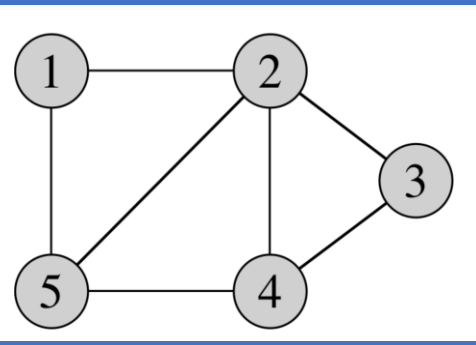
CS141: Intermediate Data Structures and Algorithms

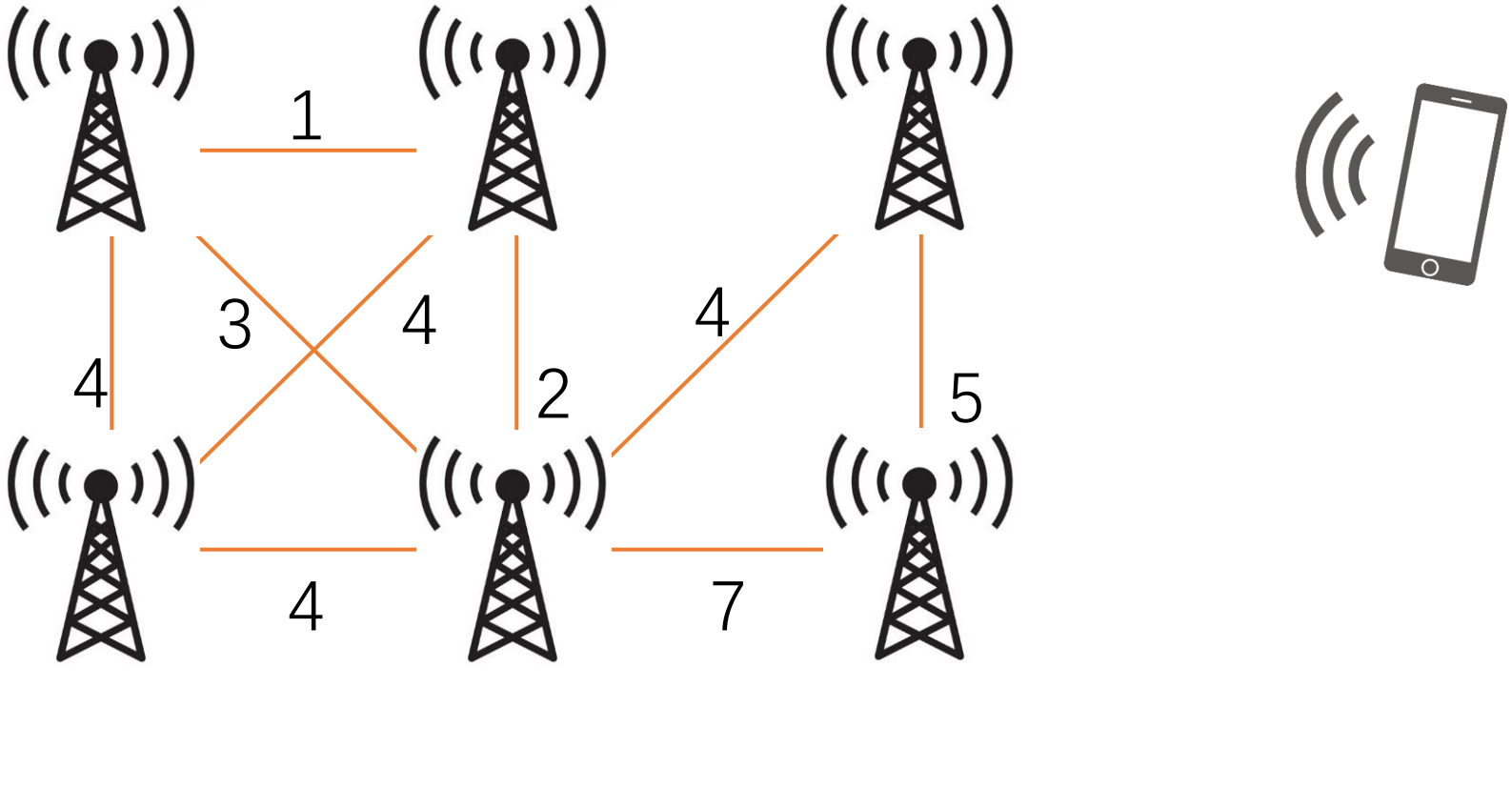


Minimum Spanning Tree (MST)

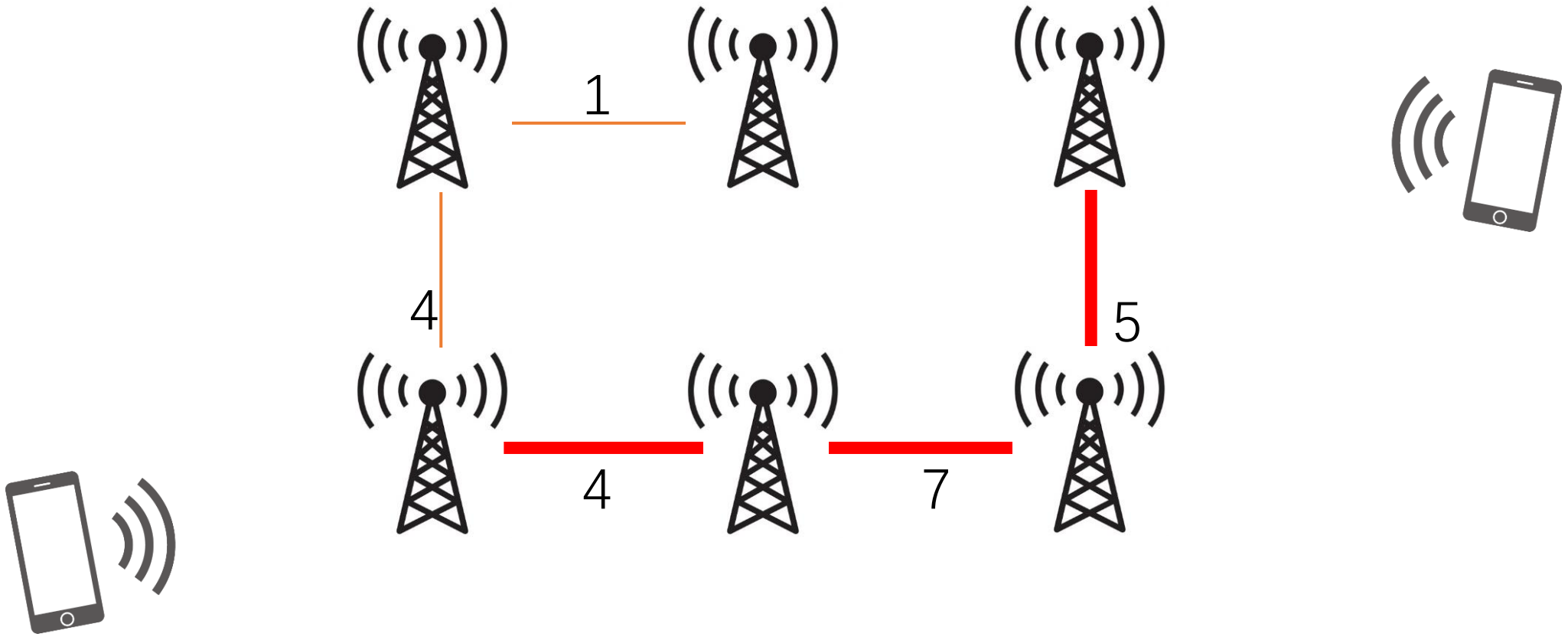
Yihan Sun

This lecture covers Section 22.1-22.3 of CLRS

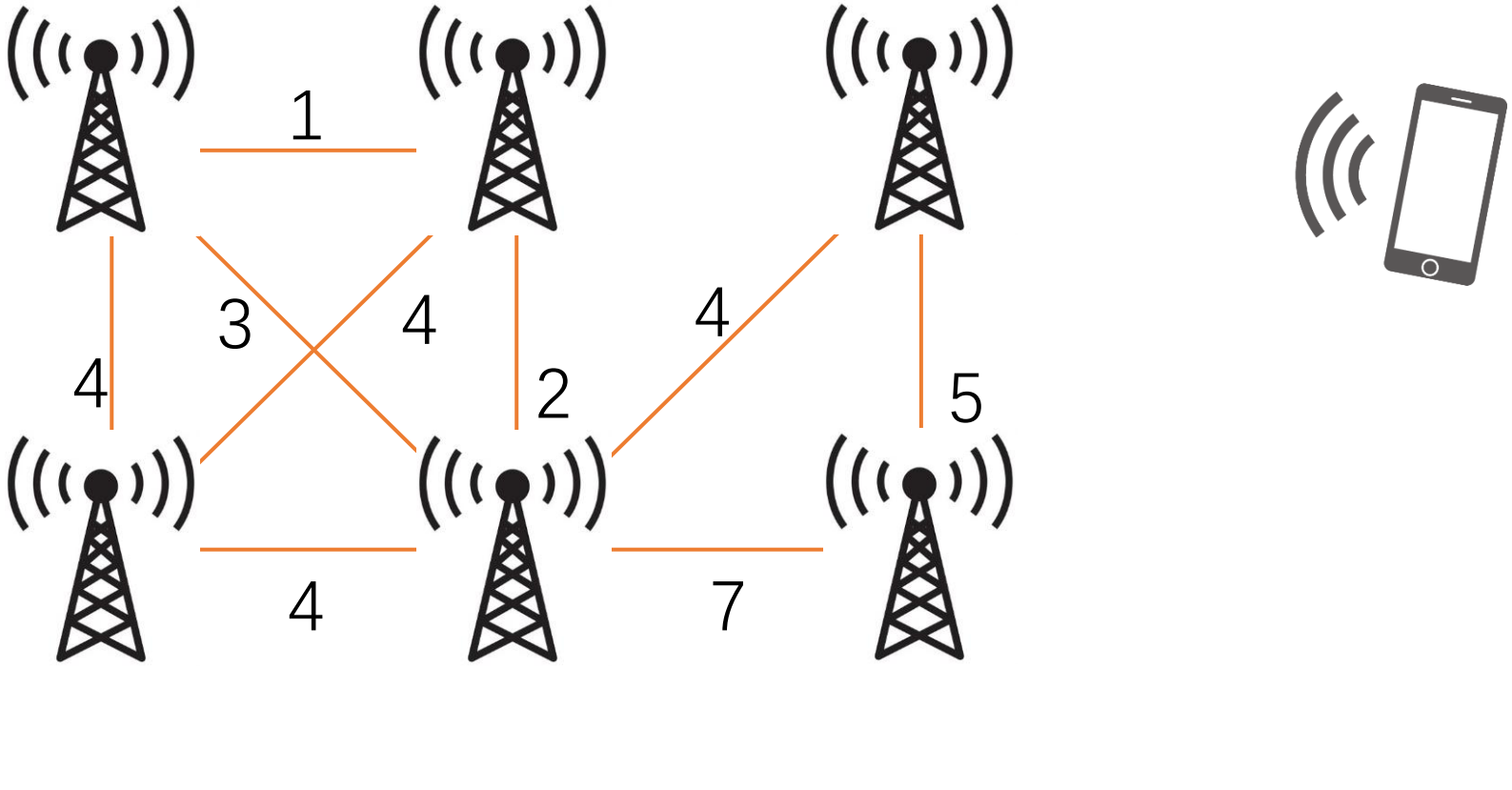
An example for building a mobile network



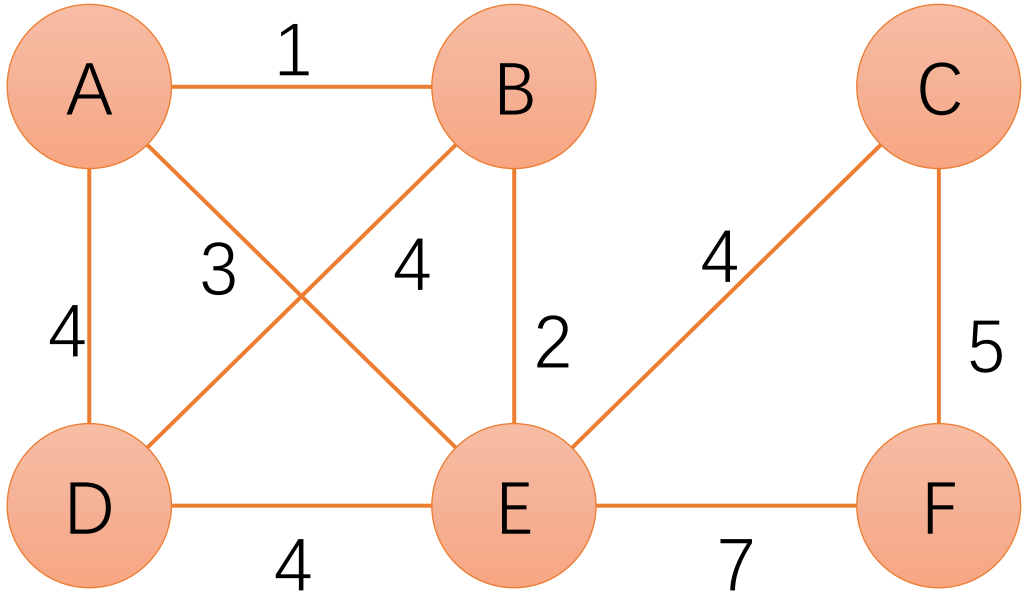
An example for building a mobile network



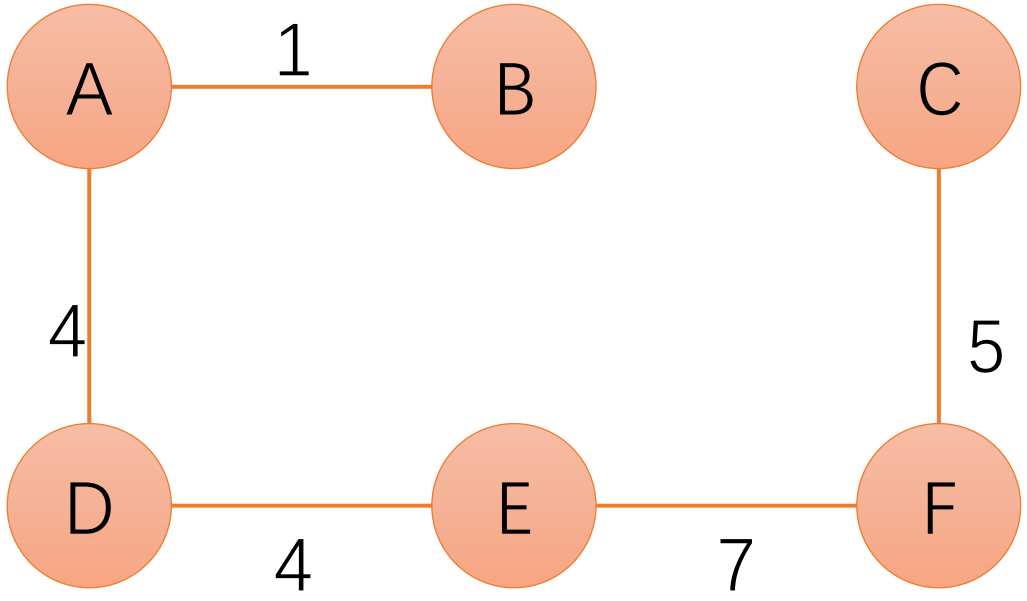
An example for building a mobile network



Abstracting it as a graph problem

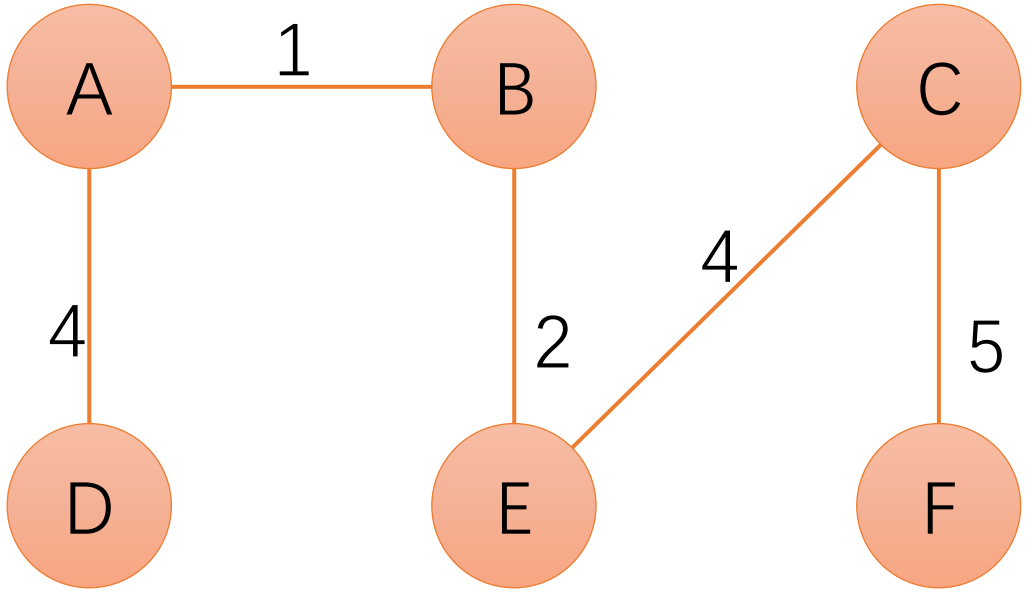


Non-minimum Spanning Tree



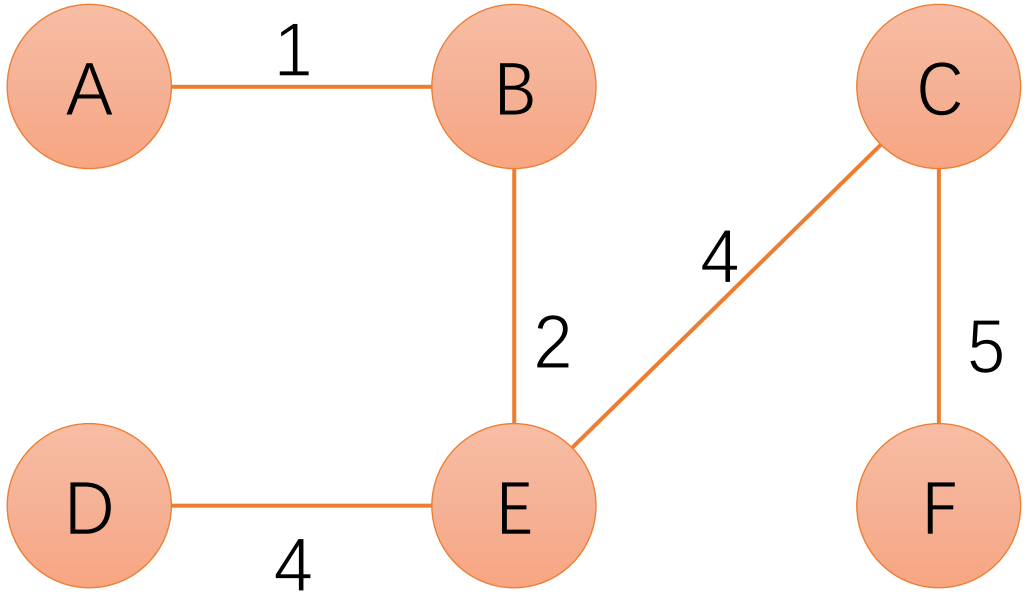
Total weight = 21

A Minimum Spanning Tree



Total weight = 16

Another MST

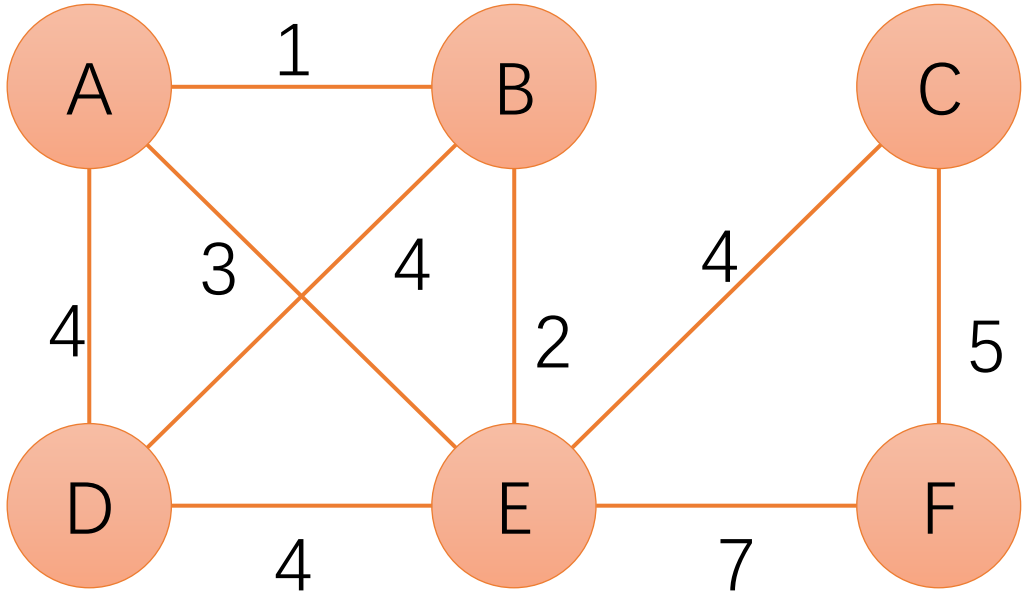


Total weight = 16

Minimum Spanning Tree (MST)

- Given an **undirected** weighted graph, find $n - 1$ edges that connects all vertices with minimum total weights
- Have been studied since 1920s
- Widely applied in many other graph algorithms and graph problems

Which edge must be included?

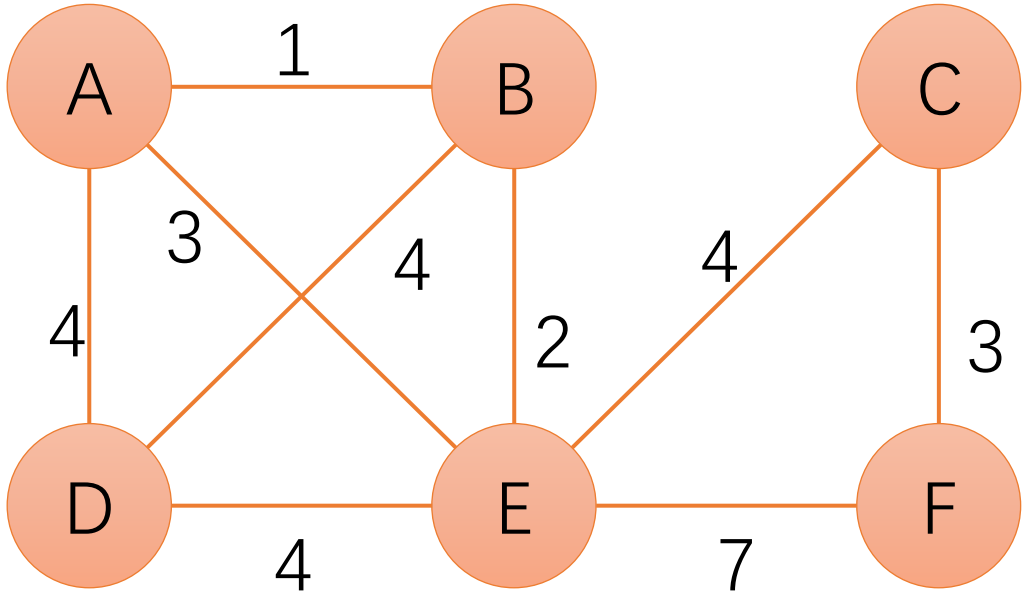


Kruskal's Algorithm

Kruskal's Algorithm (a greedy algorithm)

- Sort all the edges by weight
- Scan the edges by weight from lowest to highest
- If an edge introduces a cycle, drop it
- If an edge does not introduce a cycle, pick it
- Terminate when n-1 edges are picked

Kruskal's MST in Action



(u,v,w):

➡ (A,B,1)

(B,E,2)

(A,E,3)

(C,F,3)

(A,D,4)

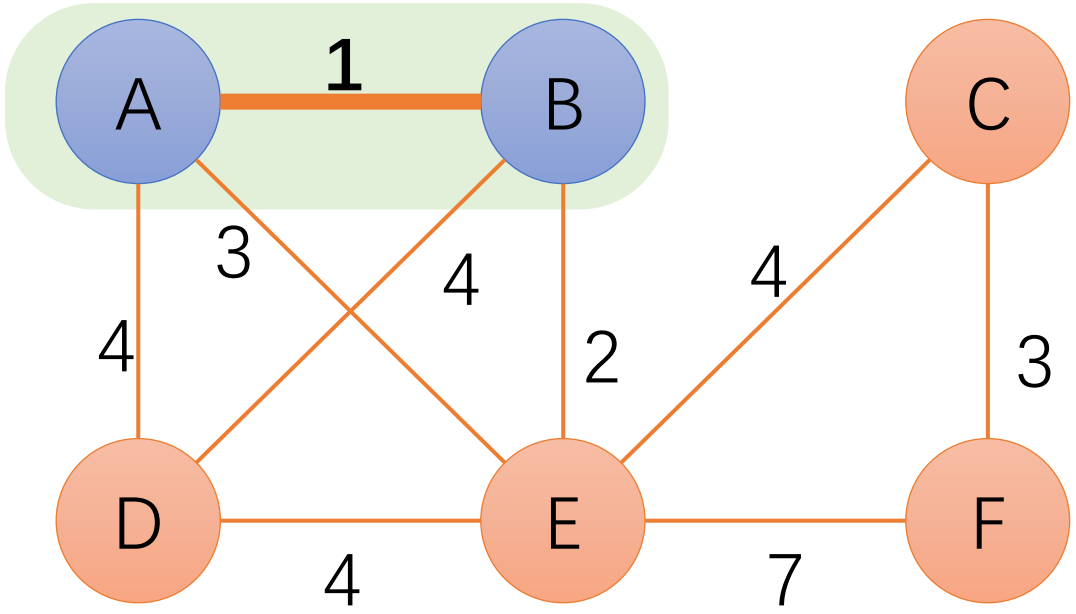
(B,D,4)

(D,E,4)

(C,E,4)

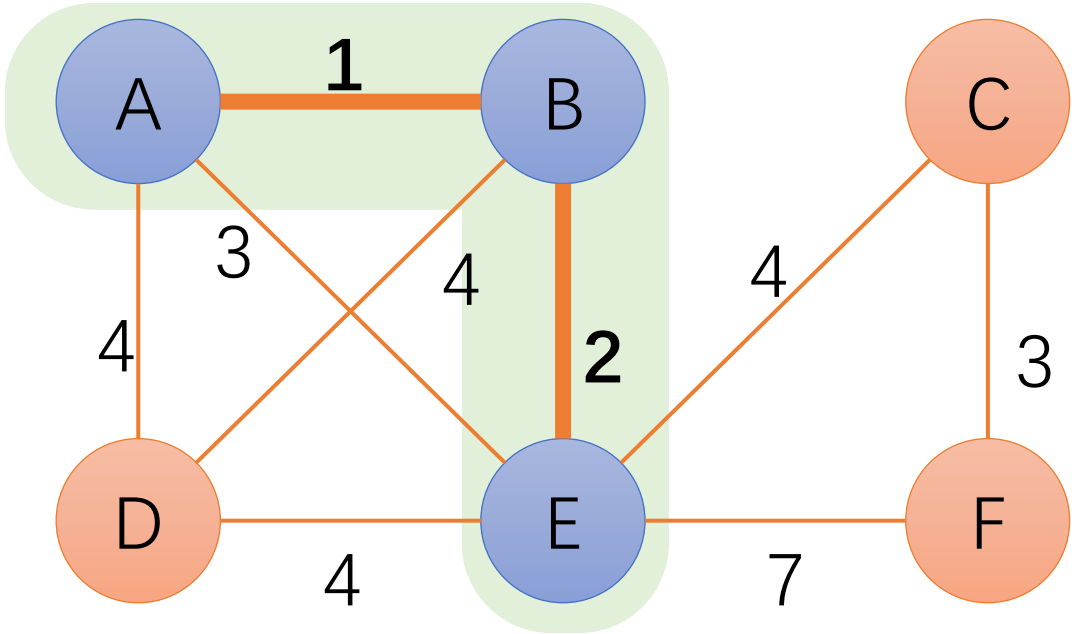
(E,F,7)

Kruskal's MST in Action



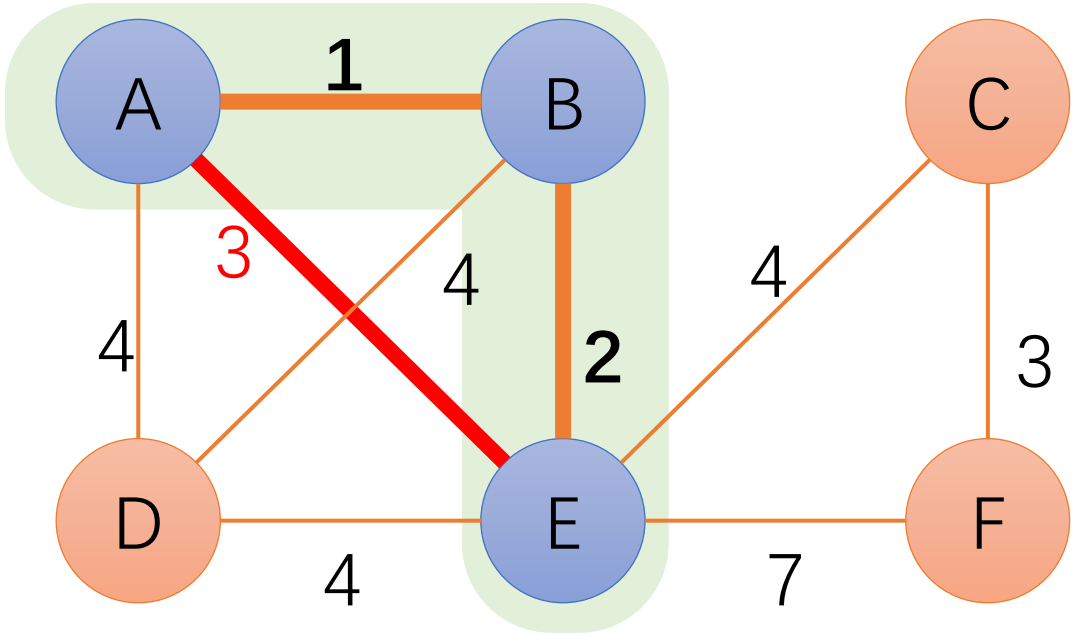

```
(u,v,w):
(A,B,1)
(B,E,2)
(A,E,3)
(C,F,3)
(A,D,4)
(B,D,4)
(D,E,4)
(C,E,4)
(E,F,7)
```

Kruskal's MST in Action



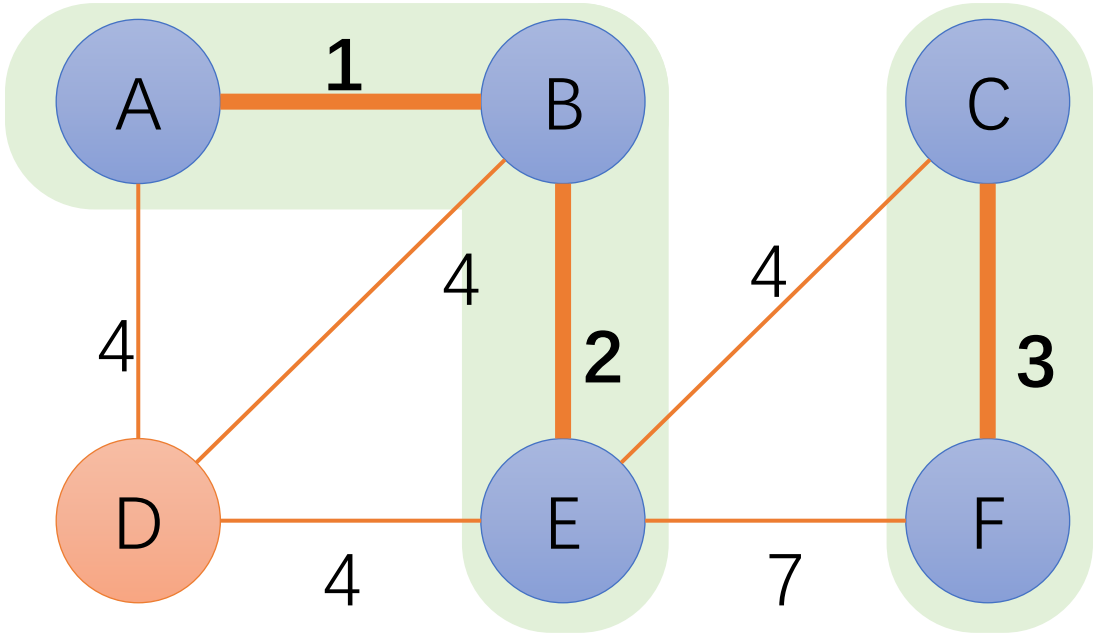

```
(u,v,w):
(A,B,1)
(B,E,2)
(A,E,3)
(C,F,3)
(A,D,4)
(B,D,4)
(D,E,4)
(C,E,4)
(E,F,7)
```

Kruskal's MST in Action



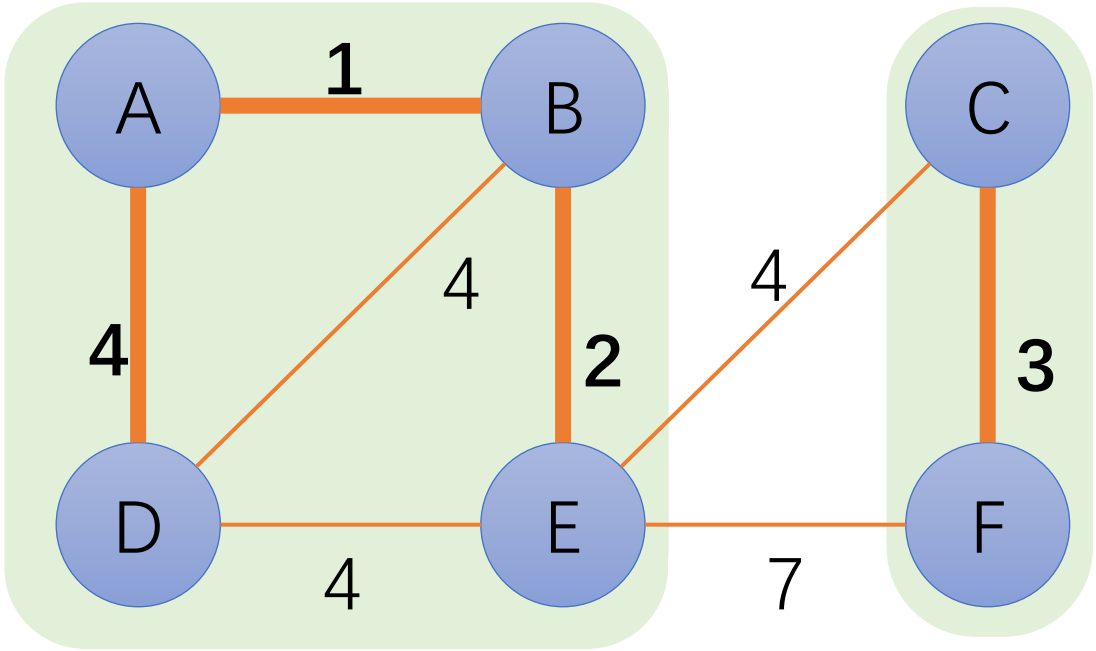

```
(u,v,w):
(A,B,1)
(B,E,2)
(A,E,3)
(C,F,3)
(A,D,4)
(B,D,4)
(D,E,4)
(C,E,4)
(E,F,7)
```

Kruskal's MST in Action



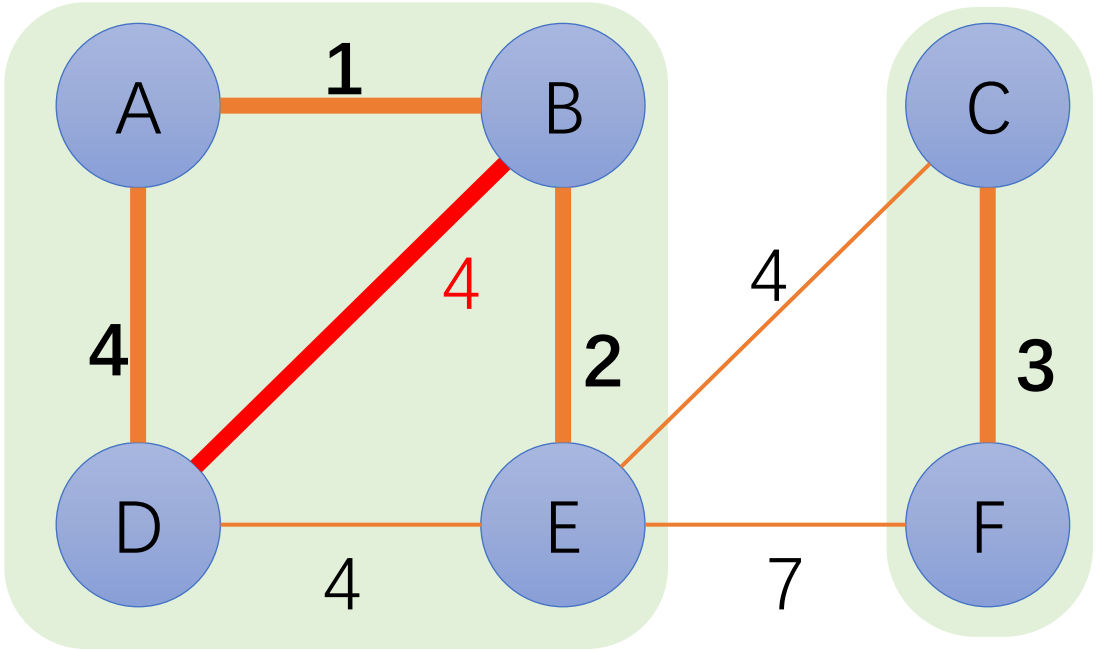

```
(u,v,w):
(A,B,1)
(B,E,2)
(A,E,3)
(C,F,3)
(A,D,4)
(B,D,4)
(D,E,4)
(C,E,4)
(E,F,7)
```

Kruskal's MST in Action



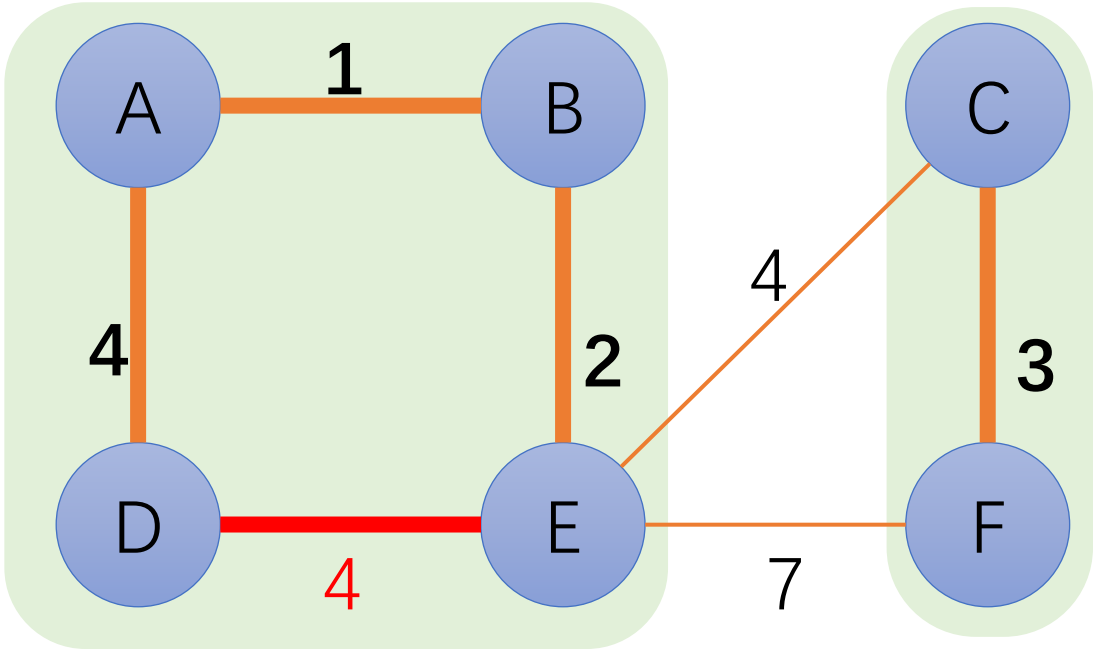

```
(u,v,w):
(A,B,1)
(B,E,2)
(A,E,3)
(C,F,3)
(A,D,4)
(B,D,4)
(D,E,4)
(C,E,4)
(E,F,7)
```

Kruskal's MST in Action



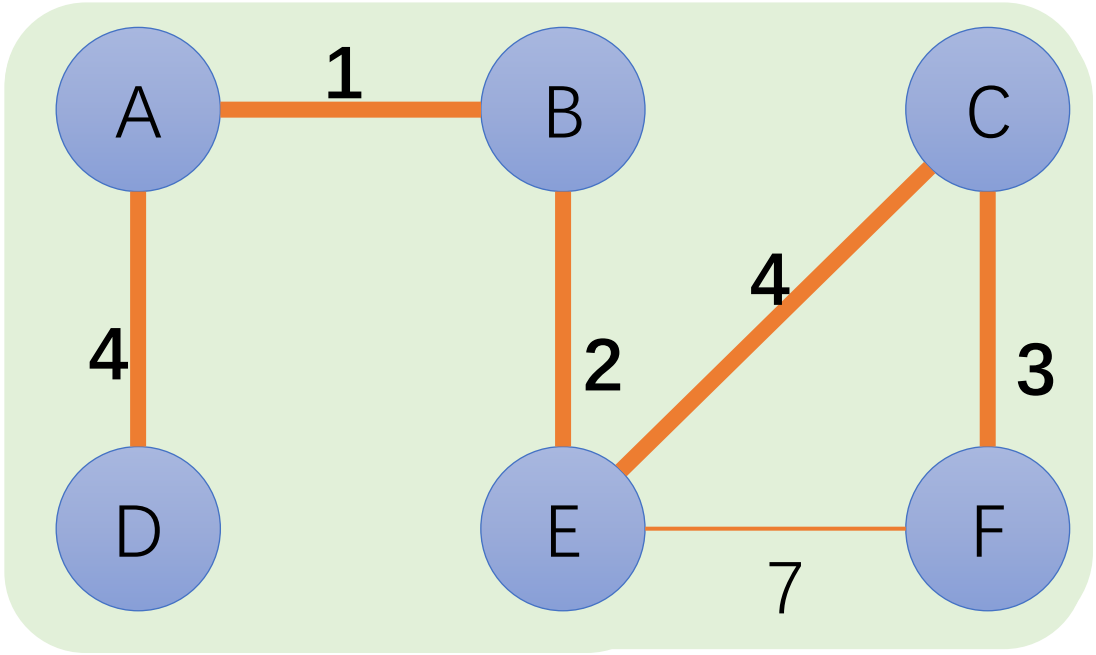

```
(u,v,w):
(A,B,1)
(B,E,2)
(A,E,3)
(C,F,3)
(A,D,4)
(B,D,4)
(D,E,4)
(C,E,4)
(E,F,7)
```

Kruskal's MST in Action



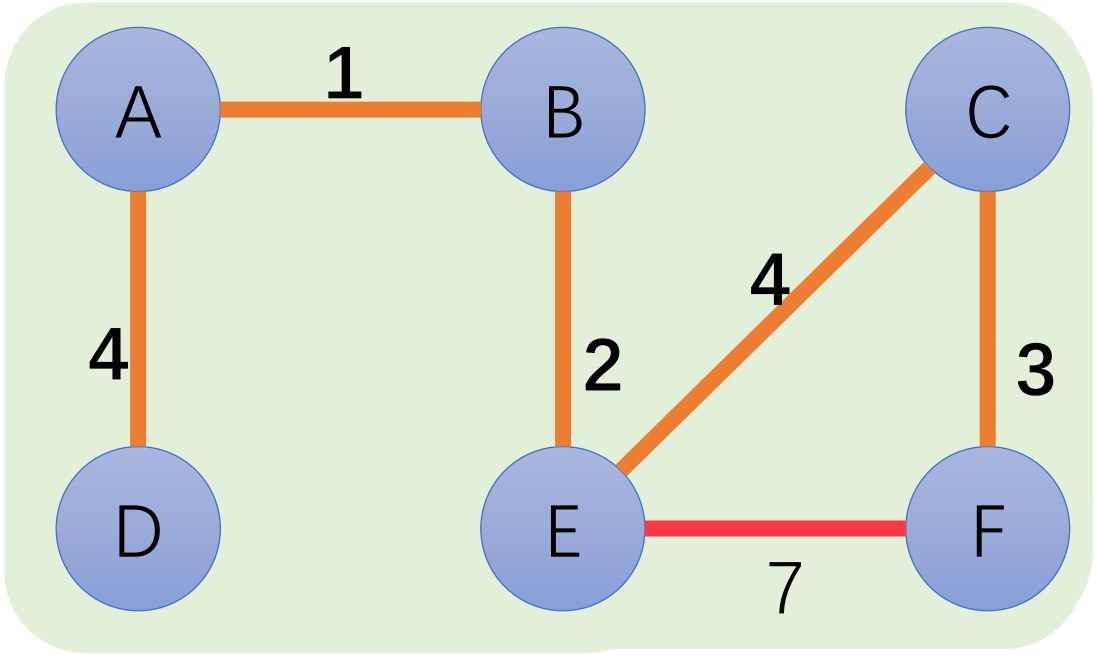

```
(u,v,w):
(A,B,1)
(B,E,2)
(A,E,3)
(C,F,3)
(A,D,4)
(B,D,4)
(D,E,4)
(C,E,4)
(E,F,7)
```

Kruskal's MST in Action



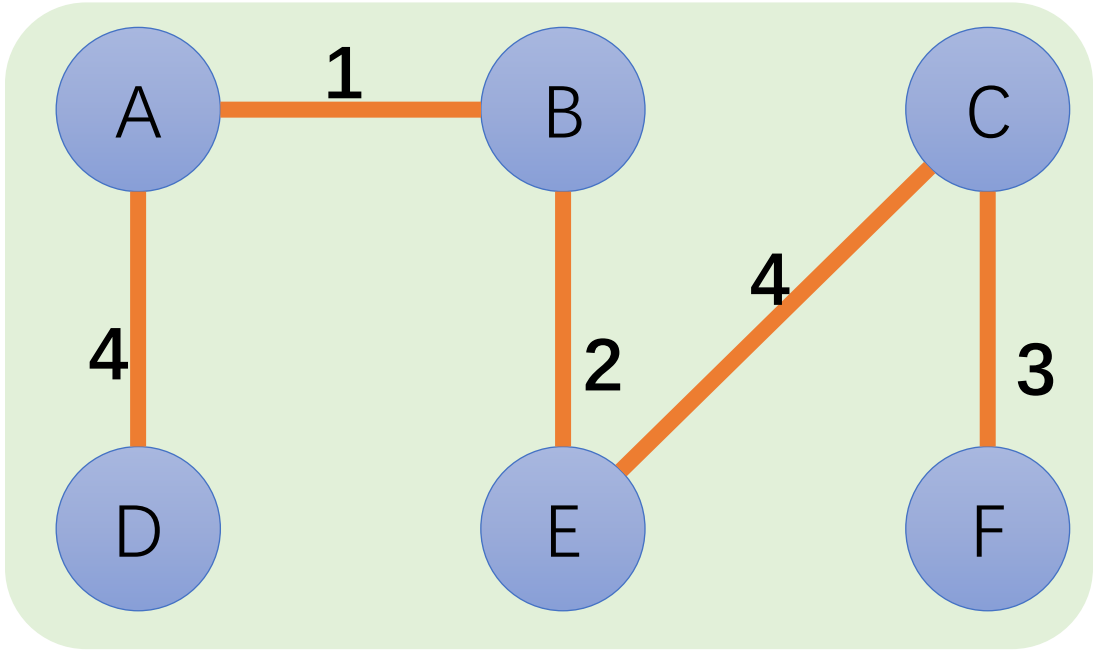

```
(u,v,w):
(A,B,1)
(B,E,2)
(A,E,3)
(C,F,3)
(A,D,4)
(B,D,4)
(D,E,4)
(C,E,4)
(E,F,7)
```

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

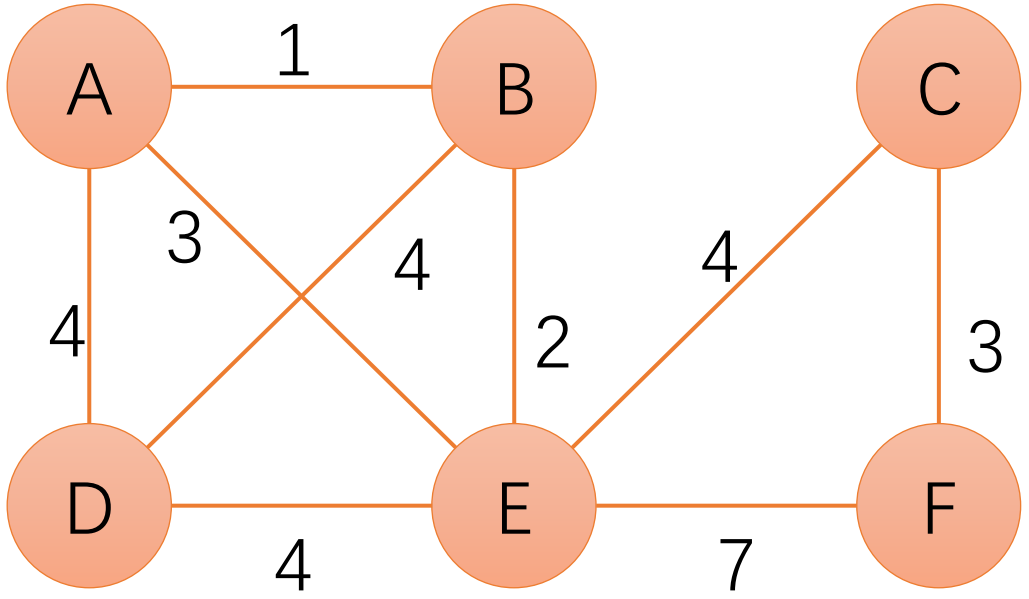



```
(u,v,w):
(A,B,1)
(B,E,2)
(A,E,3)
(C,F,3)
(A,D,4)
(B,D,4)
(D,E,4)
(C,E,4)
(E,F,7)
```

Kruskal's Algorithm

```
KRUSKAL(G, w)
  A = ∅
  for each vertex v ∈ G.V   // n iterations
      MAKE-SET(v)
  sort the edges of G.E into nondecreasing order by weight w
  for each (u, v) taken from the sorted list // m iterations
      if FIND-SET(u) ≠ FIND-SET(v)    // m iterations
          A = A ∪ {(u, v)}
          UNION(u, v)                 // n iterations
  return A
```

Kruskal's MST in Array Implementation




```
(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 Algorithm (simple implementation)

- Use an array to check the connectivity

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

```
                KRUSKAL(G, w)
O(1)            A = ∅
                for each vertex v ∈ G.V   // n iterations
O(1)                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)                if FIND-SET(u) ≠ FIND-SET(v)    // m iterations
O(1)                    A = A ∪ {(u, v)}
O(n)                    UNION(u, v)                 // n iterations
                return A
```

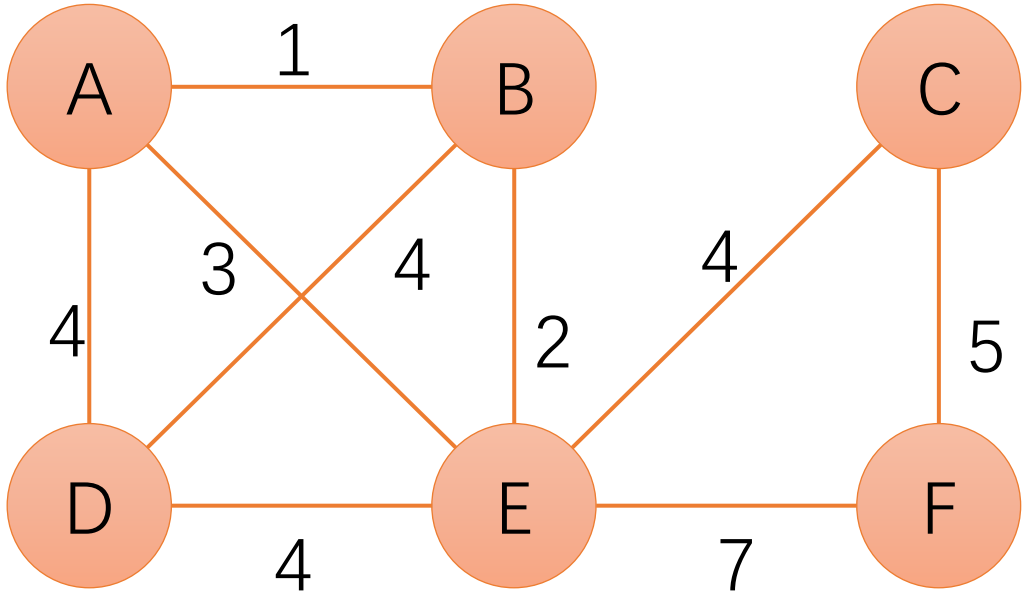
Kruskal's Algorithm (advanced implementation)

- **Use the union-find data structure to check the connectivity (covered in CS 218, S21)**

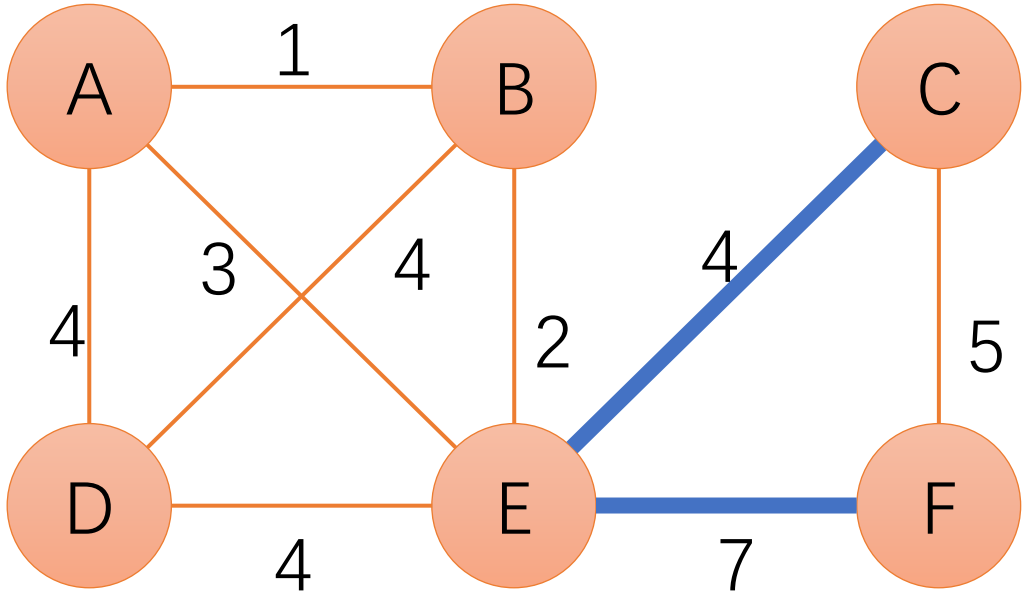
$O(m \log n)$

```
KRUSKAL(G, w)
A = ∅                                              O(1)
for each vertex v ∈ G.V                            // V iterations
    MAKE-SET(v)                                    O(1)
sort the edges of G.E into nondecreasing order by weight w   O(m log m)
for each (u, v) taken from the sorted list         // E iterations
    if FIND-SET(u) ≠ FIND-SET(v)                   O(α(n))   // E iterations
        A = A ∪ {(u, v)}                           O(1)
        UNION(u, v)                                O(α(n))   // V iterations
return A
```

Which edge must be included?

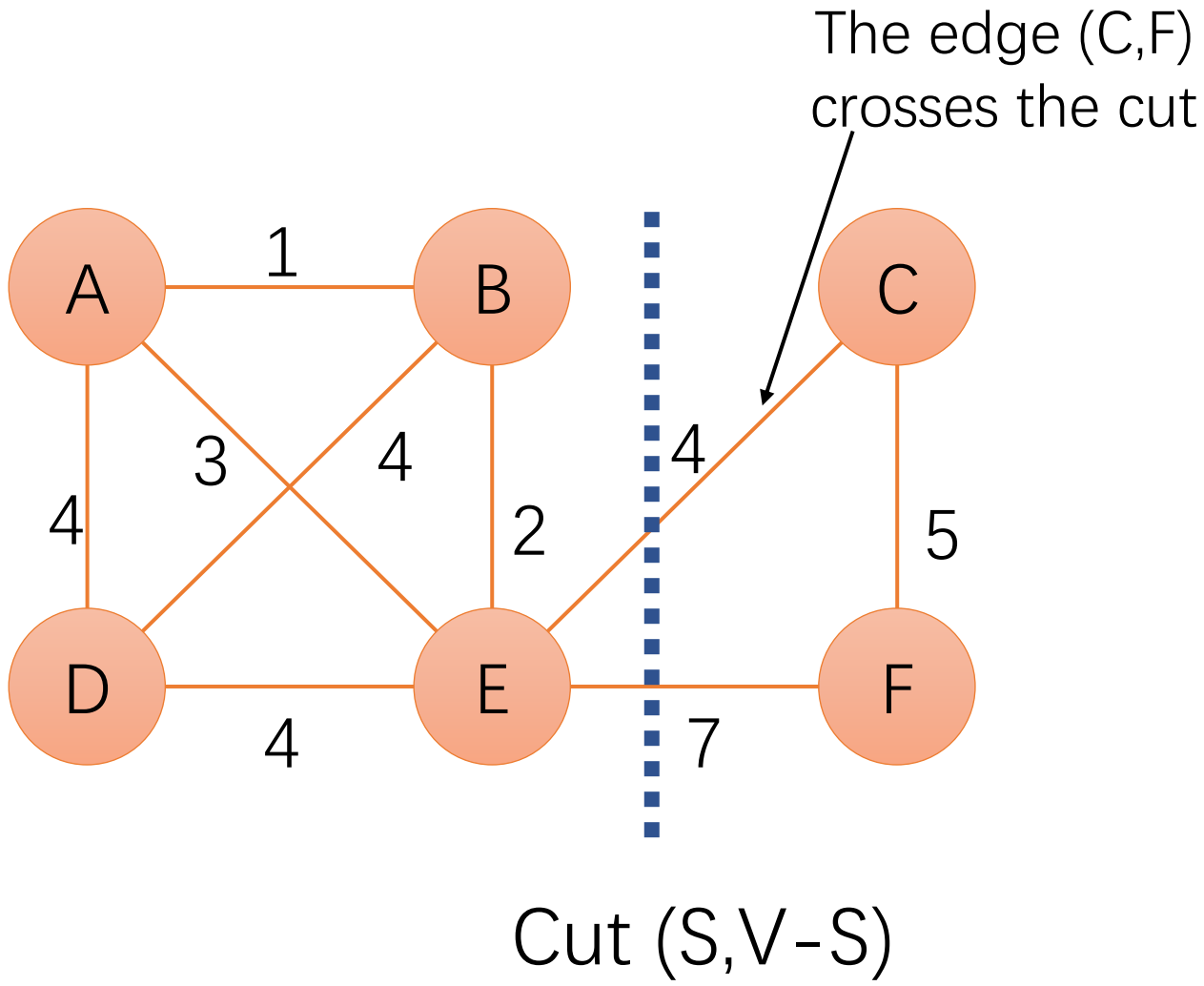


Which of them must be included?



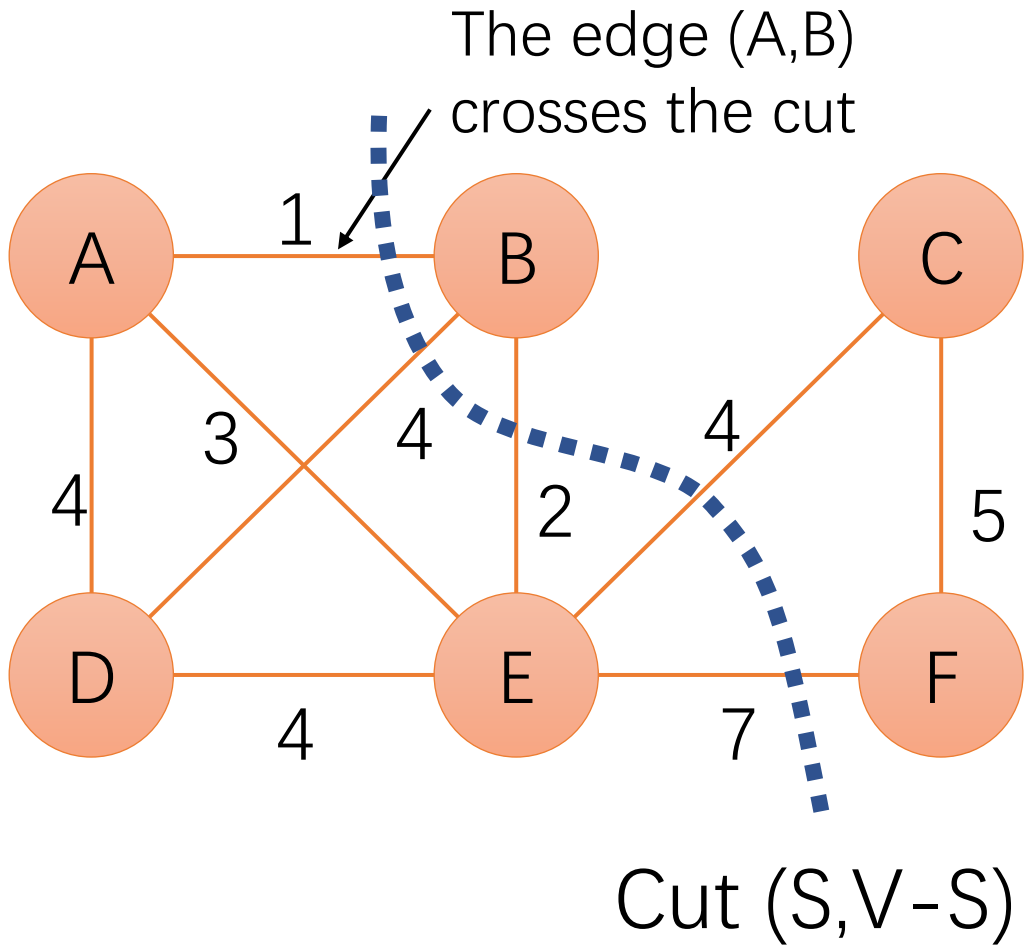
A cut of a graph

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



A cut of a graph

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

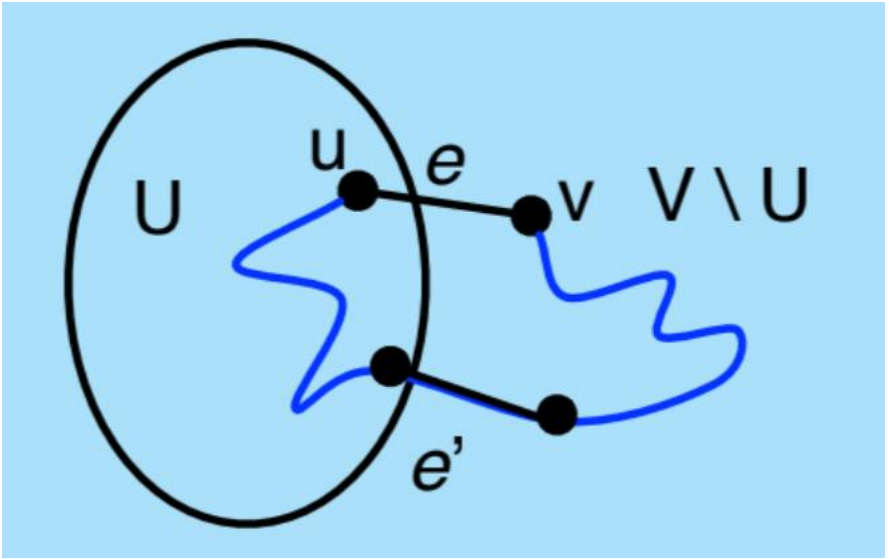


Light edge

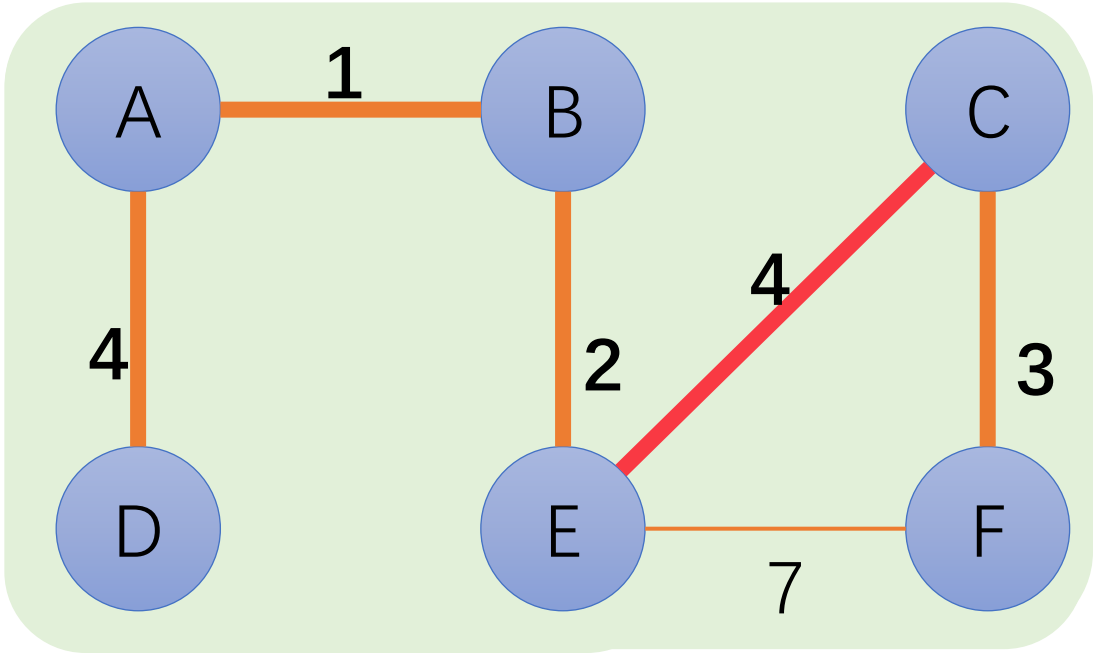
- A light edge: is a crossing edge with the minimum weight
- A light edge is safe to be add to MST
- (This greedy choice is GOOD!)

Light-edge property for MST - Proof (sketch)

- **Let the light edge be e**
- **Suppose e is not included in the MST**
- **Then there must be some e' in the cut in MST with $w(e') > w(e)$**
  - Otherwise the two parts are not connected!
- **What happens?**
  - We drop e' and use e get a spanning tree with smaller weight
  - Contradiction!



Why Kruskal is correct?




```
(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)
```

Any edge added to the tree **must be a light edge**!

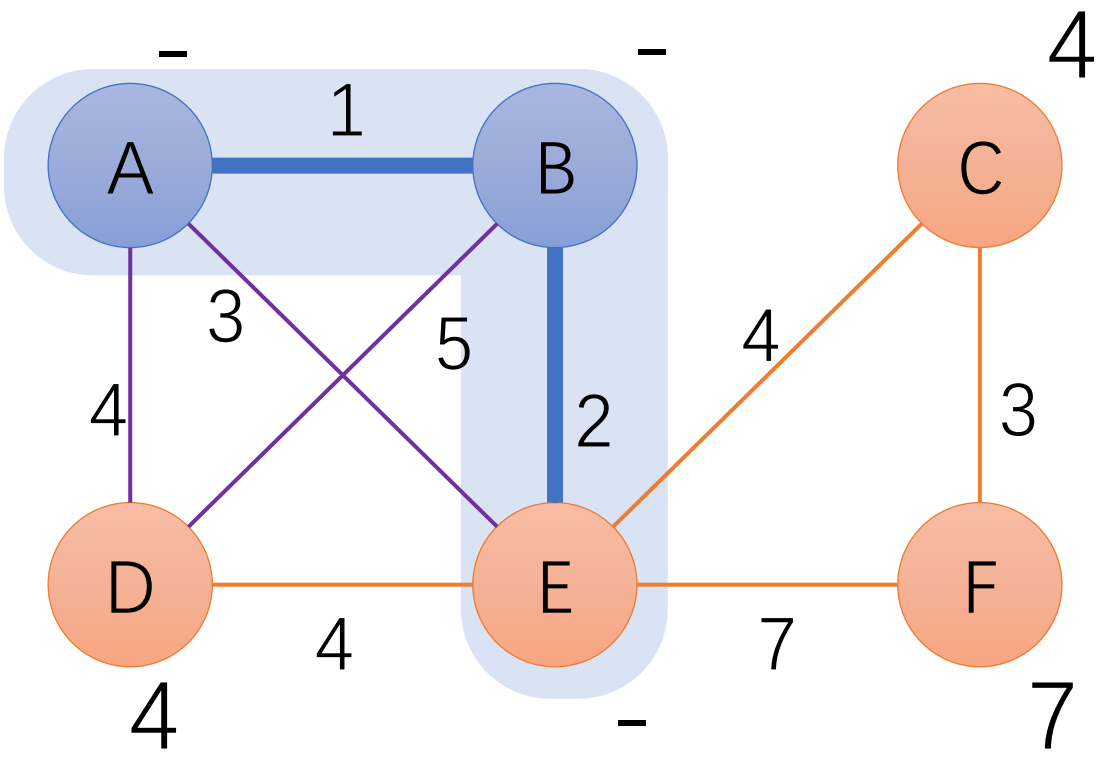
- It is an edge connecting two parts
- It is the lightest among such edges

Light edge

- A light edge: is a crossing edge with the minimum weight
- A spanning with all light edges is an MST

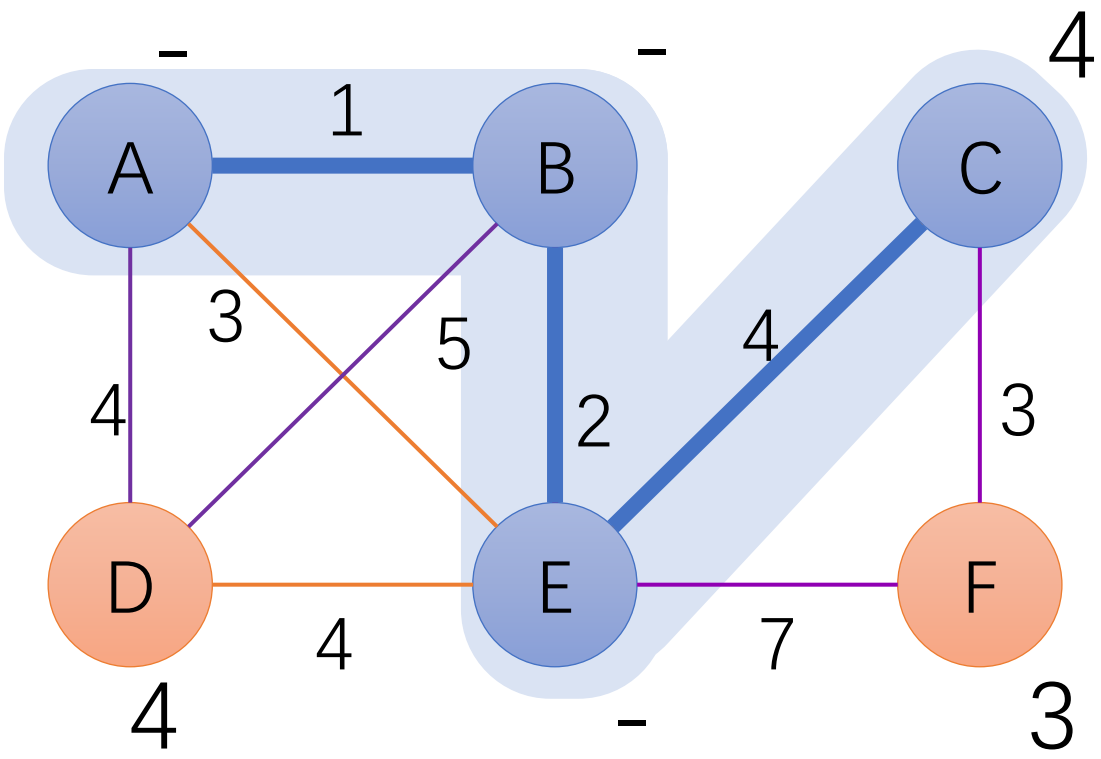
Prim’s Algorithm

Prim's



- Start from one vertex as the current MST
- Add the smallest edge incident on the current MST

Prim's



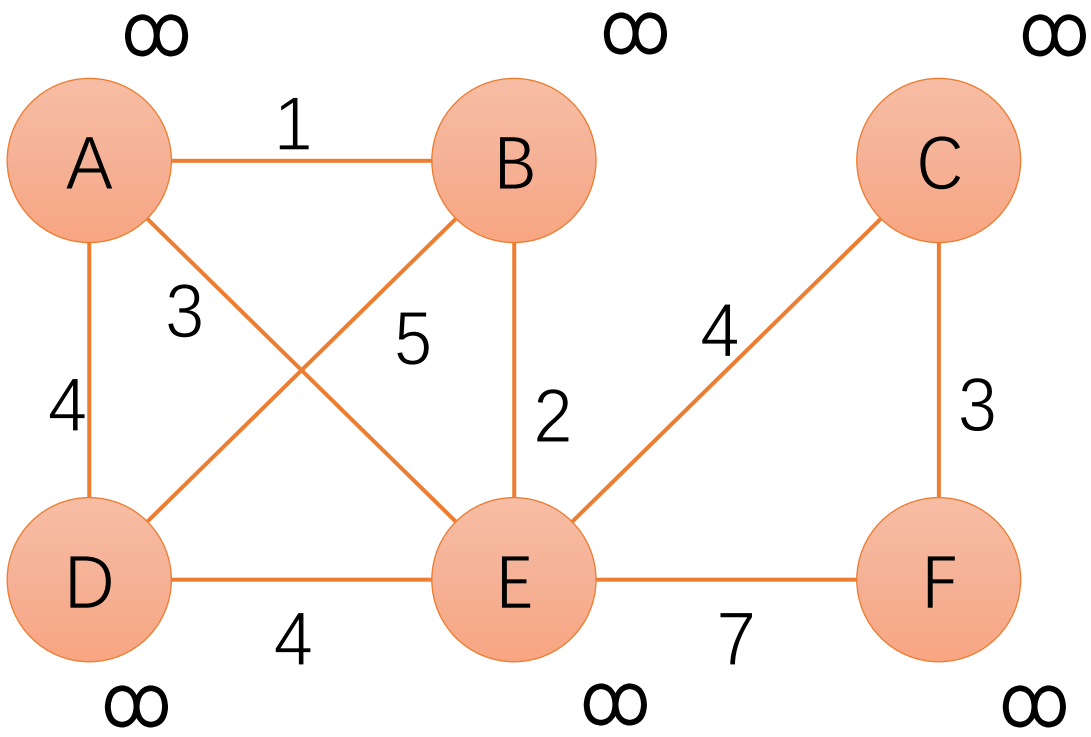
- Start from one vertex as the current MST
- Add the smallest edge incident on the current MST

- Maintain the tentative distance for every vertex (lightest edge to the current MST)
- Find the vertex with smallest tentative distance
- Add it to MST
- Relax all its neighbors

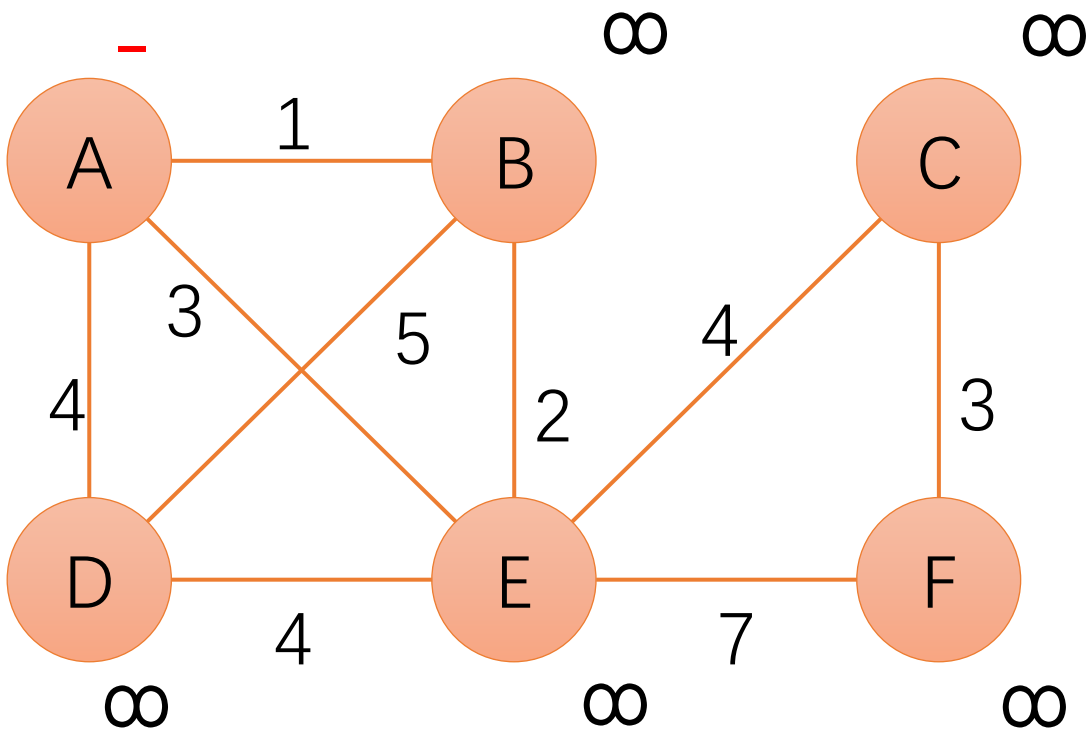
Prim's Algorithm

- Start from any node and mark it as visited
- Set the weight of each node to the lowest weight of an incident edge with a visited node
- Find the node with the lowest weight and visit it
- Repeat until all the n nodes are visited

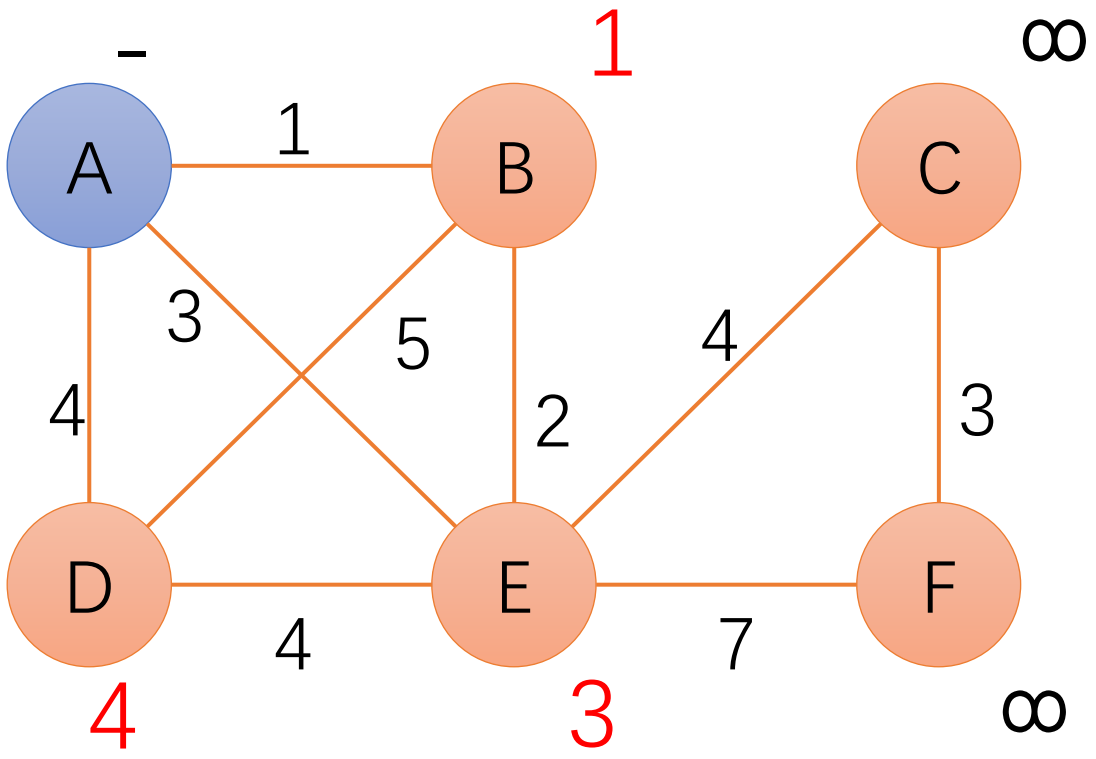
Prim's MST in Action



Prim's MST in Action

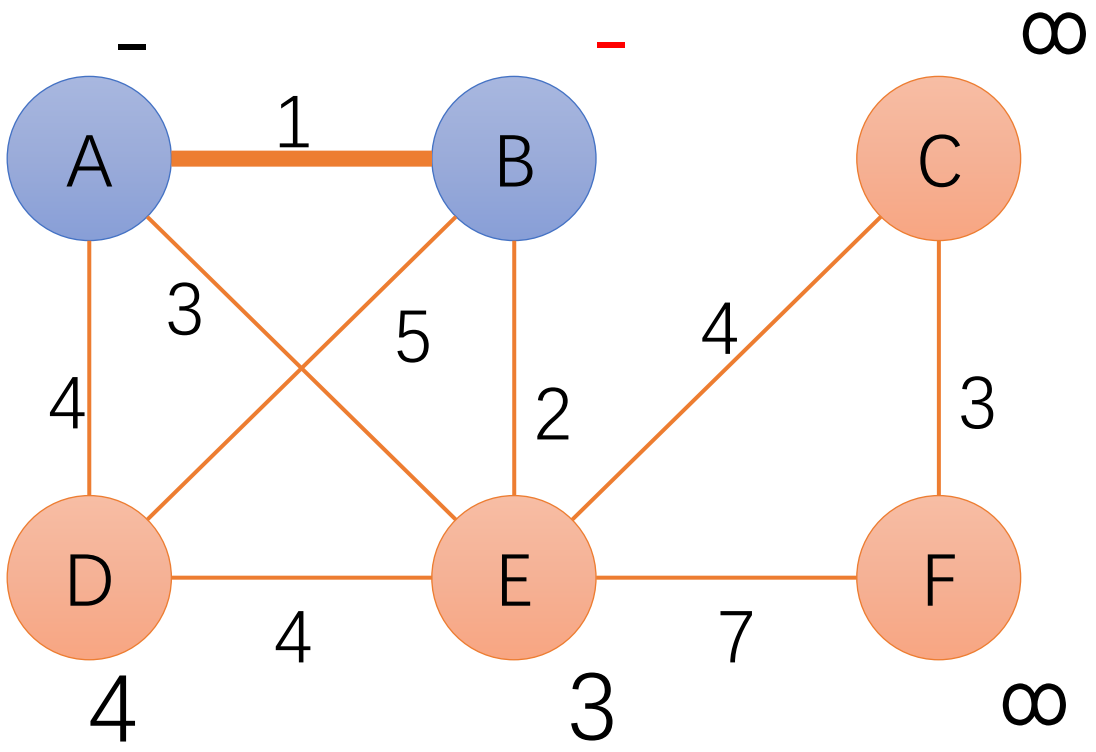


Prim's MST in Action



(B, 1)
(E, 3)
(D, 4)

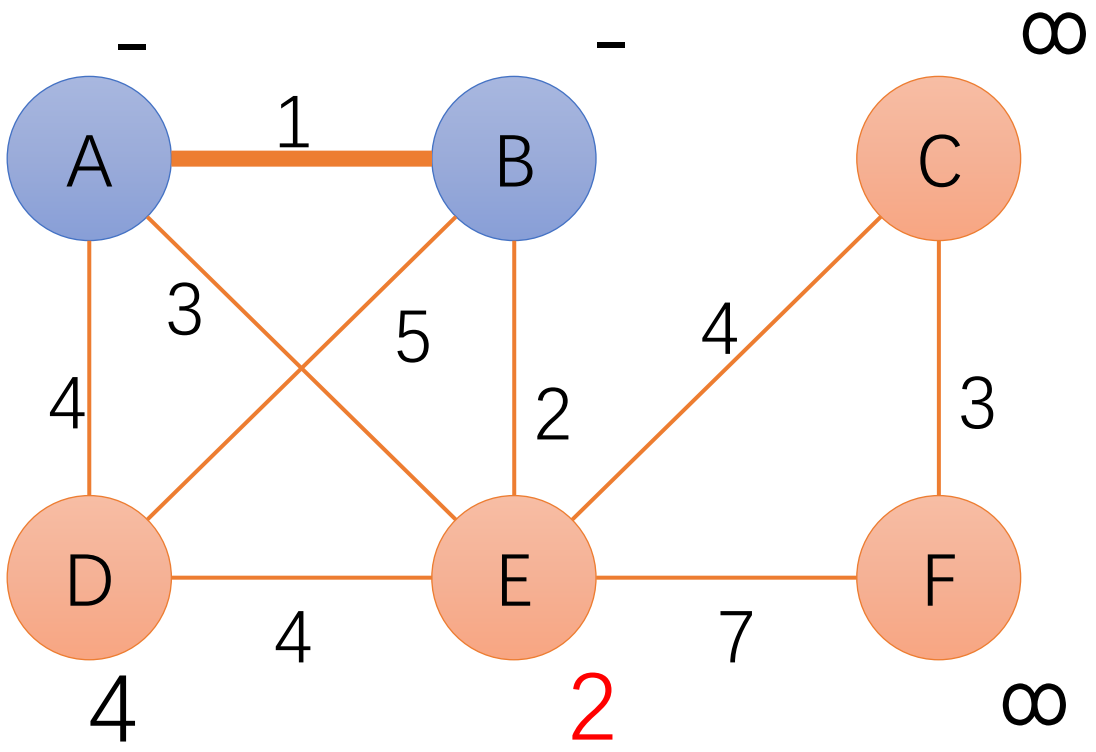
Prim's MST in Action



(E, 3)

(D, 4)

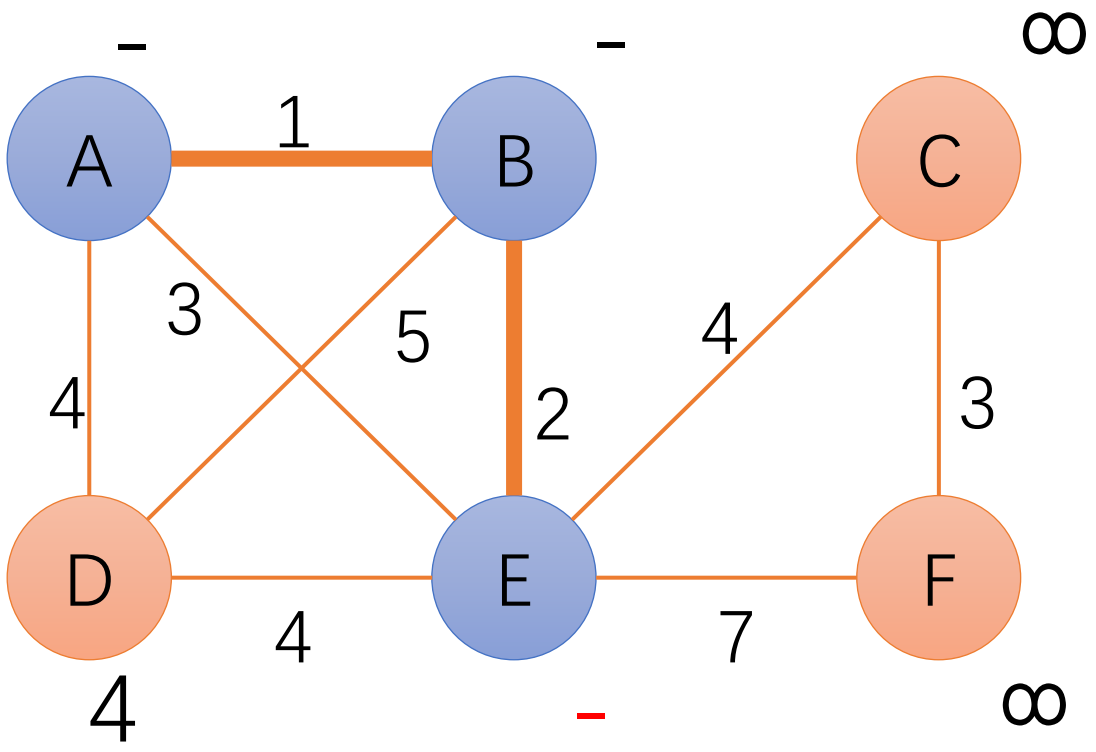
Prim's MST in Action



(E, 2)

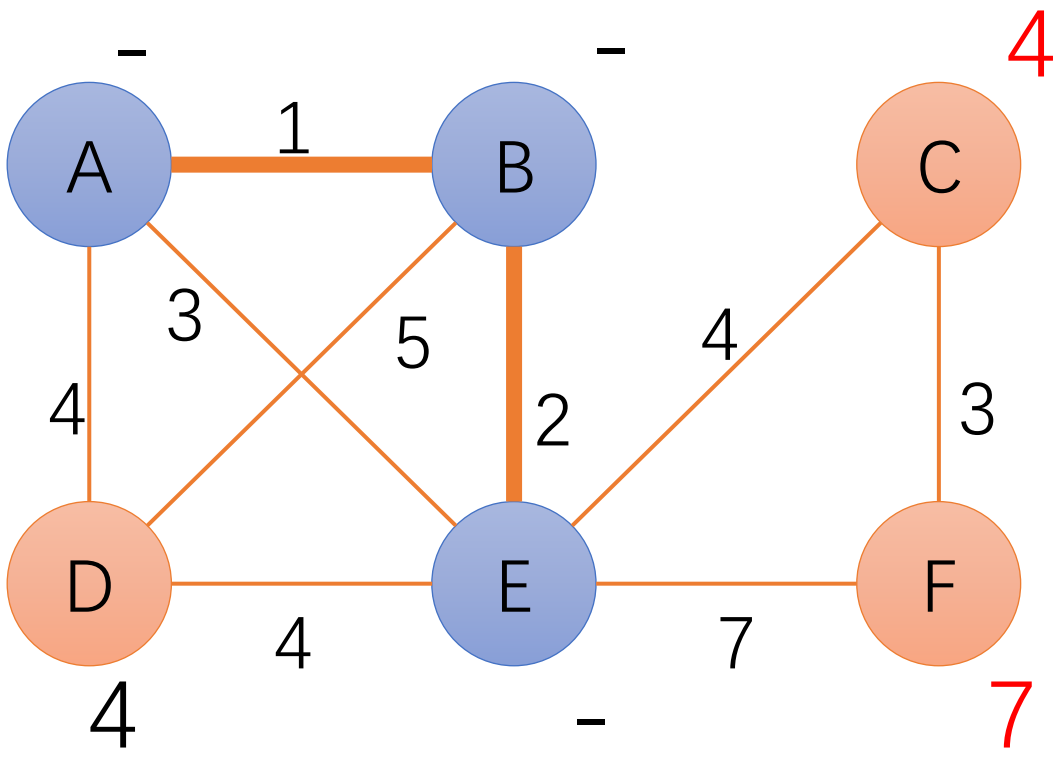
(D, 4)

Prim's MST in Action



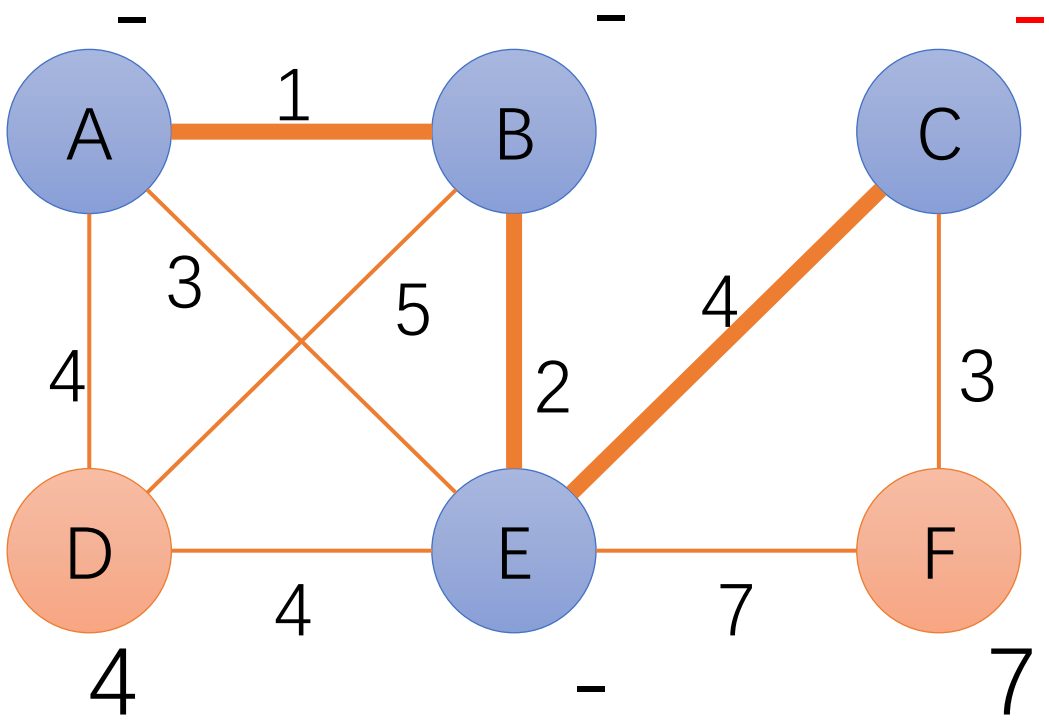
(D, 4)

Prim's MST in Action



(C, 4)
(D, 4)
(F, 7)

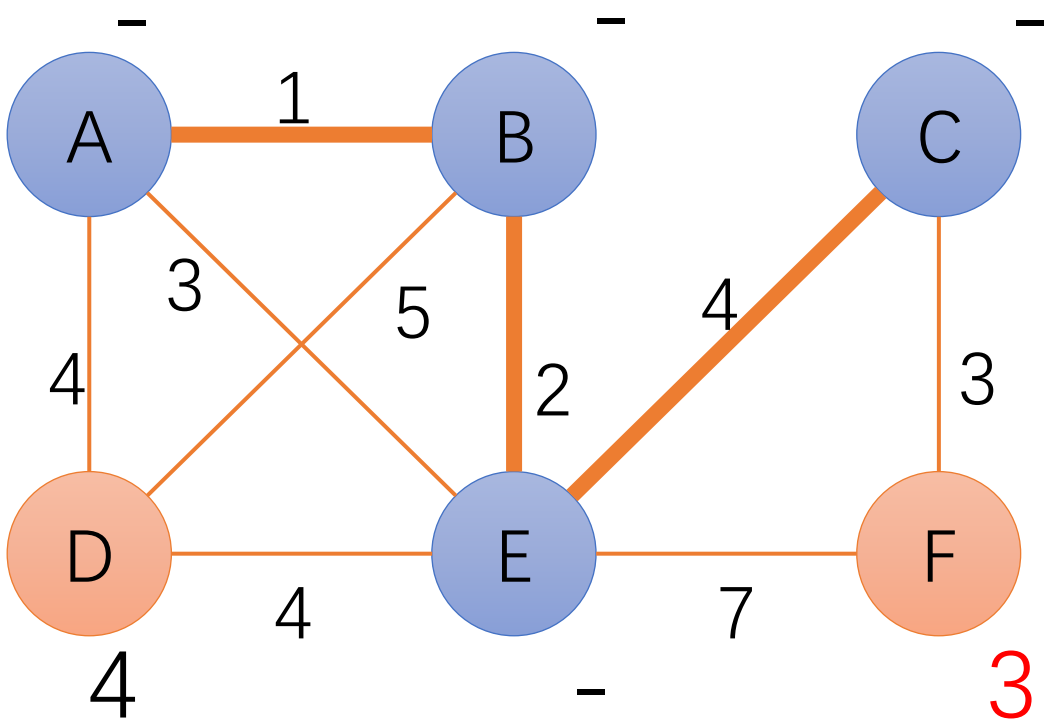
Prim's MST in Action



(D, 4)

(F, 7)

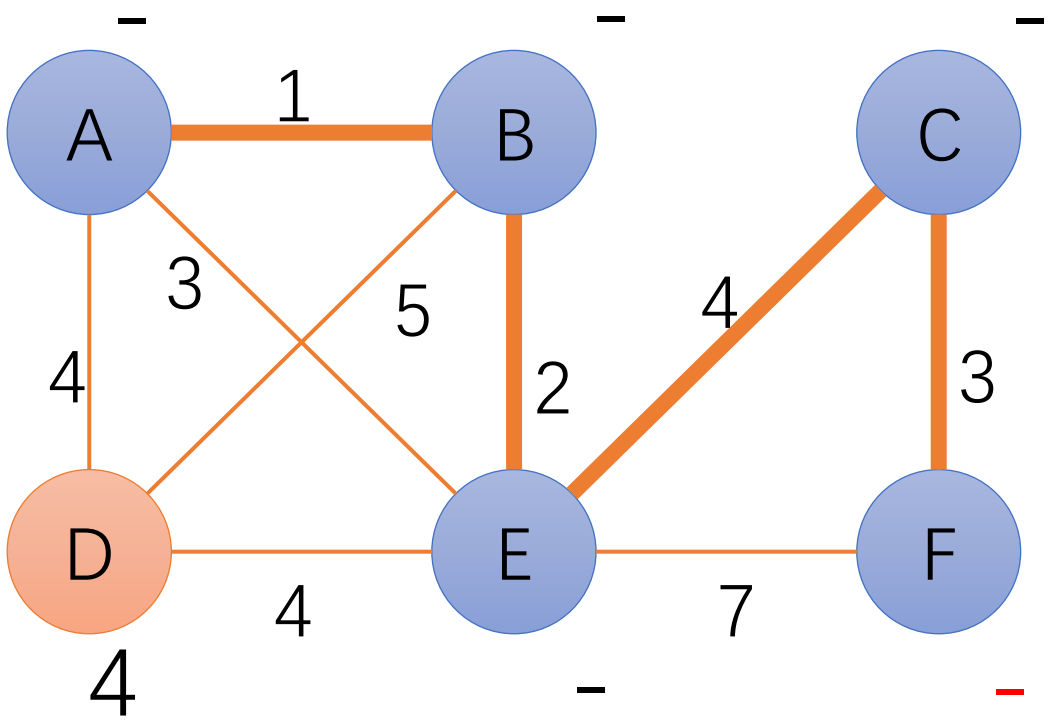
Prim's MST in Action



(F, 3)

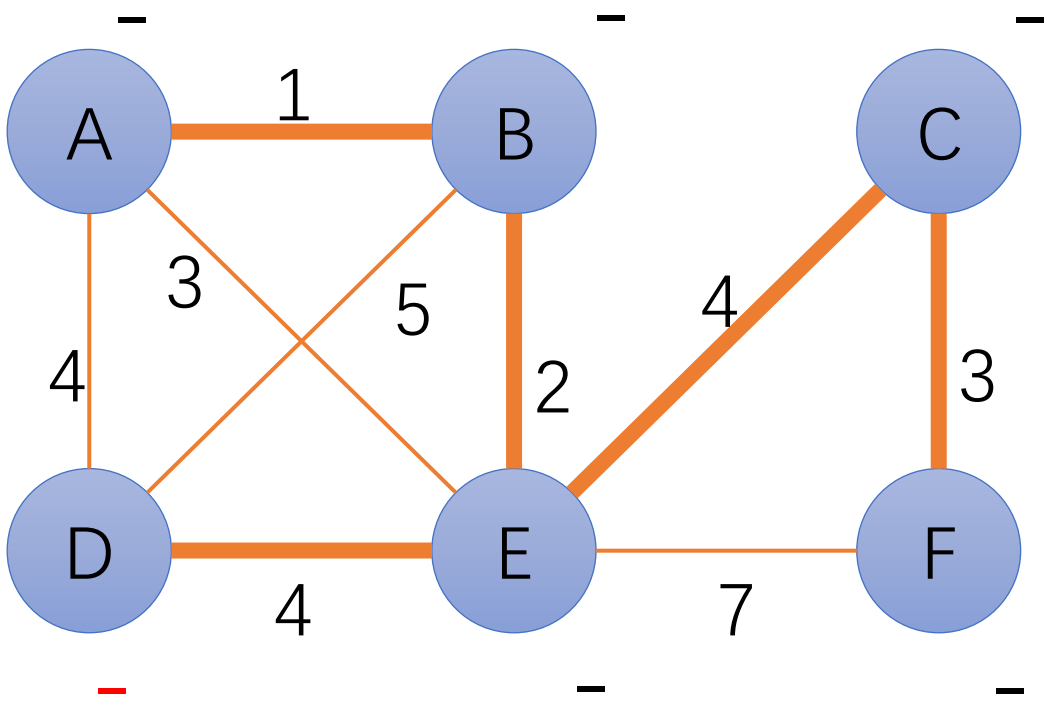
(D, 4)

Prim's MST in Action

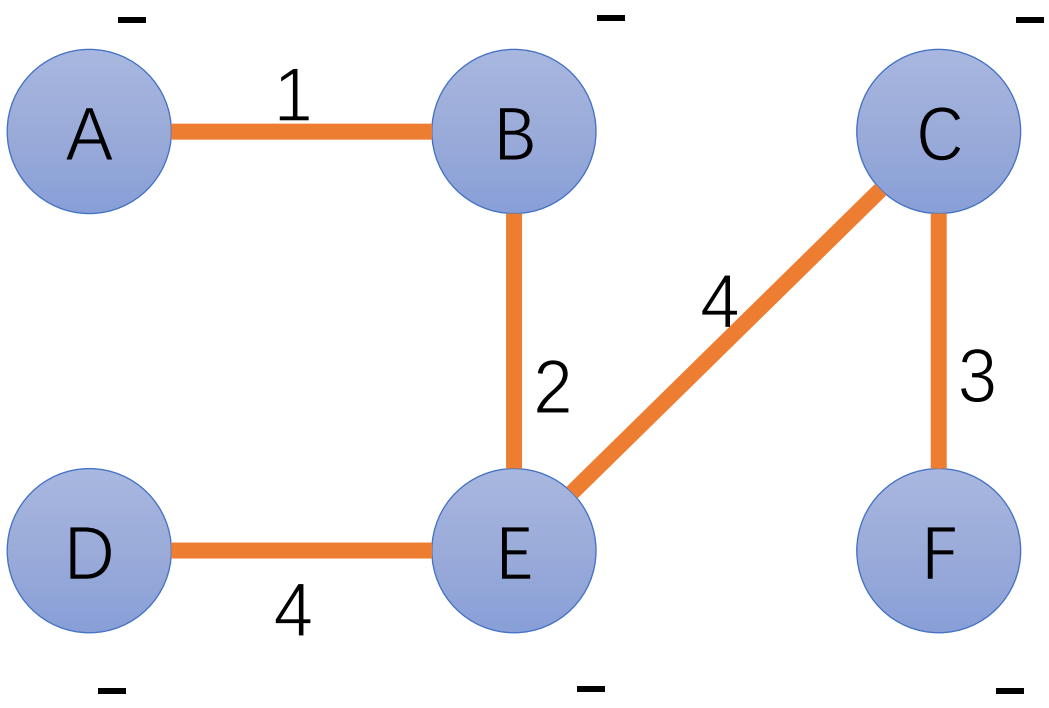


(D, 4)

Prim's MST in Action



Prim's MST in Action



Prim's Algorithm (simple array-based implementation)

$O(n)$ • $\delta(u) = \infty$ **for all** $u \in V$, 0 **for** $\delta(s)$

$O(1)$ • $MST = \emptyset$

$O(1)$ • $Q = \{s\}$

• **while** $Q \neq \emptyset$ // n iterations

$O(n)$ • $u =$ **Extract-Min**(Q)

• $MST = MST \cup \{(u, from(u))\}$

• **for each** $v \in N(u)$ // m iterations

• **if** $w(u,v) < \delta(v)$ and v is not visited

$O(1)$ • $\delta(v) = w(u,v)$

• $from(v) = u$

• **(update** v **in** Q**)**

$O(n^2)$

Prim's Algorithm (based on binary heap)

$O(n)$ • $\delta(u) = \infty$ **for all** $u \in V$, 0 **for** $\delta(s)$

$O(1)$ • $MST = \emptyset$

$O(1)$ • $Q = \{s\}$

• **while** $Q \neq \emptyset$ // n iterations

$O(\log n)$ • $u =$ **Extract-Min**(Q)

• $MST = MST \cup \{(u, from(u))\}$

• **for each** $v \in N(u)$ // m iterations

• **if** $w(u,v) < \delta(v)$ and v is not visited

• $\delta(v) = w(u,v)$

• $from(v) = u$

$O(\log n)$ • **(update** v **in** Q**)**

$O(m \log n)$

Prim's Algorithm (Fibonacci heap, CLRS Section 19)

$O(n)$ • $\delta(u) = \infty$ **for all** $u \in V$, 0 **for** $\delta(s)$

$O(1)$ • $MST = \emptyset$

$O(1)$ • $Q = \{s\}$

• **while** $Q \neq \emptyset$ // n iterations

$O(\log n)$ • $u =$ **Extract-Min**(Q)

• $MST = MST \cup \{(u, from(u))\}$

• **for each** $v \in N(u)$ // m iterations

• **if** $w(u,v) < \delta(v)$ and v is not visited

• $\delta(v) = w(u,v)$

• $from(v) = u$

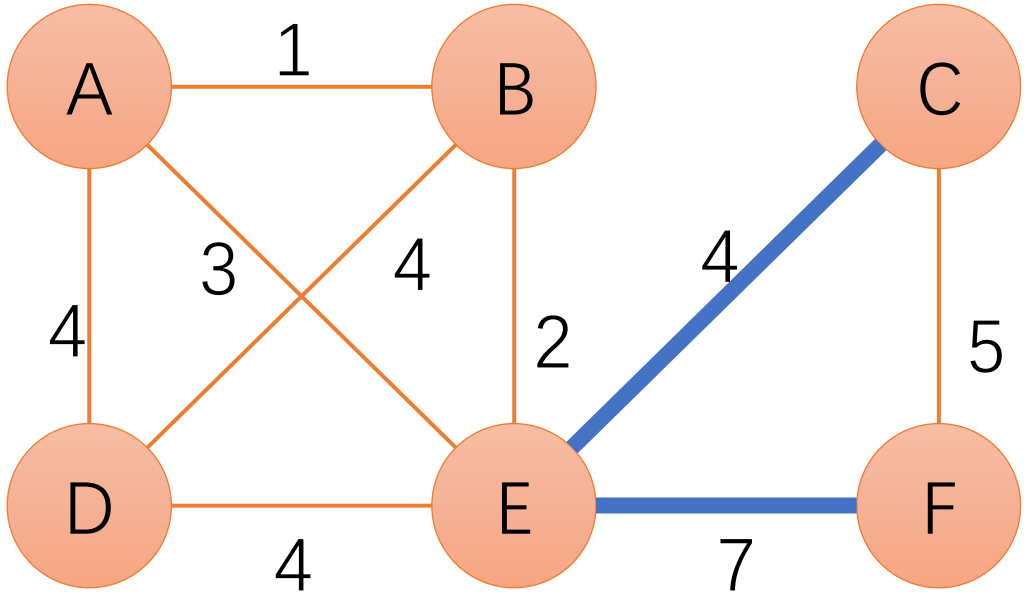
$O(1)$ • **(update** v **in** Q**)**

amortized

$O(m + n \log n)$

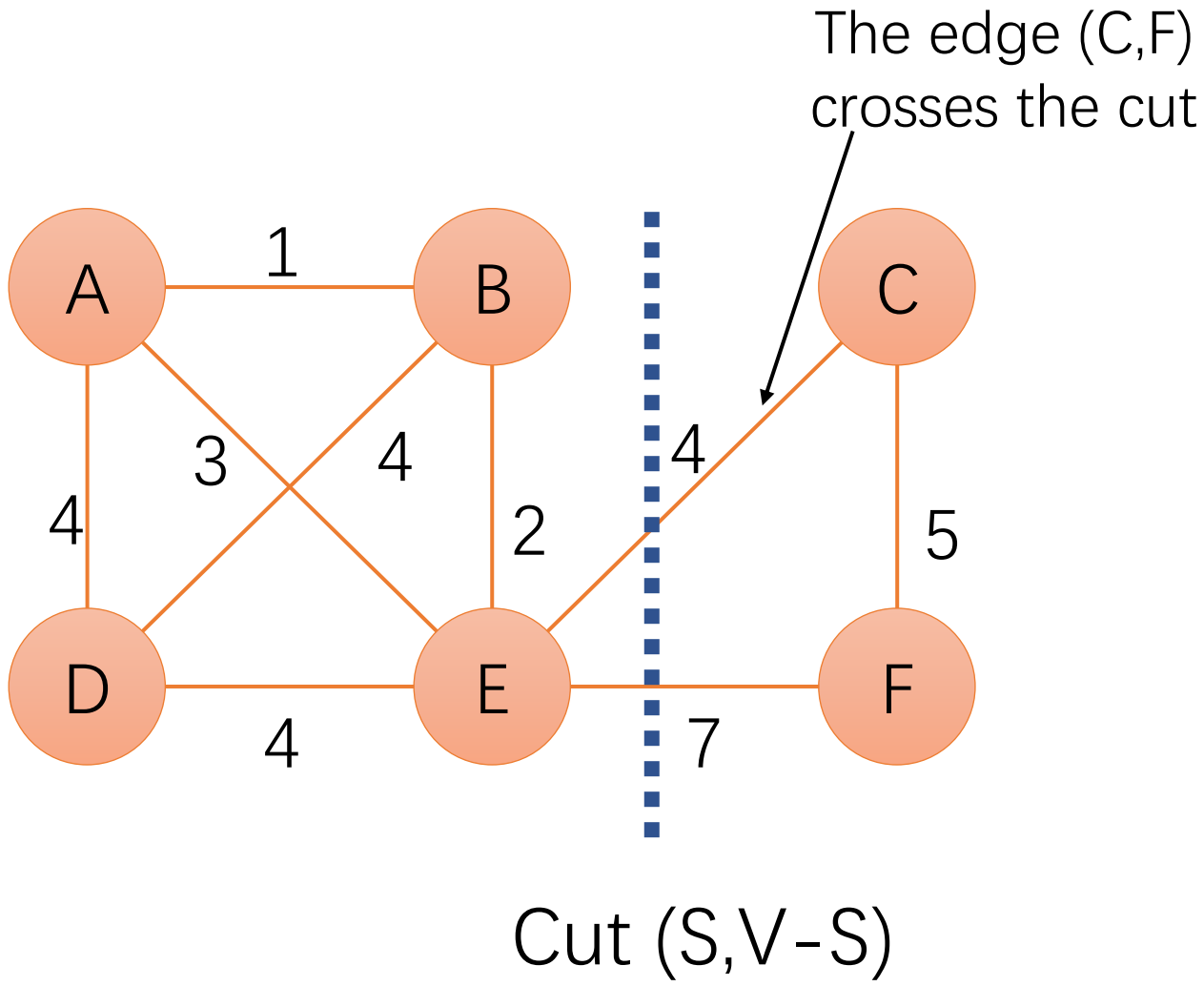
Proof of Optimality

Which of them must be included?



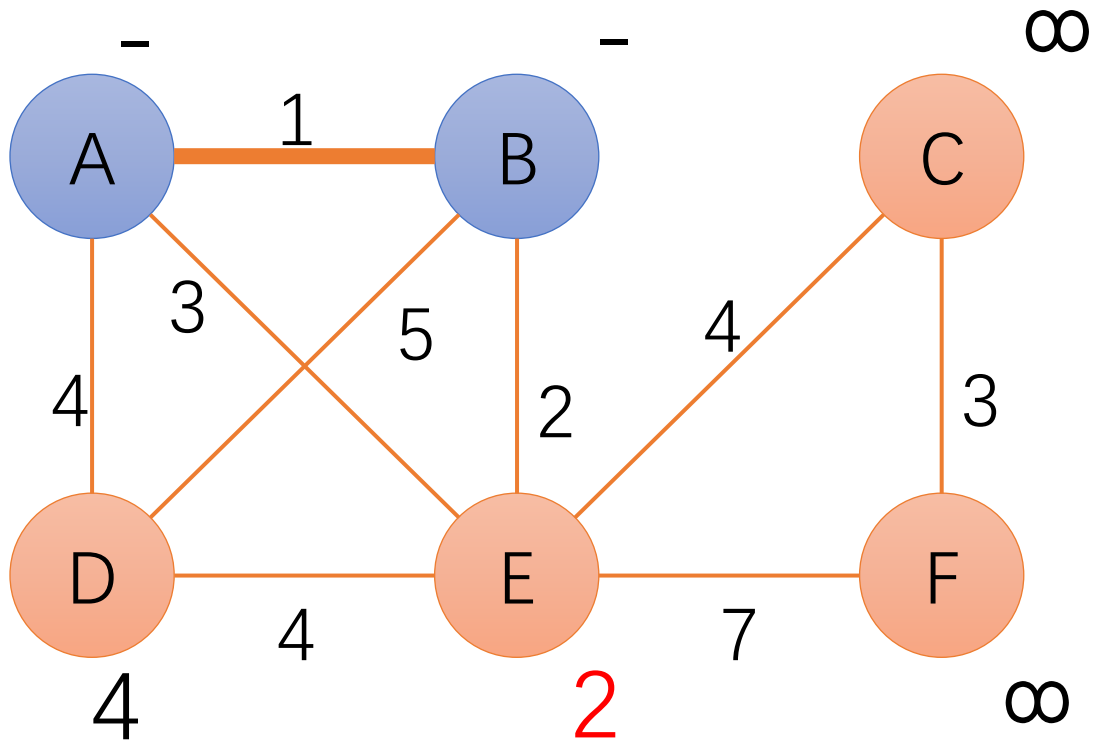
A cut of a graph

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



Do all edges chosen by Prim's light edges?

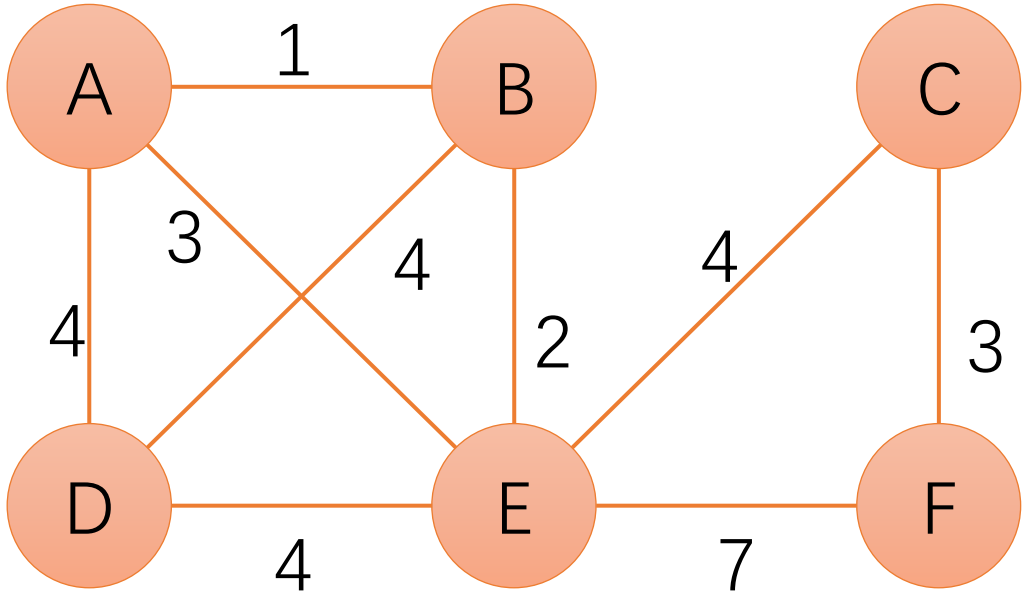
- **Every time we choose an edge, it**
  - Must be connecting two parts (the current tree and the rest)
  - Must be the shortest among such edges (compare and find min of them)



(E, 2)

(D, 4)

Similar for Kruskal's MST algorithm




```
(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)
```

Summary

Minimum spanning tree

- **Find a (spanning) tree of a graph with minimum sum of weights**
  - Can also be defined to minimize maximum weight, with degree-bounded, or minimizing max/min ratio
- **Kruskal's algorithm: sort the edges by weight, and add edges in turn**
  - Basic requirement: understand the algorithm, implement the algorithm using an array
  - Full requirement: understand the correctness, and how union-find can help
- **Prim's algorithm: grow the connected part by a vertex at a time**
  - Basic requirement: understand the algorithm, implement the algorithm using an array
  - Full requirement: understand the correctness, and how binary heap can help
- **About minimum spanning tree**
  - Understand that the min-edge on any cut must be on MST, and use it to analyze other MST algorithms and design graph algorithms for related problems