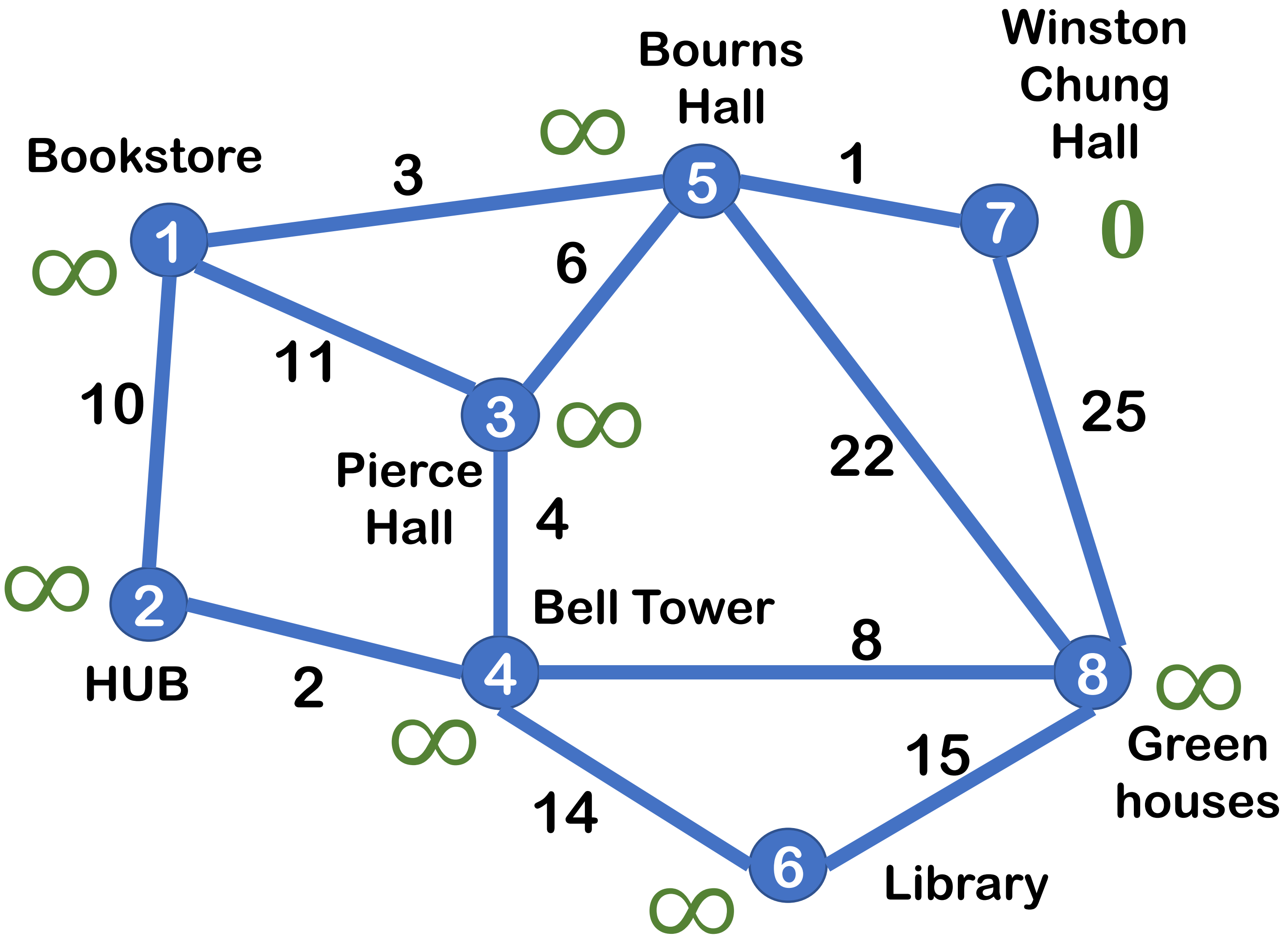
- Boundaries: $D_{s,0} = 0$, $D_{i,0} = \infty$ $(i \neq s)$
- Final answer to vertex i is $D_{i,n-1}$



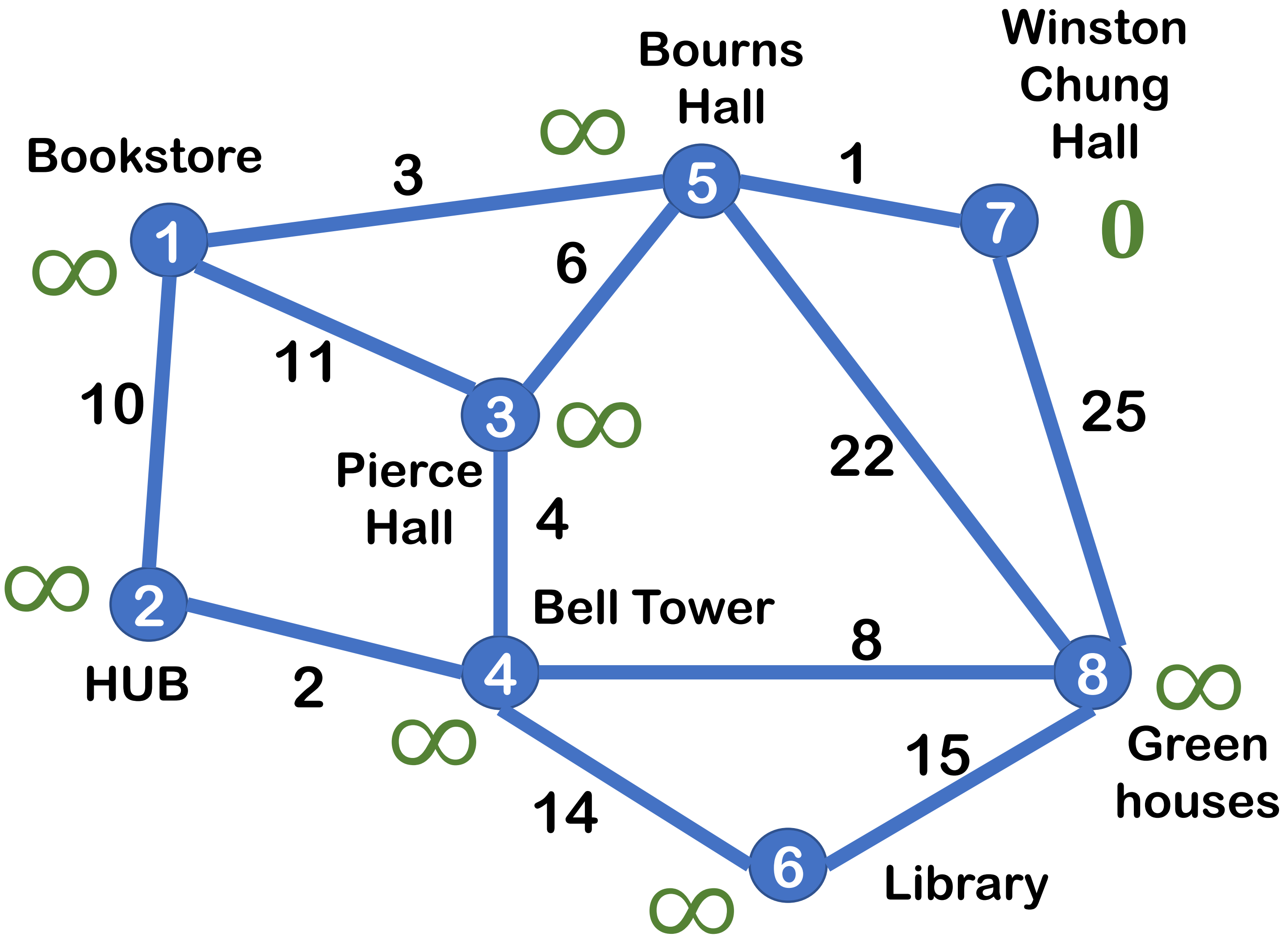
Walking in the campus as an example



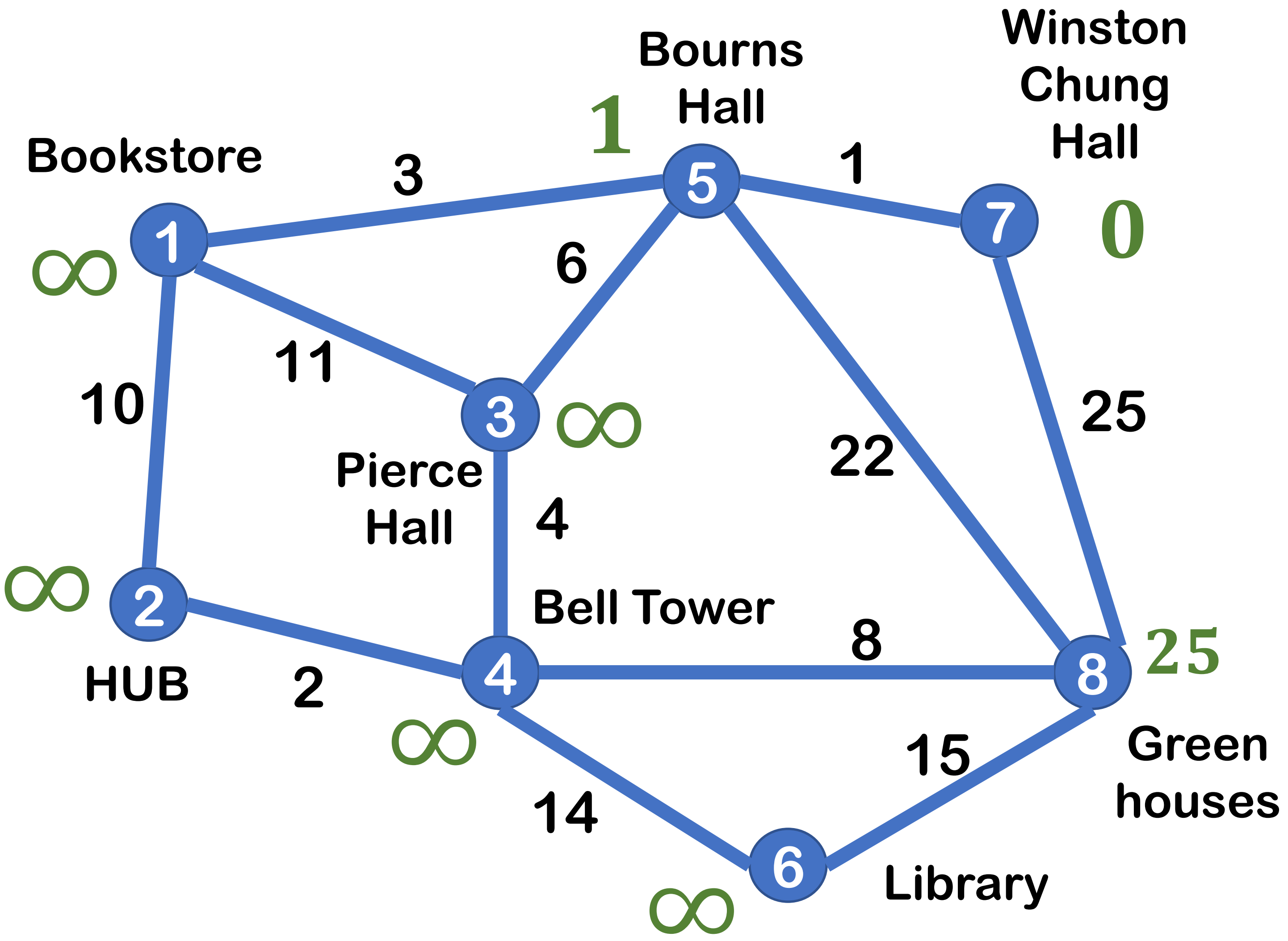
Step 0: boundaries



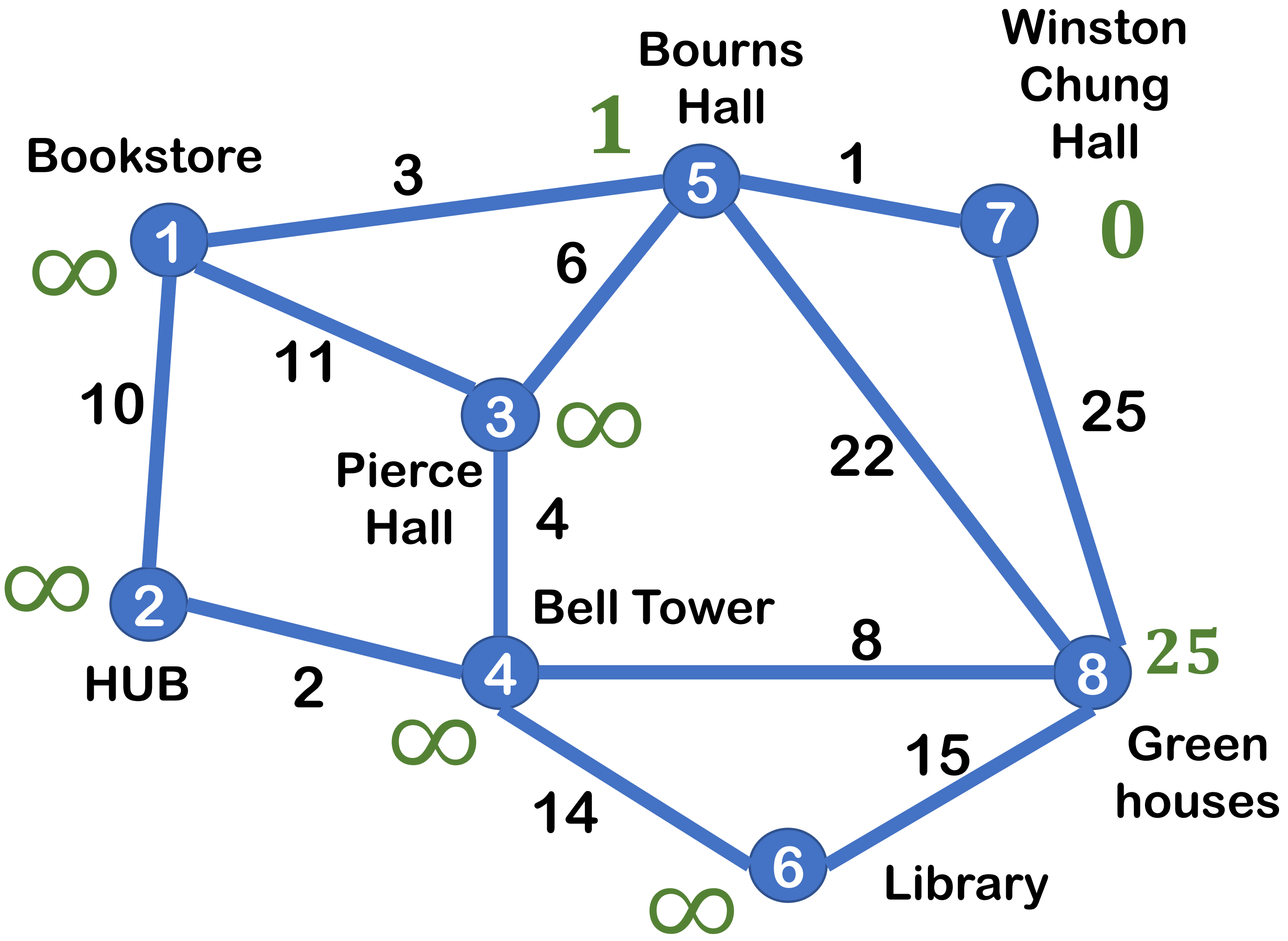
Step 1: Given $D_{i,0}$, to compute $D_{i,1}$



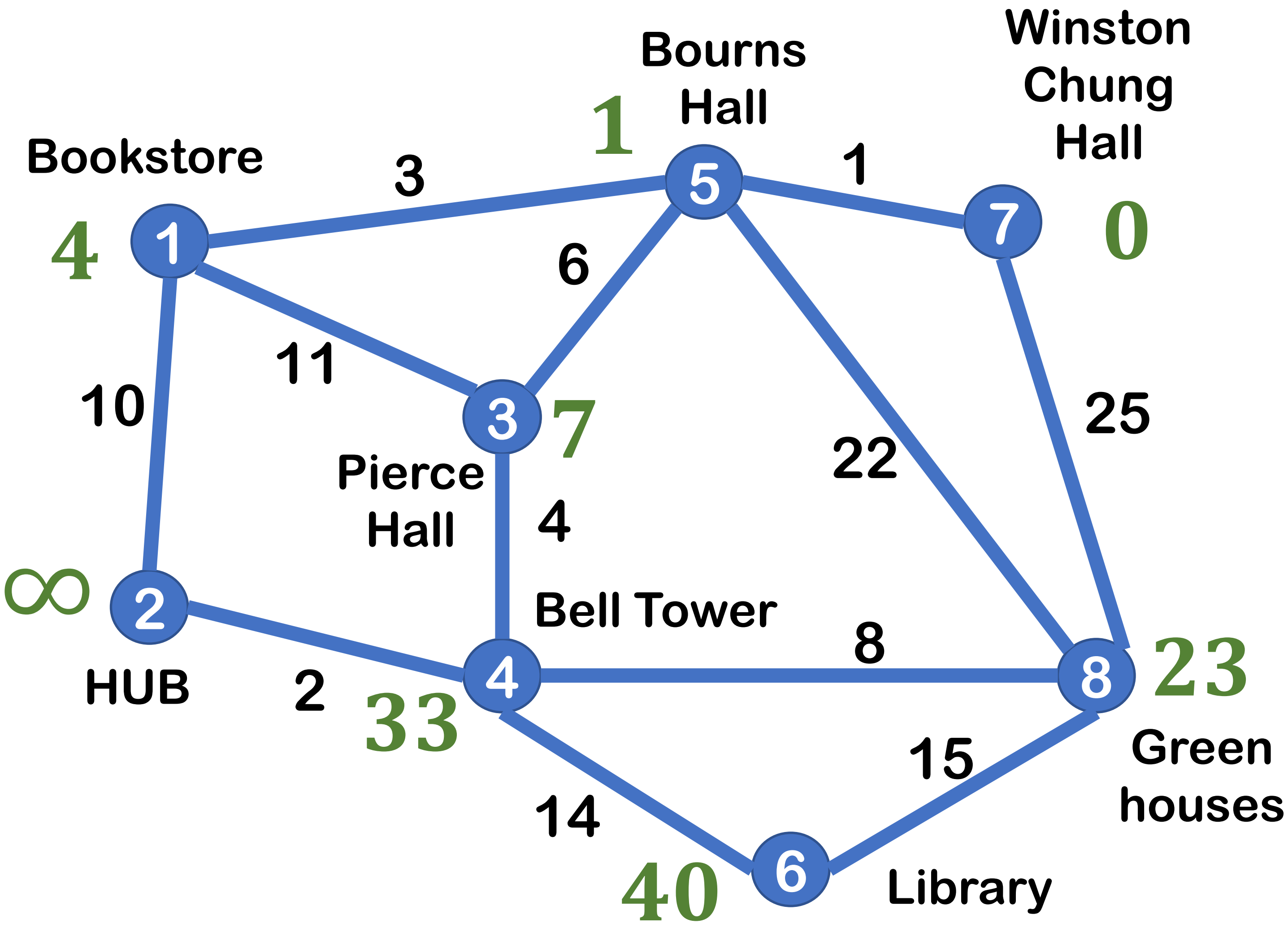
Step 1: after all updates



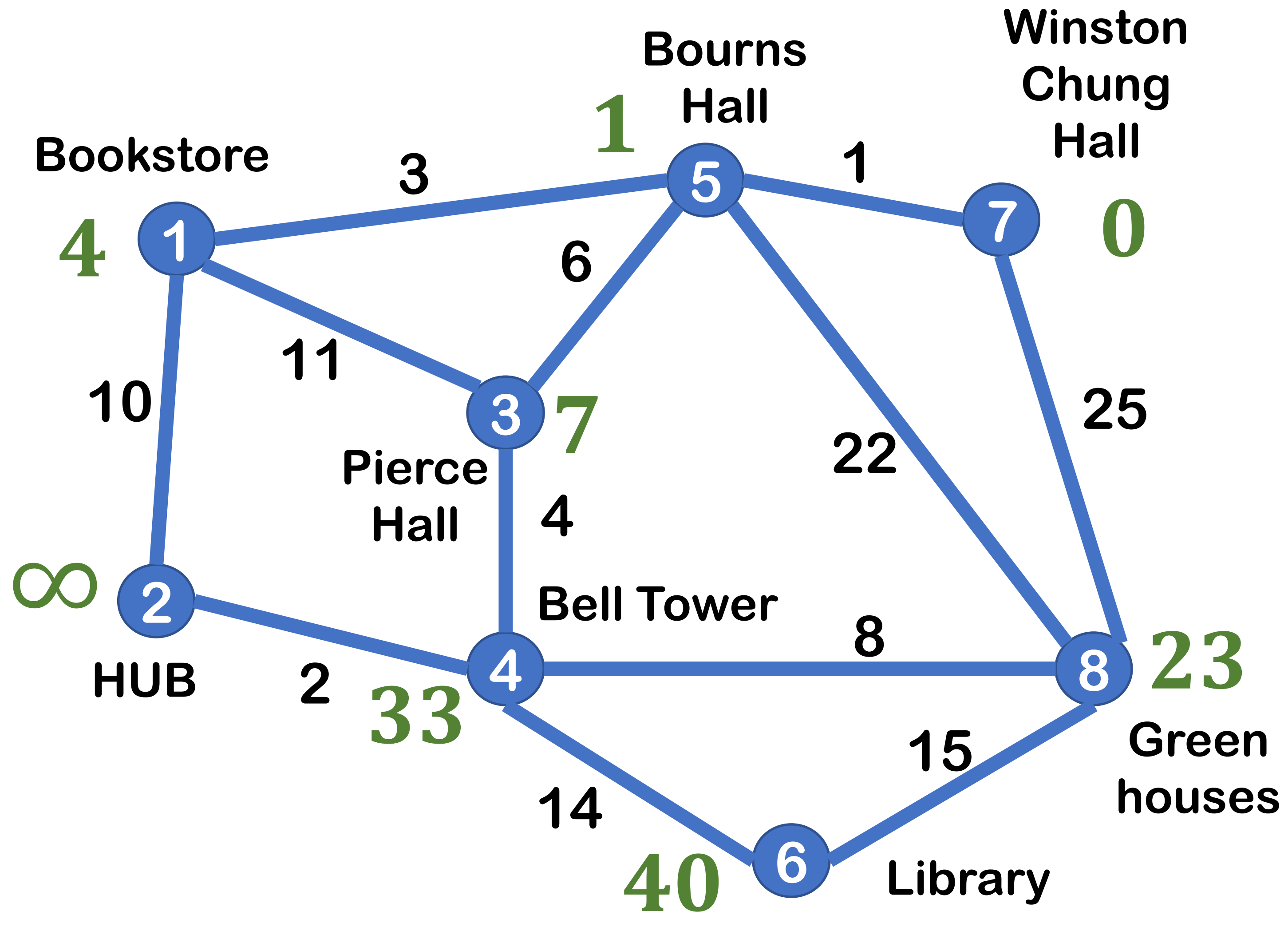
Step 2: Given $D_{i,1}$, to compute $D_{i,2}$



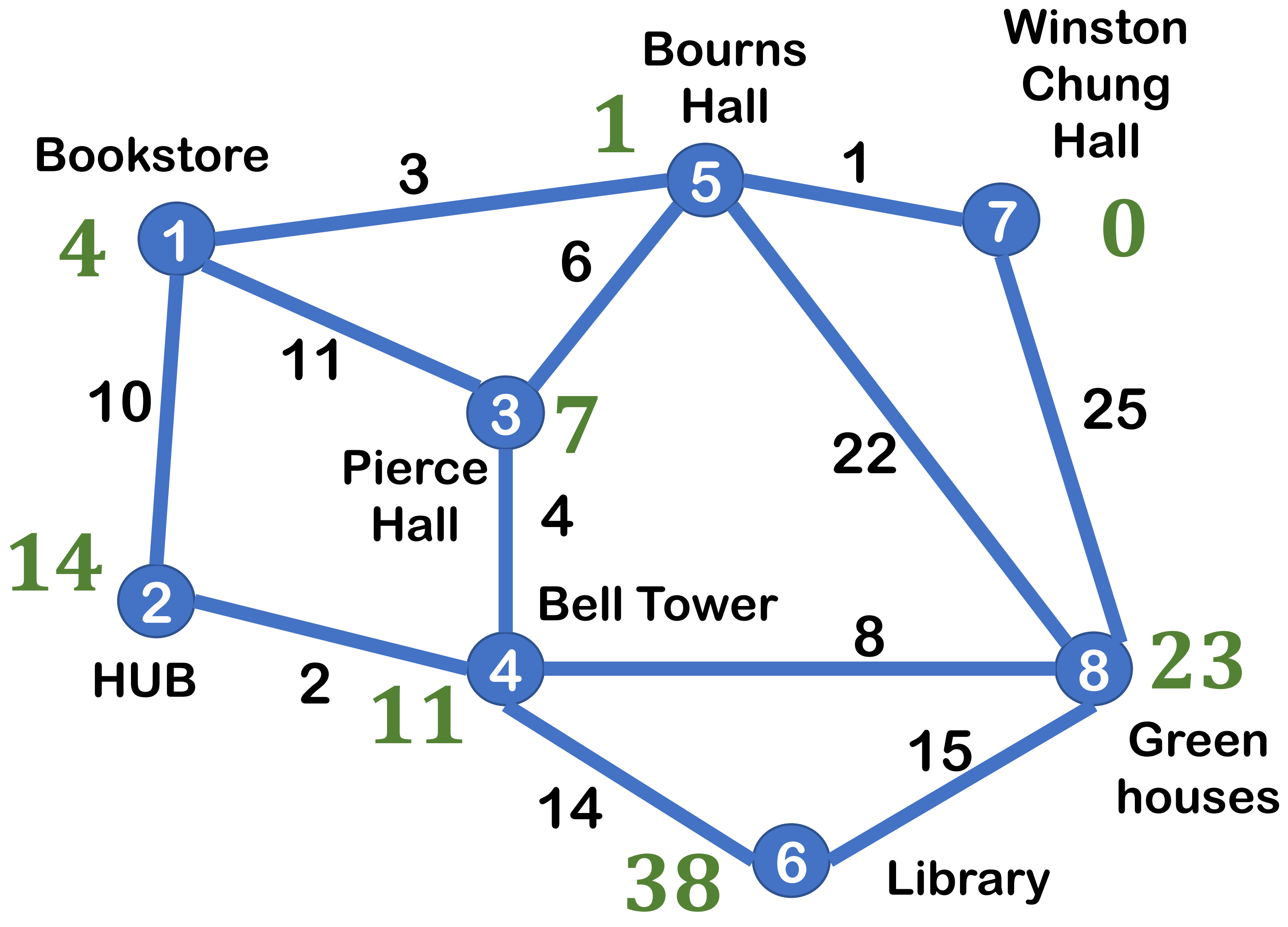
Step 2: after all updates



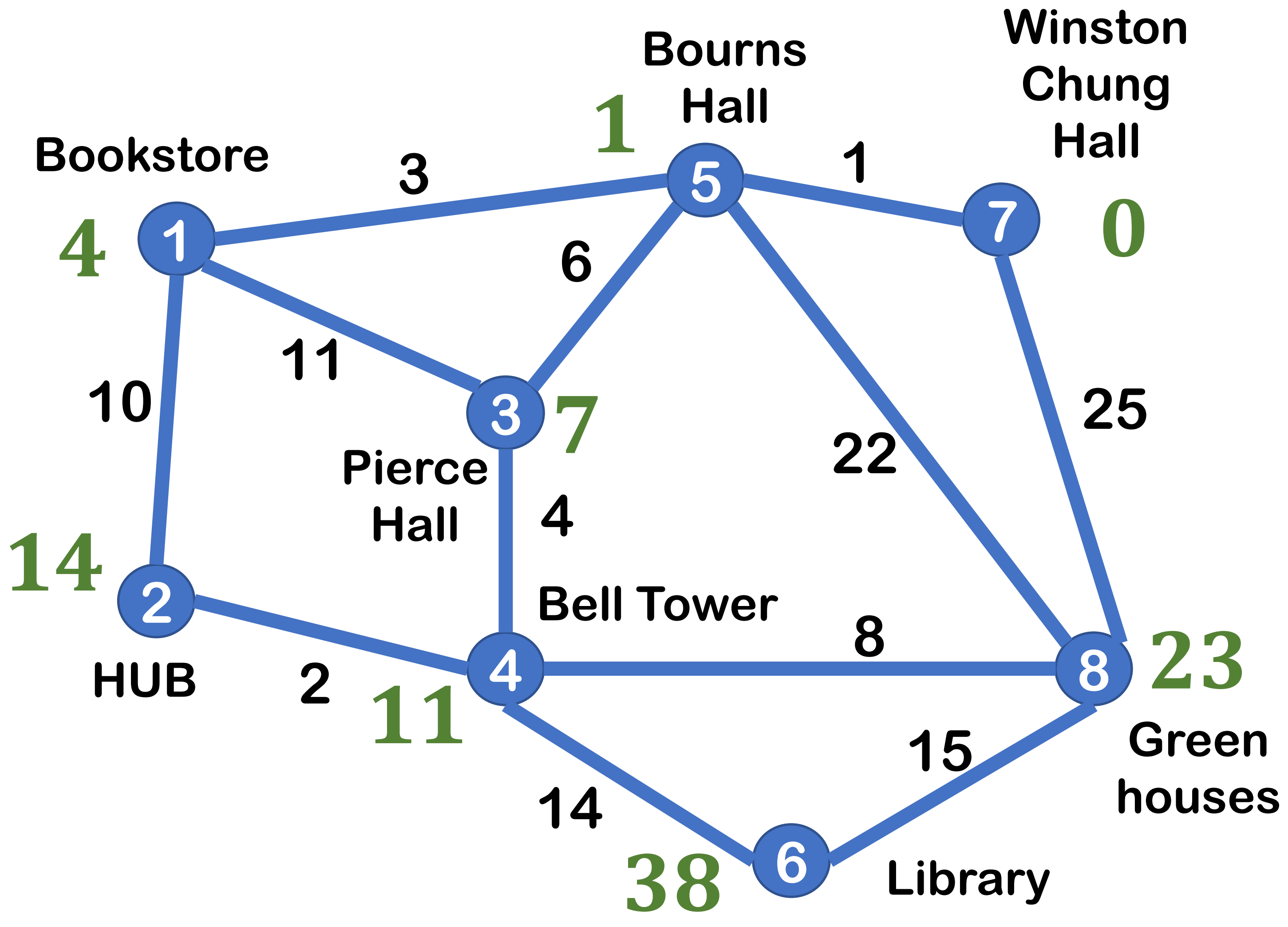
Step 3 : Given $D_{i,2}$, to compute $D_{i,3}$



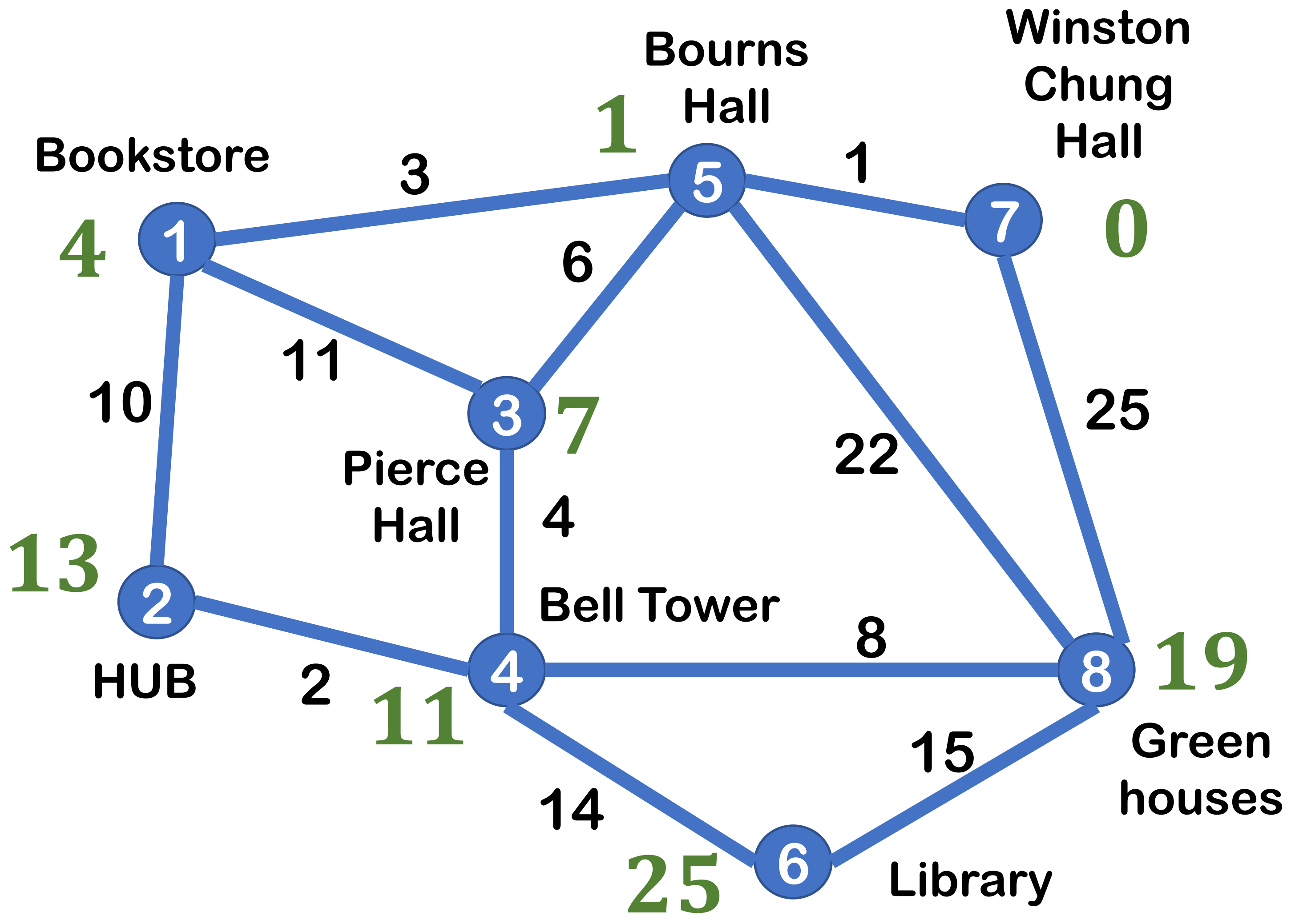
Step 3: after all updates



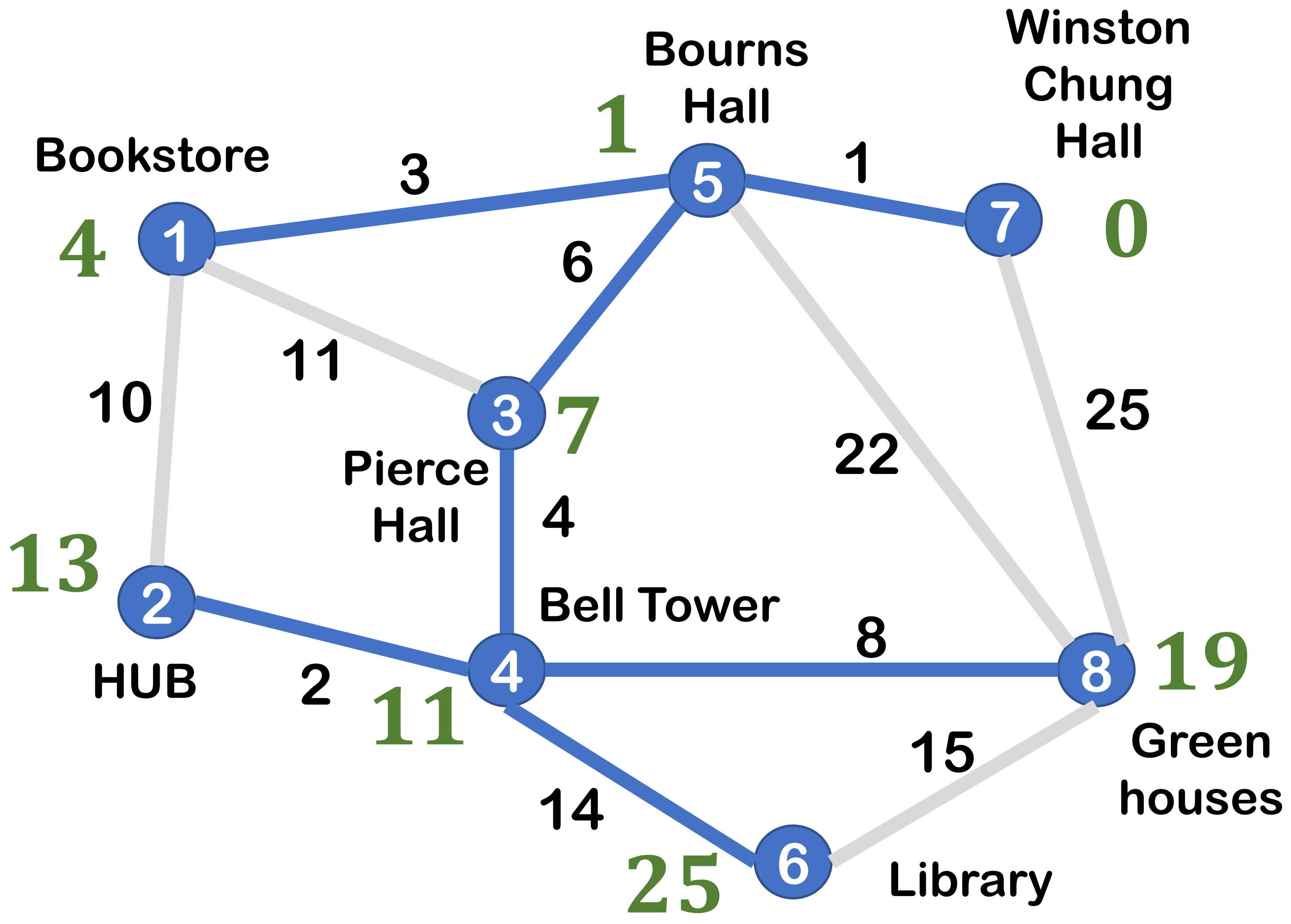
Step 4: Given $D_{i,3}$, to compute $D_{i,4}$



Step 4: after all updates



The shortest-path tree



Bellman-Ford Algorithm(G,s)

```
for k=1 to n-1 do
  for i=1 to n do
    D[k][i]=MAX
D[0][s]=0
for k=1 to n-1 do
  for each (i,j) in E do
    if (D[k-1][i]+w(i,j)<D[k][j])
        D[k][j]=D[k-1][i]+w(i,j), from[j]=i
  optional optimization: if no distance is updated, break
```

$$D_{i,k} = \min \begin{cases} D_{i,k-1} \\ \min_{(j,i)\in E} \{D_{j,k-1} + w(j,i)\} \end{cases}$$

Cost: $O(nm)$

Bellman-Ford Algorithm(G,s)

```
for i=1 to n do
  D[i]=MAX
D[s]=0
for k=1 to n-1 do
  for each (i,j) in E do
    if (D[i]+w(i,j)<D[j])
       D[j]=D[i]+w(i,j), from[j]=i
  optional optimization: if no distance is updated, break
```

$$D_{i,k} = \min \begin{cases} D_{i,k-1} \\ \min_{(j,i)\in E} \{D_{j,k-1} + w(j,i)\} \end{cases}$$

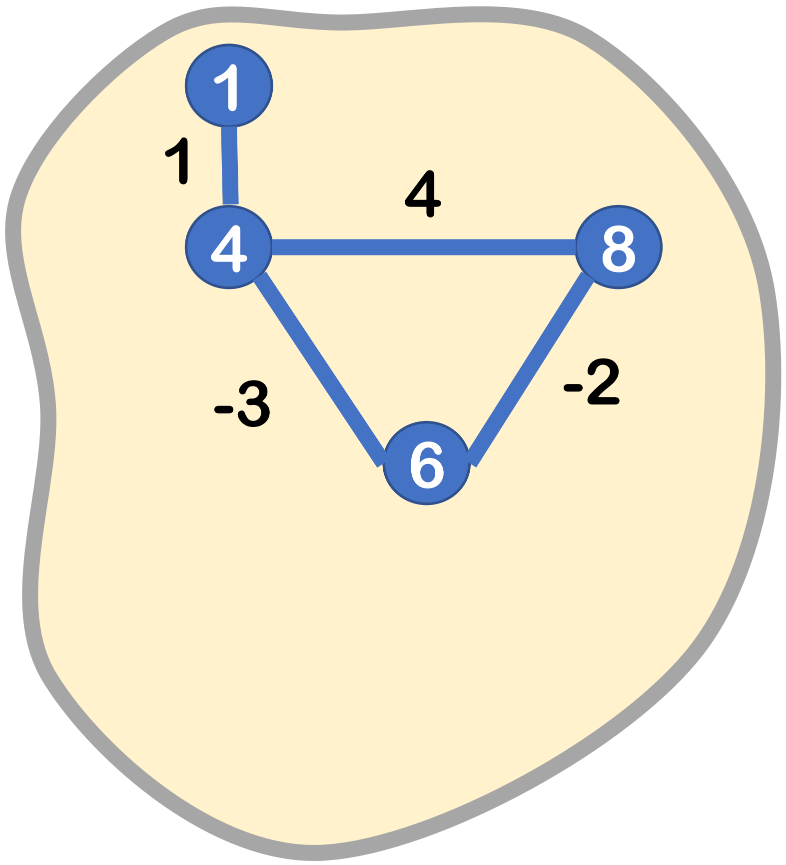
Cost: $O(nm)$

Detecting negative cycles

- The graph has a negative cycle $\Leftrightarrow$ run for another (n-th) round, and there are still edges being updated

- Intuitively, the distance to a vertex will be negative infinity

- Formal proof in CLRS 24.1



Summary

Single-source shortest-paths (SSSP)

- **One of the most widely used algorithms**
- **On unweighted graphs: BFS**
- **On positive-weighted graphs: Dijkstra's algorithm**
  - Similar to Prim's algorithm (greedy approach)
- **On any weighted graphs: Bellman-Ford algorithm**
  - Based on dynamic programming
- **There are many other SSSP algorithms**
  - Gabow's algorithm (scaling algorithm, on integer-weight graphs)
  - Thorup's algorithm ($O(m)$ work on certain restricted integer-weight graphs)
  - Parallel algorithms (Delta-Stepping, Radius-Stepping, Rho-stepping)