

CS2040C

# Data Structures and Algorithms

All about minimum spanning trees...

# Roadmap

---

Today: Minimum Spanning Trees

- Prim's Algorithm
- Kruskal's Algorithm
- Boruvka's Algorithm

Variations:

- Constant weight edges
- Bounded integer edge weights
- Euclidean
- Directed graphs
- Maximum Spanning Tree
- Steiner Tree

# Roadmap

---

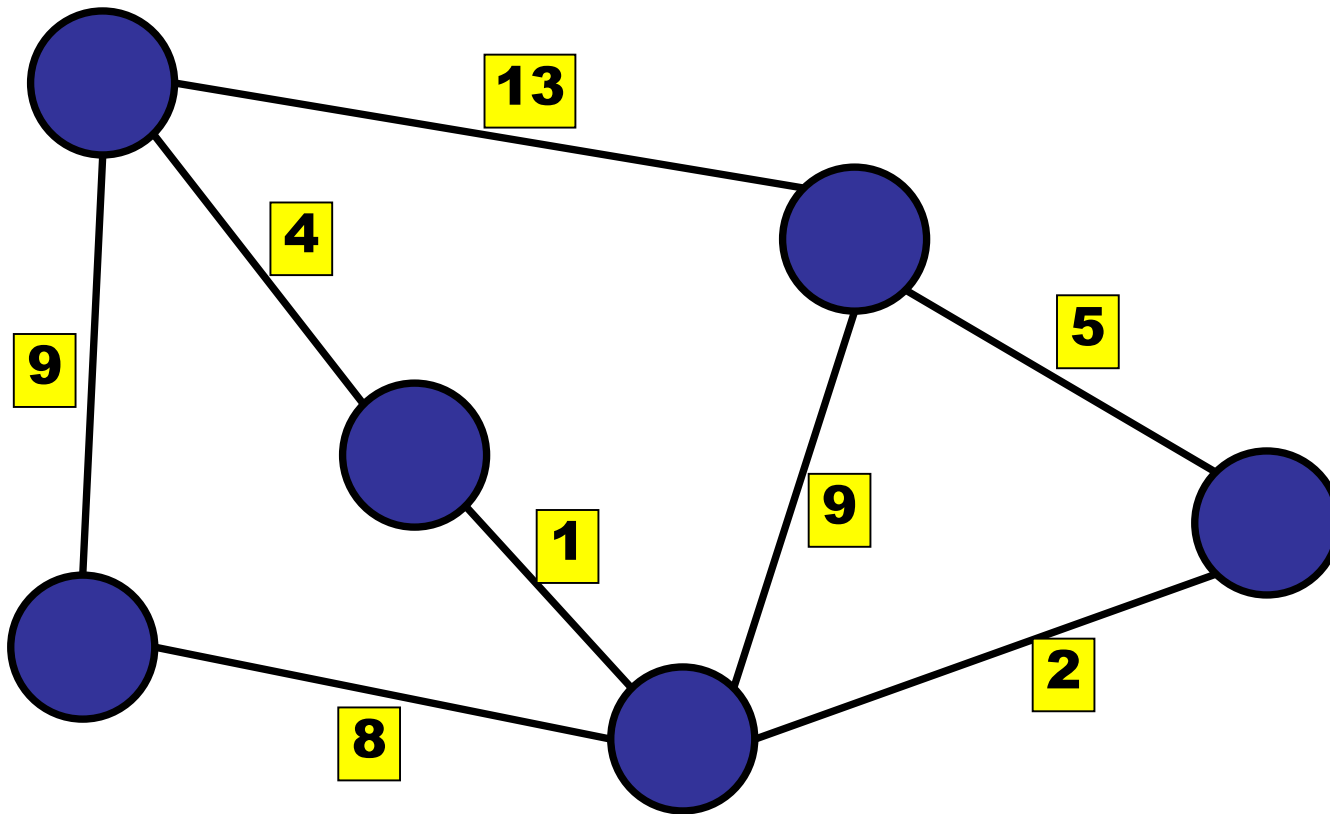
## Minimum Spanning Trees

- **The MST Problem**
- Basic Properties of an MST
- Generic MST Algorithm
- Prim's Algorithm
- Kruskal's Algorithm
- Boruvka's Algorithm
- Variations

# Spanning Tree

---

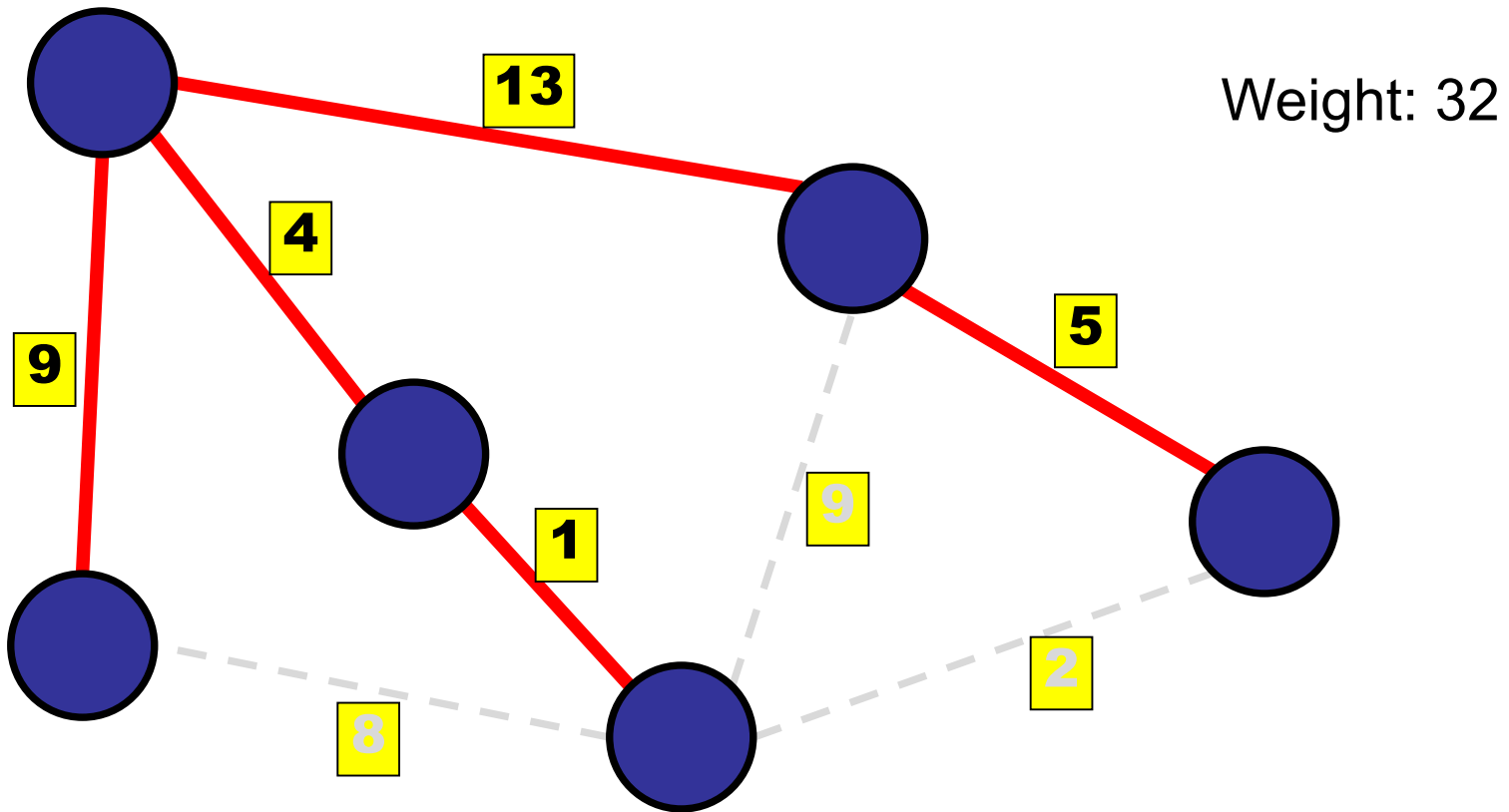
Weighted, undirected graph:



# Spanning Tree

---

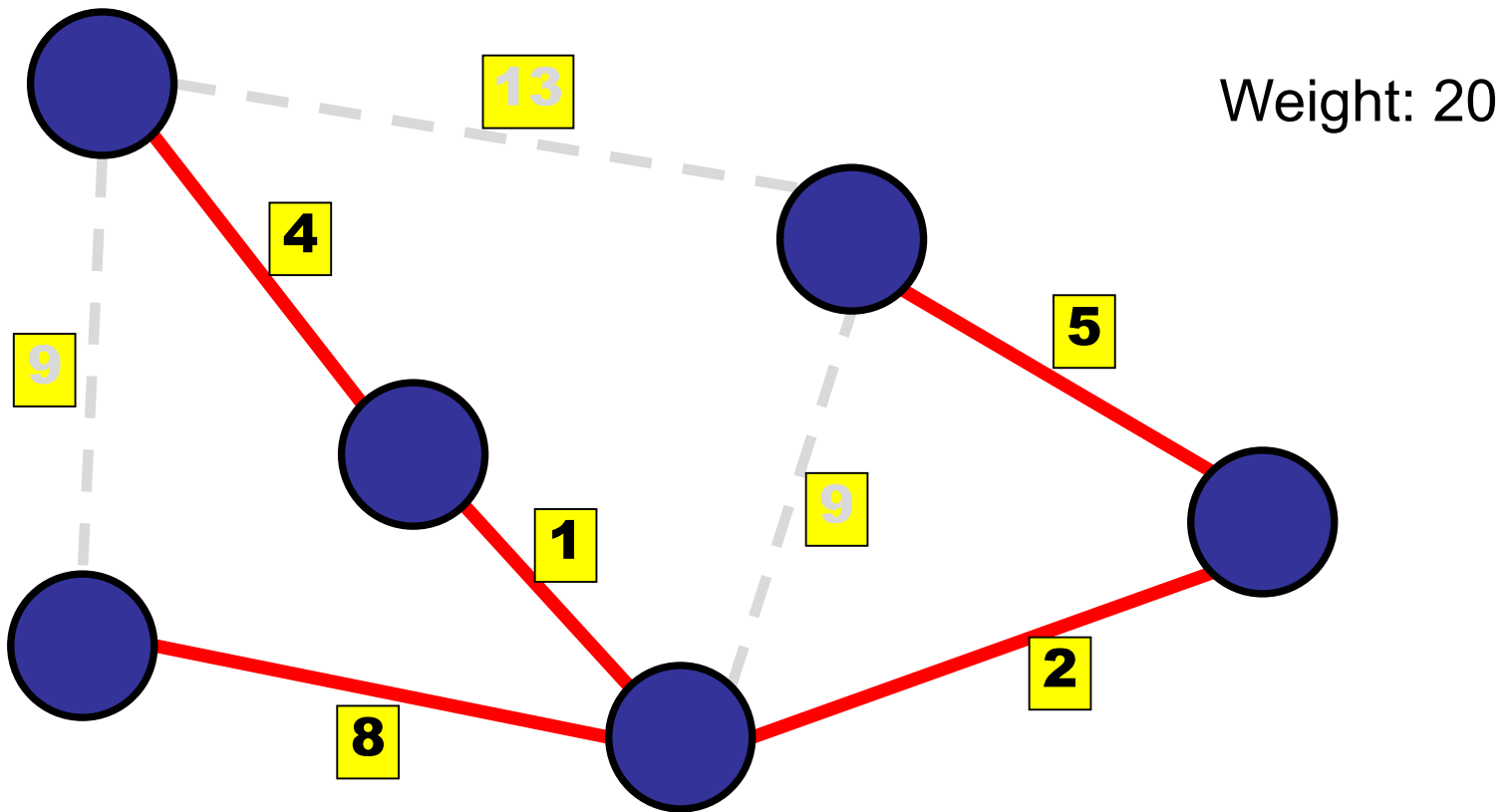
Definition: a **spanning tree** is an acyclic subset of the edges that connects all nodes



# Minimum Spanning Tree

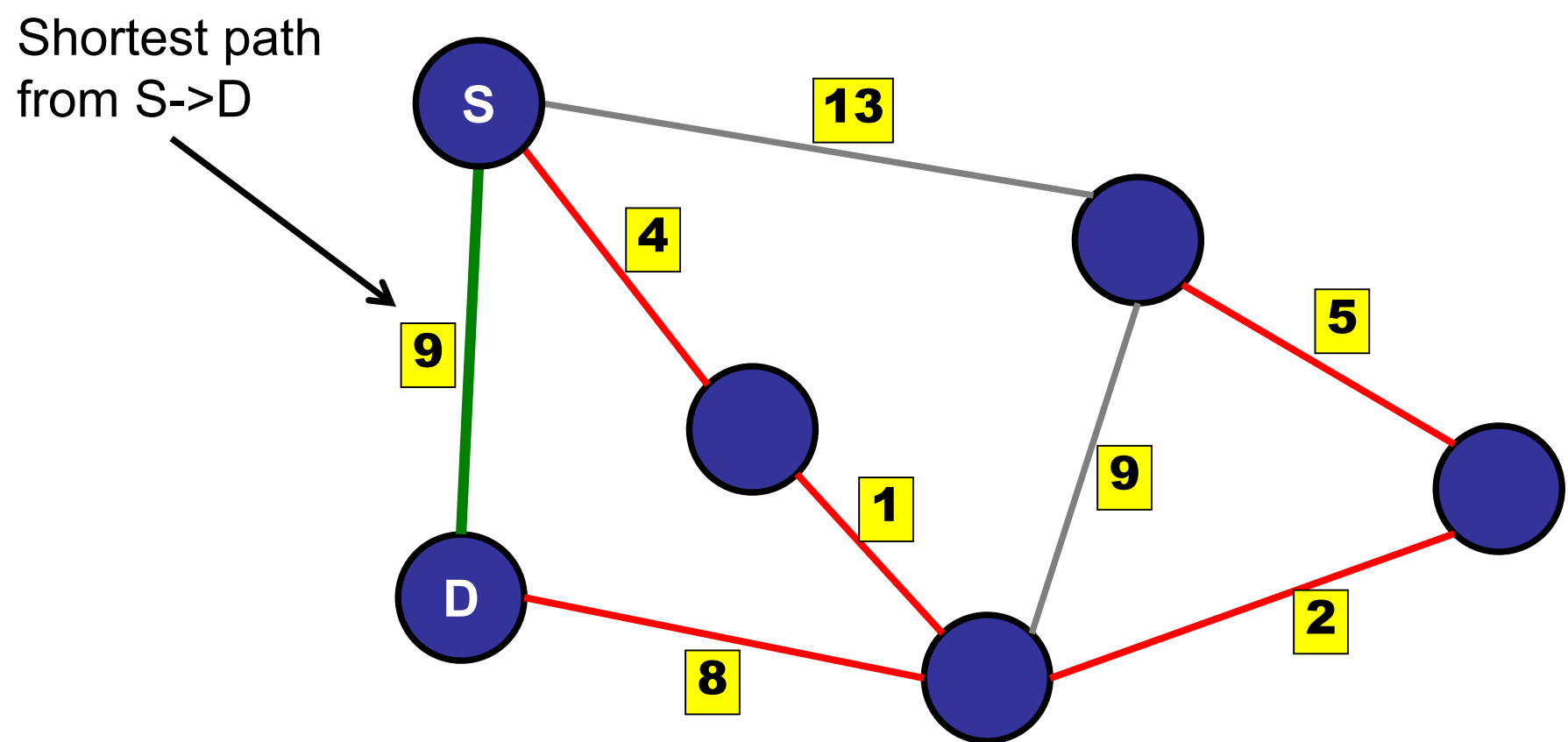
---

Definition: a spanning tree with minimum weight



# Minimum Spanning Tree

Not the same a shortest paths:



# Applications of MST

---

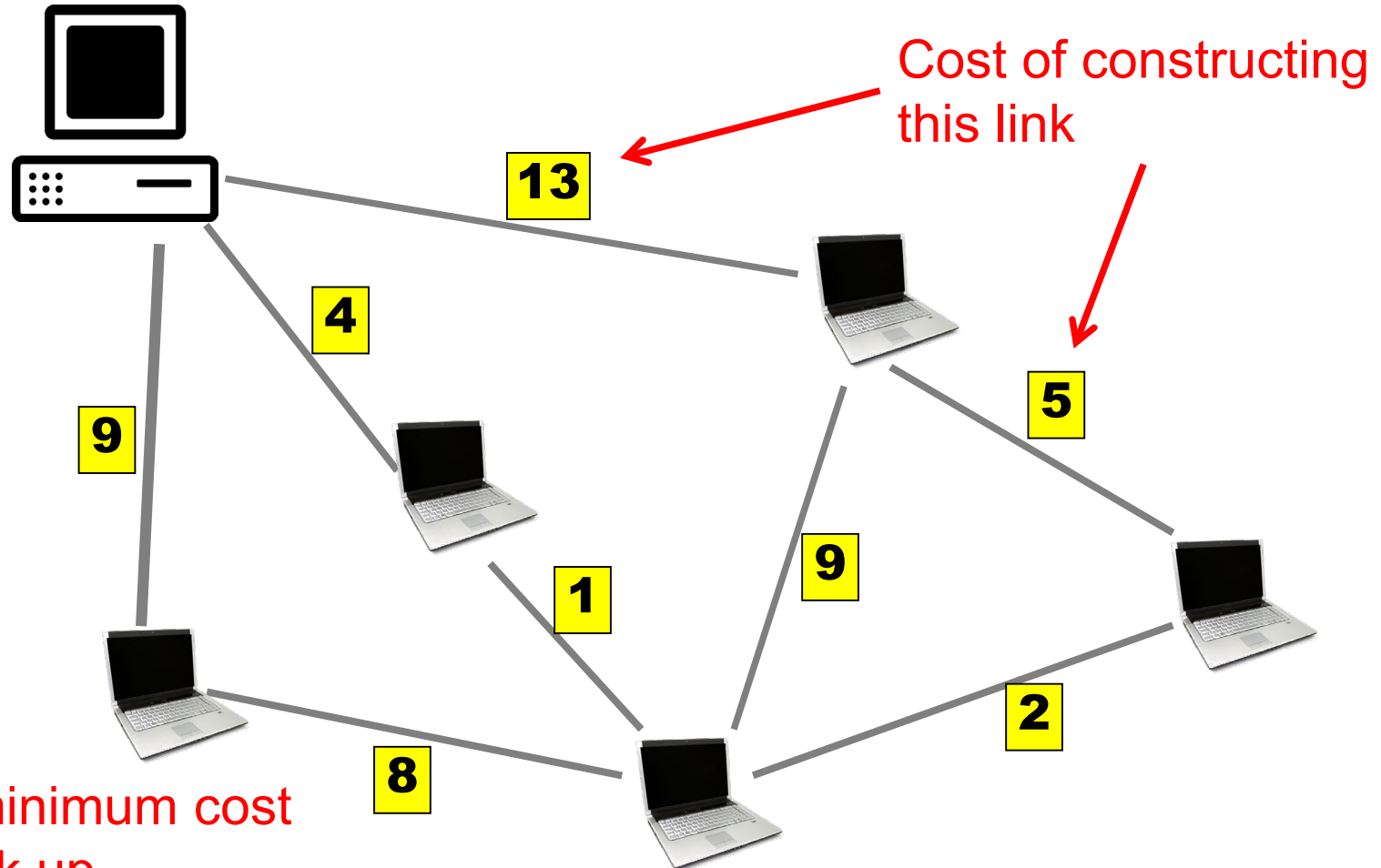
Many applications:

- Network design
  - Telephone networks
  - Electrical networks
  - Computer networks
  - Ethernet autoconfig
  - Road networks
  - Bottleneck paths



# Data distribution

Network:



What is the minimum cost network to link up all the users?

# Applications of MST

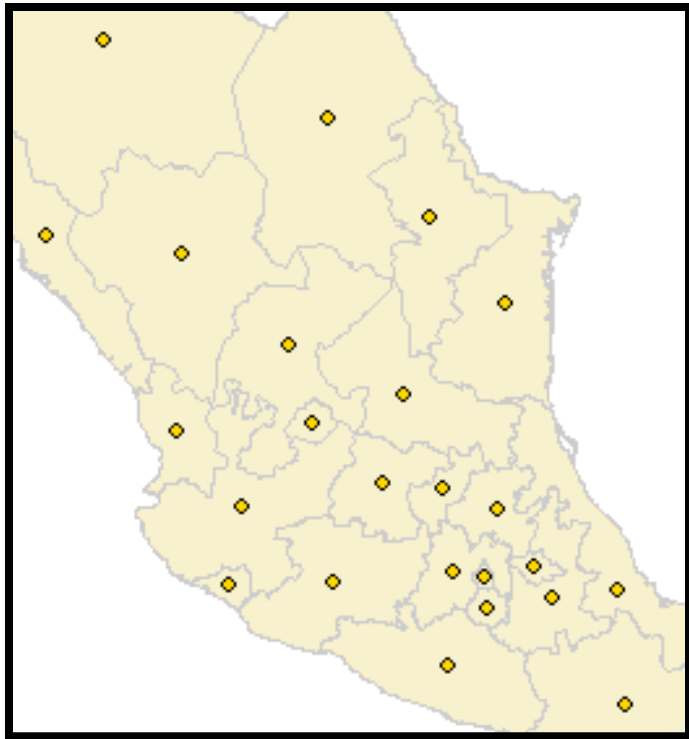
---

Many applications:

- Many other
  - Error correcting codes
  - Face verification
  - Cluster analysis
  - Image registration

# Euclidean Minimal Spanning Tree

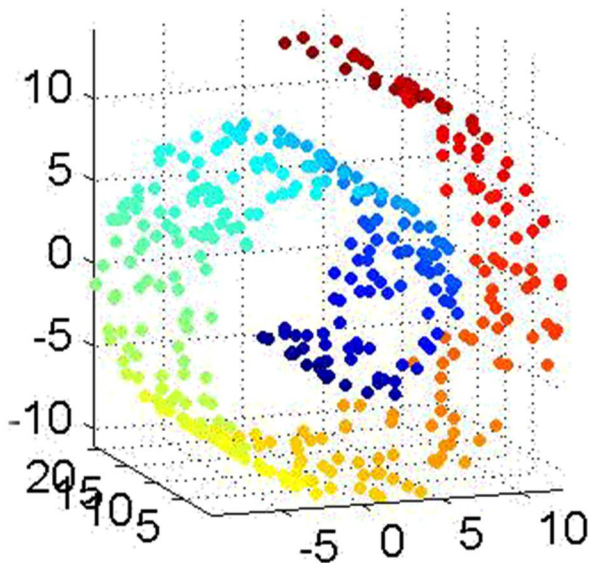
- Given point set  $P$ ,  $EMST(P)$  is the tree that spans  $P$  and the sum of lengths of all edges is minimal



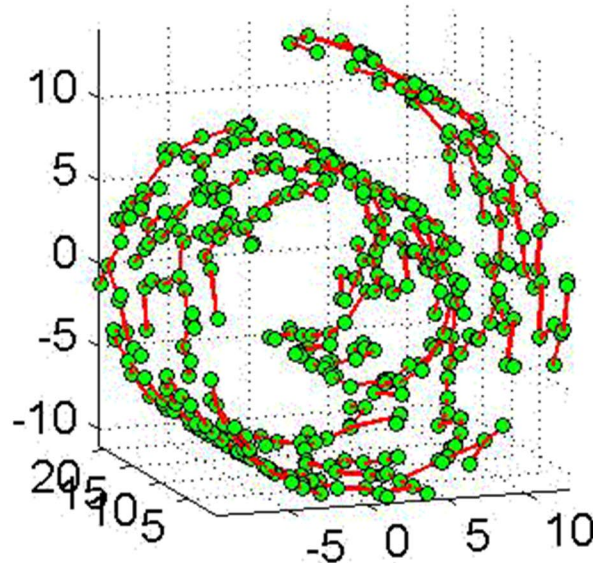
# Discovering structures (mainly manifold) for high dimensional data

- In machine learning, pattern recognition, data mining, etc

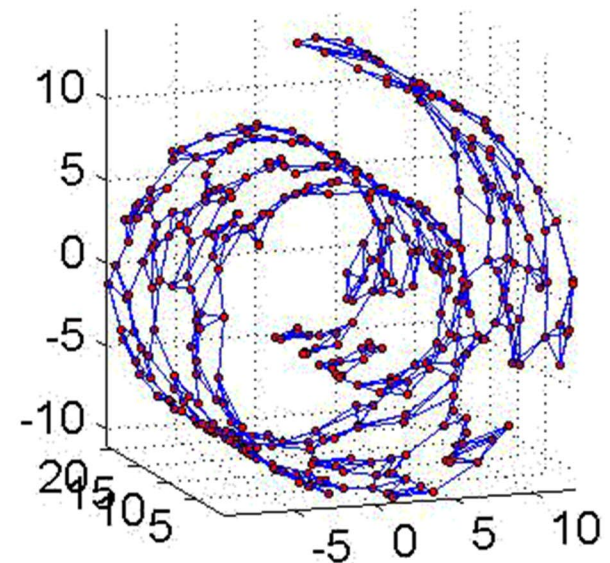
Data points



GMST

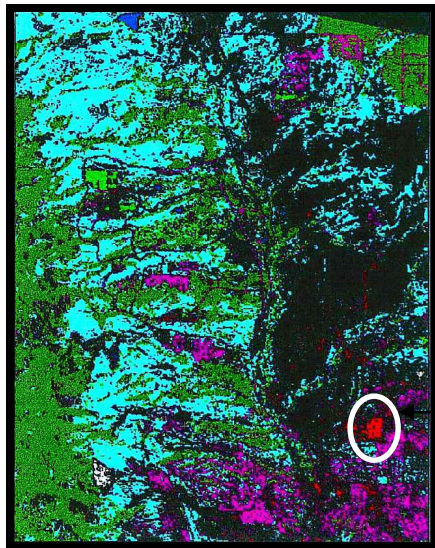


4-NN

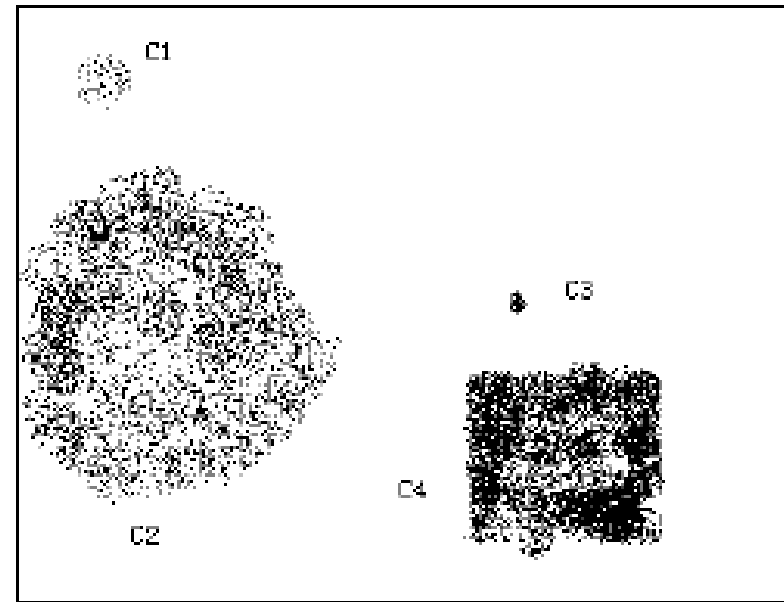


# Anomaly Detection

- Anomaly is a pattern in the data that does not conform to the expected behavior.
- E.g. Cyber intrusions, credit card fraud, air traffic safety



Anomaly



# Roadmap

---

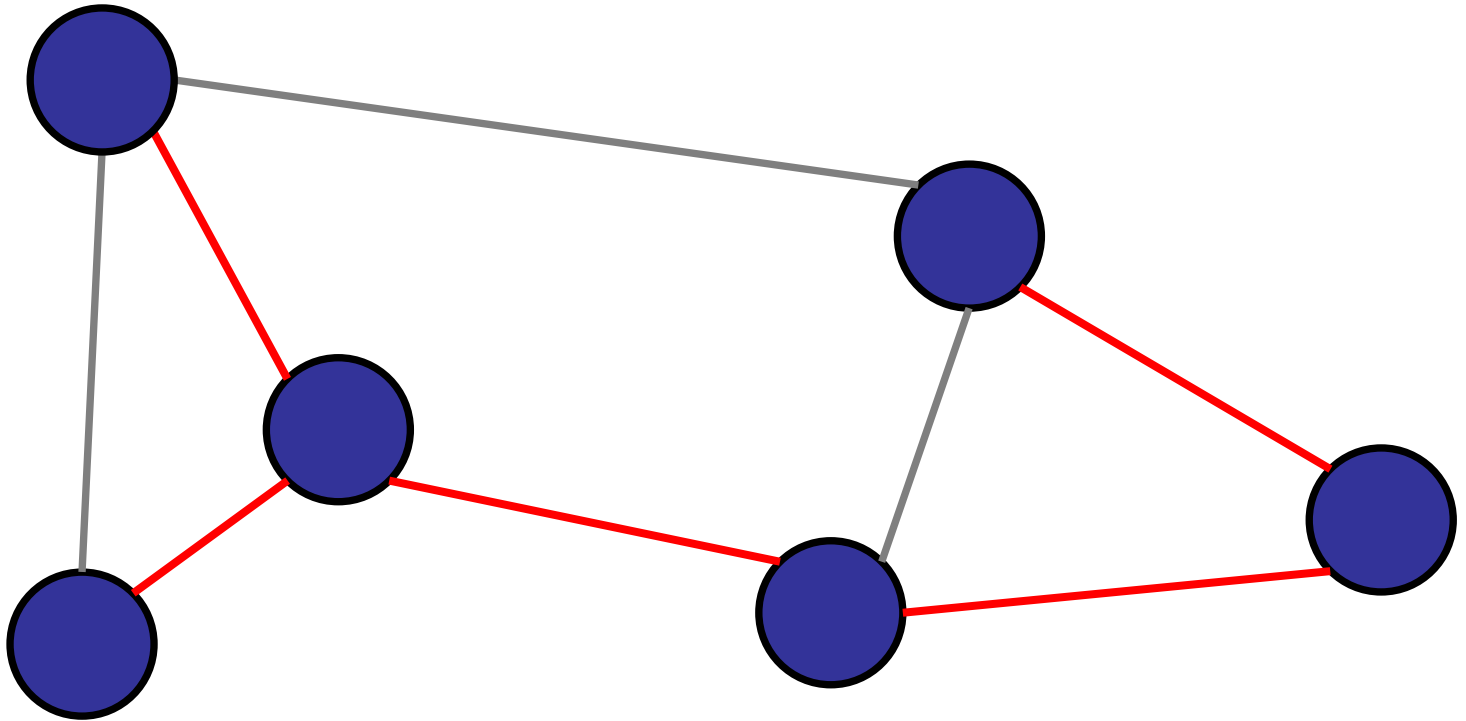
## Minimum Spanning Trees

- The MST Problem
- **Basic Properties of an MST**
- Generic MST Algorithm
- Prim's Algorithm
- Kruskal's Algorithm
- Boruvka's Algorithm
- Variations

# Properties of MST

---

## Property 1: No cycles

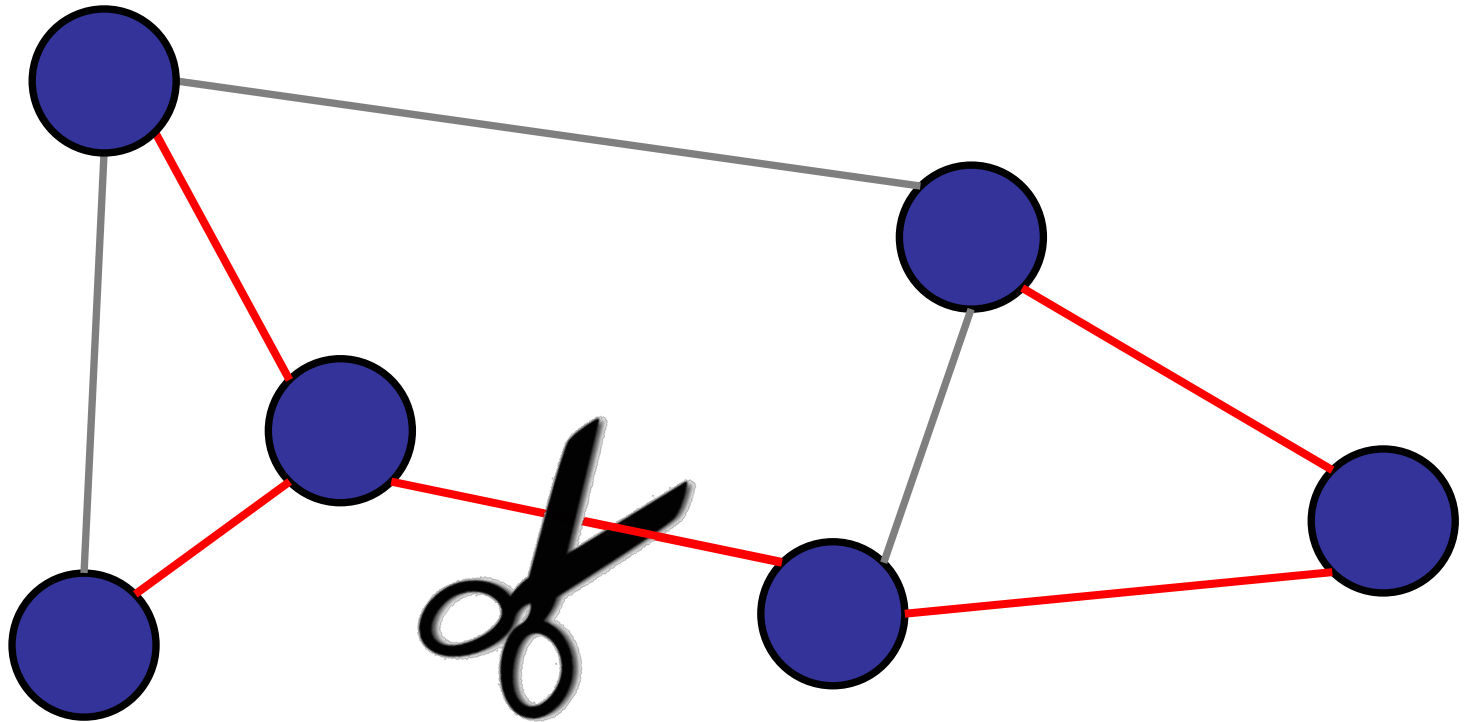


Why? If there were cycles, we could remove one edge and reduce the weight!

# Properties of MST

---

What happens if you cut an MST into T1 and T2??

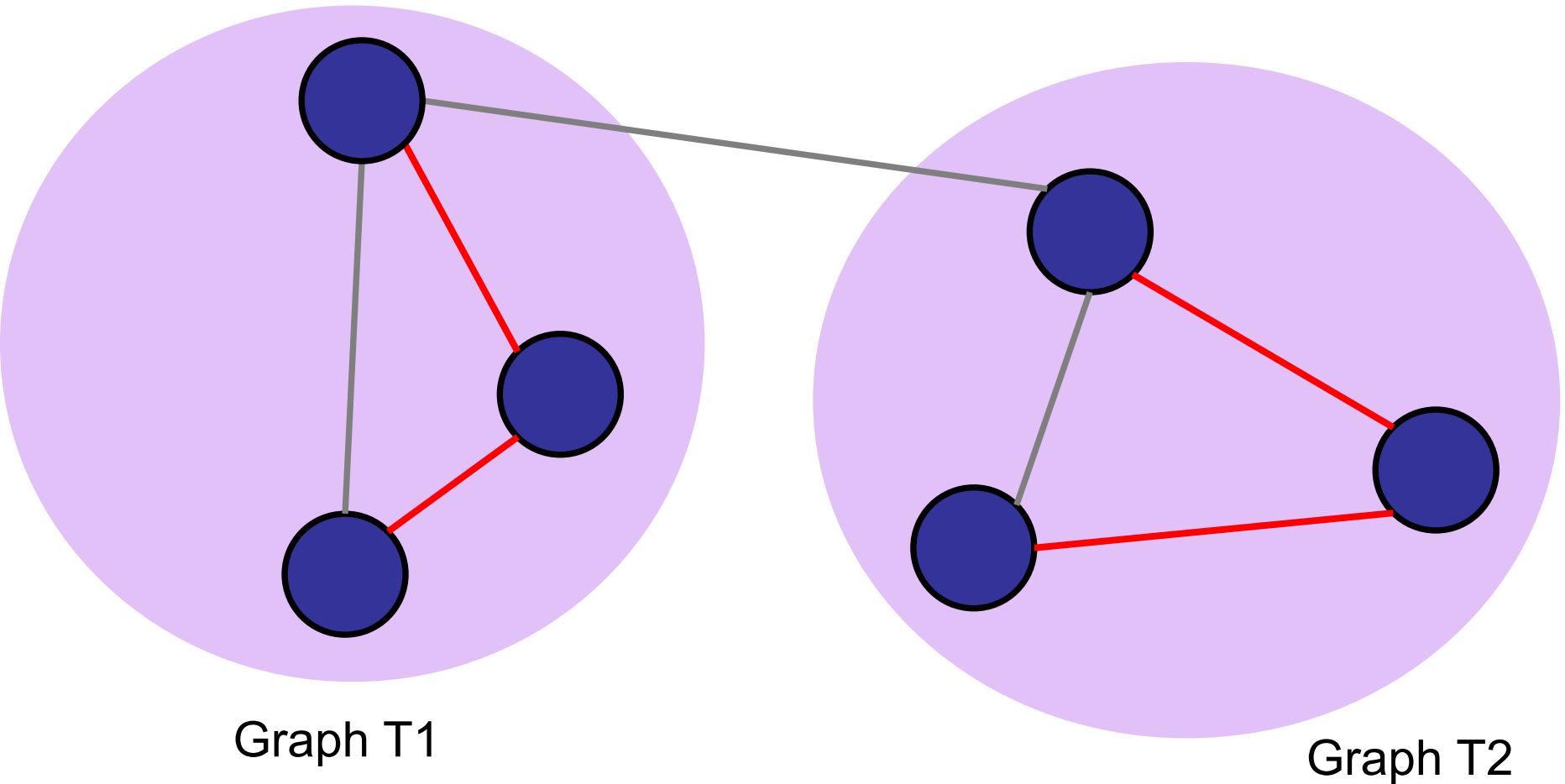




# Properties of MST

---

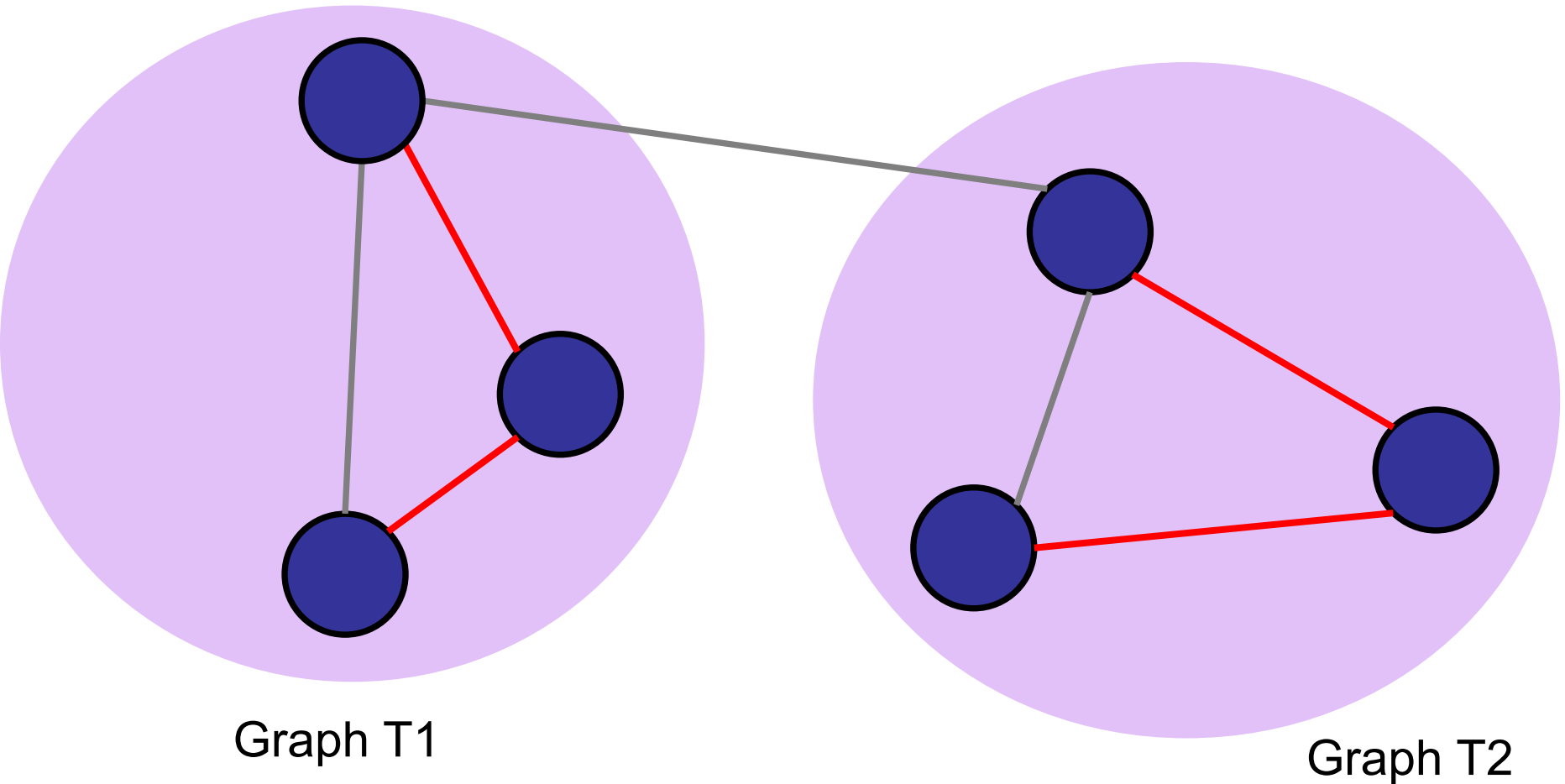
What happens if you cut an MST into T1 and T2?



# Properties of MST

---

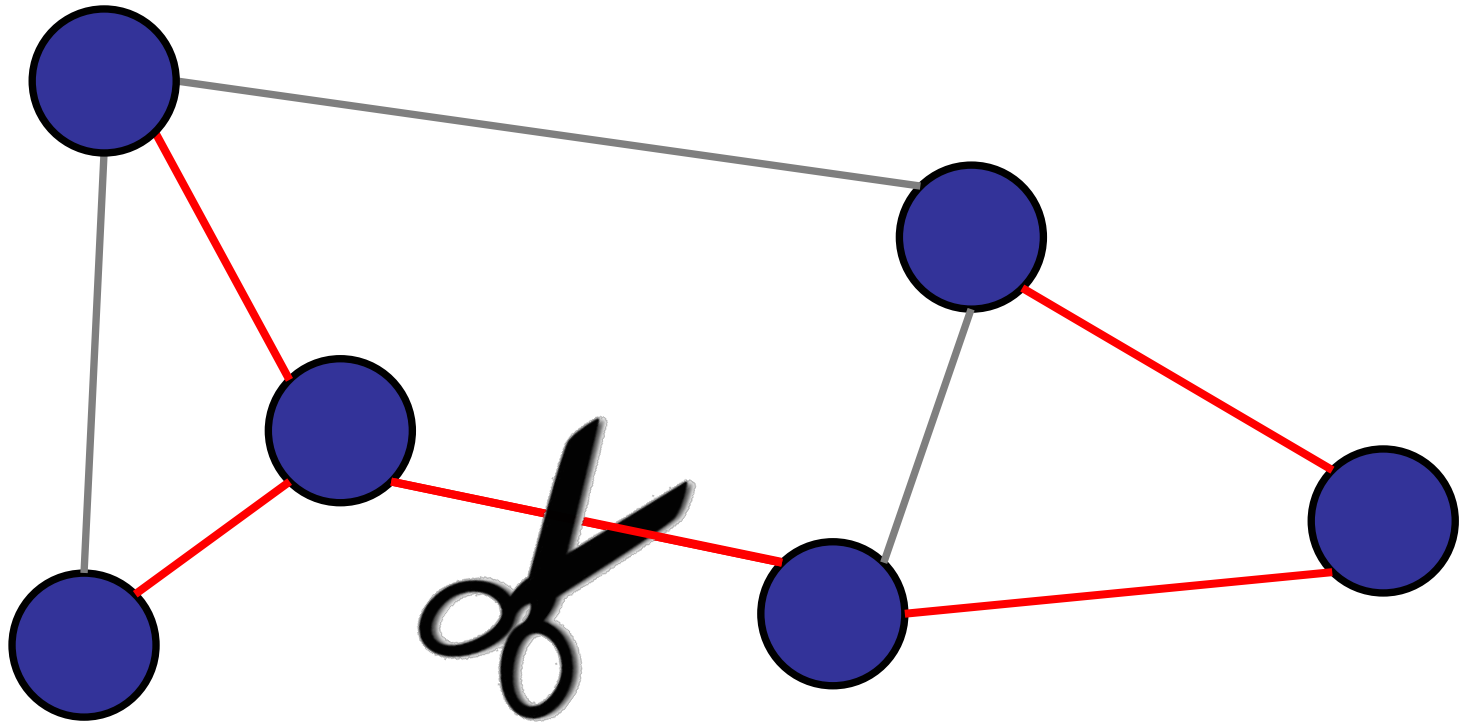
Theorem: T1 is an MST and T2 is an MST.



# Properties of MST

---

Property 2: If you cut an MST, the two pieces are both MSTs.



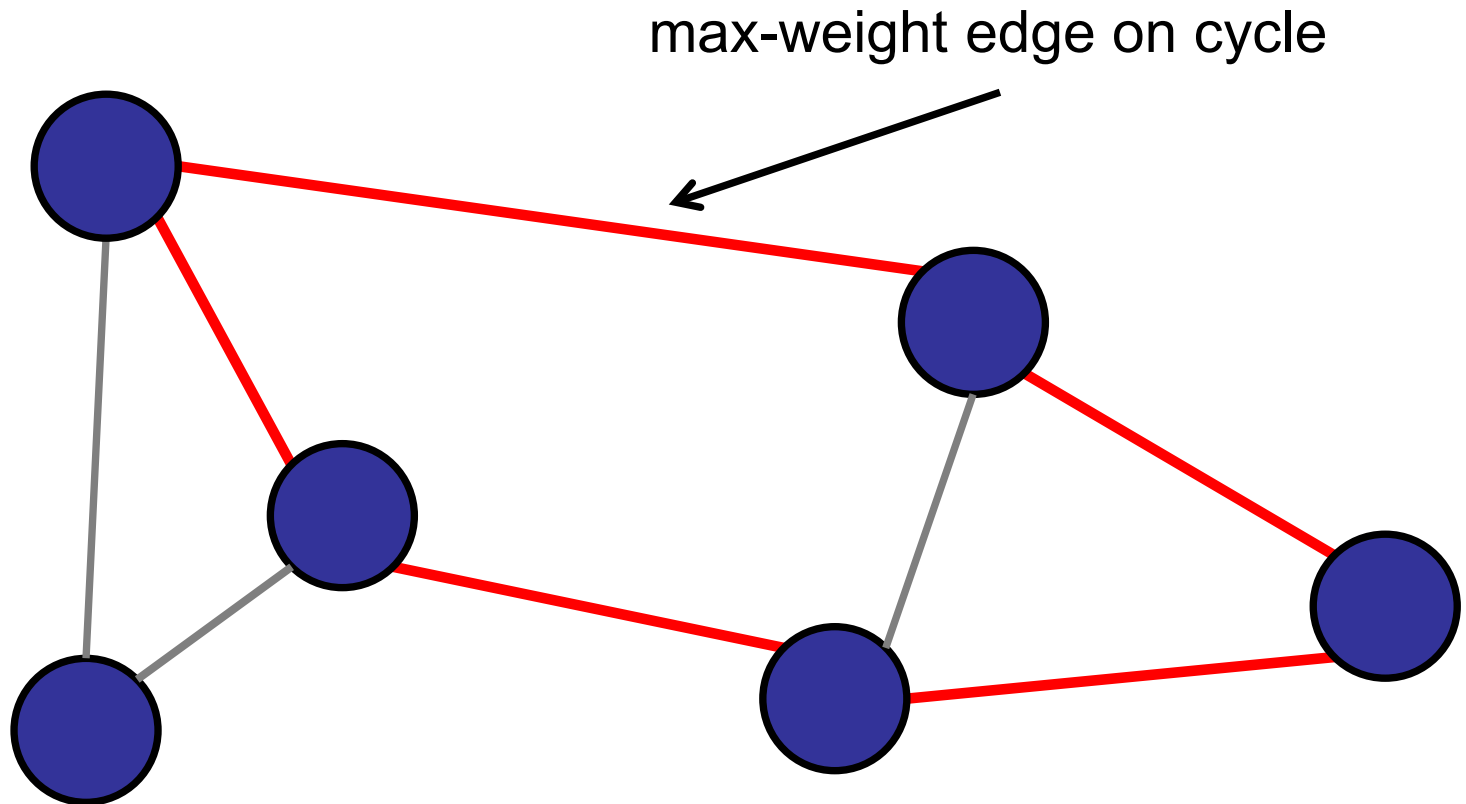
*Overlapping sub-problems! Dynamic programming? Yes, but better...*

# Properties of MST

---

## Property 3: Cycle property

For every cycle, the maximum weight edge is not in the MST.

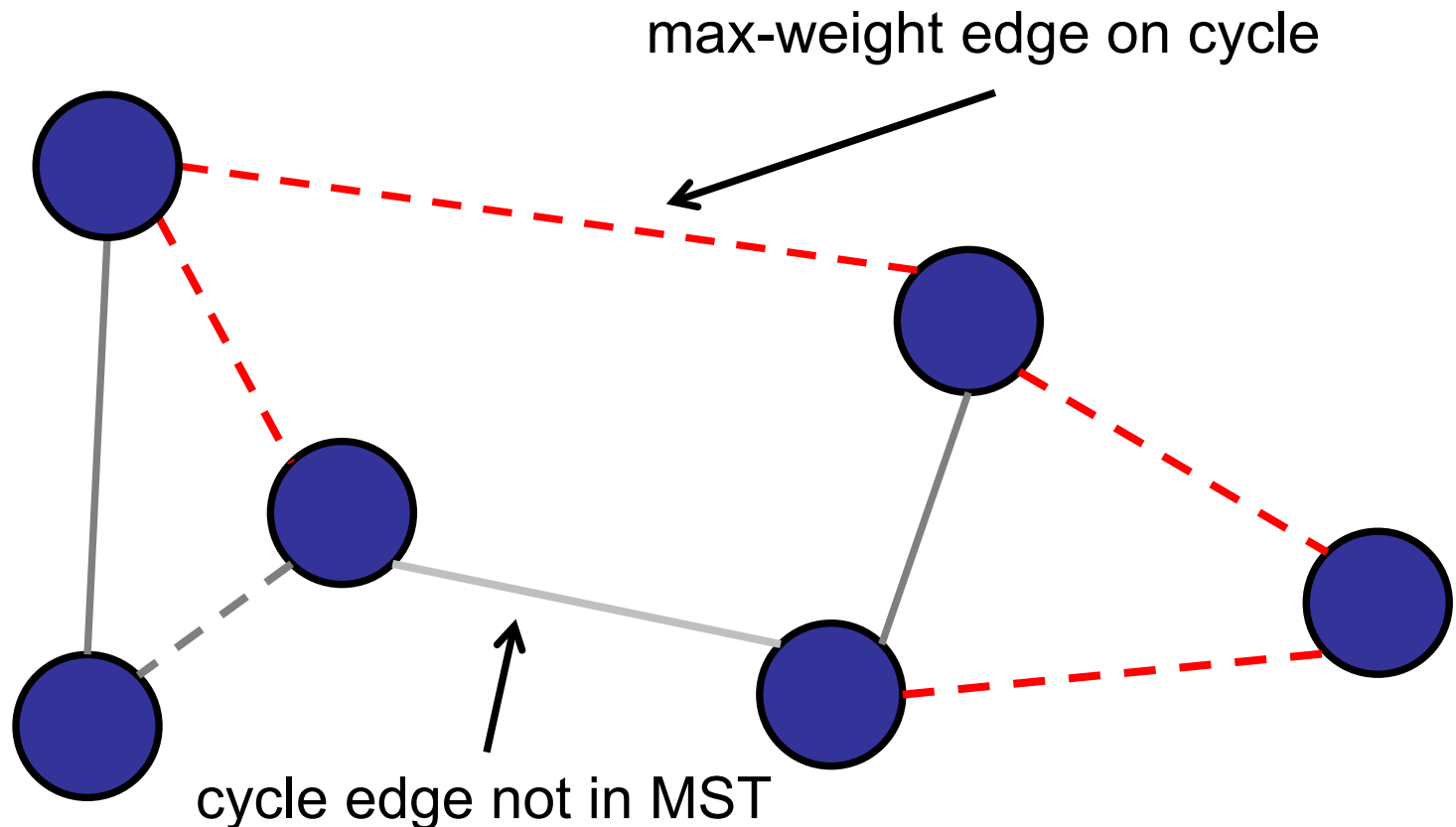


# Properties of MST

---

Proof: Cut-and-paste

Assume heavy edge is in the MST.



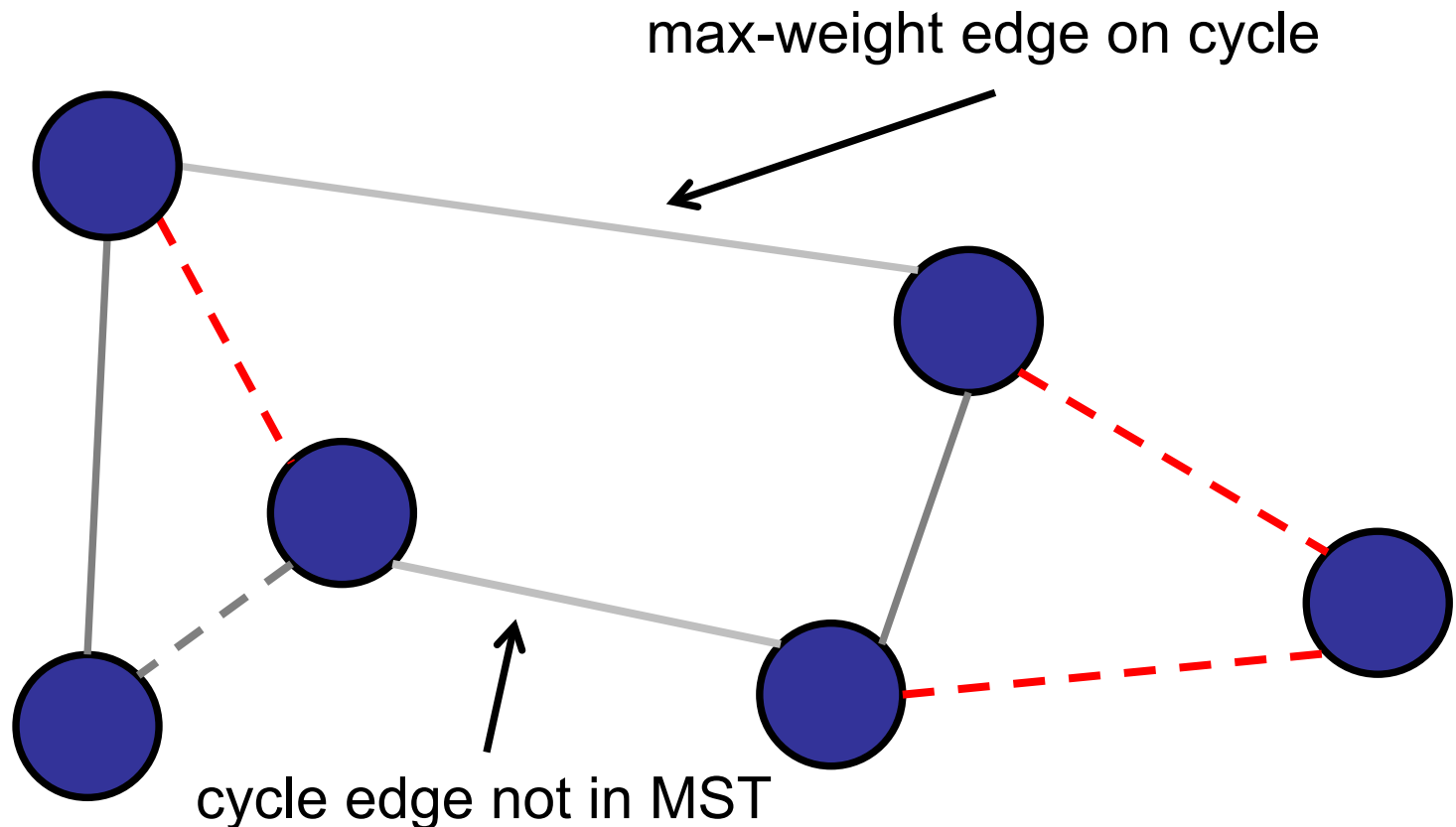
# Properties of MST

---

Proof: Cut-and-paste

Assume heavy edge is in the MST.

Remove max-weight edge; cuts graph.

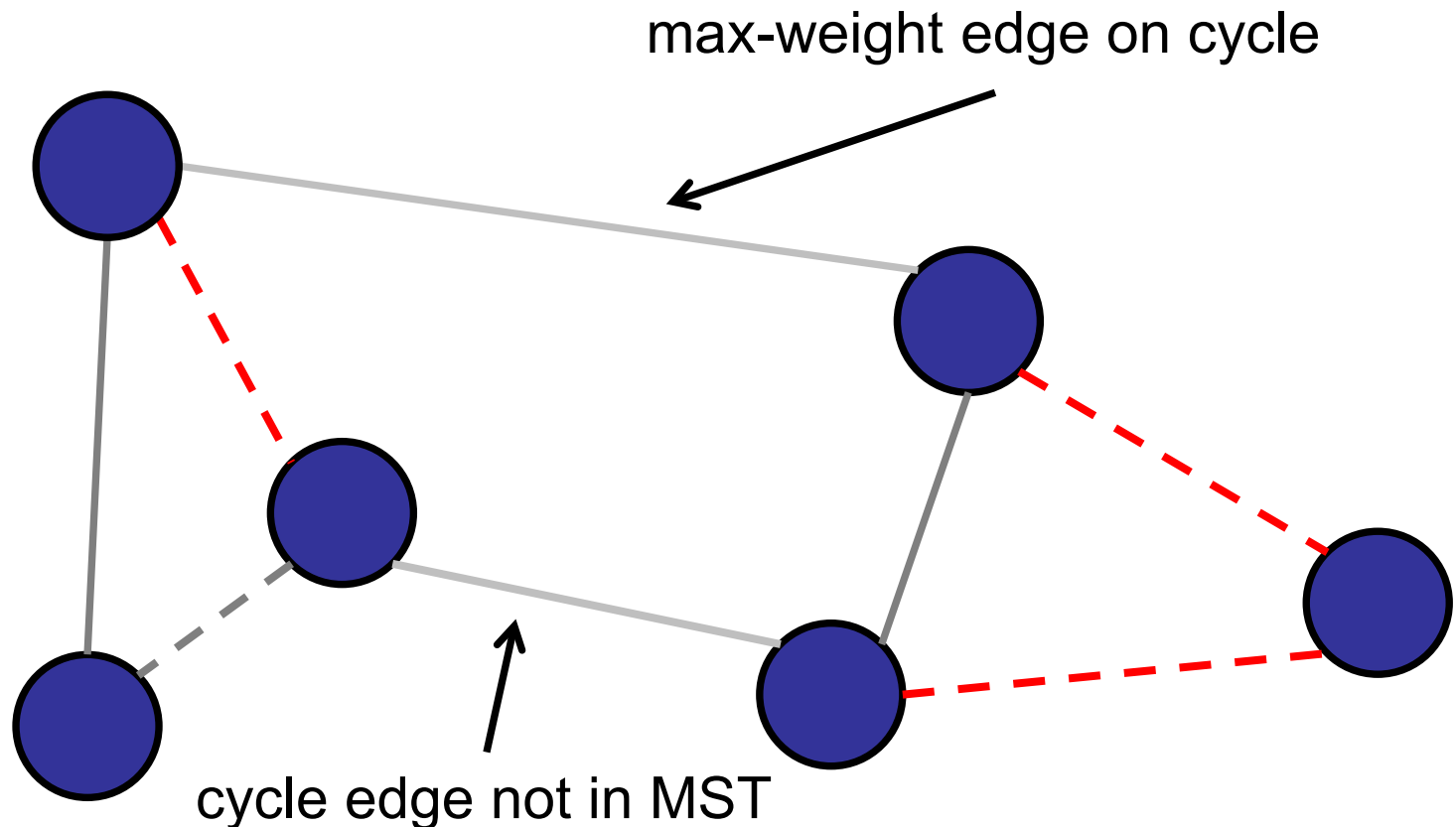


# Properties of MST

---

## Proof: Cut-and-paste

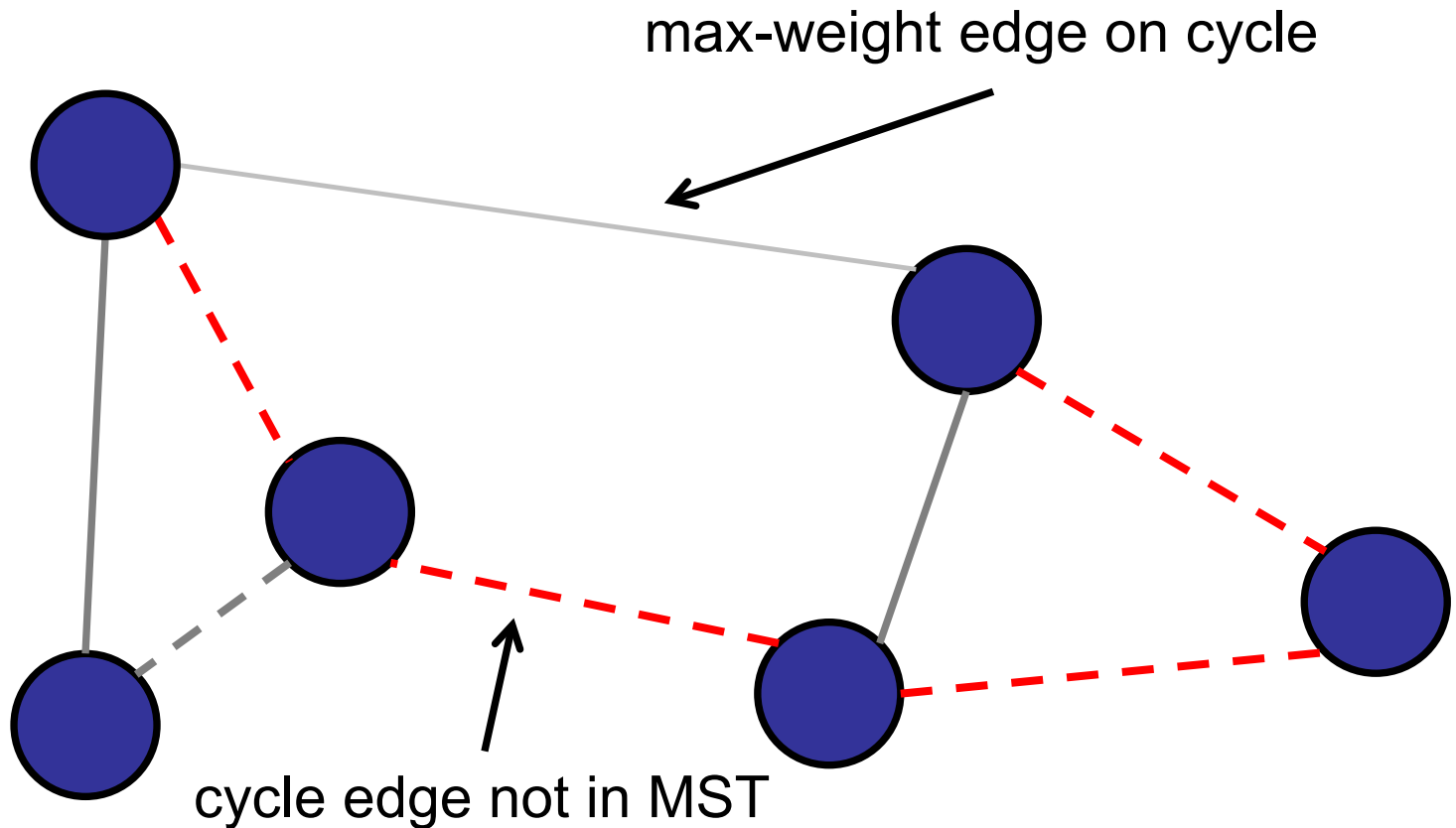
There exists another cycle edge that crosses the cut. (Even # of cycle edges across cut.)



# Properties of MST

# Proof: Cut-and-paste

Replace heavy edge with lighter edge.  
Still a spanning tree: Property 2.



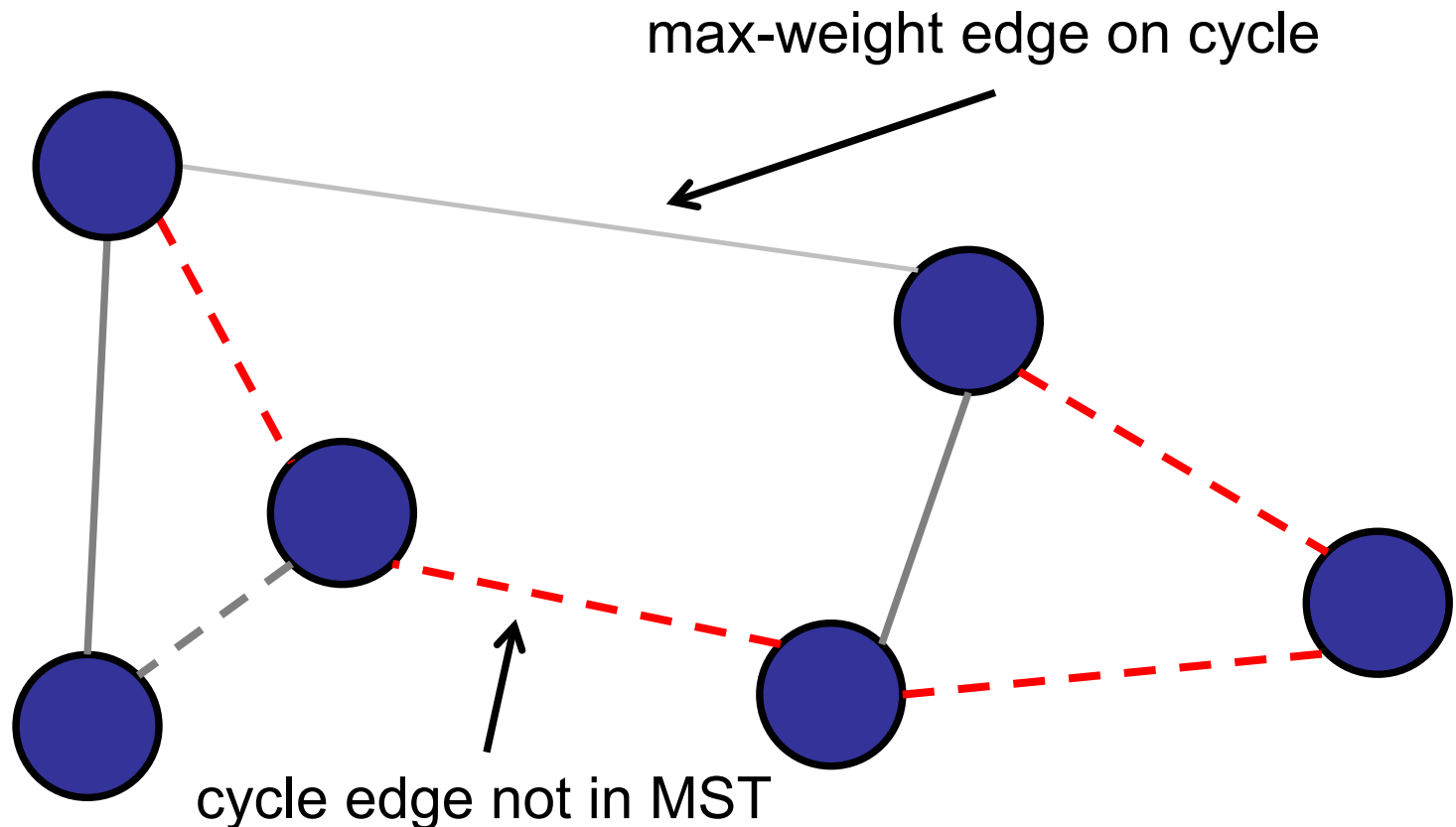


# Properties of MST

---

Proof: Cut-and-paste

Replace heavy edge with lighter edge.  
Less weight! Contradiction...

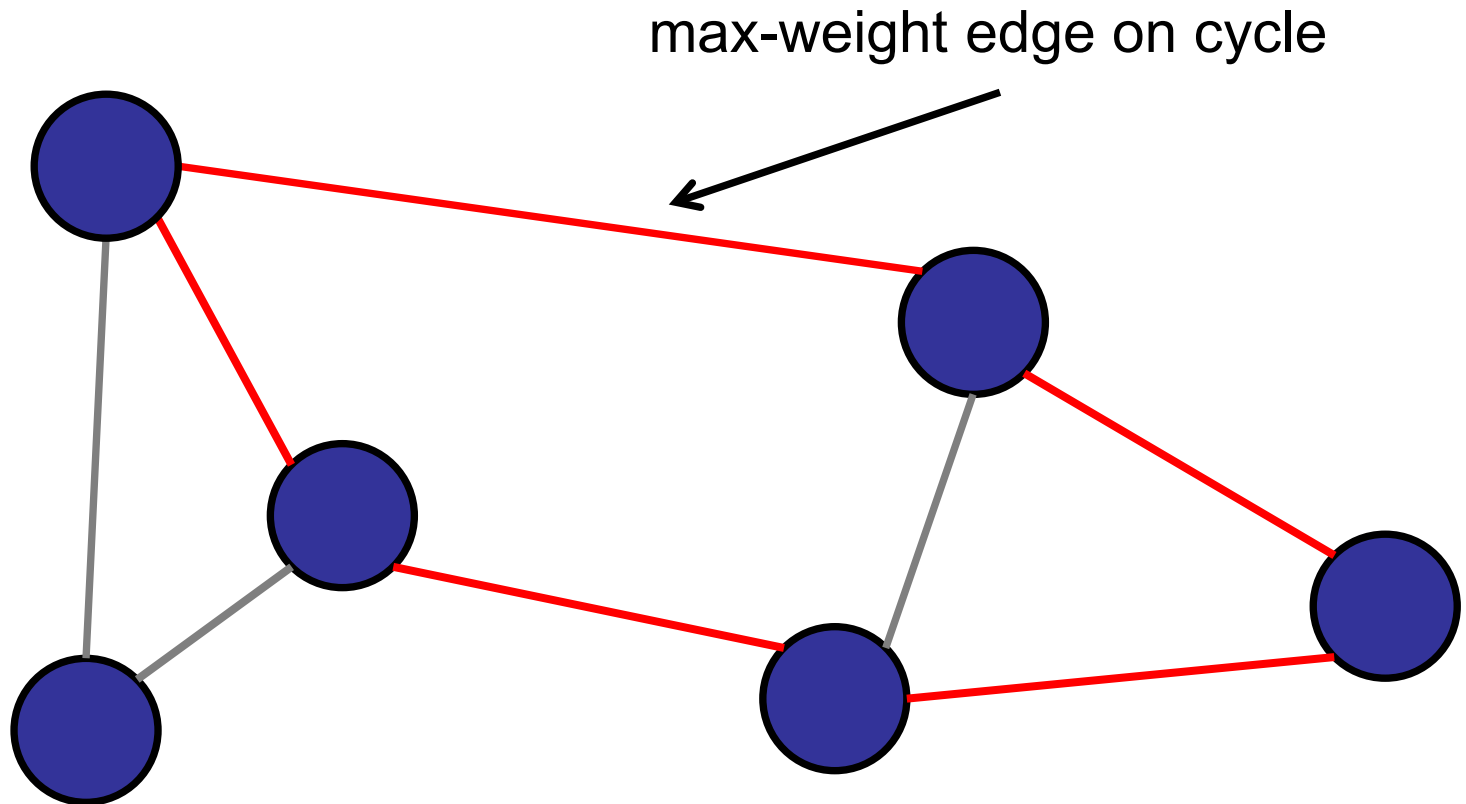


# Properties of MST

---

## Property 3: Cycle property

For every cycle, the maximum weight edge is not in the MST.

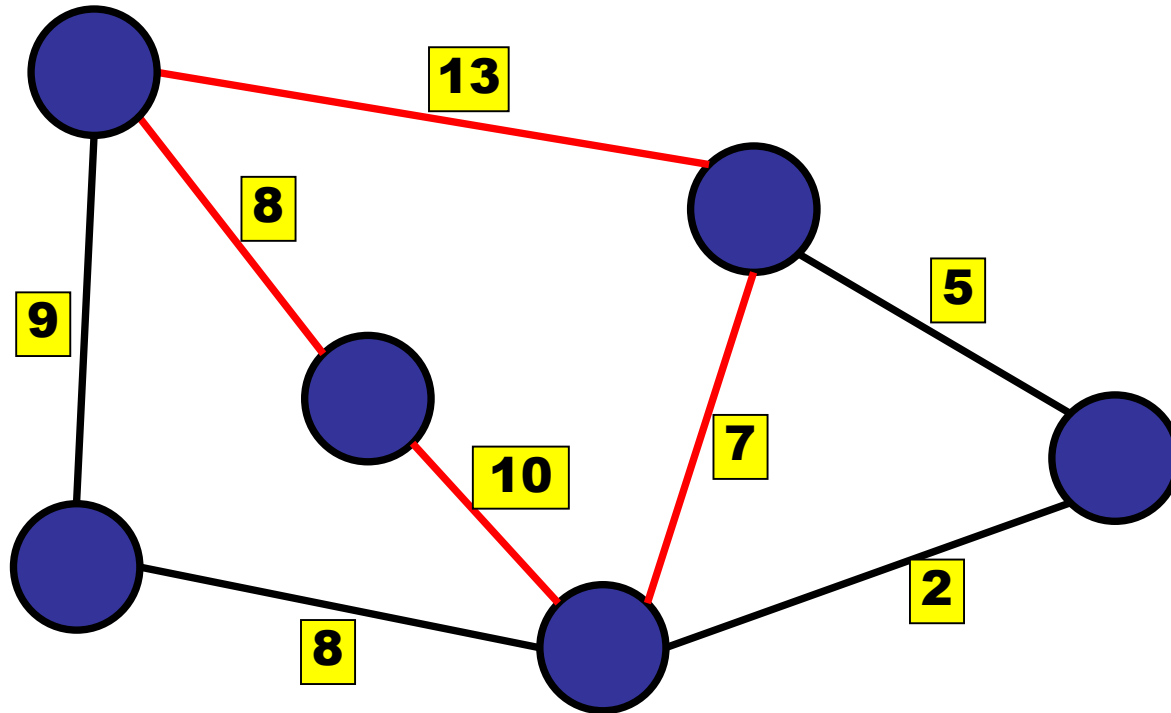


# Properties of MST

---

## Property 3: False Cycle property

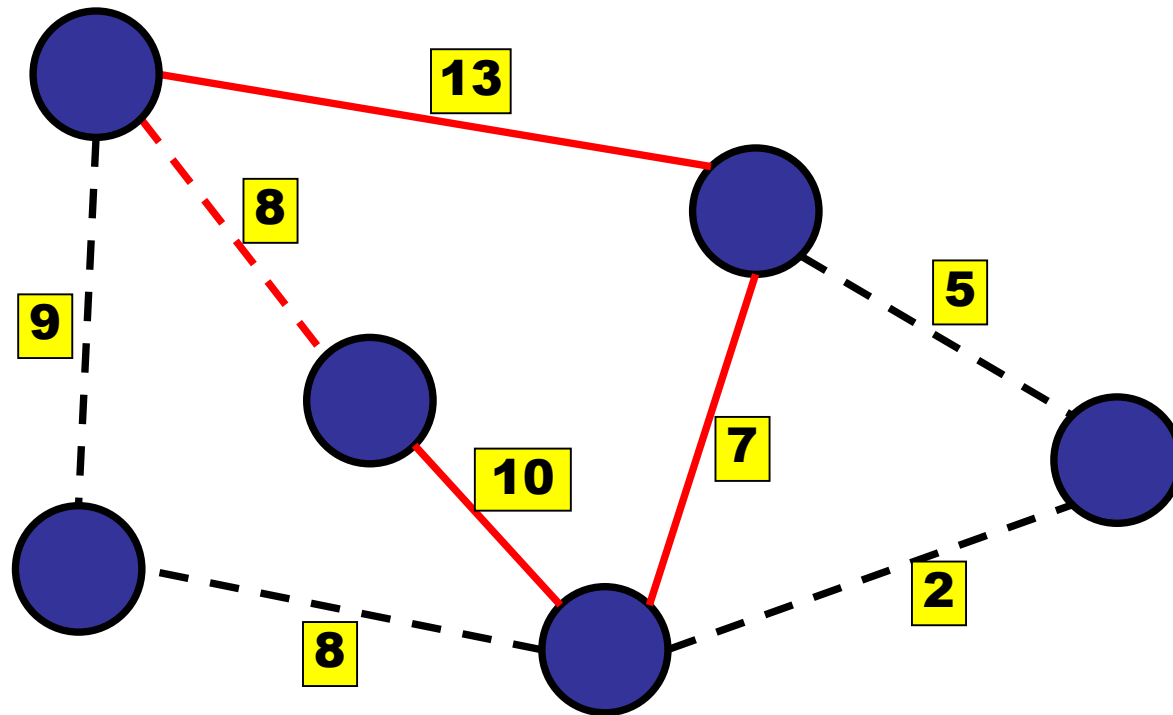
For every cycle, the minimum weight edge may or may not be in the MST.



# Properties of MST

## Property 3: False Cycle property

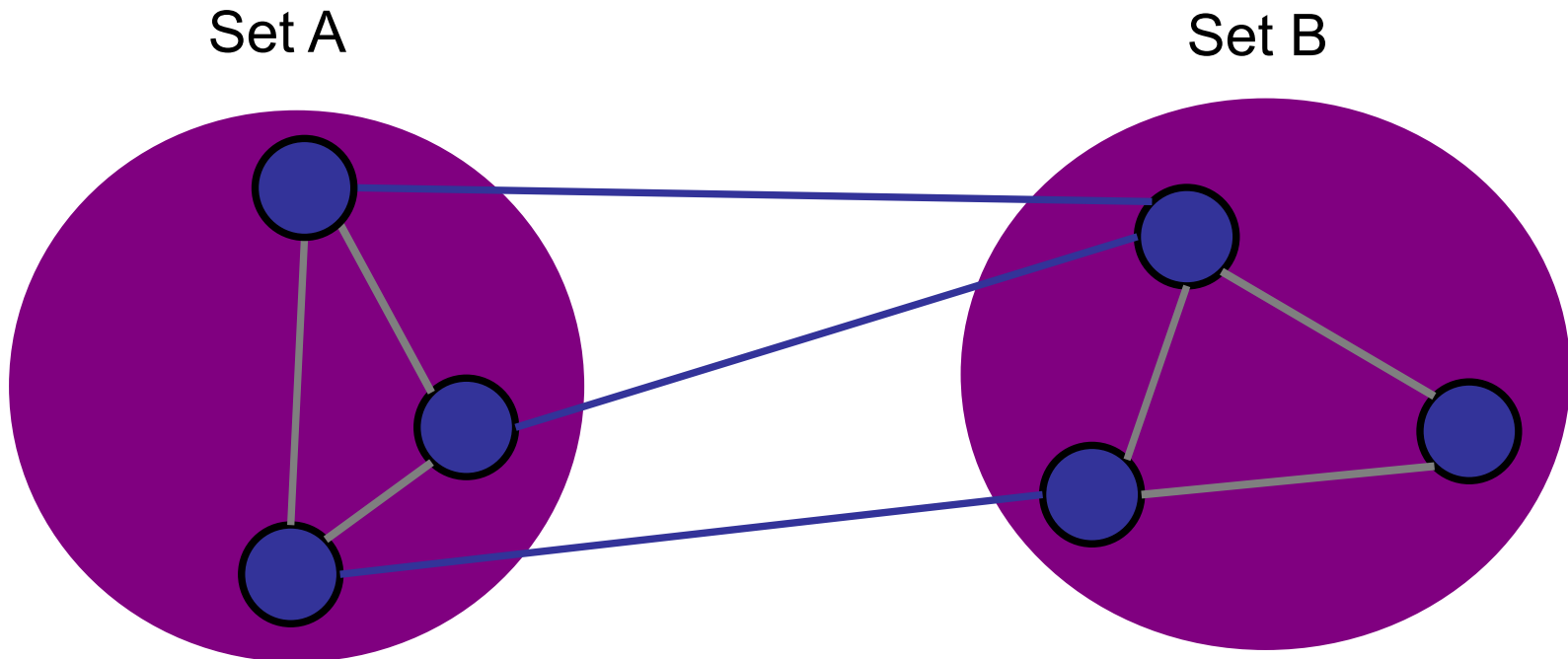
For every cycle, the minimum weight edge may or may not be in the MST



# Properties of MST

---

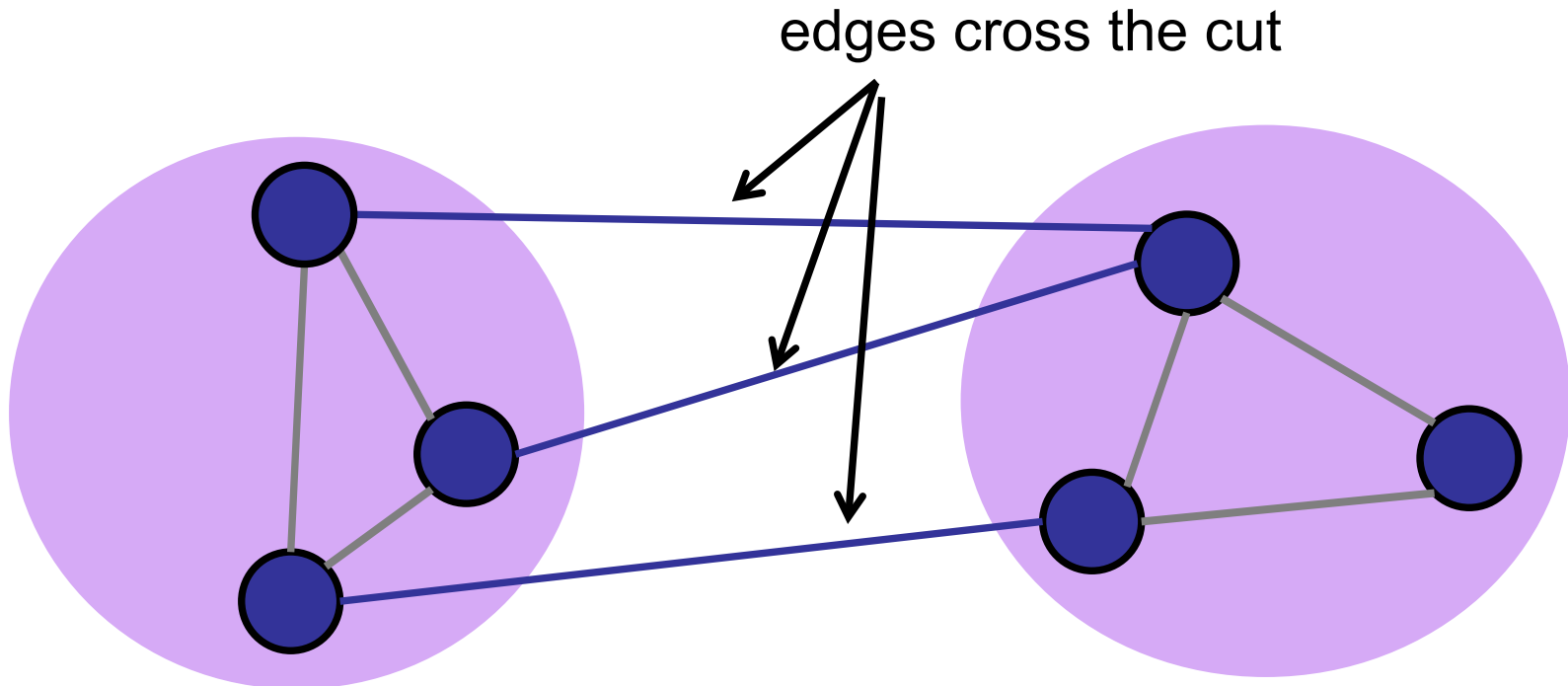
Definition: A *cut* of a graph  $G=(V,E)$  is a partition of the vertices  $V$  into two disjoint subsets.



# Properties of MST

---

Definition: An edge *crosses a cut* if it has one vertex in each of the two sets.

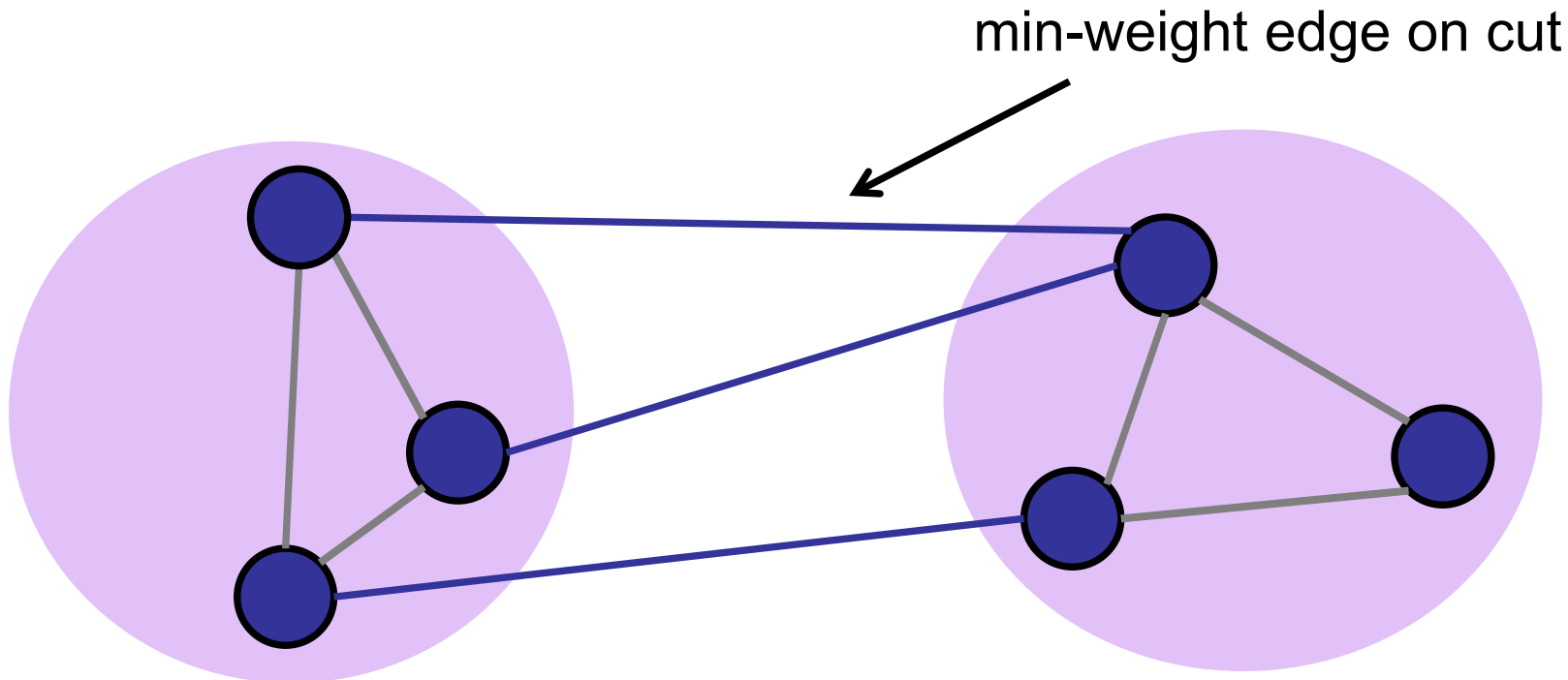


# Properties of MST

---

## Property 4: Cut property

For every partition of the nodes, the minimum weight edge across the cut is in the MST.

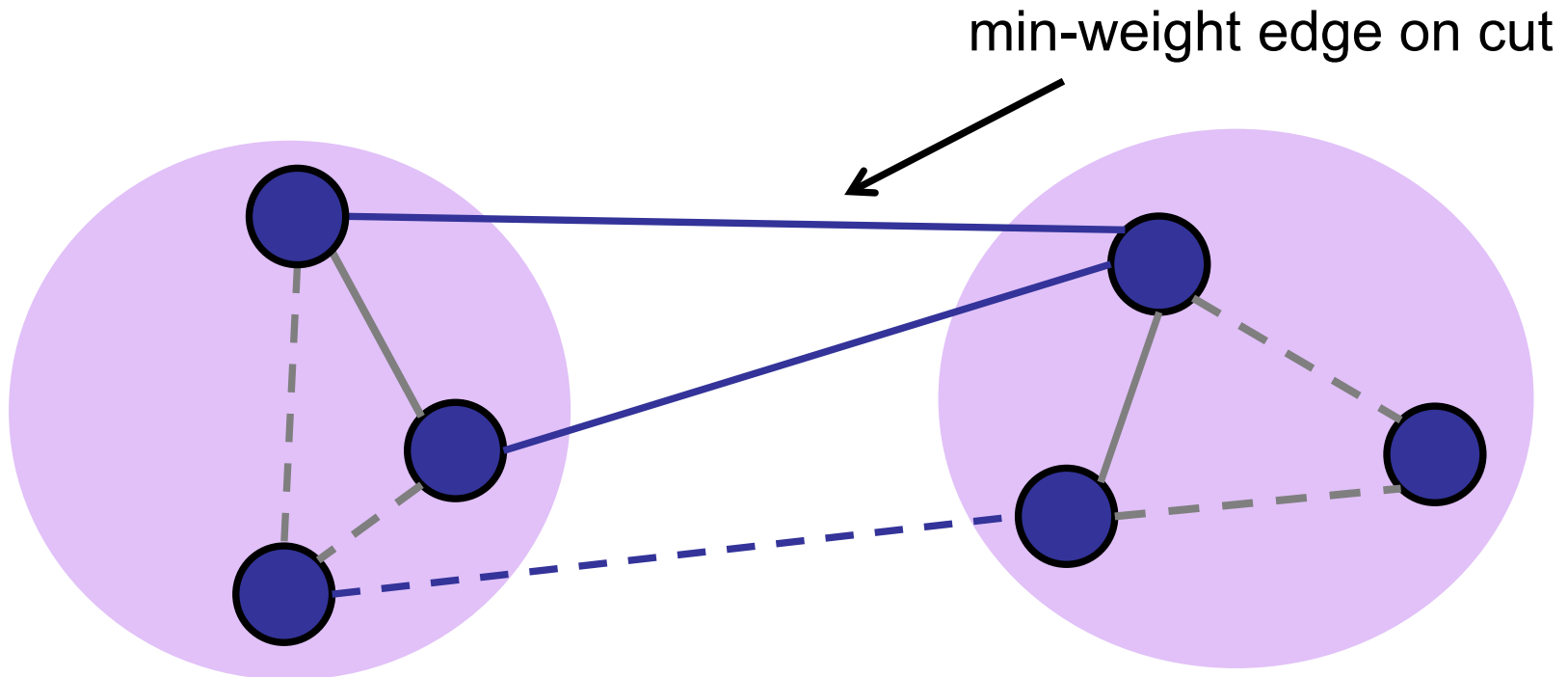


# Properties of MST

---

Proof: Cut-and-paste

Assume not.





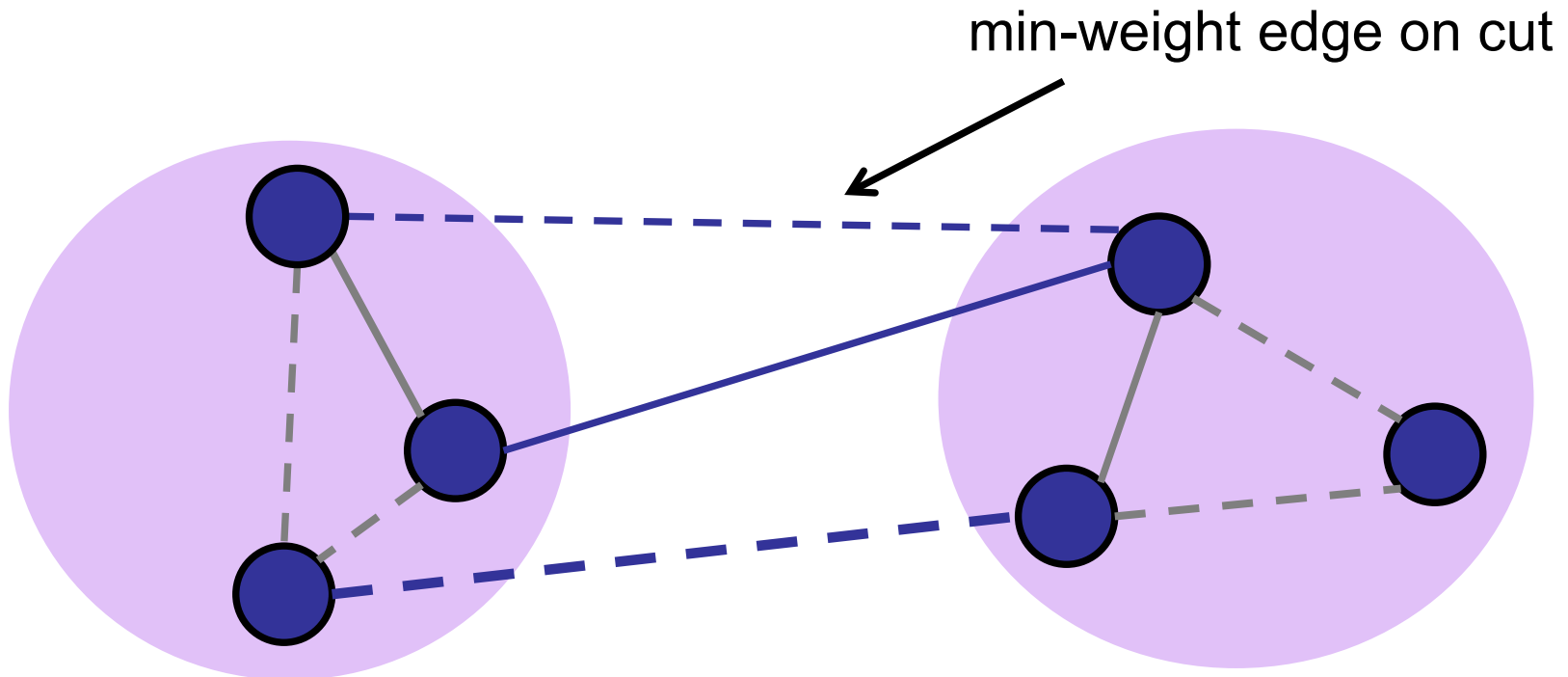
# Properties of MST

---

Proof: Cut-and-paste

Assume not.

Add min-weight edge on cut.



# Properties of MST

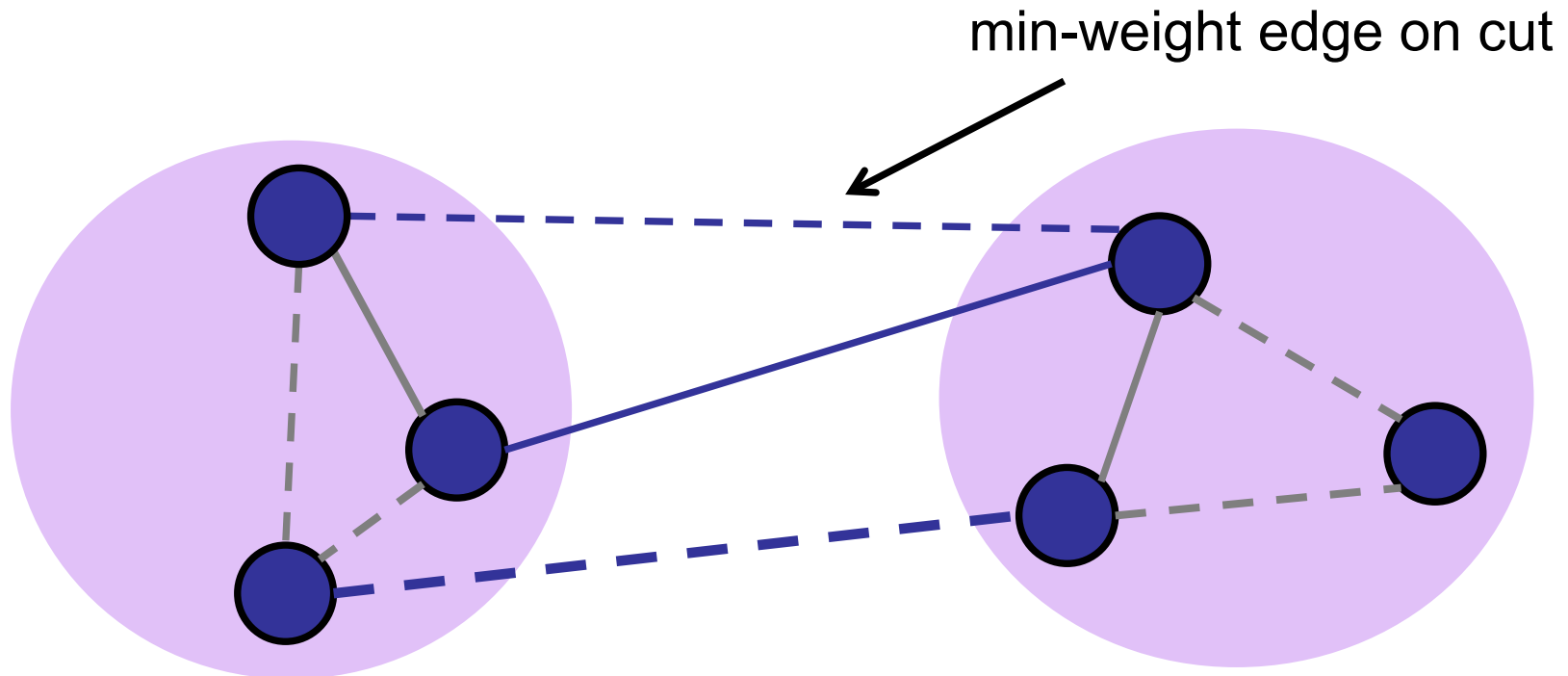
---

Proof: Cut-and-paste

Assume not.

Add min-weight edge on cut.

Oops, creates a cycle!



# Properties of MST

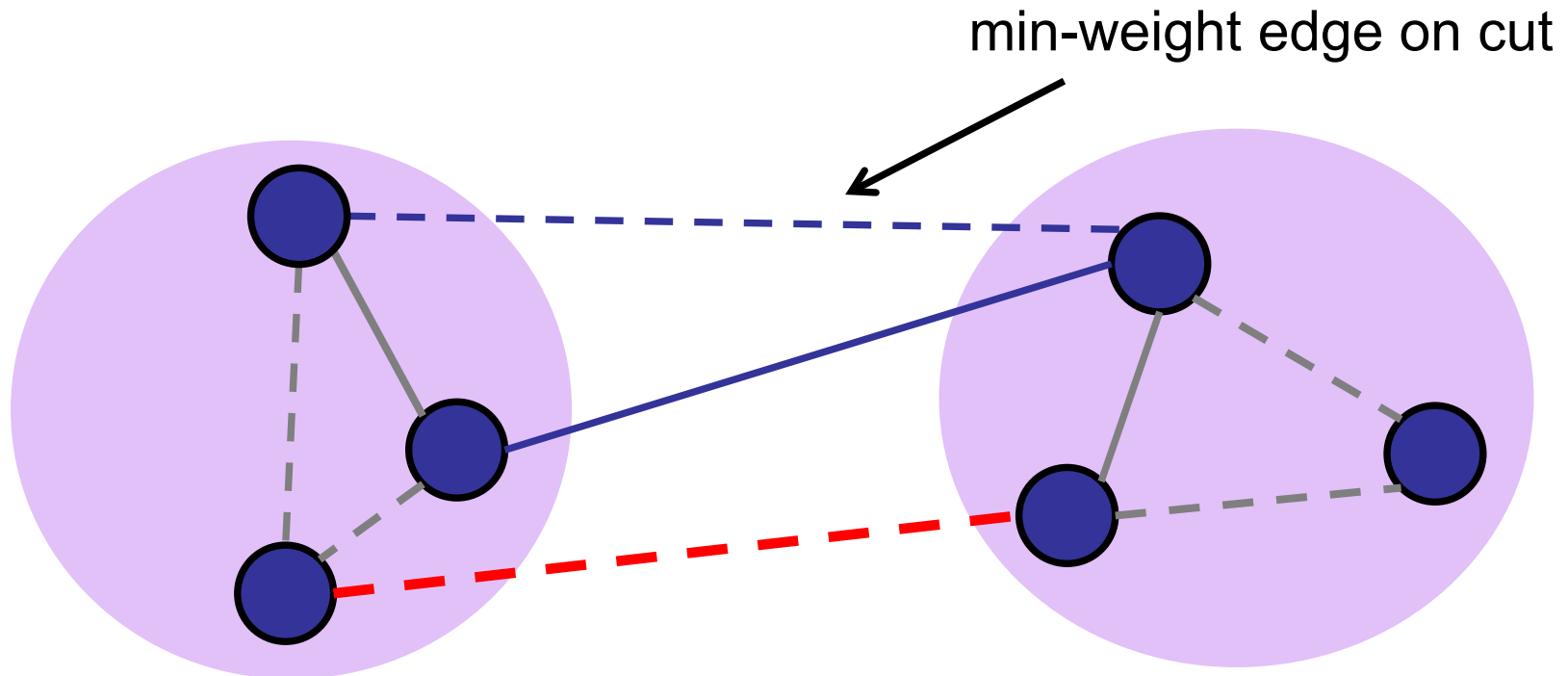
---

Proof: Cut-and-paste

Assume not.

Add min-weight edge on cut.

Remove heaviest edge on cycle.



# Properties of MST

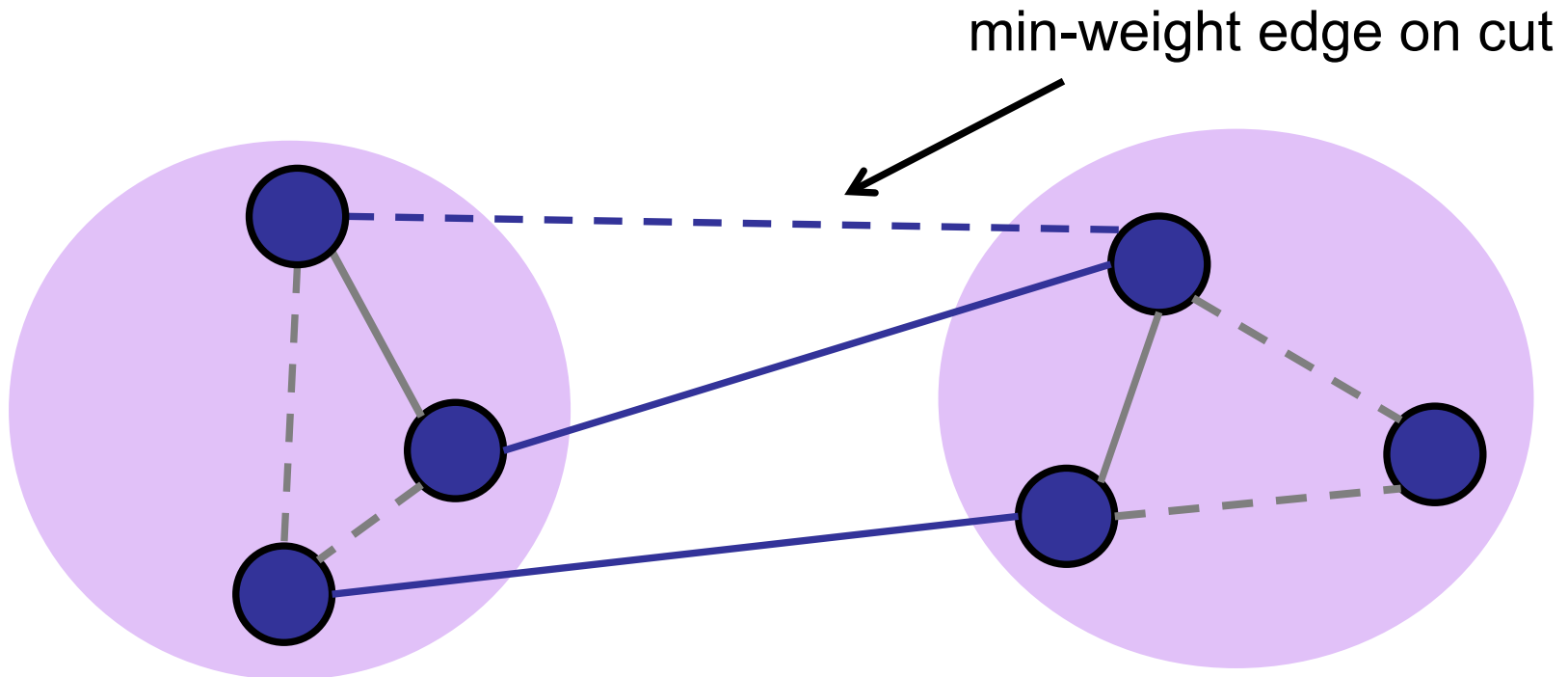
---

Proof: Cut-and-paste

Assume not.

Add min-weight edge on cut.

Remove heaviest edge on cycle.

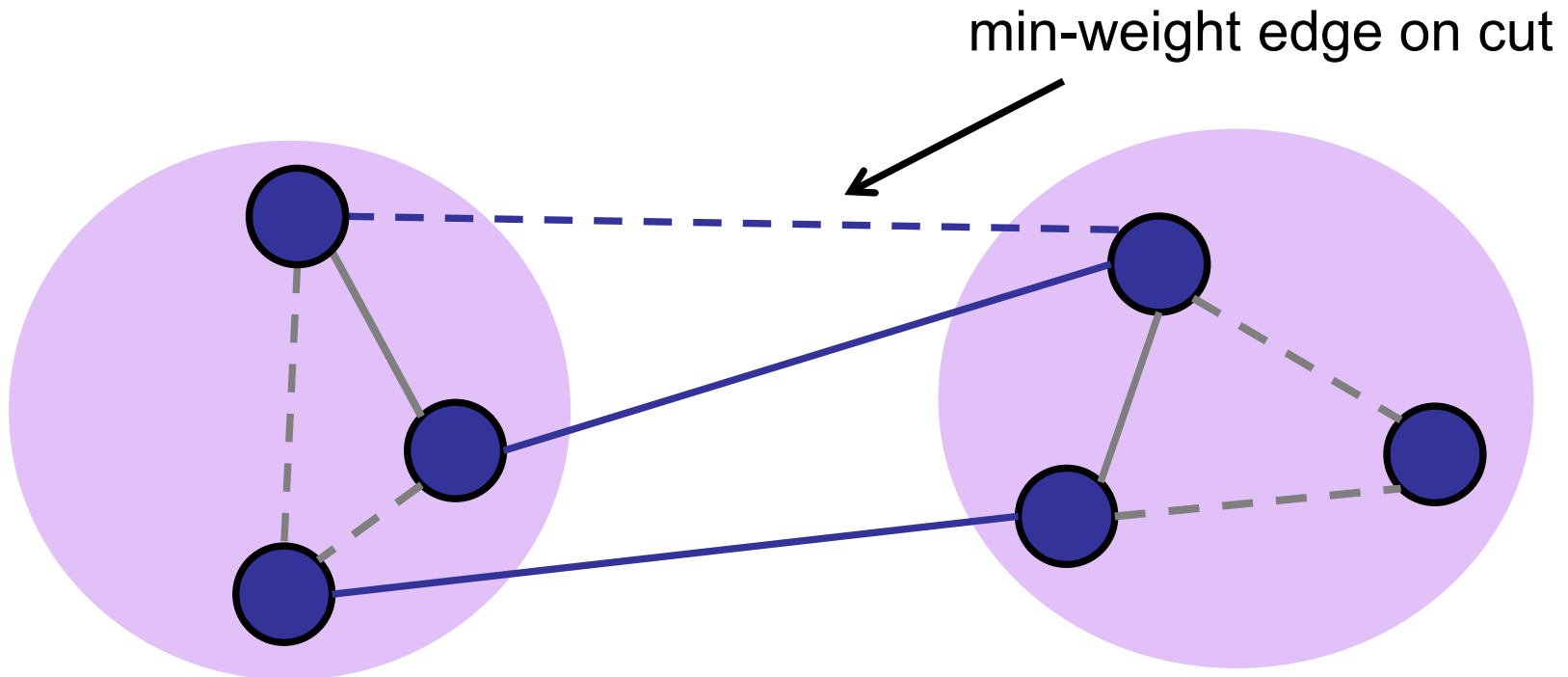


# Properties of MST

---

Proof: Cut-and-paste

Result: a new spanning tree.

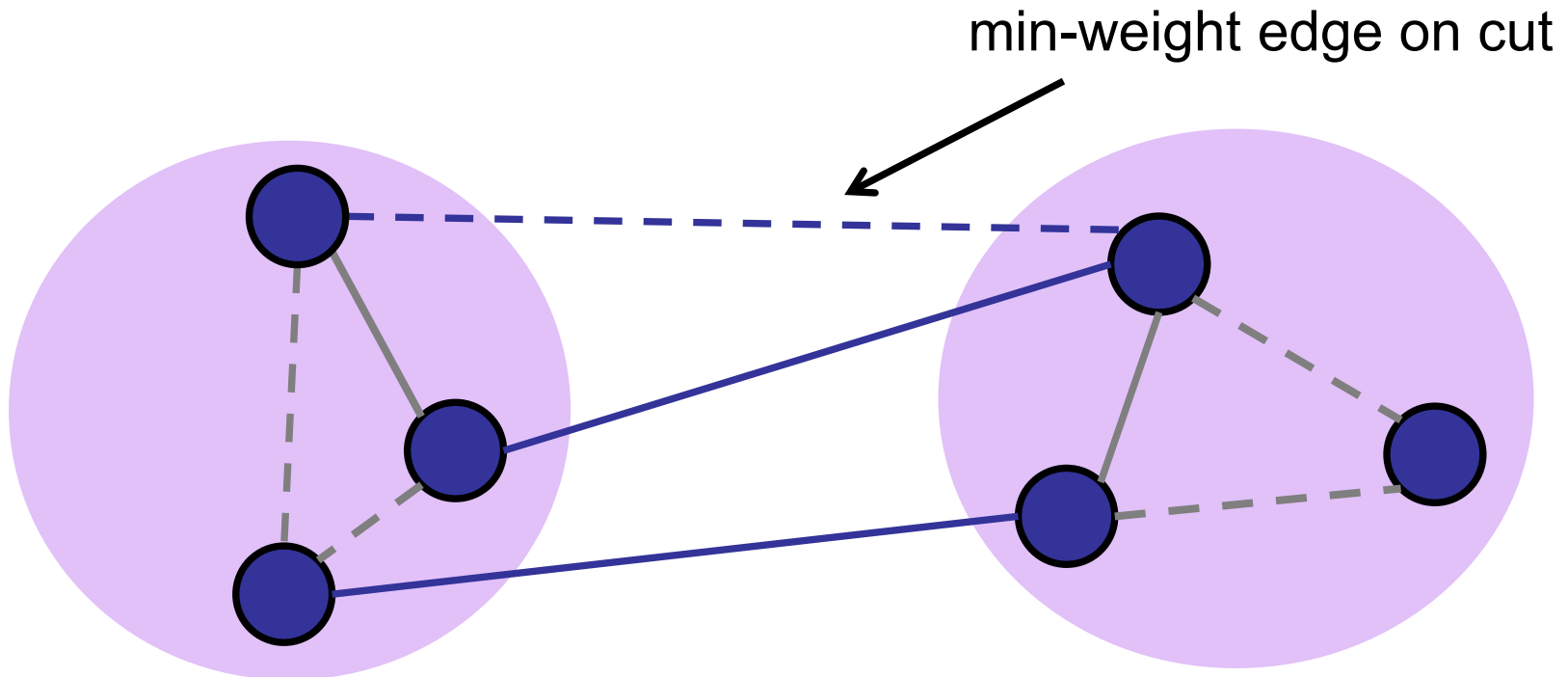


# Properties of MST

---

## Proof: Cut-and-paste

Less weight: replaced heavier edge with lighter edge.

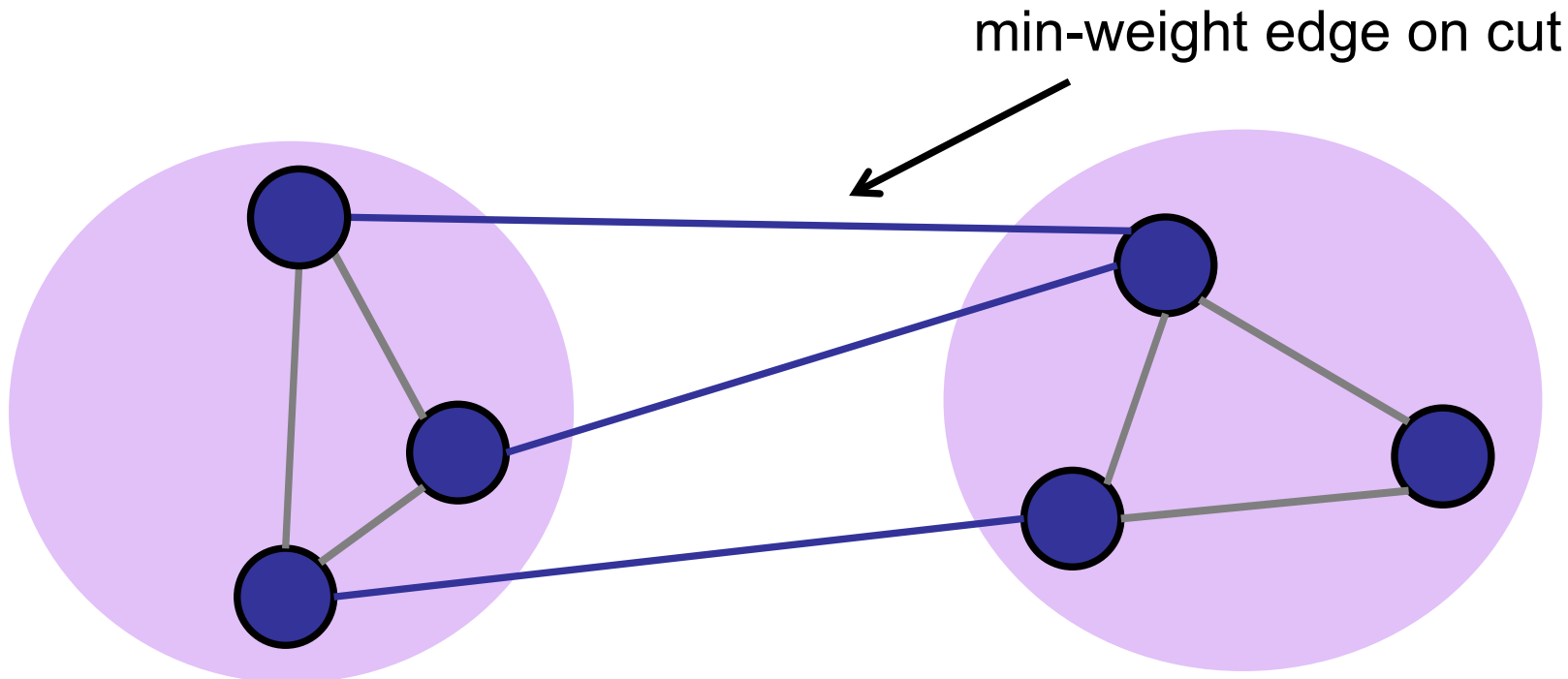


# Properties of MST

---

## Property 4: Cut property

For every partition of the nodes, the minimum weight edge across the cut *is* in the MST.

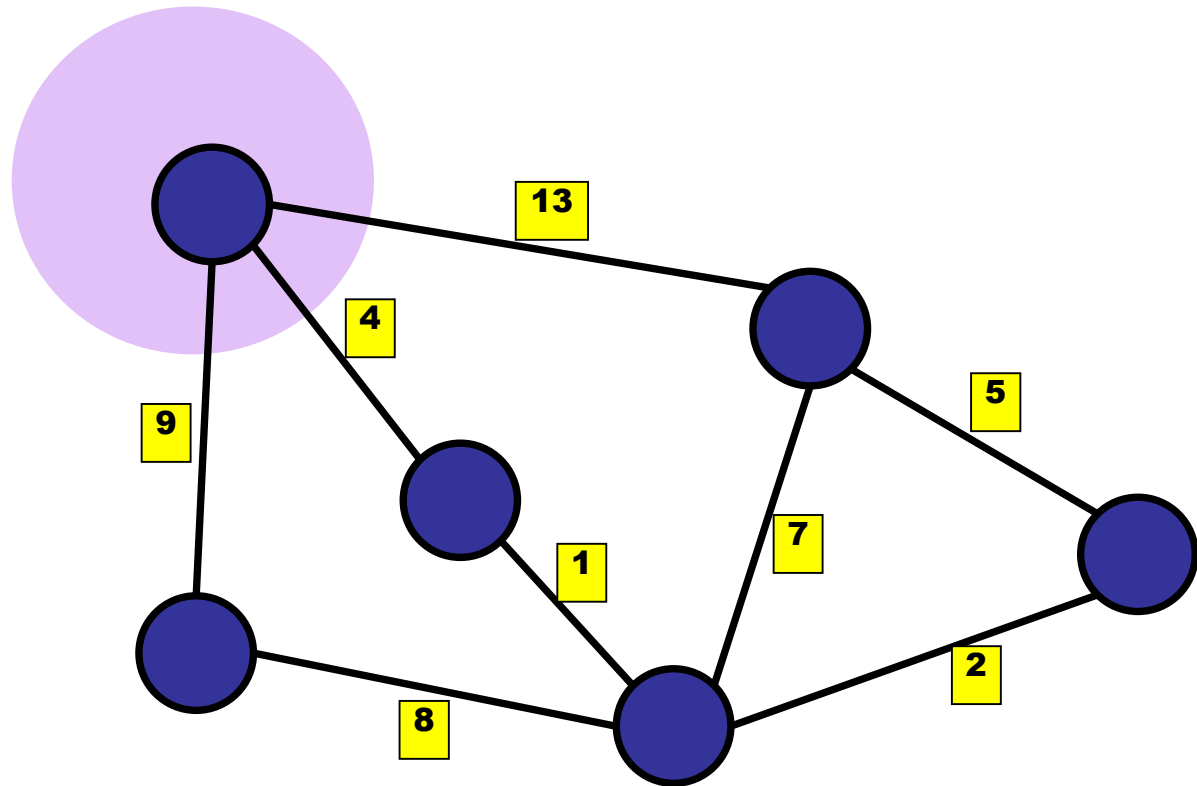


# Properties of MST

---

## Property 4: Cut property

For every partition of the nodes, the minimum weight edge across the cut *is* in the MST.



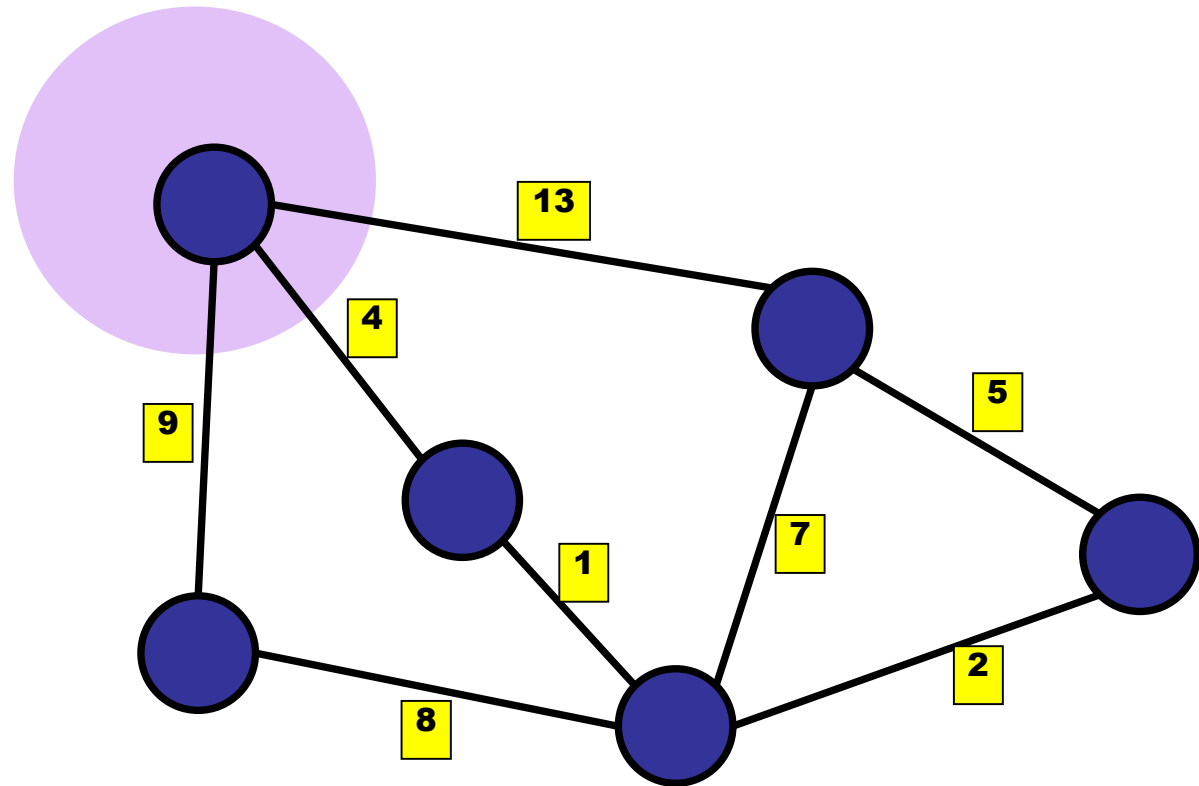


# Properties of MST

---

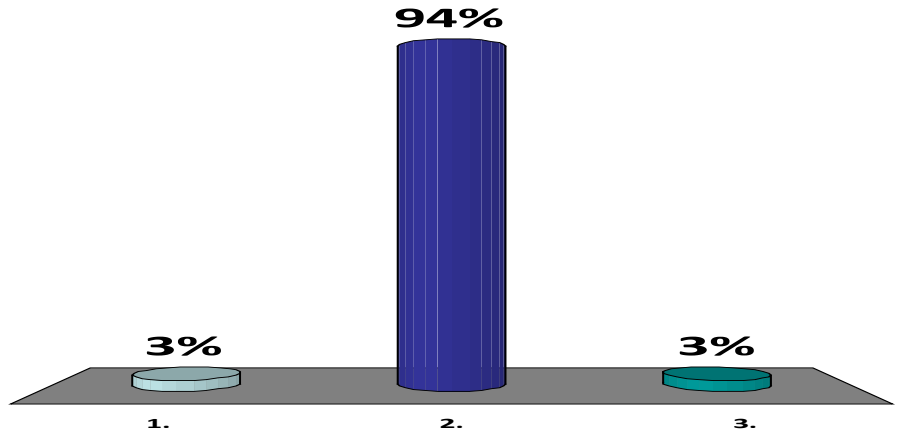
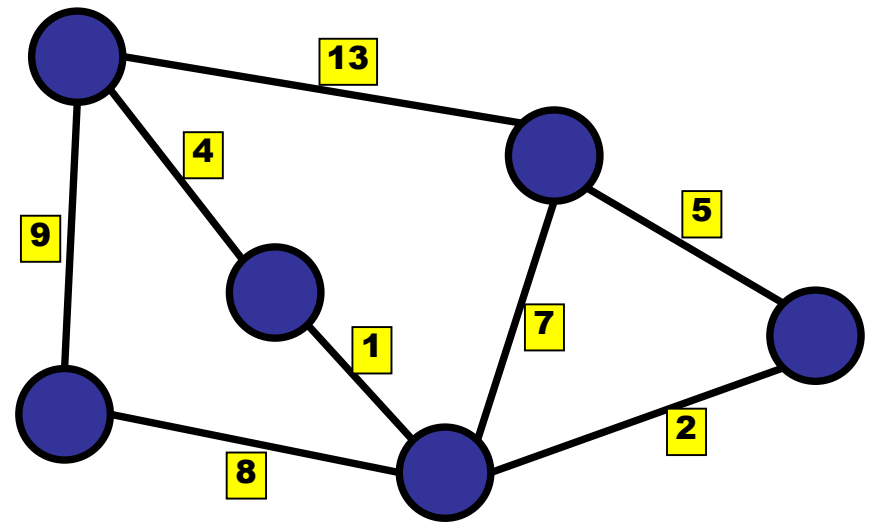
## Property 4**b**: Cut property

For every vertex, the minimum outgoing edge is always part of the MST



For every vertex, the maximum outgoing edge is never part of the MST.

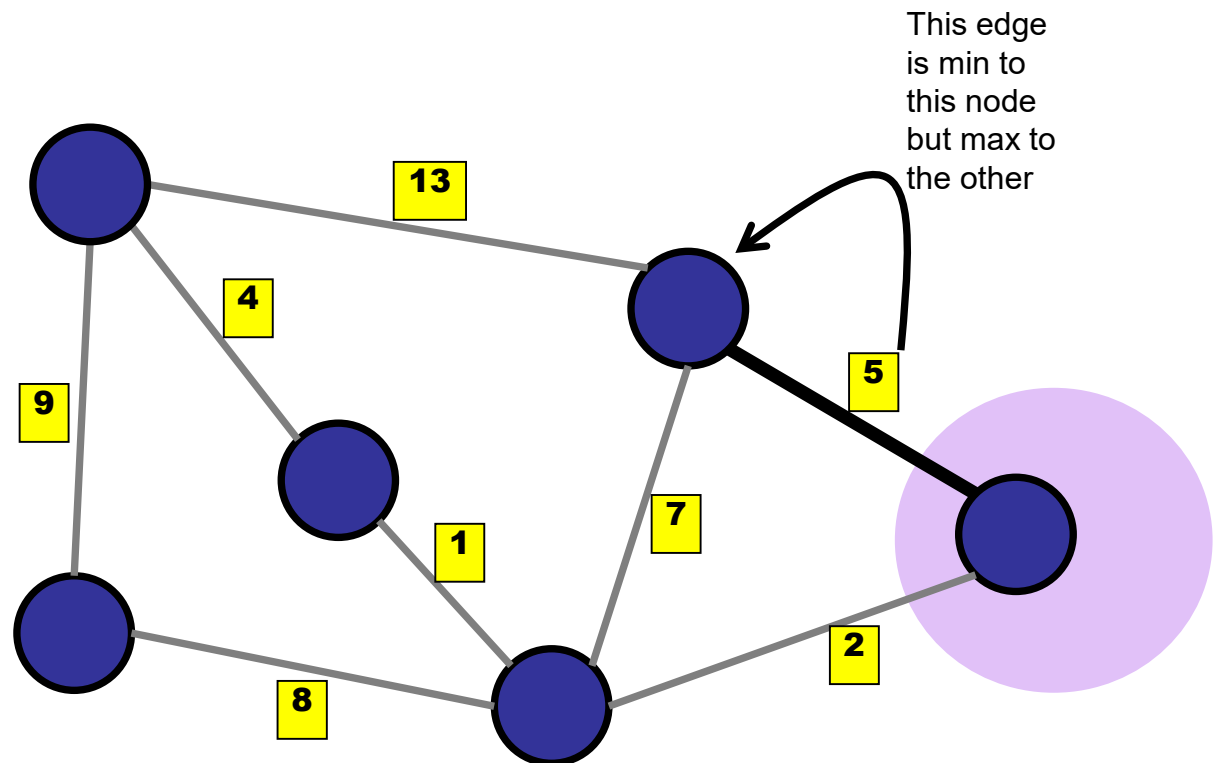
1. True
- ✓ 2. False
3. I don't know.



# Properties of MST

## Property 4: Cut property

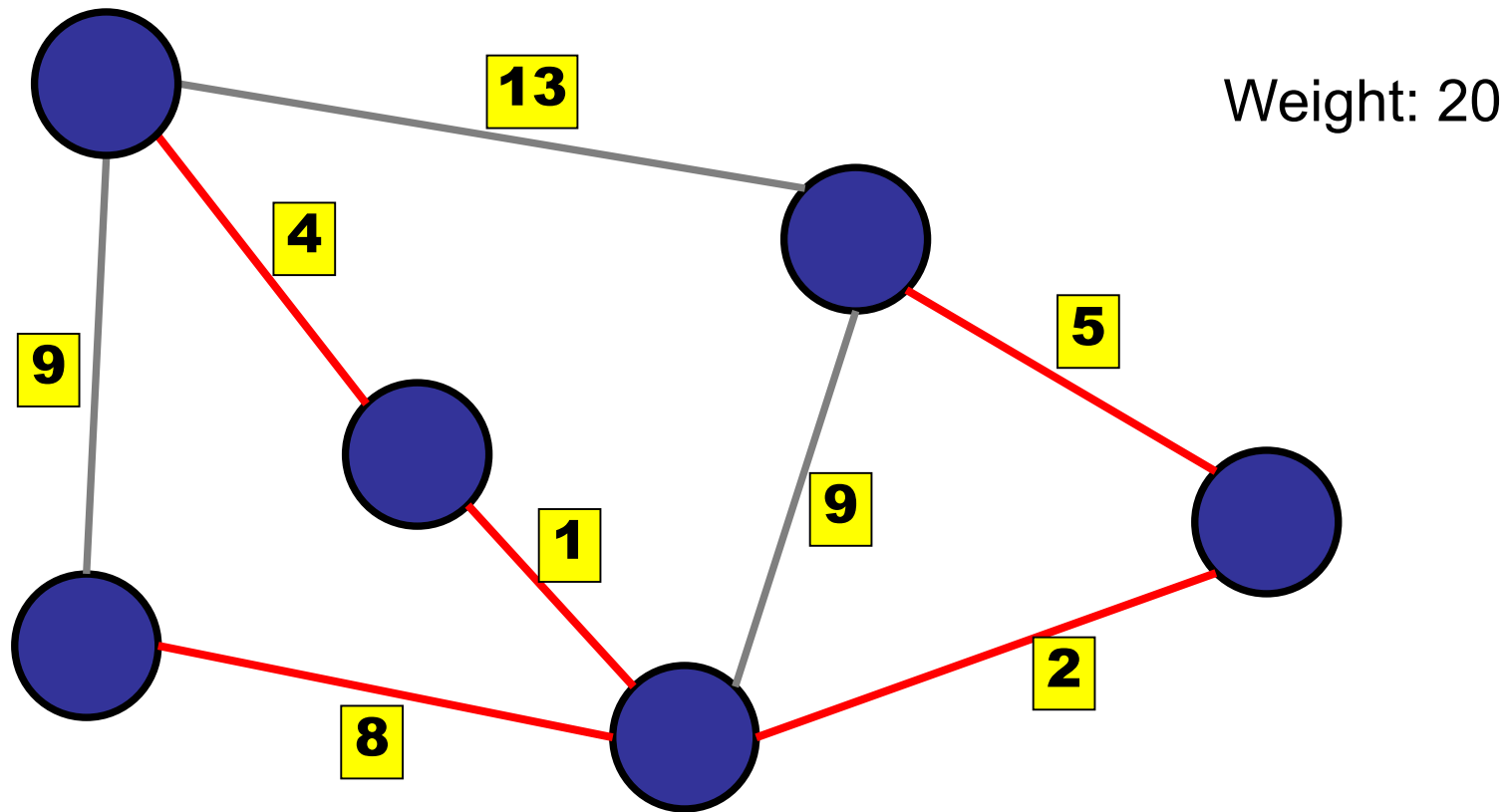
For every partition of the nodes, the minimum weight edge across the cut *is* in the MST.



# Minimum Spanning Tree

---

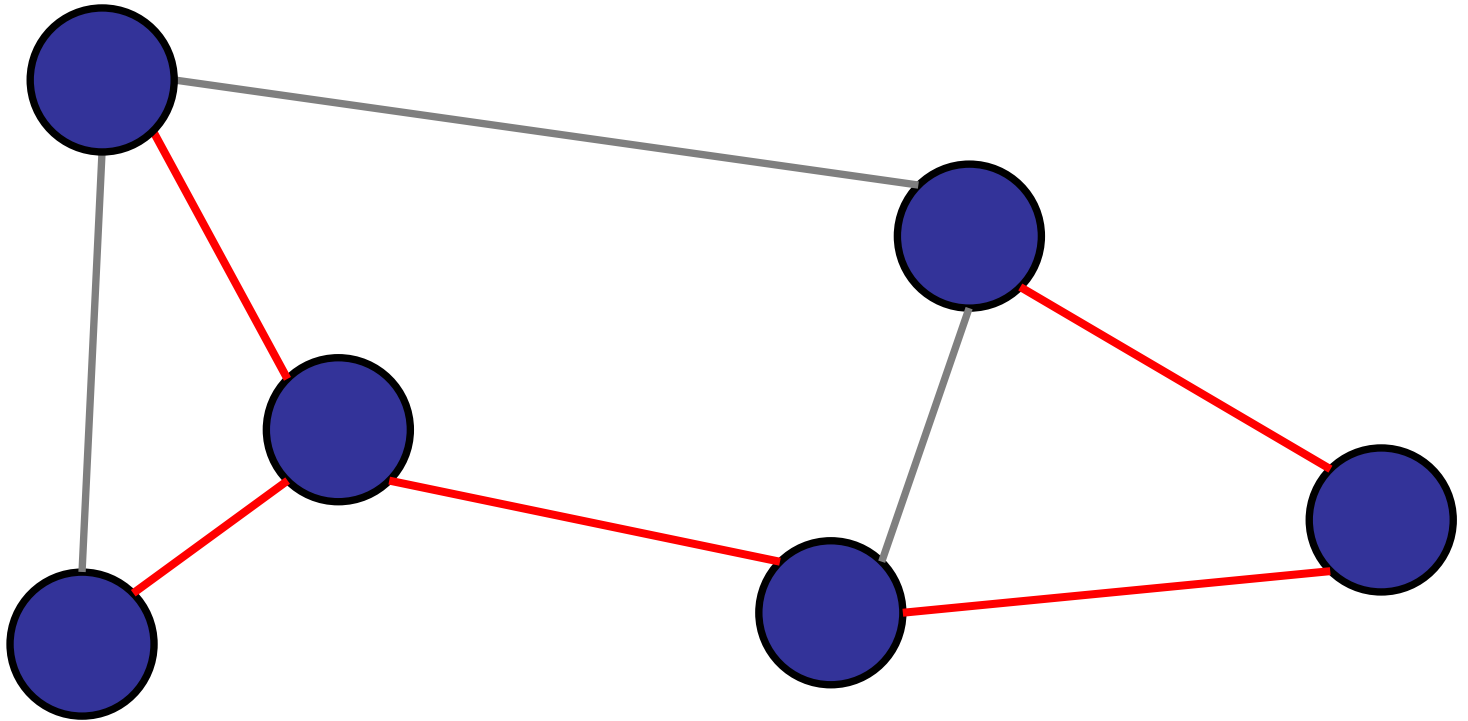
Definition: a spanning tree with minimum weight



# Properties of MST

---

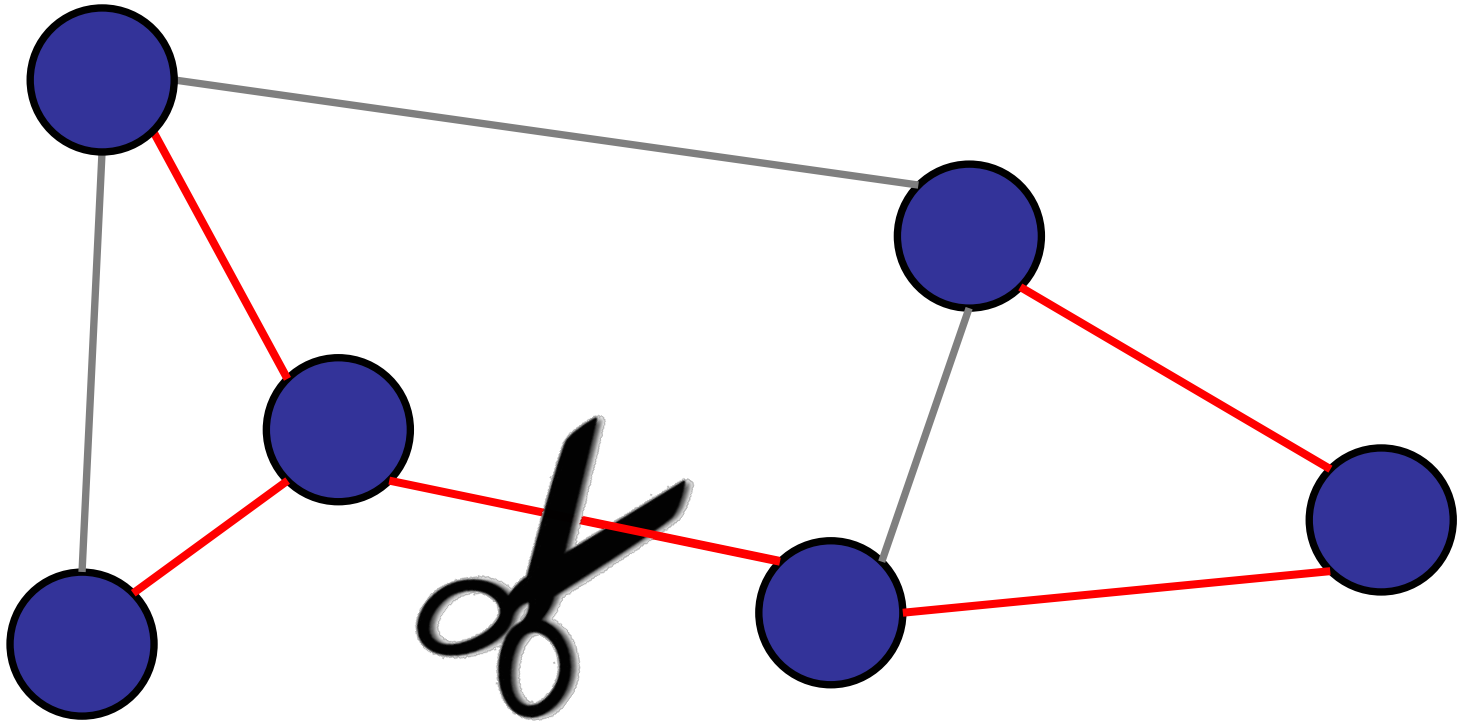
Property 1: No cycles



# Properties of MST

---

Property 2: If you cut an MST, the two pieces are both MSTs.

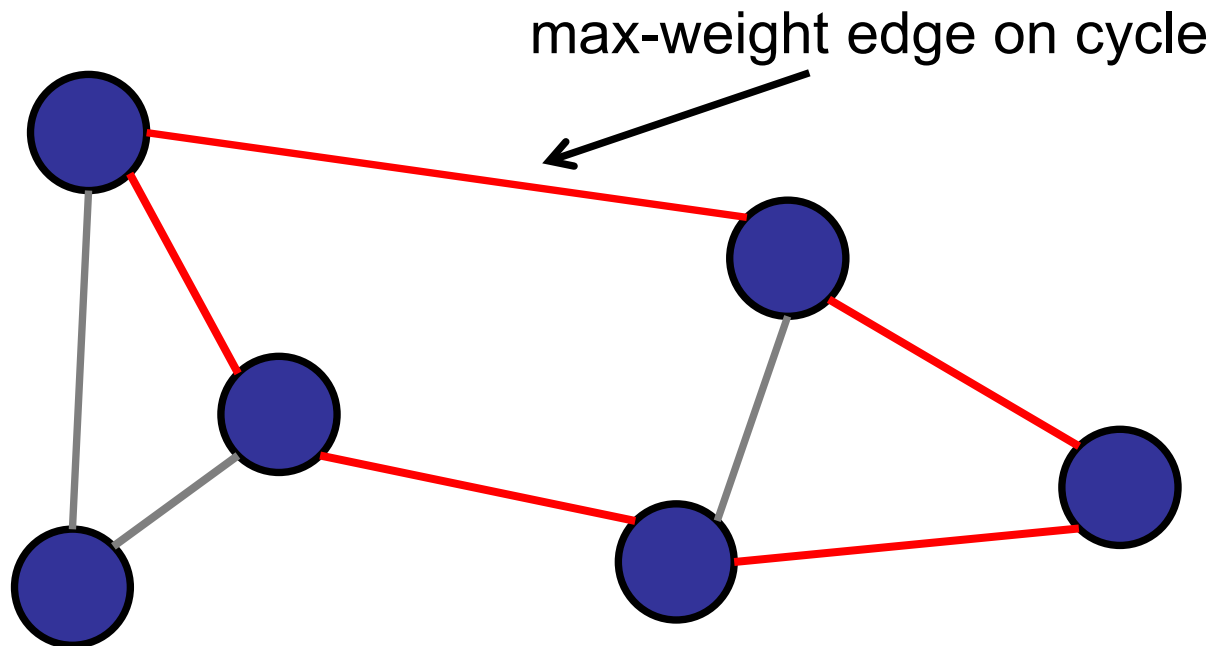


# Properties of MST

---

## Property 3: Cycle property

For every cycle, the maximum weight edge is *not* in the MST.

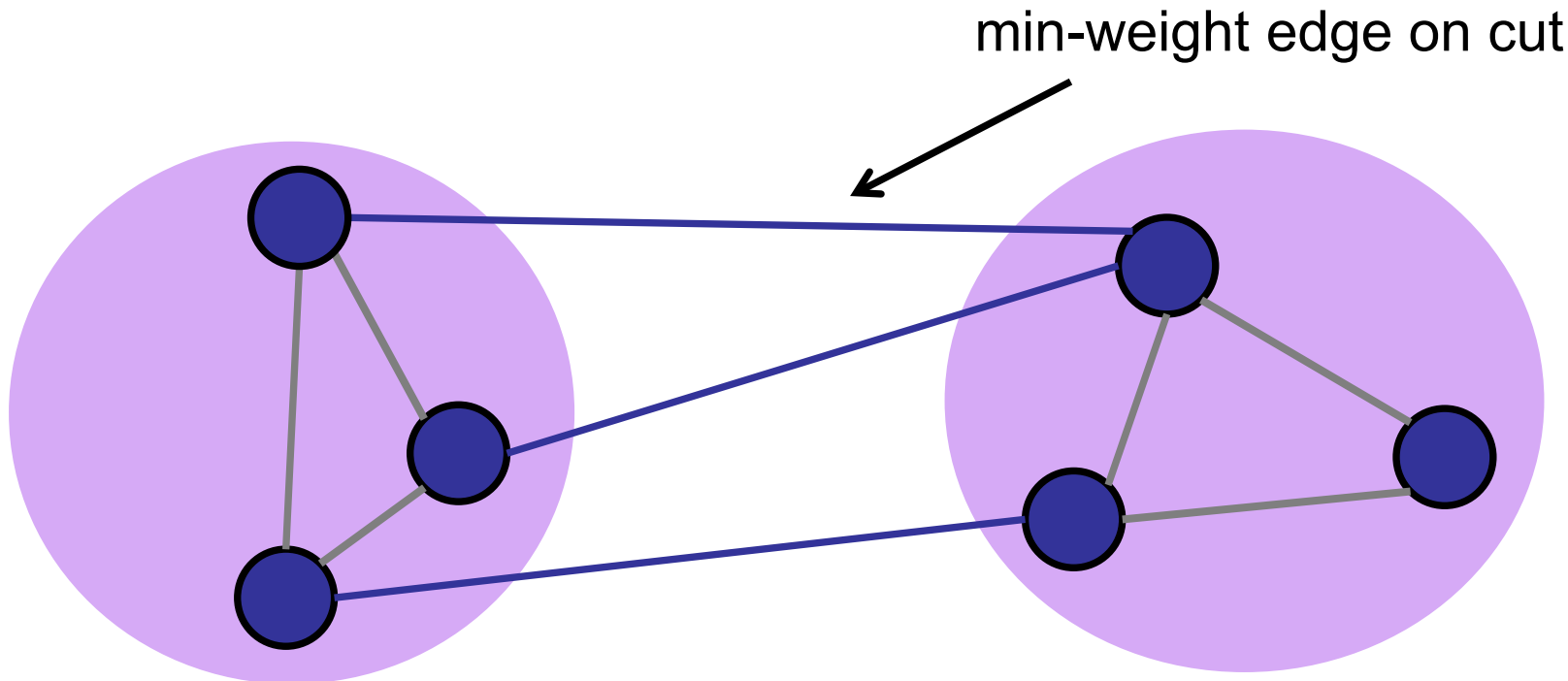


# Properties of MST

---

## Property 4: Cut property

For every cut  $D$ , the minimum weight edge that crosses the cut *is* in the MST.





# Property of MST

---

- No cycles
- If you cut an MST, the two pieces are both MSTs.
- Cycle property
  - For every cycle, the maximum weight edge is not in the MST.
- Cut property
  - For every cut  $D$ , the minimum weight edge that crosses the cut is in the MST.

# Roadmap

---

## Minimum Spanning Trees

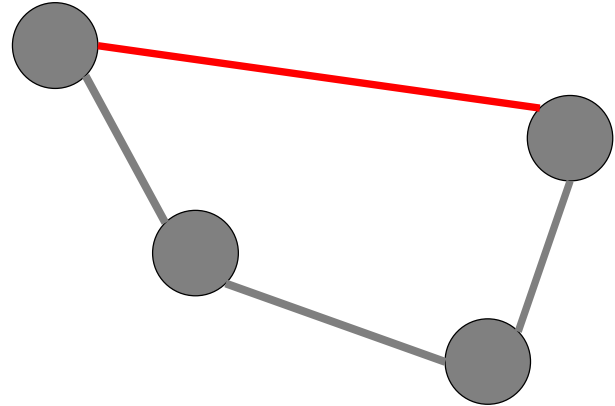
- The MST Problem
- Basic Properties of an MST
- **Generic MST Algorithm**
- Prim's Algorithm
- Kruskal's Algorithm
- Boruvka's Algorithm
- Variations

# Generic MST Algorithm

---

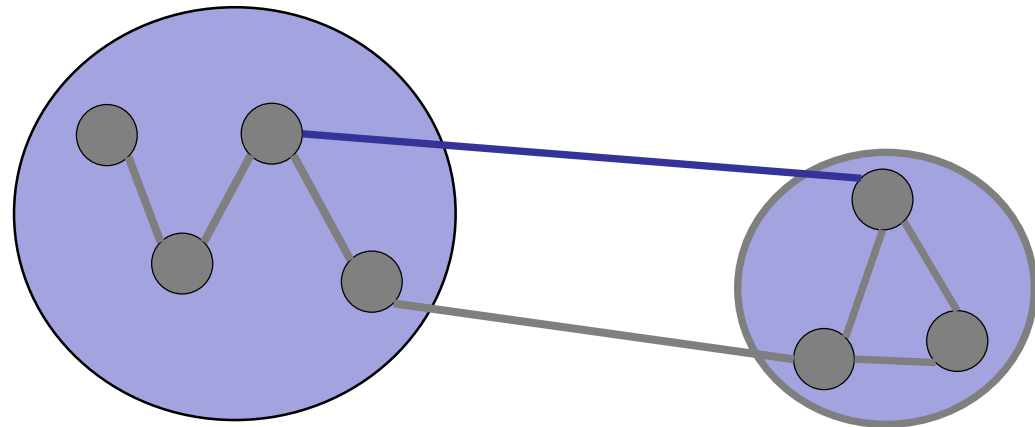
## **Red** rule: (Property 3)

If  $C$  is a cycle with no red arcs, then color the max-weight edge in  $C$  red.



## **Blue** rule: (Property 4)

If  $D$  is a cut with no blue arcs, then color the min-weight edge in  $D$  blue.



# Generic MST Algorithm

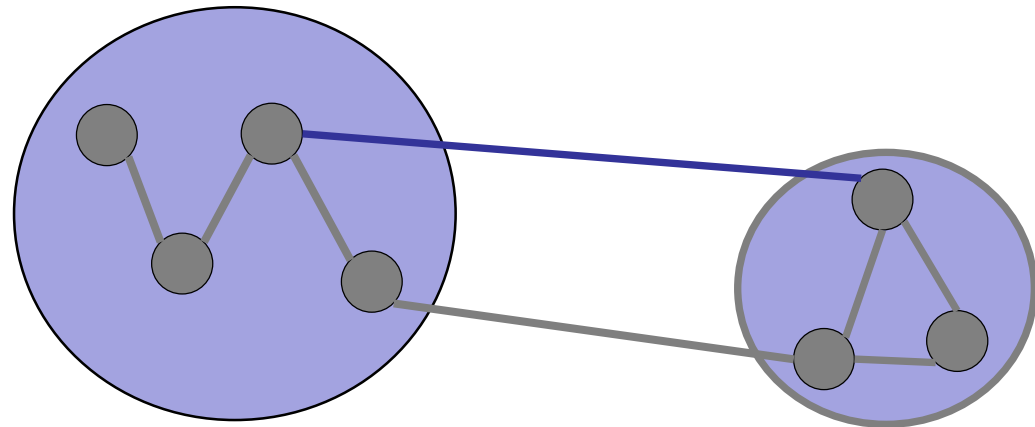
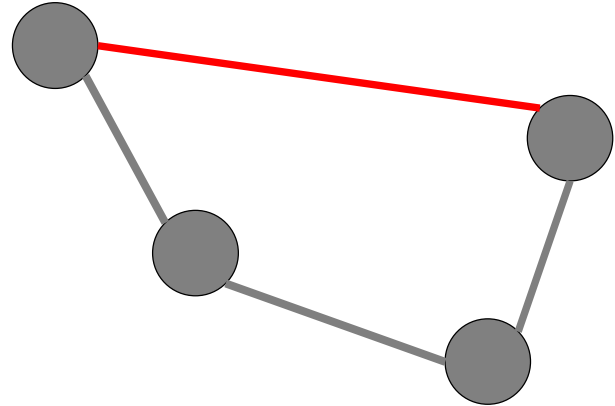
---

## Greedy Algorithm:

Repeat:

**Apply red rule or  
blue rule to an  
arbitrary edge.**

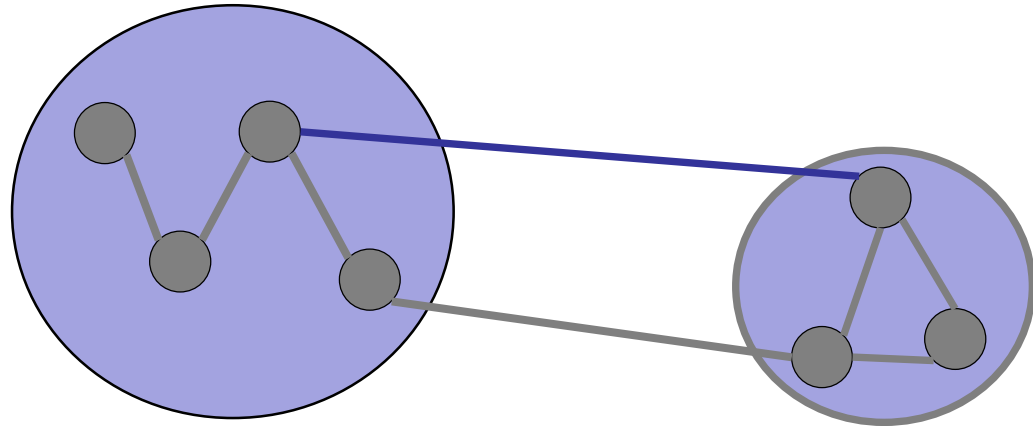
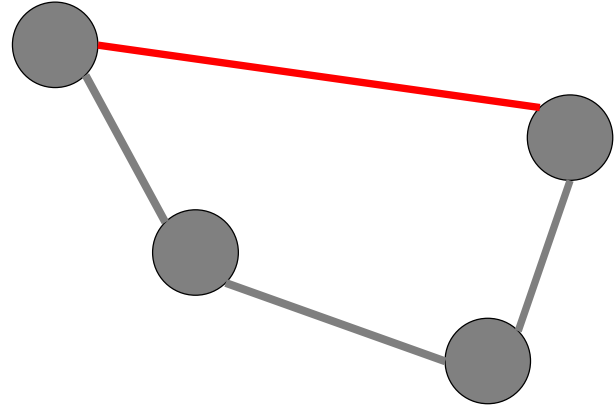
until no more edges  
can be colored.



# Generic MST Algorithm

---

**Claim:** On termination, the blue edges are an MST.



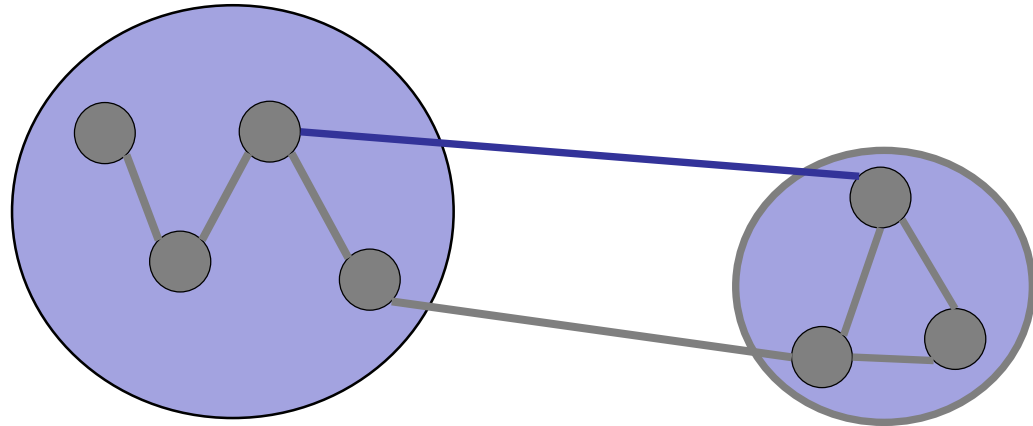
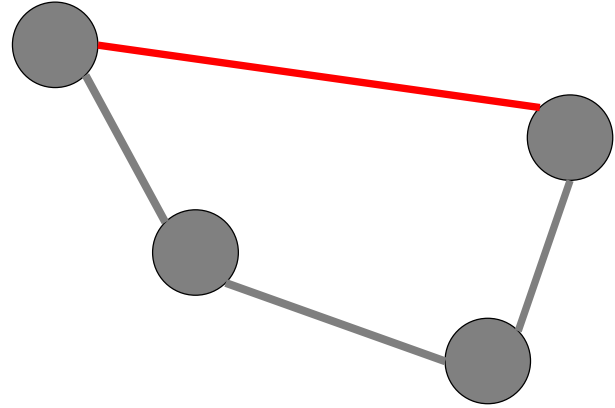
# Generic MST Algorithm

---

**Claim:** On termination, the blue edges are an MST.

On termination:

1. Every cycle has a **red** edge.  
No **blue** cycles.

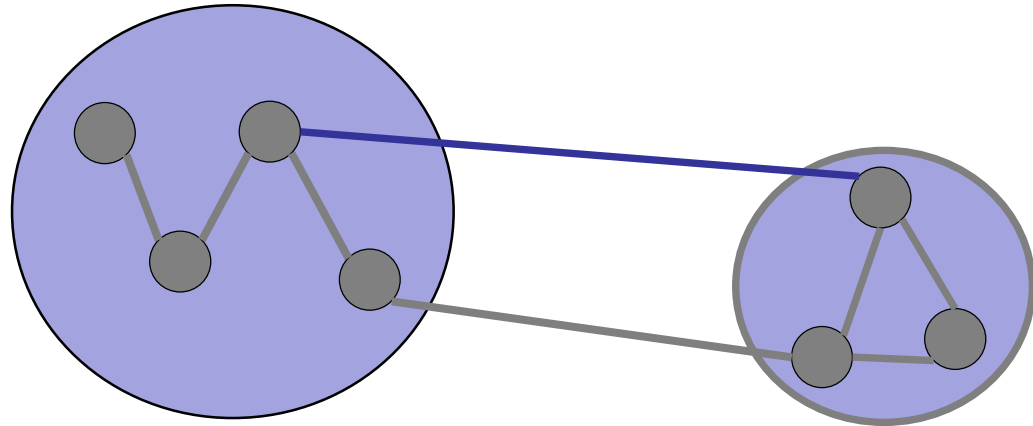
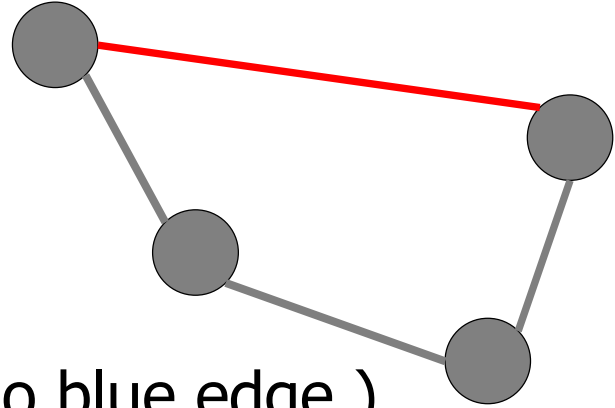


# Generic MST Algorithm

**Claim:** On termination, the blue edges are an MST.

On termination:

1. Every cycle has a **red** edge.  
No **blue** cycles.
2. **Blue** edges form a tree.  
(Otherwise, there is a cut with no blue edge.)

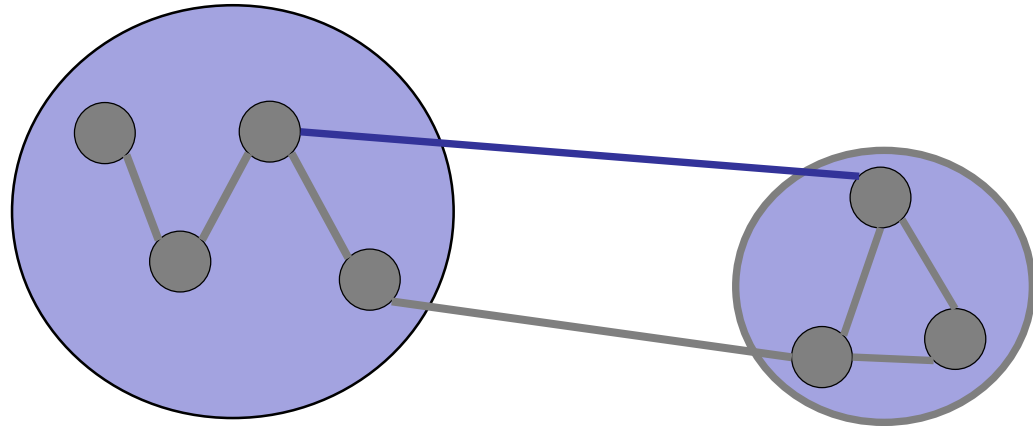
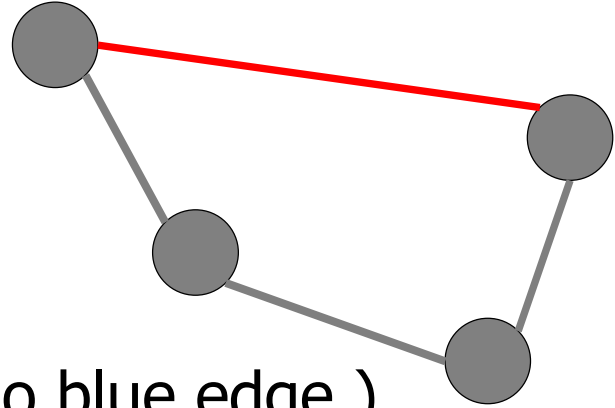


# Generic MST Algorithm

**Claim:** On termination, the blue edges are an MST.

On termination:

1. Every cycle has a **red** edge.  
No **blue** cycles.
2. **Blue** edges form a tree.  
(Otherwise, there is a cut with no blue edge.)
3. Every edge is colored.



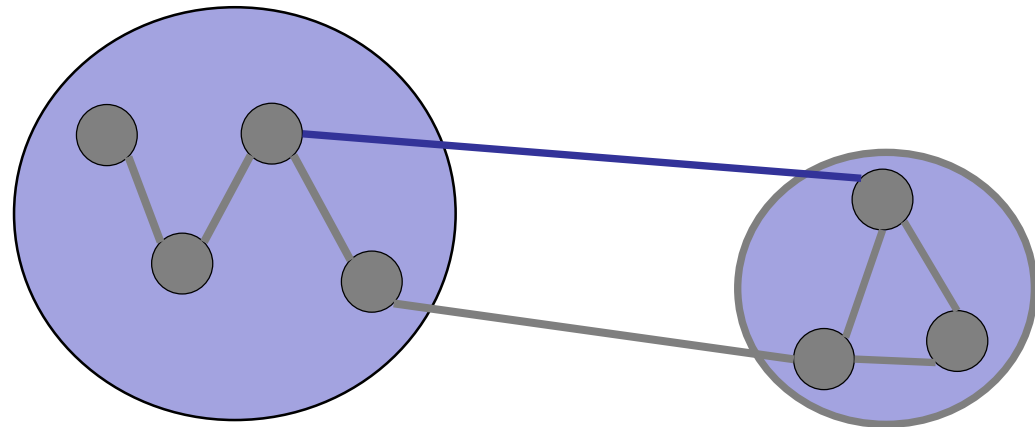
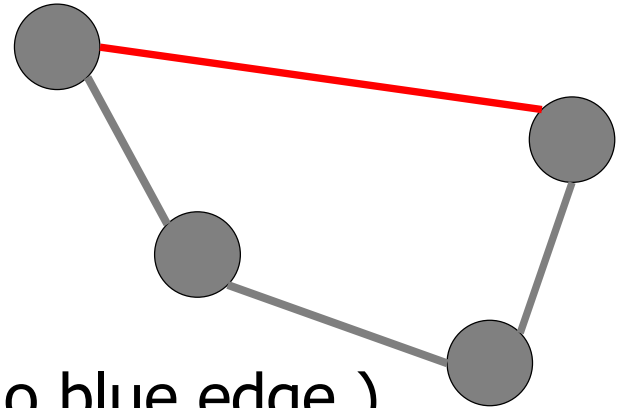


# Generic MST Algorithm

**Claim:** On termination, the blue edges are an MST.

On termination:

1. Every cycle has a **red** edge.  
No **blue** cycles.
2. **Blue** edges form a tree.  
(Otherwise, there is a cut with no blue edge.)
3. Every edge is colored.
4. Every **blue edge** is in the MST (Property 4).



# Generic MST Algorithm

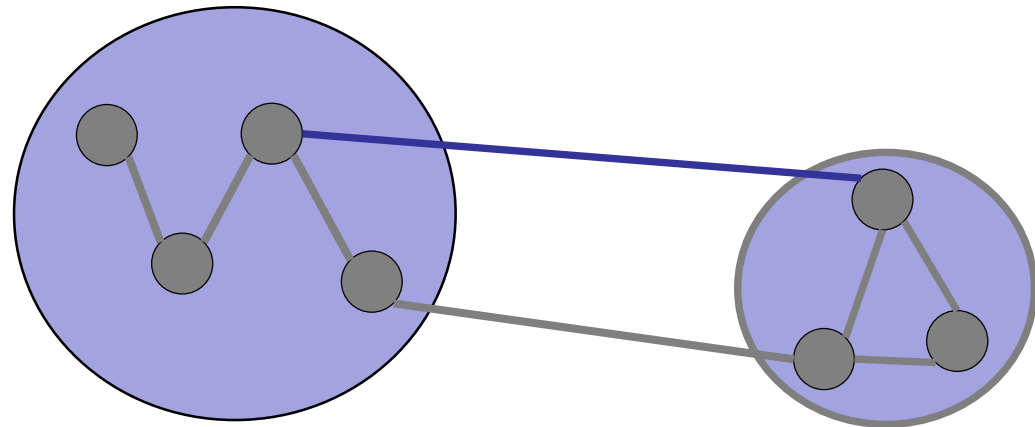
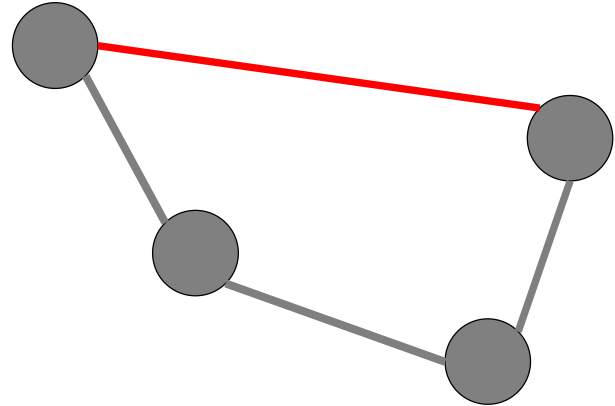
---

## Greedy Algorithm:

Repeat:

**Apply red rule or  
blue rule to an  
arbitrary edge.**

until no more edges  
can be colored.



# Roadmap

---

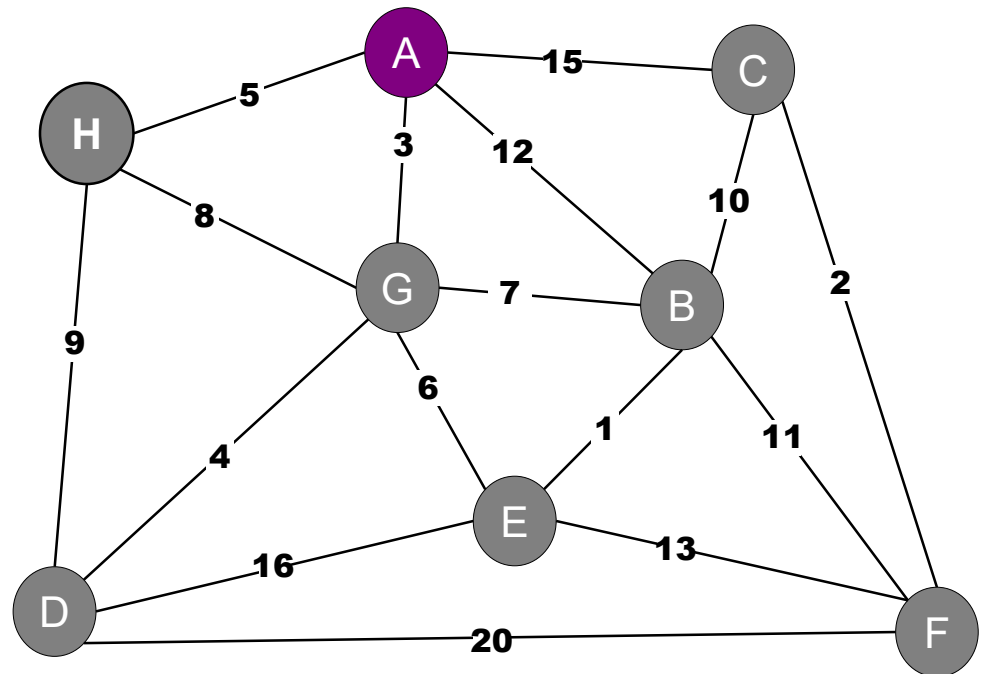
## Minimum Spanning Trees

- The MST Problem
- Basic Properties of an MST
- Generic MST Algorithm
- **Prim's Algorithm**
- Kruskal's Algorithm
- Boruvka's Algorithm
- Variations

# Prim's Algorithm

---

Prim's Algorithm. (Jarnik 1930, Dijkstra 1957, Prim 1959)



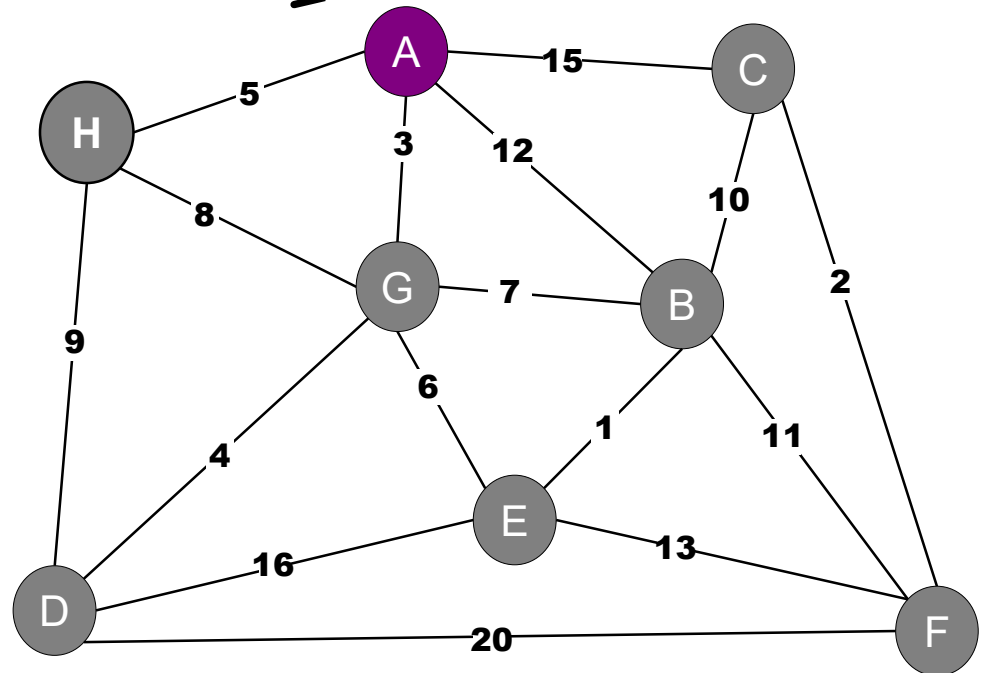
# Prim's Algorithm

---

**Prim's Algorithm.** (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- $S$  : set of nodes connected by blue edges. (An MST of a subgraph  $S$ )
- Initially:  $S = \{A\}$



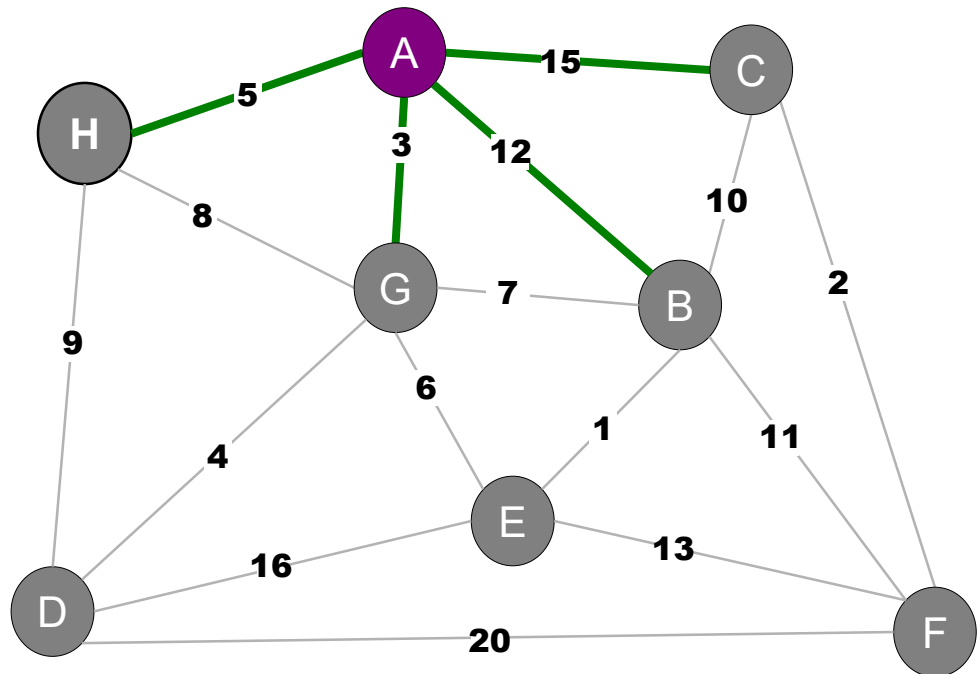
# Prim's Algorithm

---

**Prim's Algorithm.** (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- $S$  : set of nodes connected by blue edges.
- Initially:  $S = \{A\}$
- Identify cut:  $\{S, V-S\}$

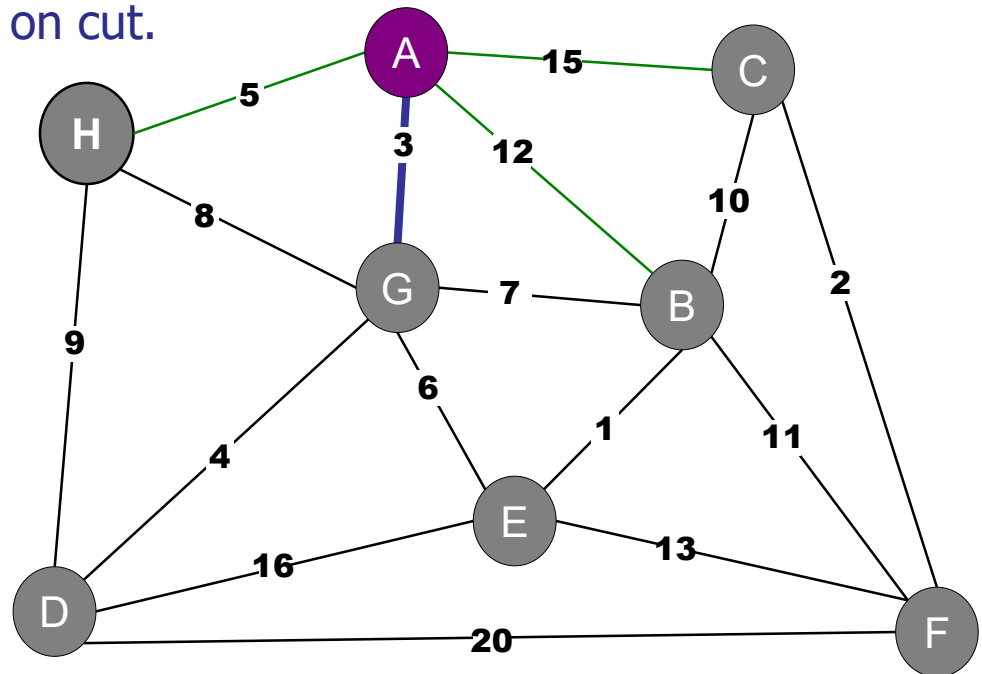


# Prim's Algorithm

**Prim's Algorithm.** (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- $S$  : set of nodes connected by blue edges.
- Initially:  $S = \{A\}$
- Identify cut:  $\{S, V-S\}$
- Find minimum weight edge on cut.

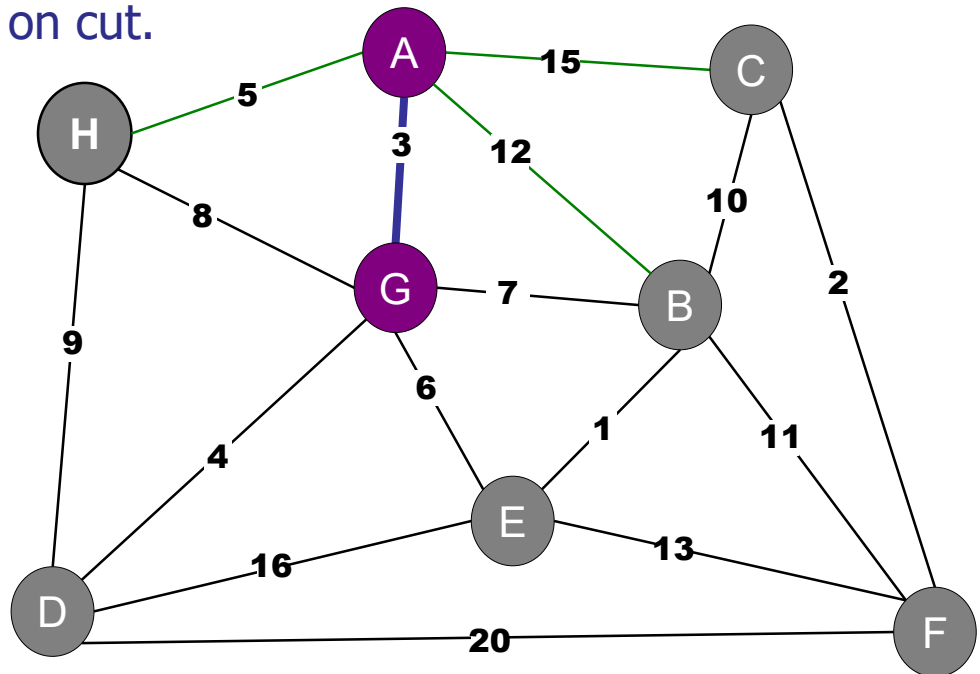


# Prim's Algorithm

**Prim's Algorithm.** (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- $S$  : set of nodes connected by blue edges.
- Initially:  $S = \{A\}$
- Identify cut:  $\{S, V-S\}$
- Find minimum weight edge on cut.
- Add new node to  $S$ .





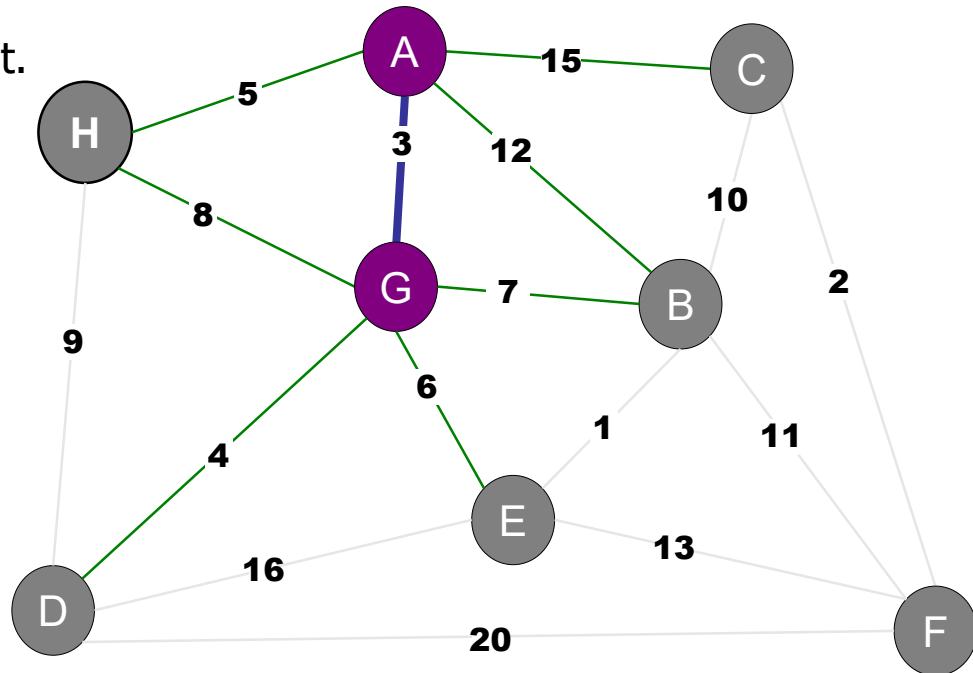
# Prim's Algorithm

---

**Prim's Algorithm.** (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- $S$  : set of nodes connected by blue edges.
- Initially:  $S = \{A\}$
- Repeat:
  - Identify cut:  $\{S, V-S\}$
  - Find minimum weight edge on cut.
  - Add new node to  $S$ .

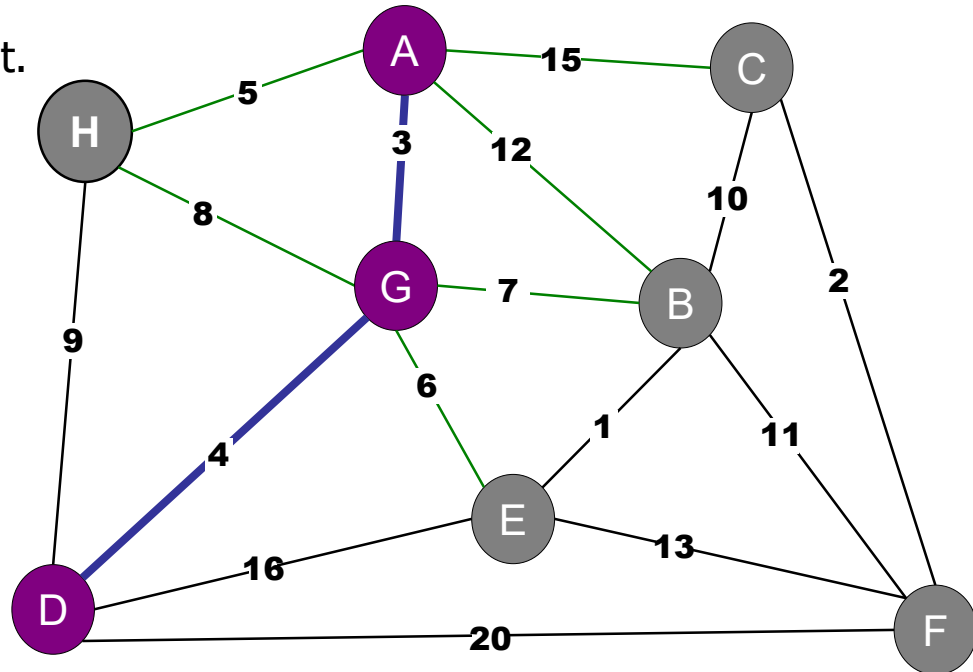


# Prim's Algorithm

**Prim's Algorithm.** (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- $S$  : set of nodes connected by blue edges.
- Initially:  $S = \{A\}$
- Repeat:
  - Identify cut:  $\{S, V-S\}$
  - Find minimum weight edge on cut.
  - Add new node to  $S$ .

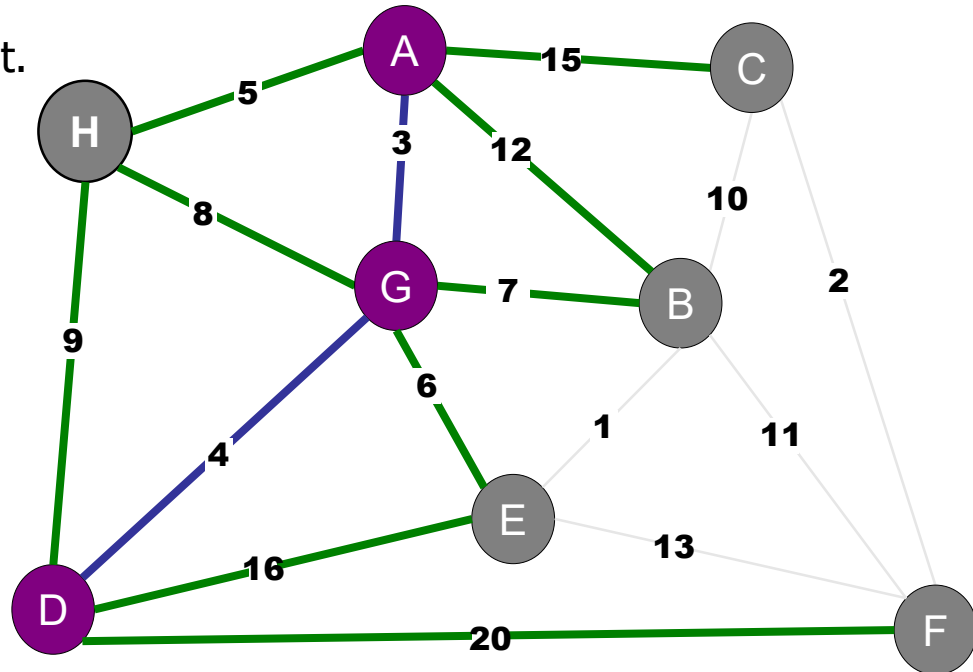


# Prim's Algorithm

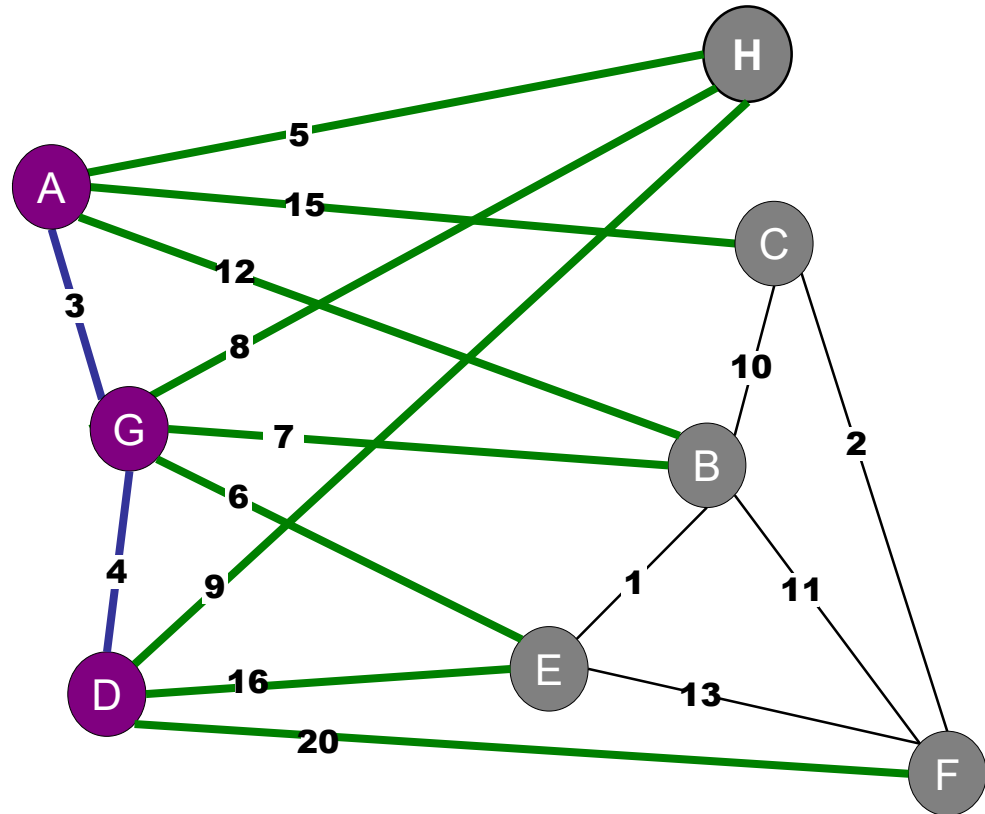
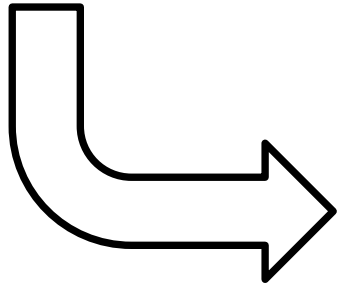
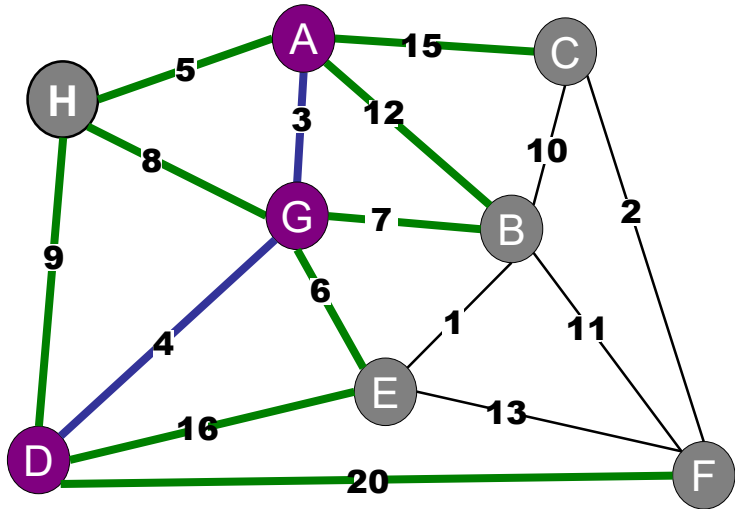
**Prim's Algorithm.** (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- $S$  : set of nodes connected by blue edges.
- Initially:  $S = \{A\}$
- Repeat:
  - Identify cut:  $\{S, V-S\}$
  - Find minimum weight edge on cut.
  - Add new node to  $S$ .



# Prim's Algorithm



# Prim's Algorithm: Initialization

---

// Initialize priority queue

```
PriorityQueue pq = new PriorityQueue();  
for (Node v : G.V()) {  
    pq.insert(v, INFTY);  
}
```

```
pq.decreaseKey(start, 0);
```

// Initialize set S

```
HashSet<Node> S = new HashSet<Node>();  
S.put(start);
```

// Initialize parent hash table

```
HashMap<Node, Node> parent = new HashMap<Node, Node>();  
parent.put(start, null);
```

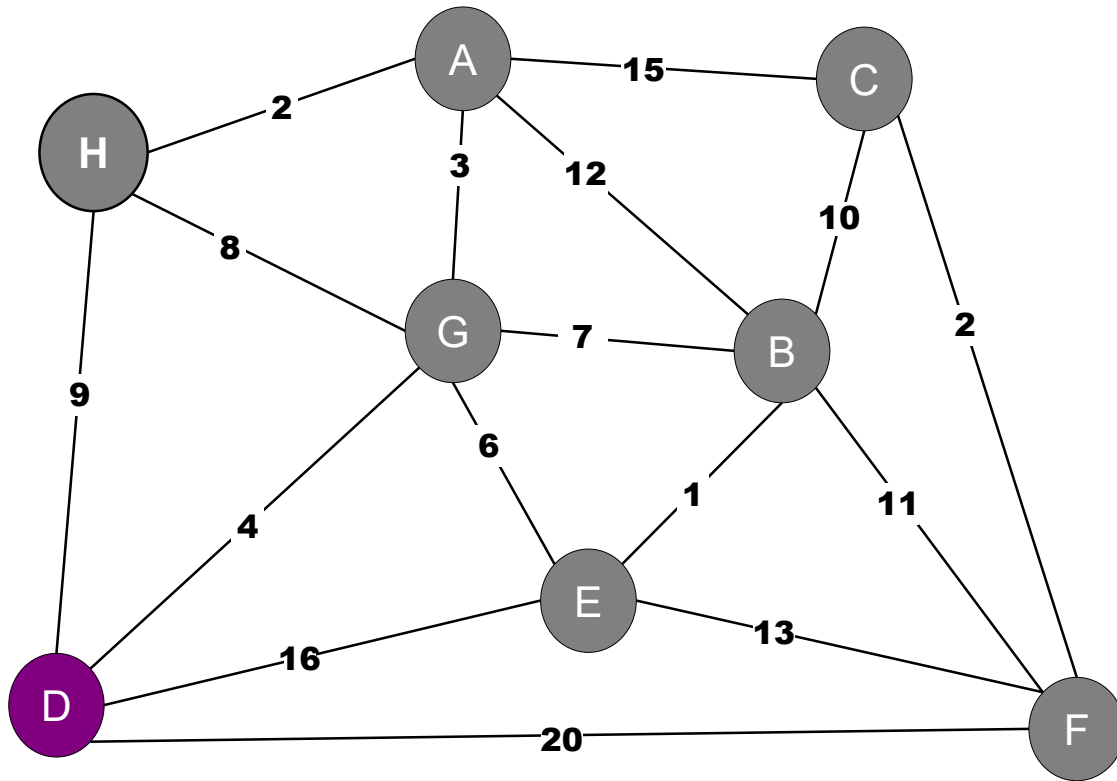
# Prim's Algorithm

---

```
while (!pq.isEmpty()) {  
    Node v = pq.deleteMin();  
    S.put(v);  
    for each (Edge e : v.edgeList()) {  
        Node w = e.otherNode(v);  
        if (!S.get(w)) {  
            pq.decreaseKey(w, e.getWeight());  
            parent.put(w, v);  
        }  
    }  
}
```

# Prim's Example

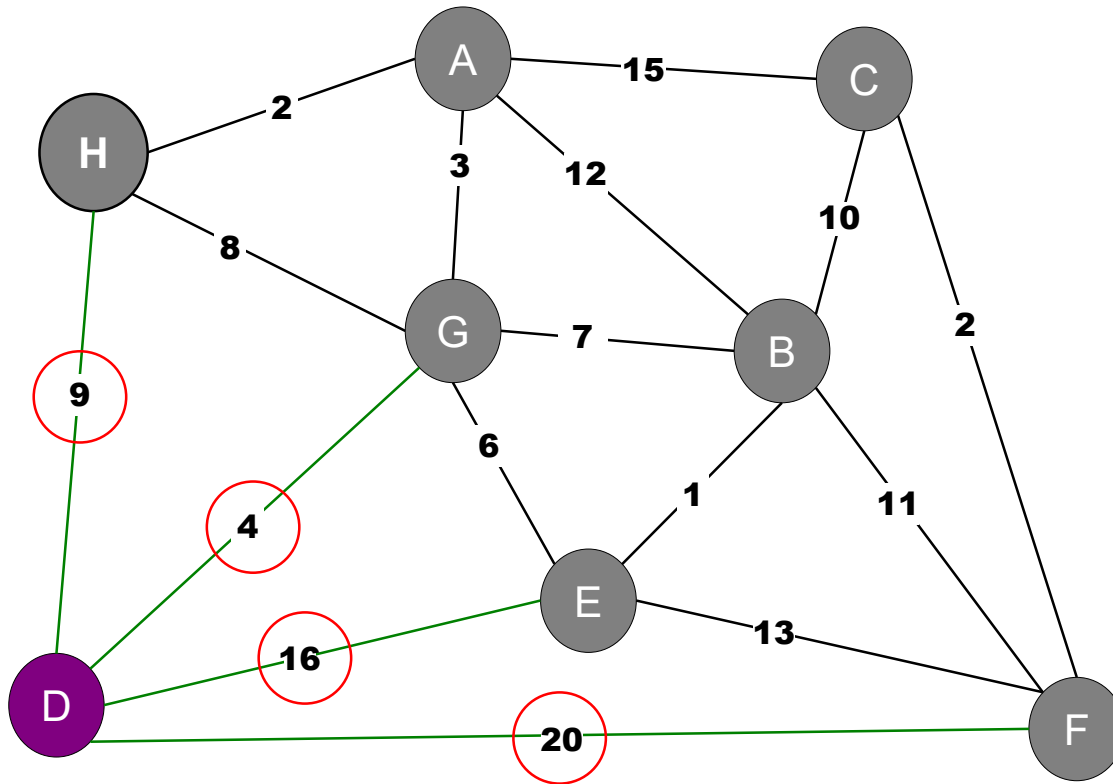
---



Vertex	Weight
D	0

# Prim's Example

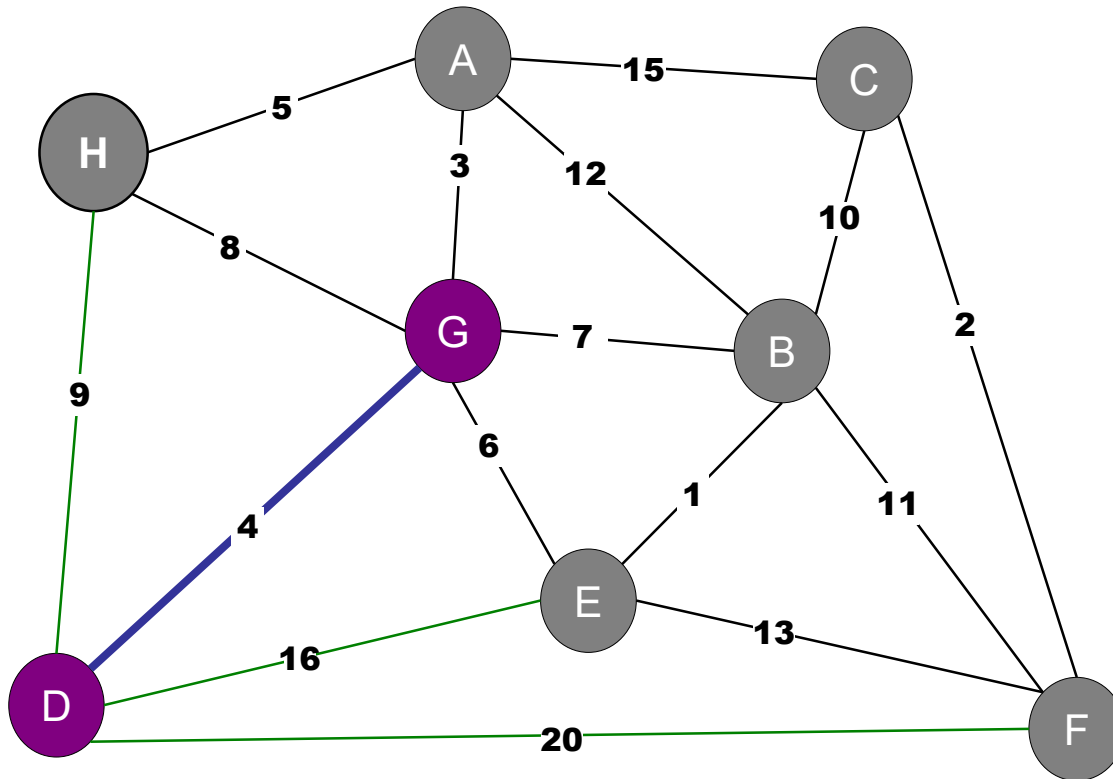
Vertex	Weight
<b>G</b>	<b>4</b>
<b>H</b>	<b>9</b>
<b>E</b>	<b>16</b>
<b>F</b>	<b>20</b>



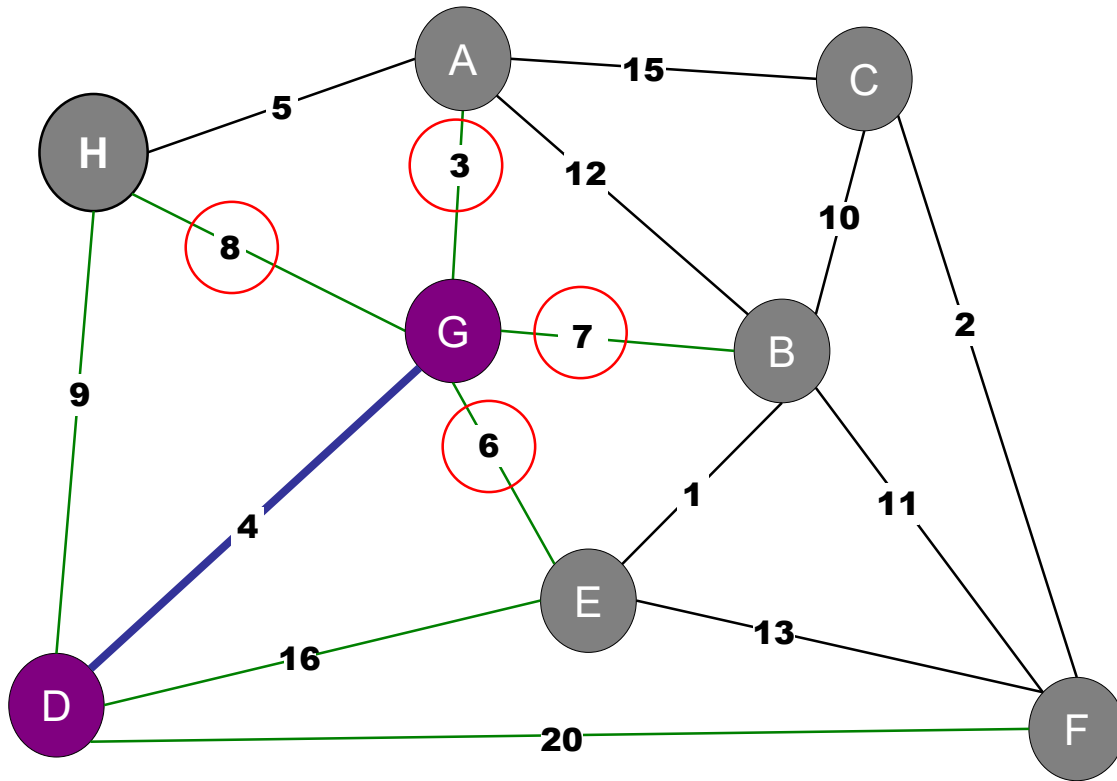


# Prim's Example

Vertex	Weight
H	9
E	16
F	20



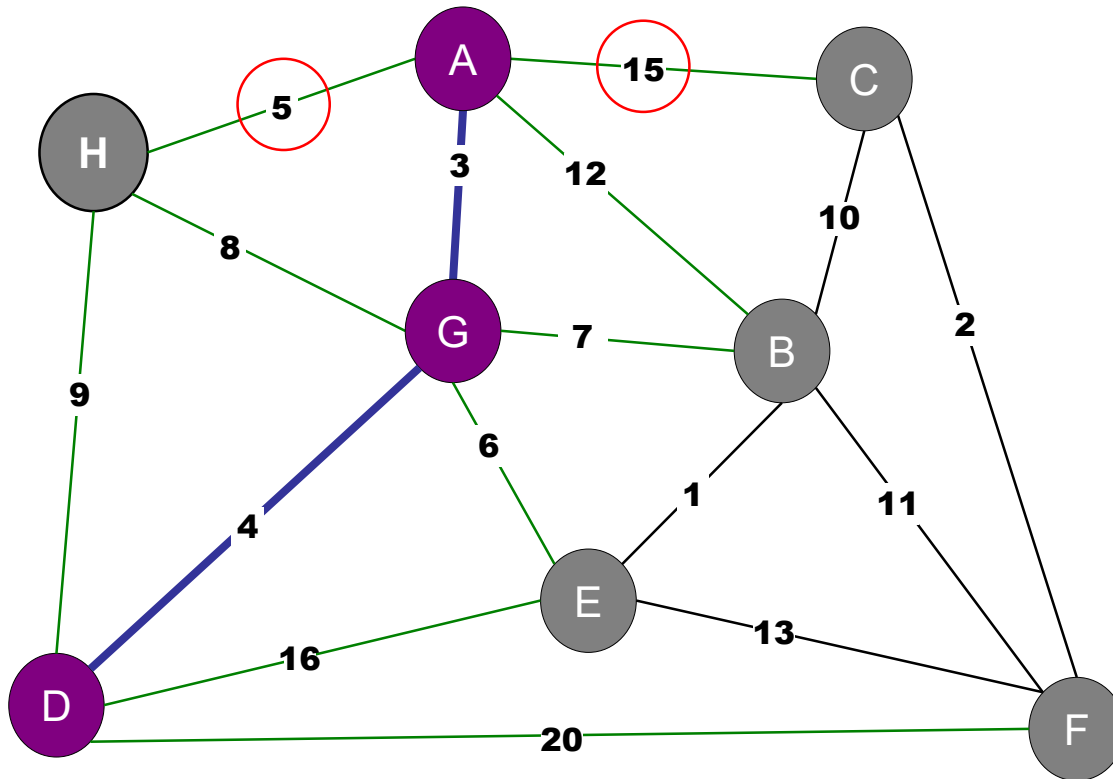
# Prim's Example



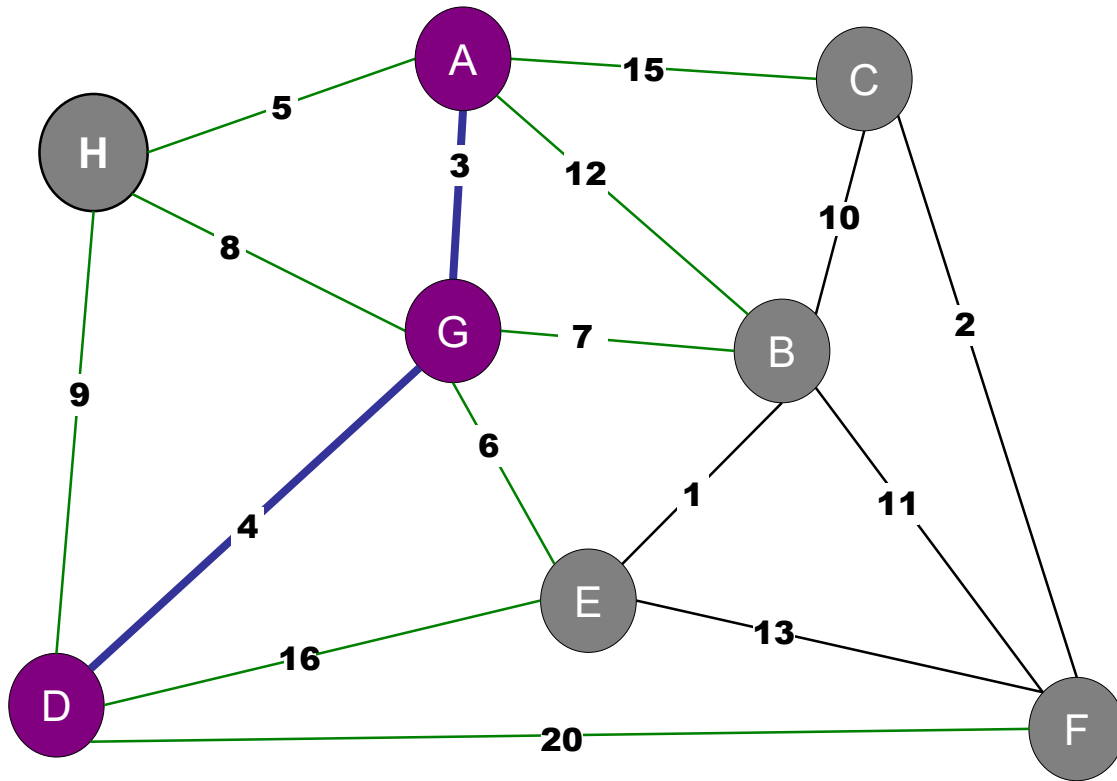
Vertex	Weight
A	3
E	16->6
B	7
H	9->8
F	20

# Prim's Example

Vertex	Weight
H	8->5
E	6
B	7
C	15
F	20



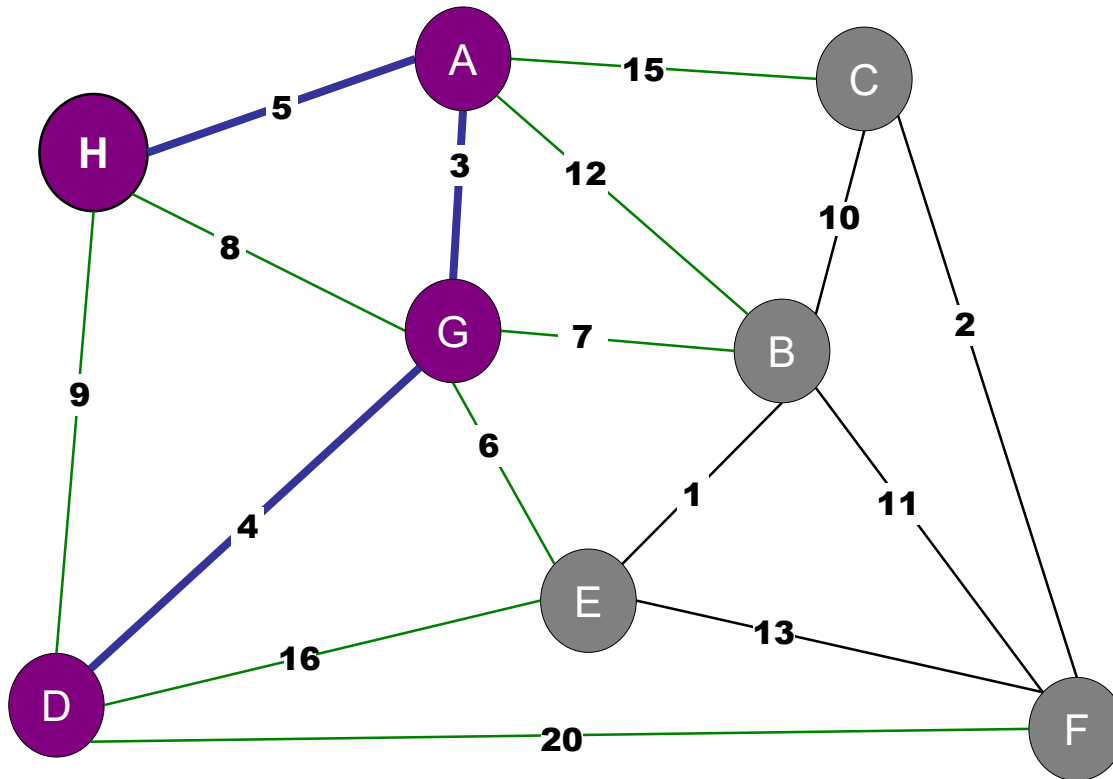
# Prim's Example



Vertex	Weight
H	5
E	6
B	7
C	15
F	20

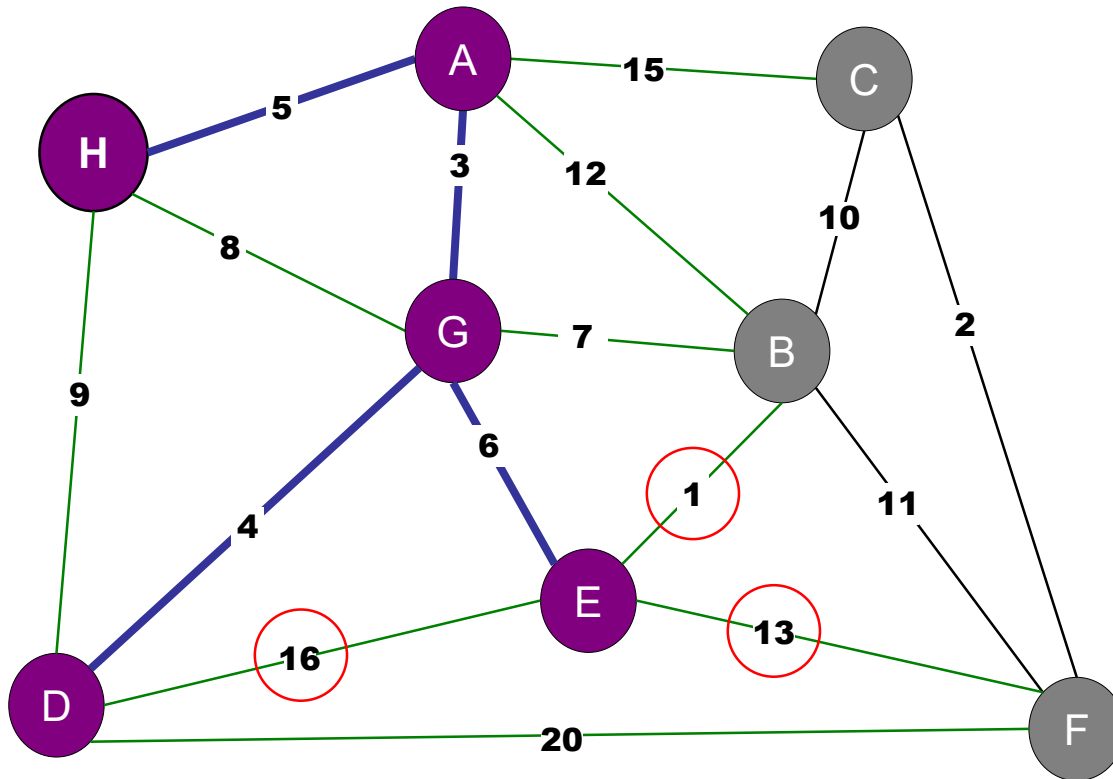
# Prim's Example

Vertex	Weight
E	6
B	7
C	15
F	20



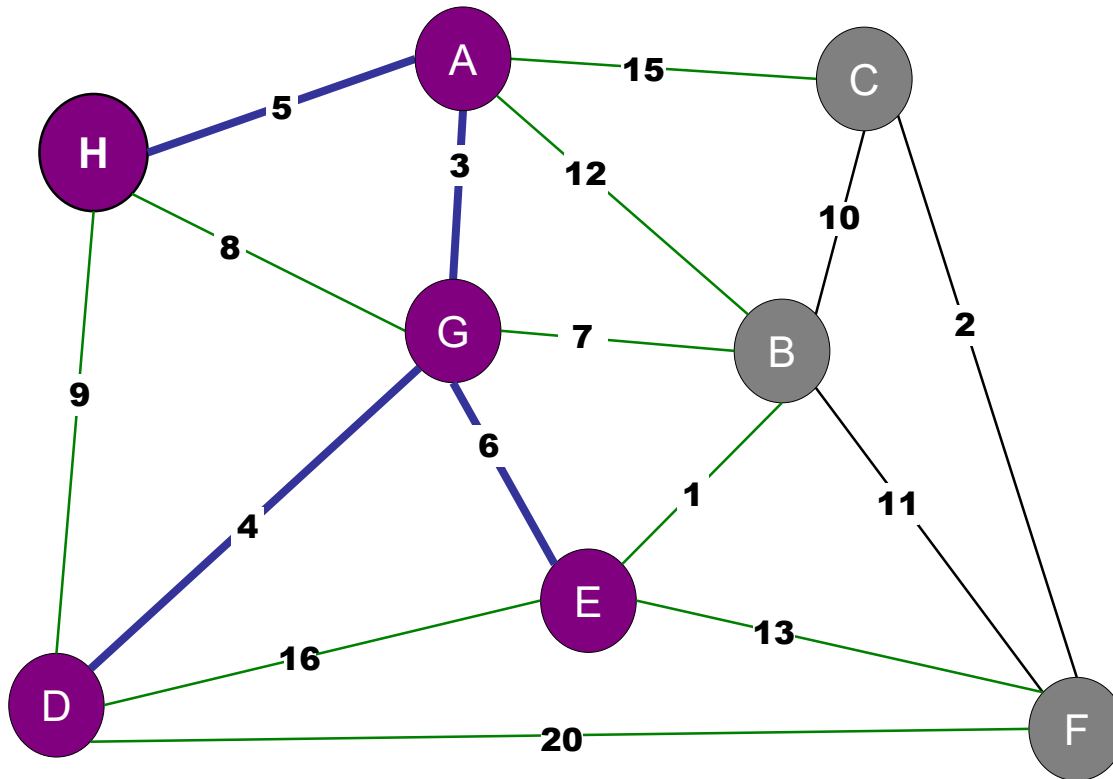
# Prim's Example

Vertex	Weight
<b>B</b>	<b>7-&gt;1</b>
C	15
<b>F</b>	<b>20-&gt;13</b>



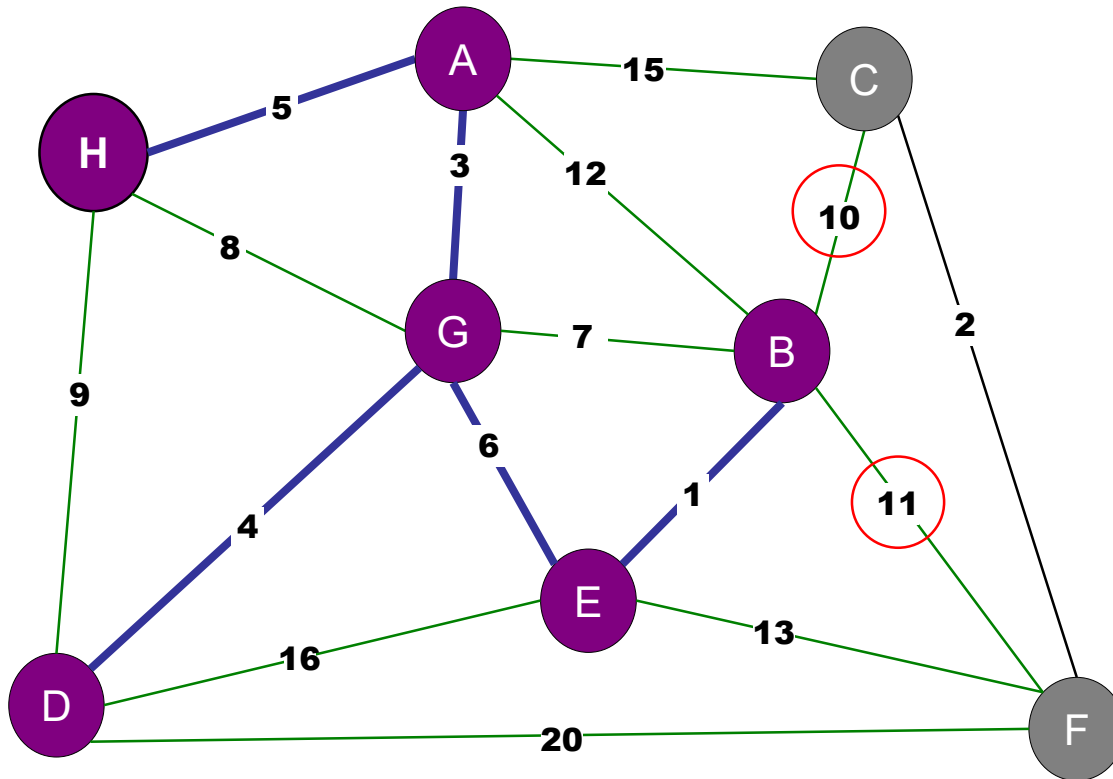
# Prim's Example

Vertex	Weight
B	1
C	15
F	13



# Prim's Example

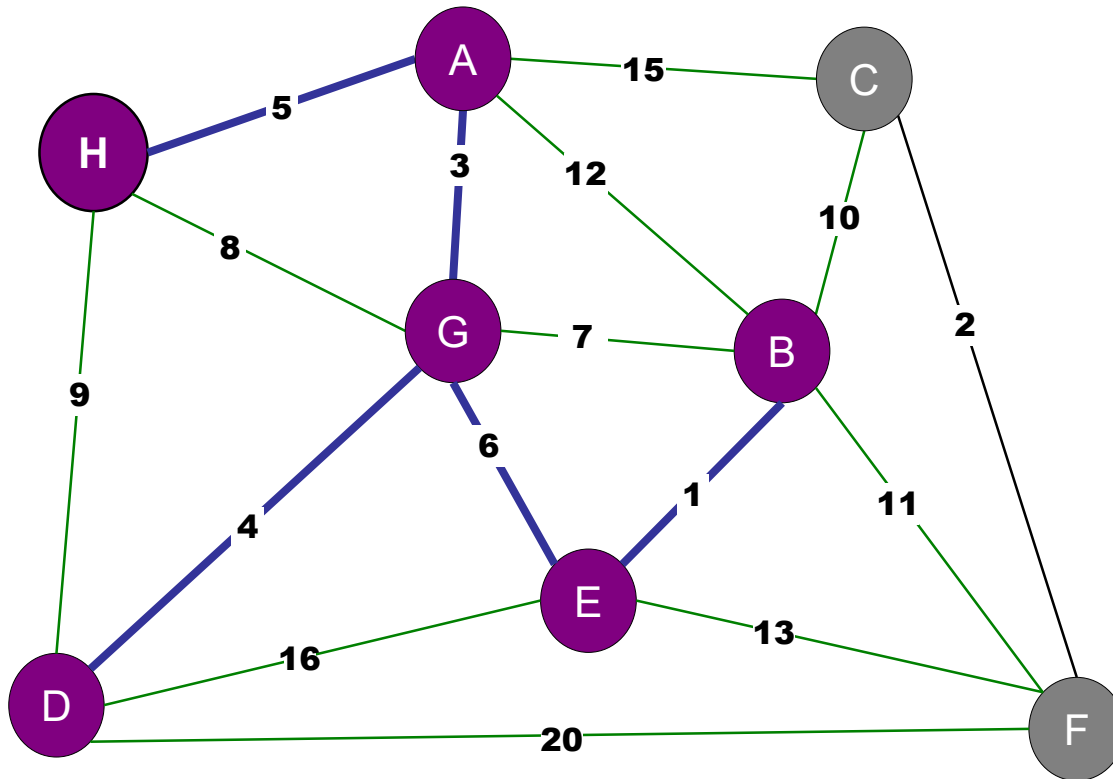
Vertex	Weight
C	<b>15-&gt;10</b>
F	<b>13-&gt;11</b>





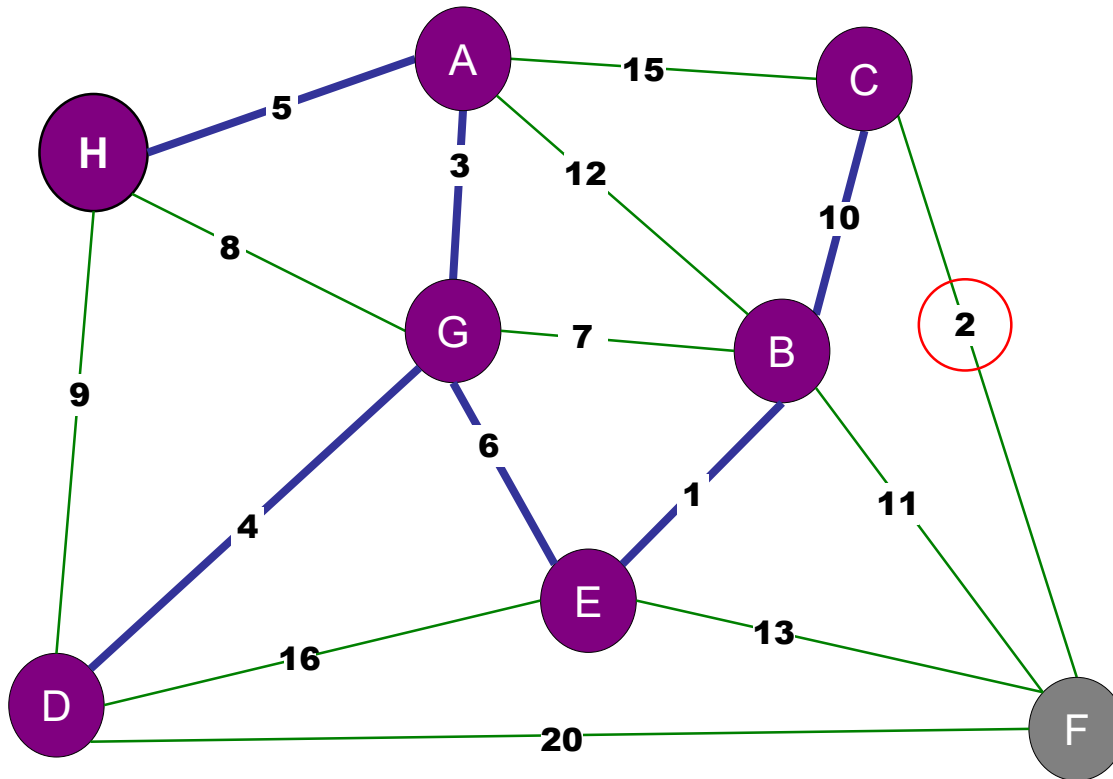
# Prim's Example

Vertex	Weight
C	10
F	11



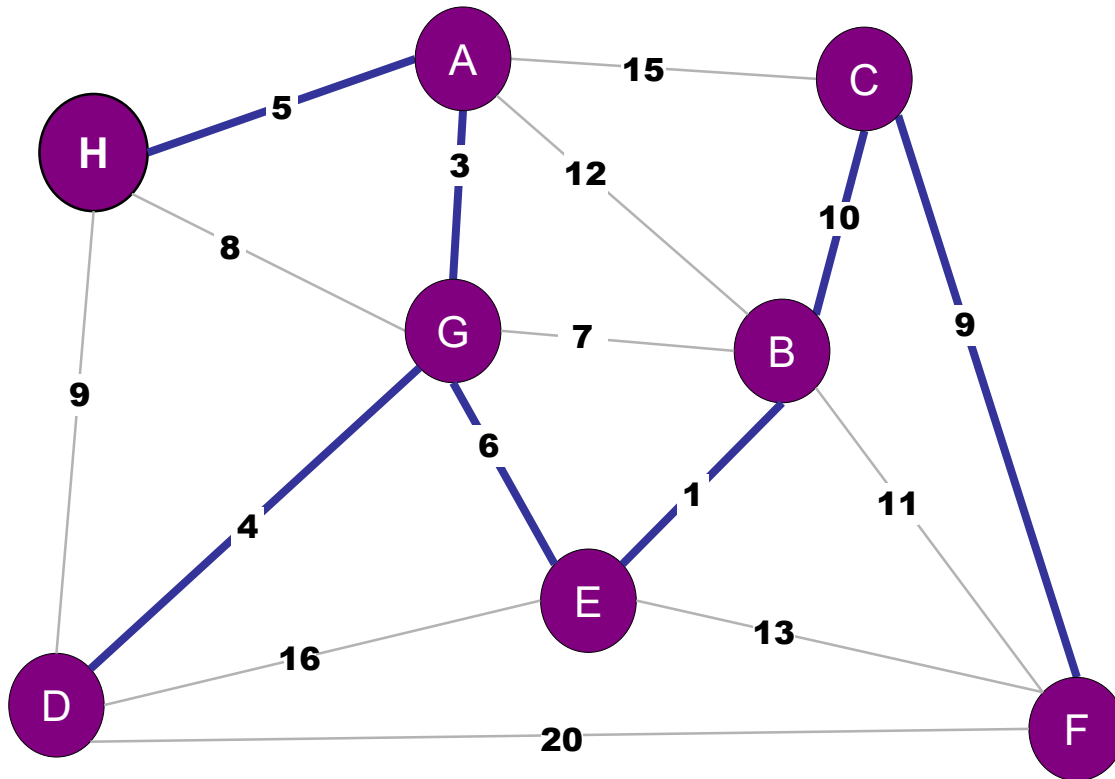
# Prim's Example

Vertex	Weight
F	11->2



# Prim's Example

Vertex	Weight



# Prim's Algorithm

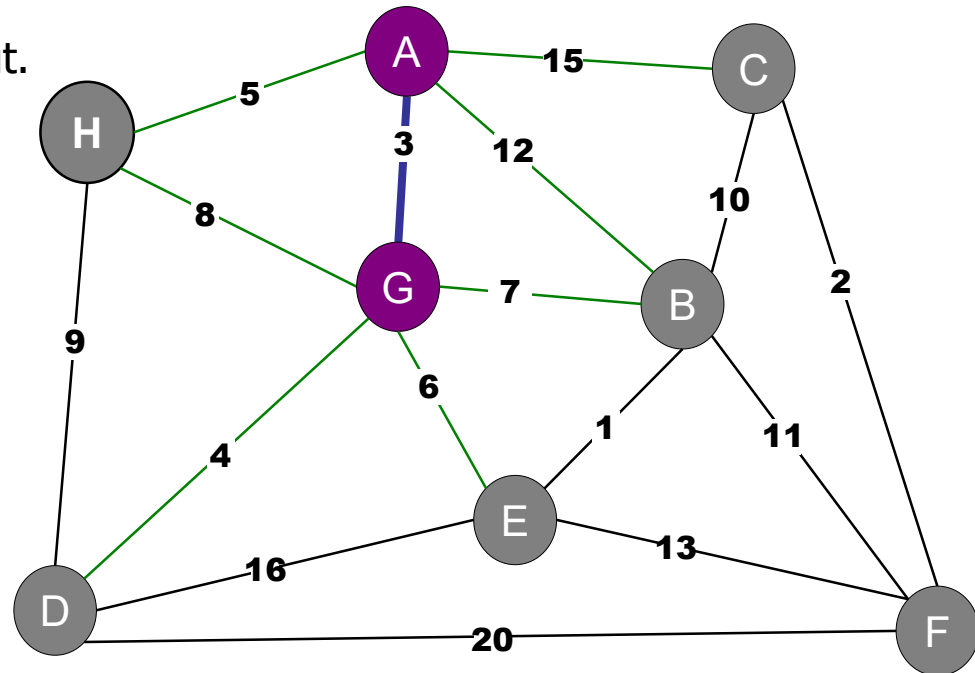
**Prim's Algorithm.** (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- $S$  : set of nodes connected by blue edges.
- Initially:  $S = \{A\}$
- Repeat:
  - Identify cut:  $\{S, V-S\}$
  - Find minimum weight edge on cut.
  - Add new node to  $S$ .

Proof:

- Each added edge is the lightest on some cut.
- Hence each edge is in the MST.



# Prim's Algorithm

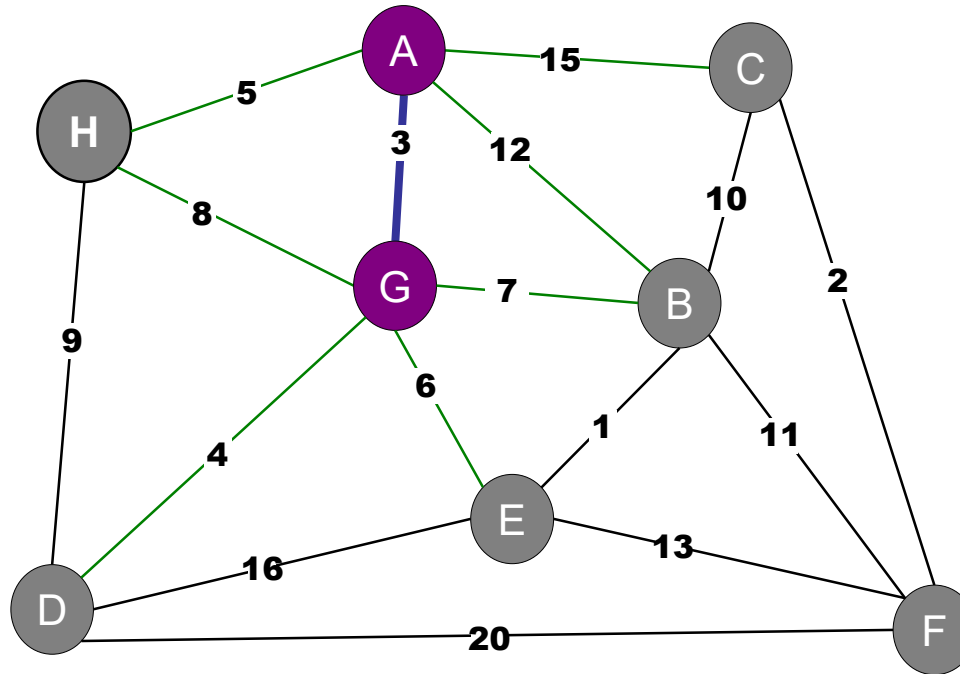
**Prim's Algorithm.** (Jarnik 1930, Dijkstra 1957, Prim 1959)

Basic idea:

- $S$  : set of nodes connected by blue edges.
- Initially:  $S = \{A\}$
- Repeat:
  - Identify cut:  $\{S, V-S\}$
  - Find minimum weight edge on cut.
  - Add new node to  $S$ .

Analysis:

- Each vertex added/removed once from the priority queue:  $O(V \log V)$
- Each edge  $\Rightarrow$  one decreaseKey:  $O(E \log V)$ .



# Two Algorithms

---

## Prim's Algorithm.

Basic idea:

- Maintain a set of visited nodes.
- Greedily grow the set by adding node connected via the **lightest** edge.
  - Use Priority Queue to order nodes by **edge weight**.

## Dijkstra's Algorithm.

Basic idea:

- Maintain a set of visited nodes.
- Greedily grow the set by adding neighboring node that is **closest to the source**.
  - Use Priority Queue to order nodes by **distance**.

# Roadmap

---

## Minimum Spanning Trees

- The MST Problem
- Basic Properties of an MST
- Generic MST Algorithm
- Prim's Algorithm
- **Kruskal's Algorithm**
- Boruvka's Algorithm
- Variations

# Generic MST Algorithm

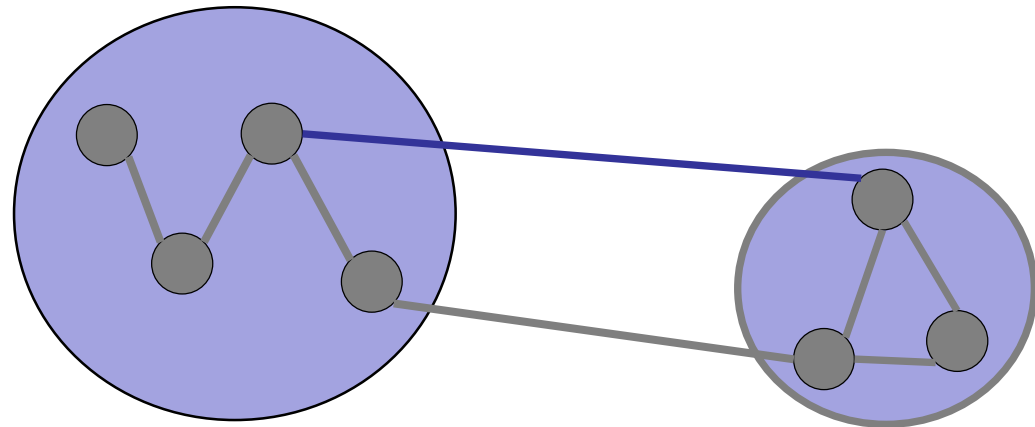
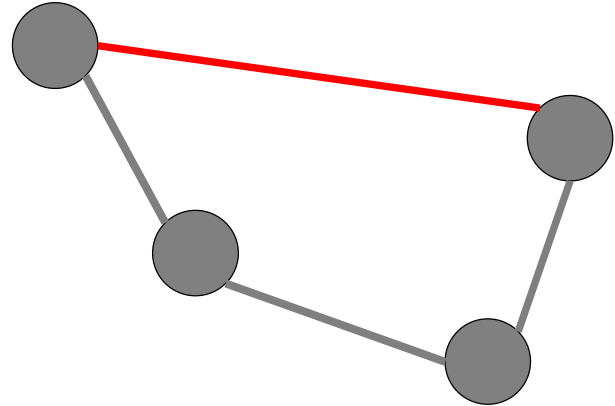
---

## Greedy Algorithm:

Repeat:

**Apply red rule or  
blue rule to an  
arbitrary edge.**

until no more edges  
can be colored.

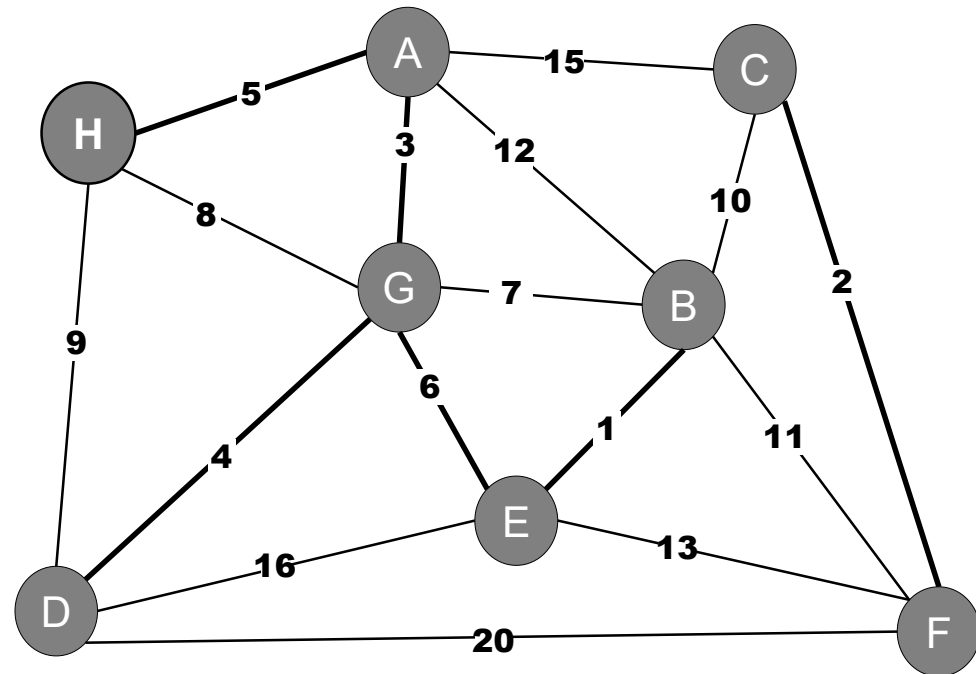




# Kruskal's Algorithm

---

Kruskal's Algorithm. (Kruskal 1956)



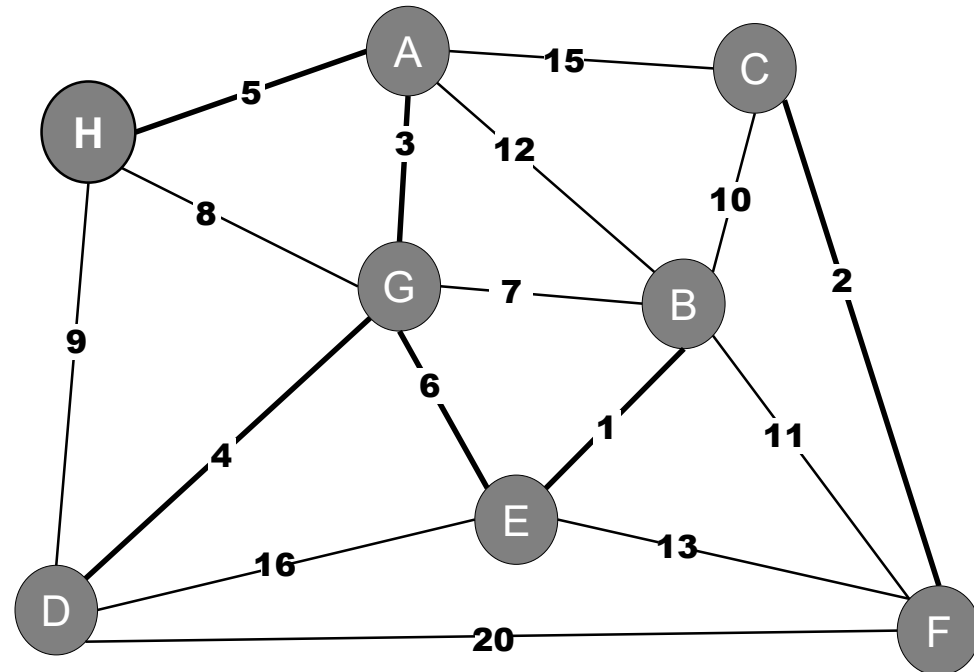
# Kruskal's Algorithm

---

## Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.

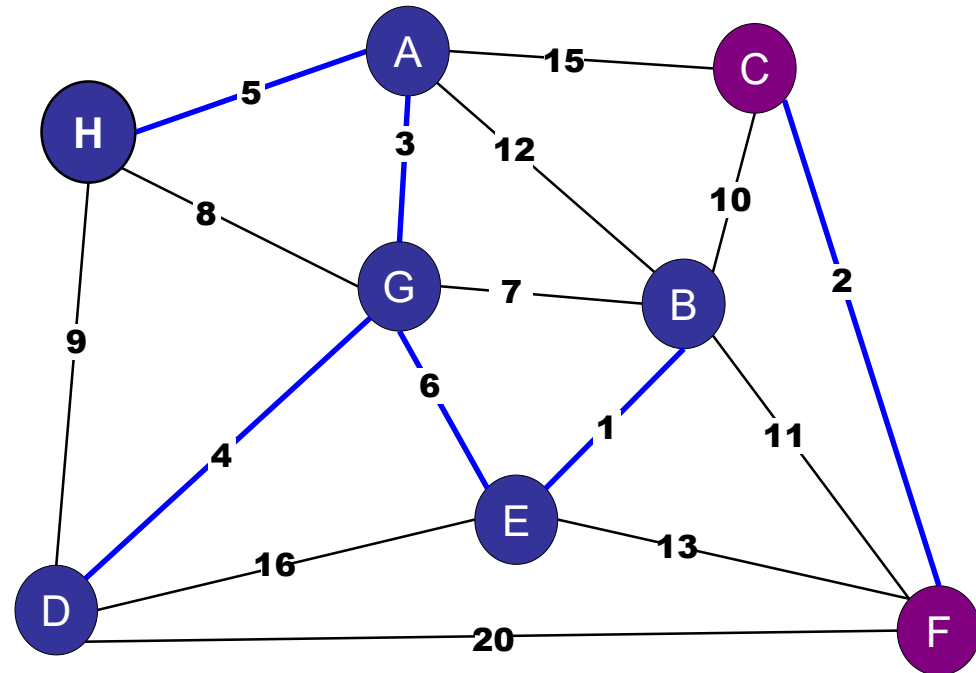


# Kruskal's Algorithm

## Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.

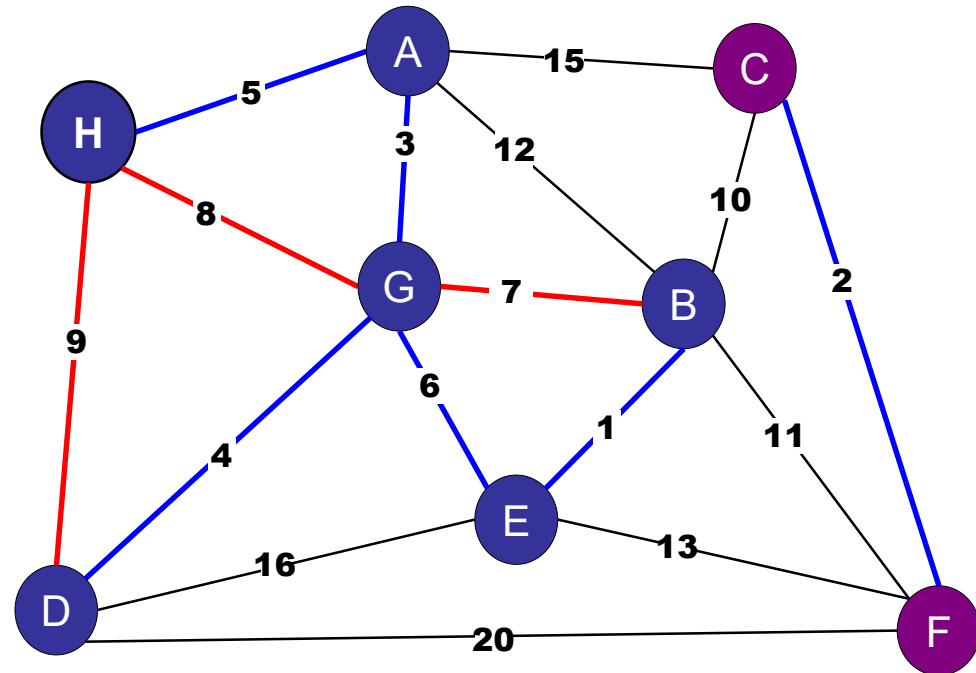


# Kruskal's Algorithm

## Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.



# Kruskal's Algorithm

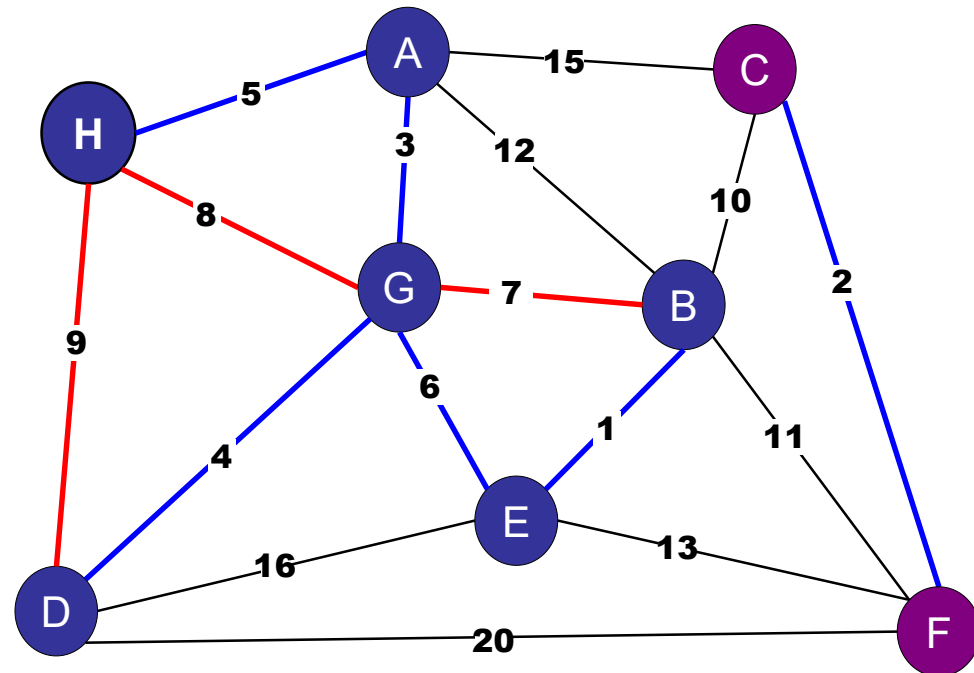
## Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.

Data structure:

- Union-Find
- Connect two nodes if they are in the same blue tree.



# Kruskal's Algorithm

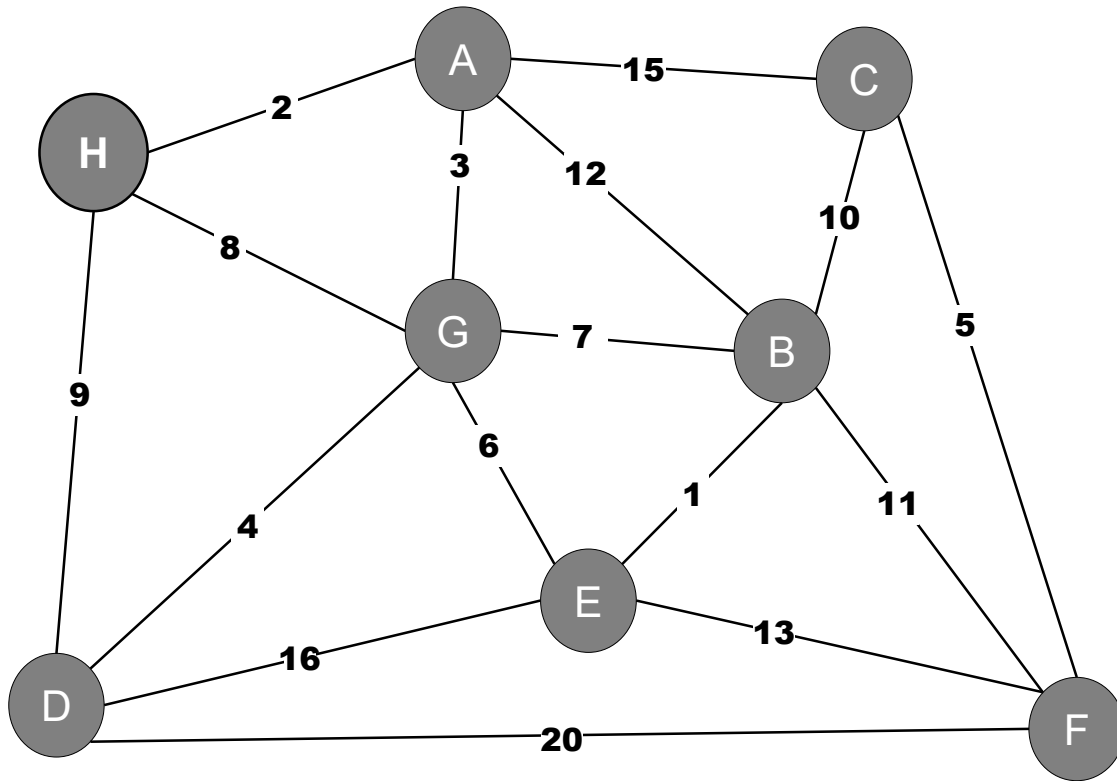
---

```
// Sort edges and initialize
Edge[] sortedEdges = sort(G.E());
ArrayList<Edge> mstEdges = new ArrayList<Edge>();
UnionFind uf = new UnionFind(G.V());

// Iterate through all the edges, in order
for (int i=0; i<sortedEdges.length; i++) {
    Edge e = sortedEdges[i]; // get edge
    Node v = e.one(); // get node endpoints
    Node w = e.two();

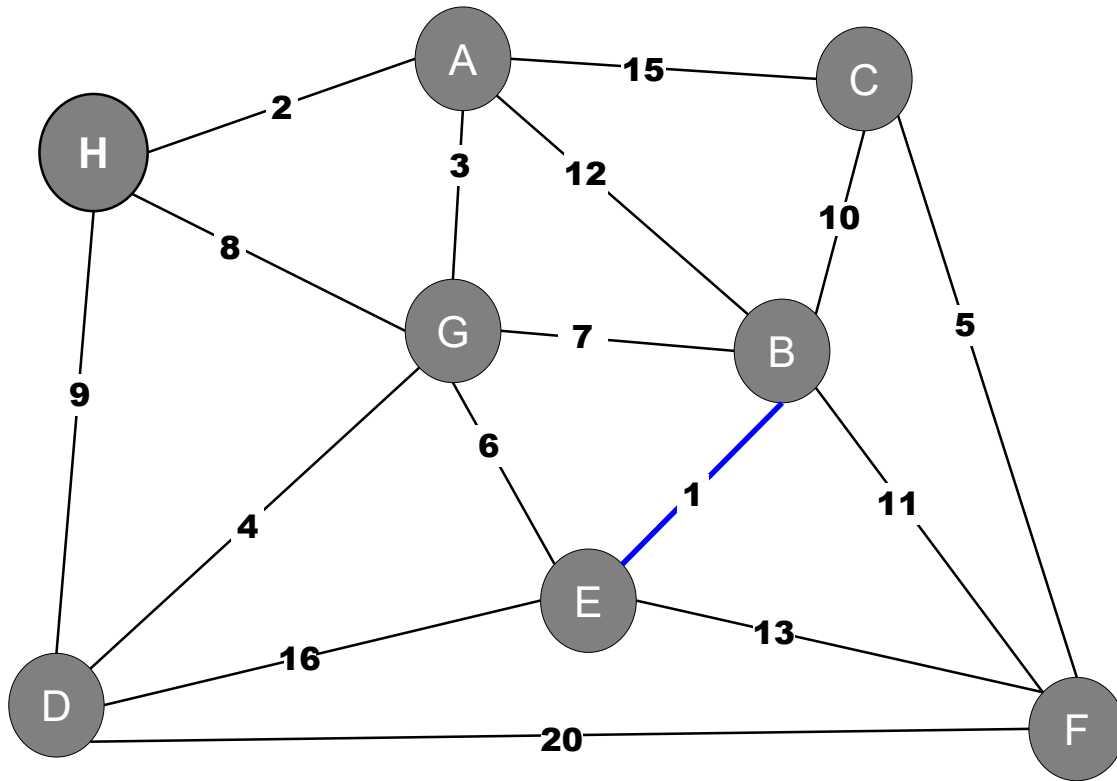
    if (!uf.find(v,w)) { // in the same tree?
        mstEdges.add(e); // save edge
        uf.union(v,w); // combine trees
    }
}
```

# Kruskal's Example



Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

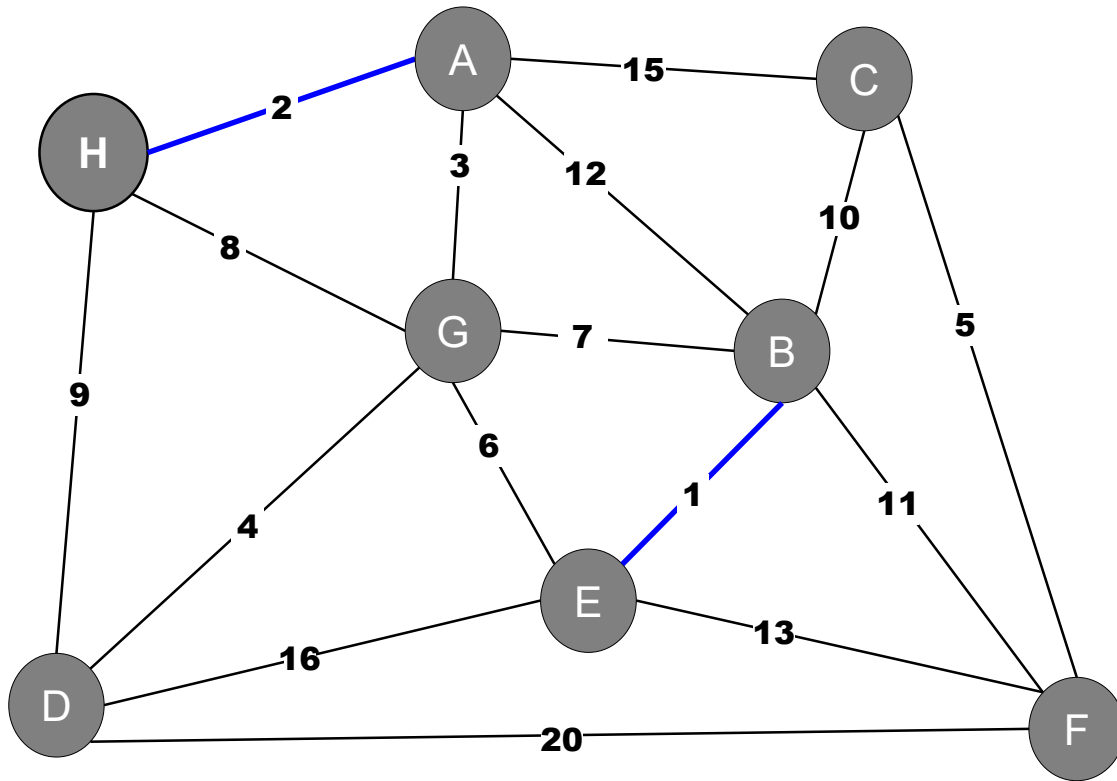
# Kruskal's Example



Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

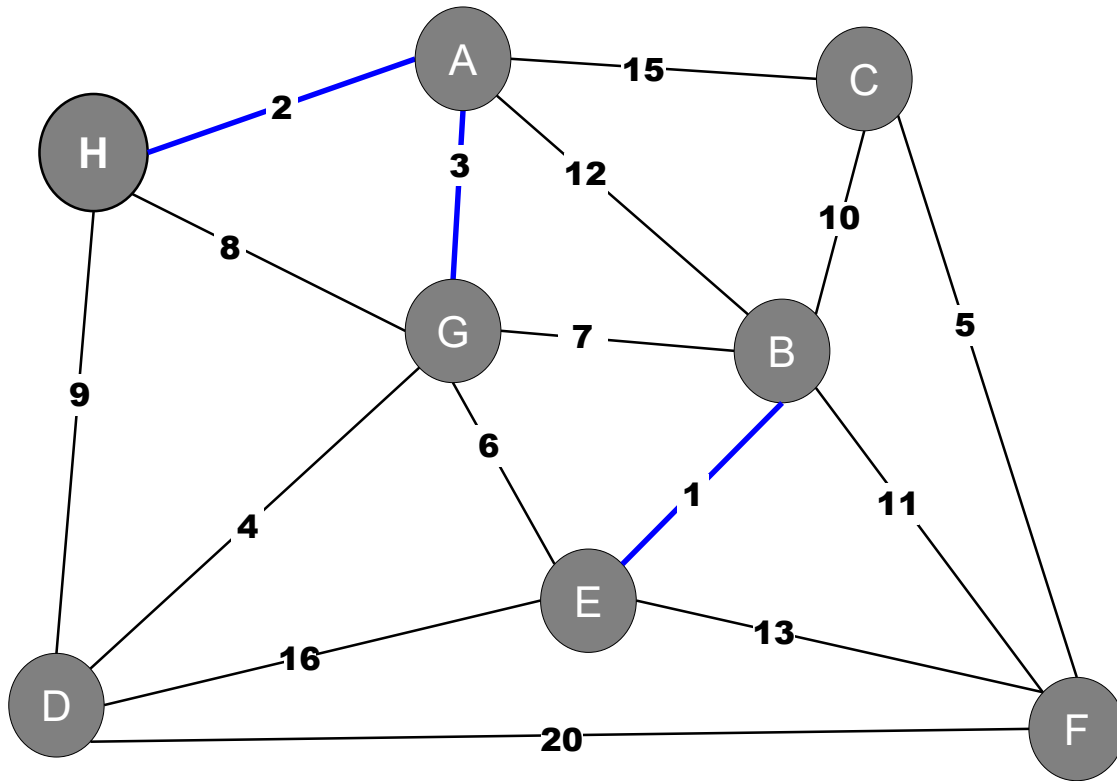


# Kruskal's Example



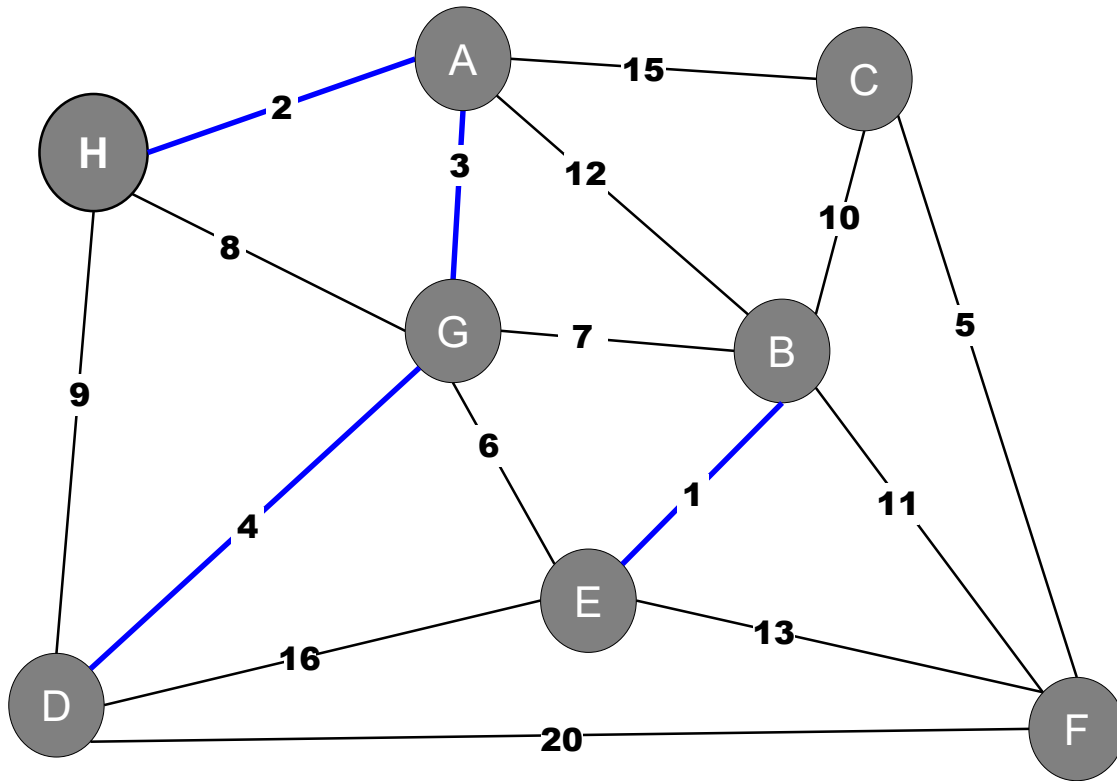
Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

# Kruskal's Example



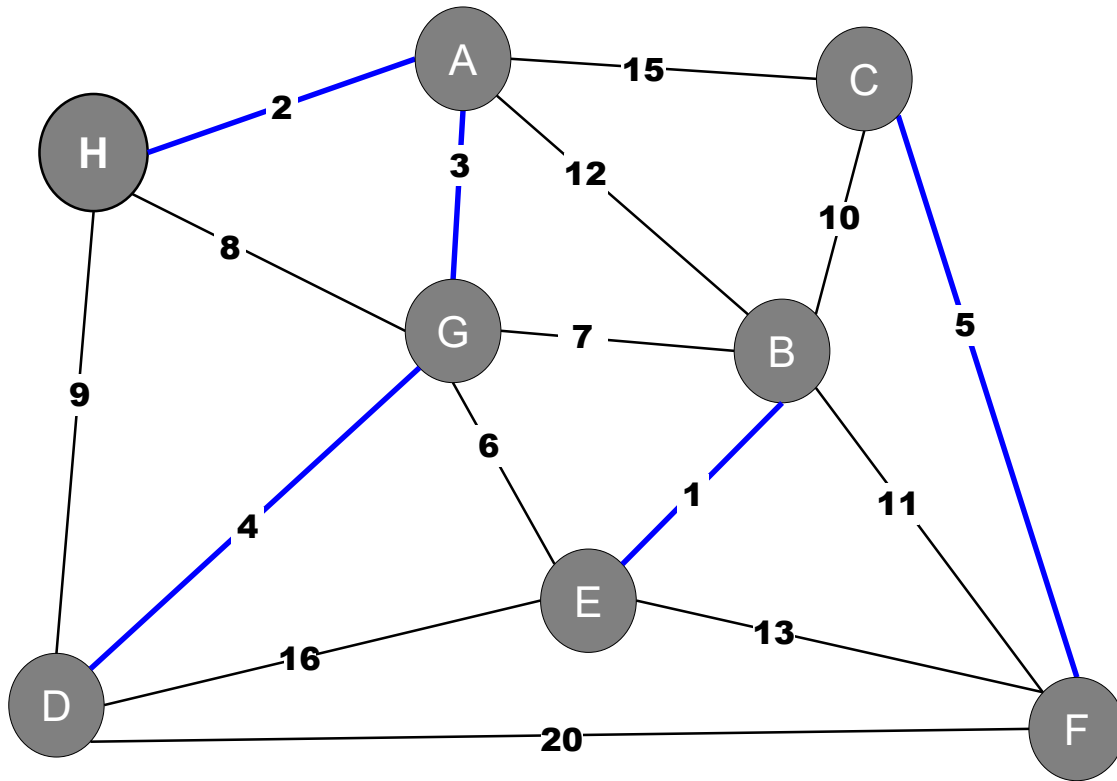
Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

# Kruskal's Example



Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	<b>(D,G)</b>
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

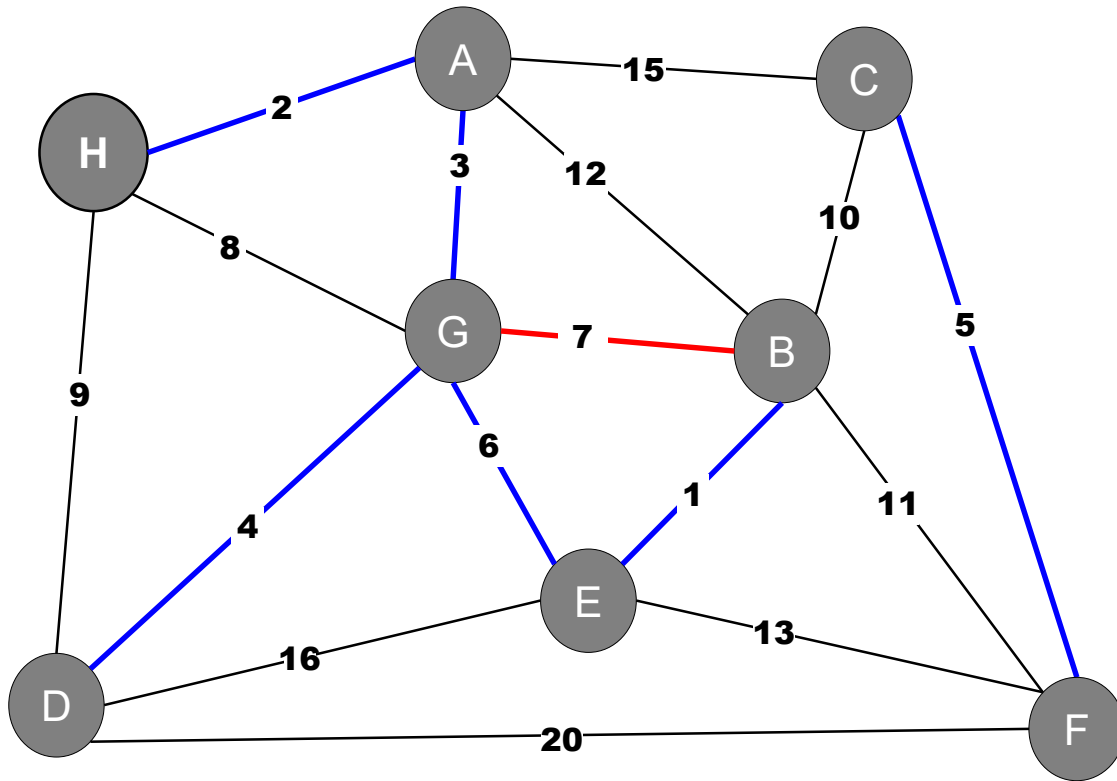
# Kruskal's Example



Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

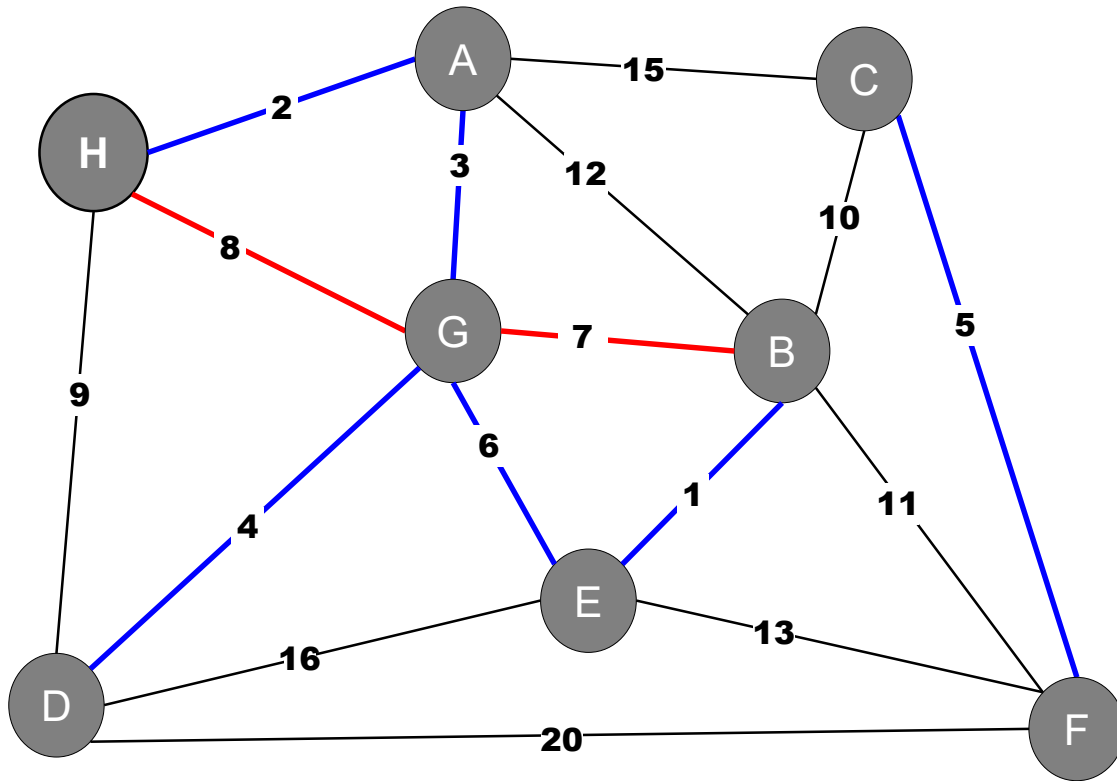


# Kruskal's Example



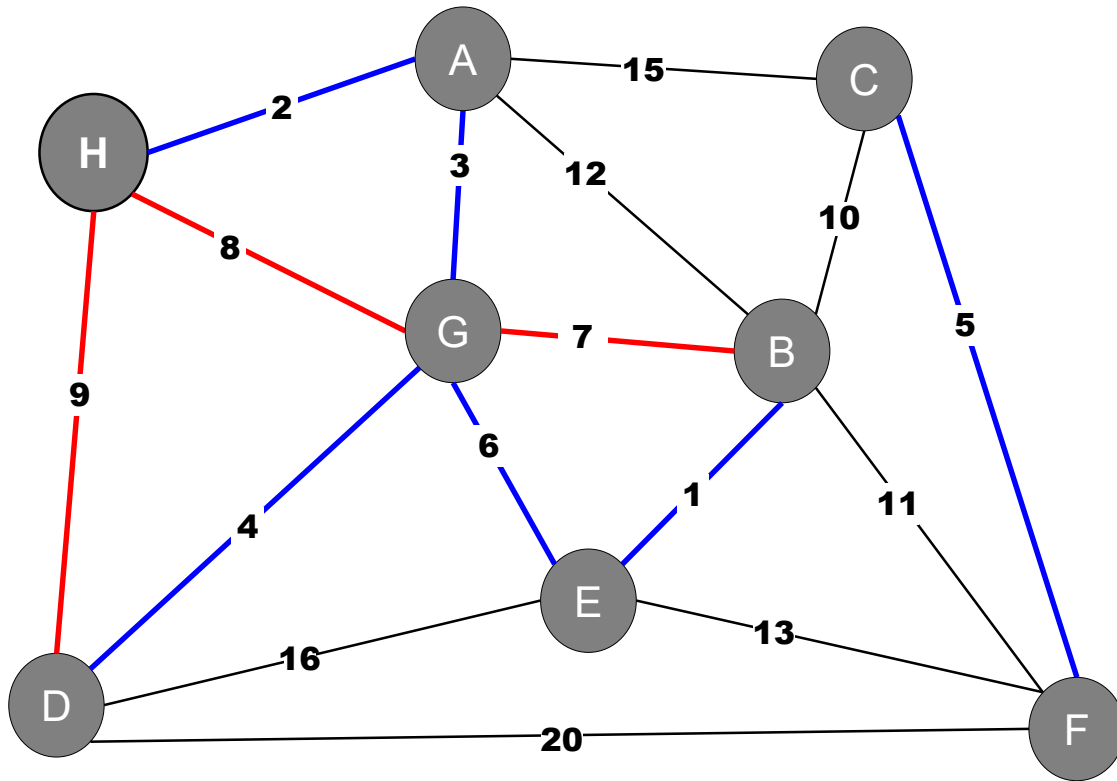
Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

# Kruskal's Example



Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

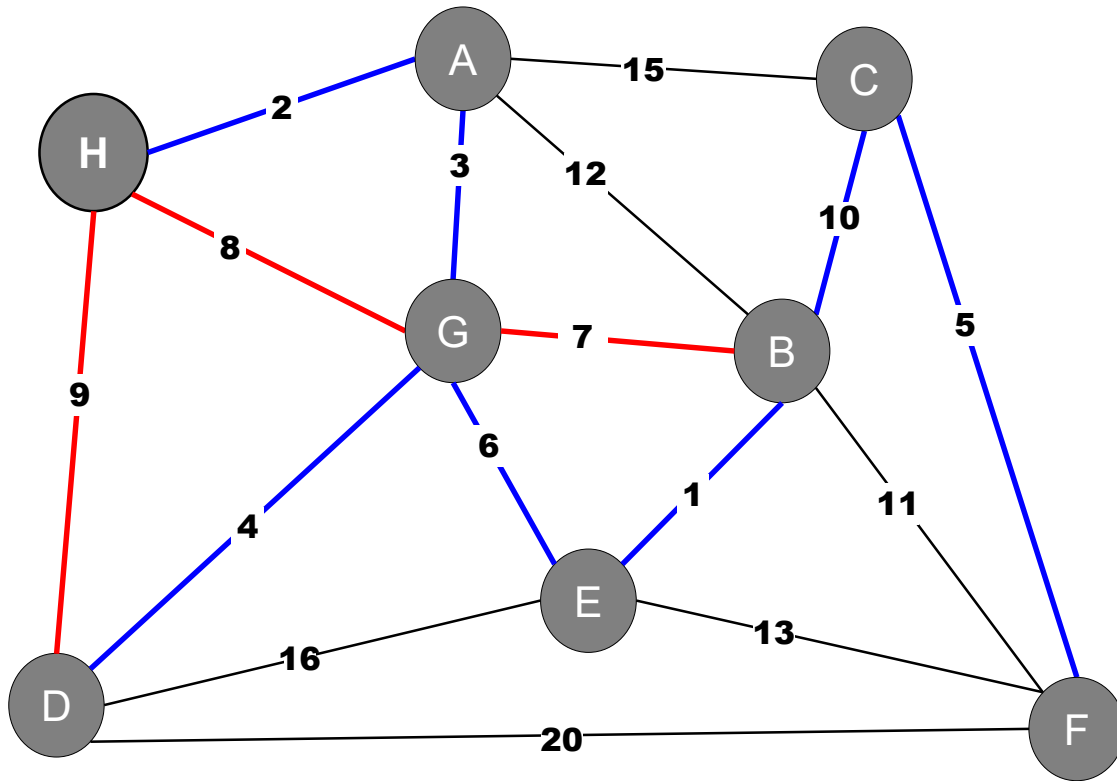
# Kruskal's Example



Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

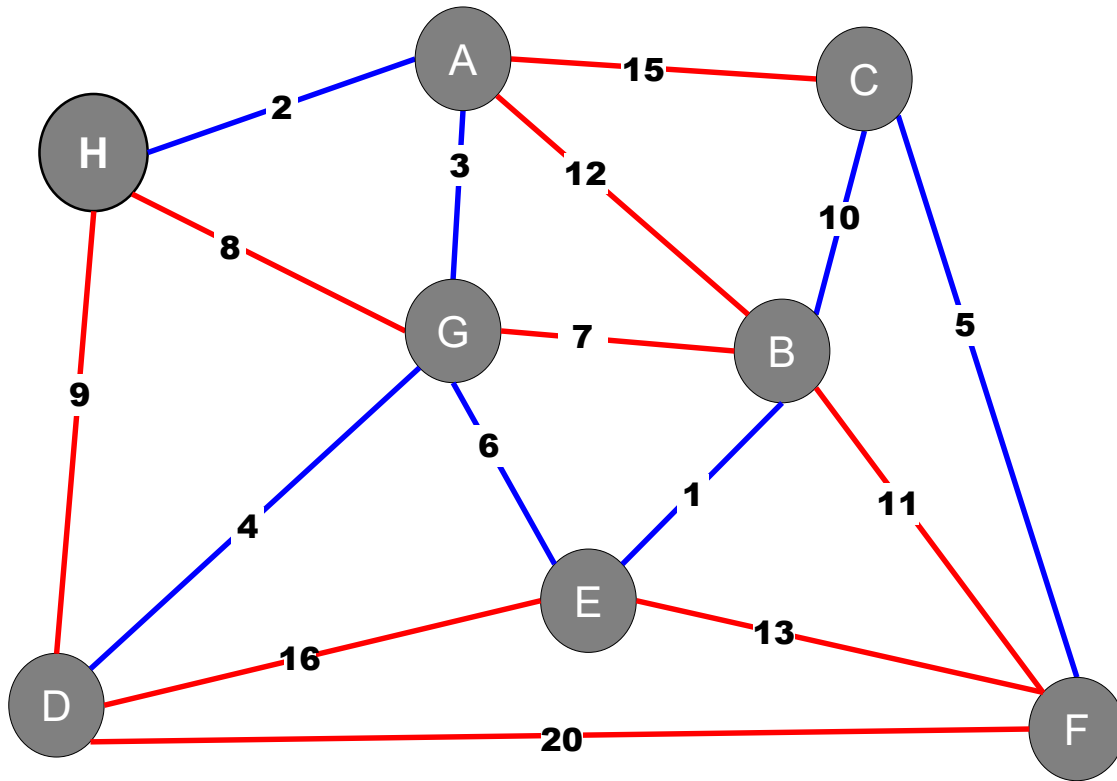


# Kruskal's Example



Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

# Kruskal's Example



Weight	Edge
1	(E,B)
2	(A,H)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,H)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

# Kruskal's Algorithm

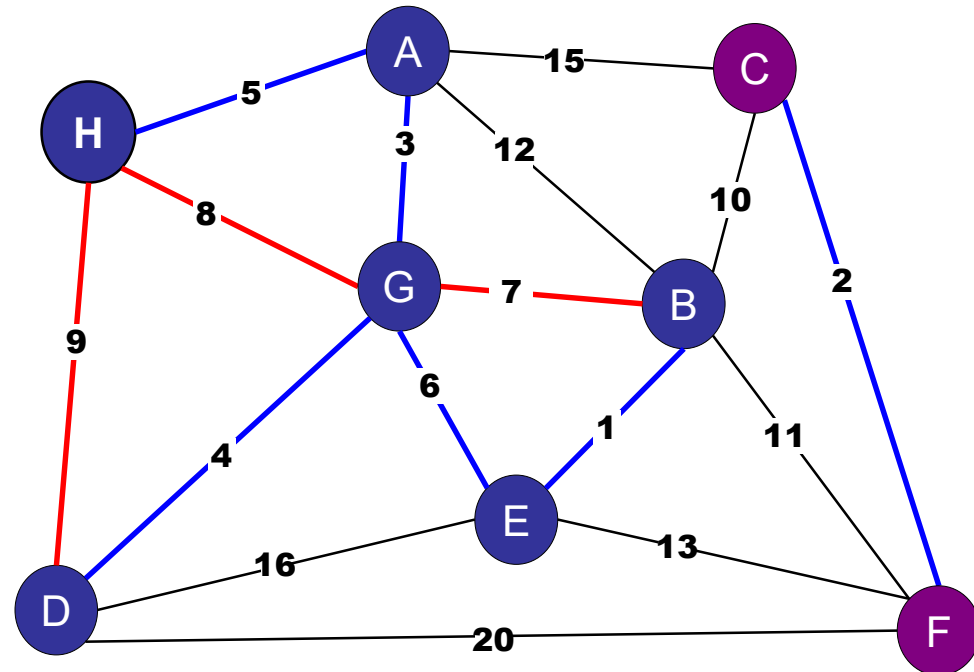
## Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.

Proof:

- Each added edge crosses a cut.
- Each edge is the lightest edge across the cut: all other lighter edges across the cut have already been considered.



# Generic MST Algorithm

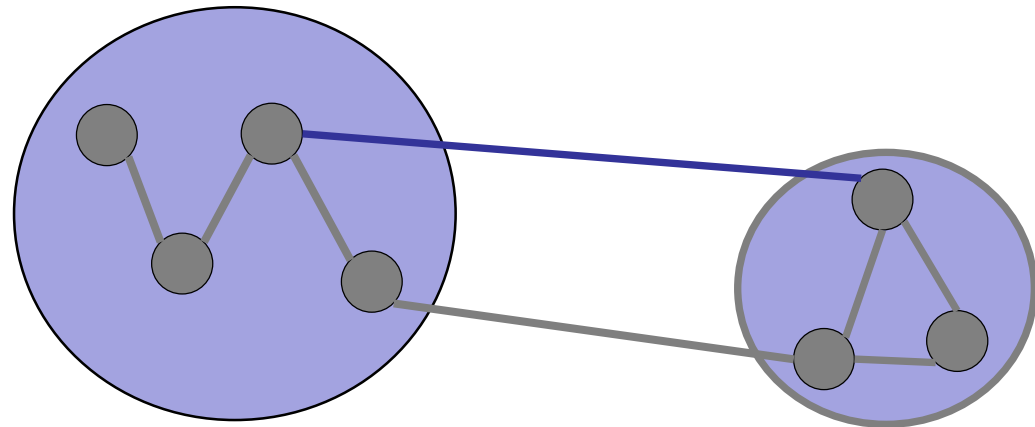
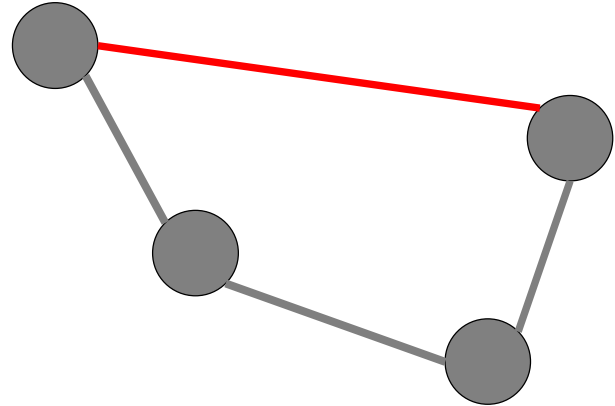
---

## Greedy Algorithm:

Repeat:

**Apply red rule or  
blue rule to an  
arbitrary edge.**

until no more edges  
can be colored.



# Kruskal's Algorithm

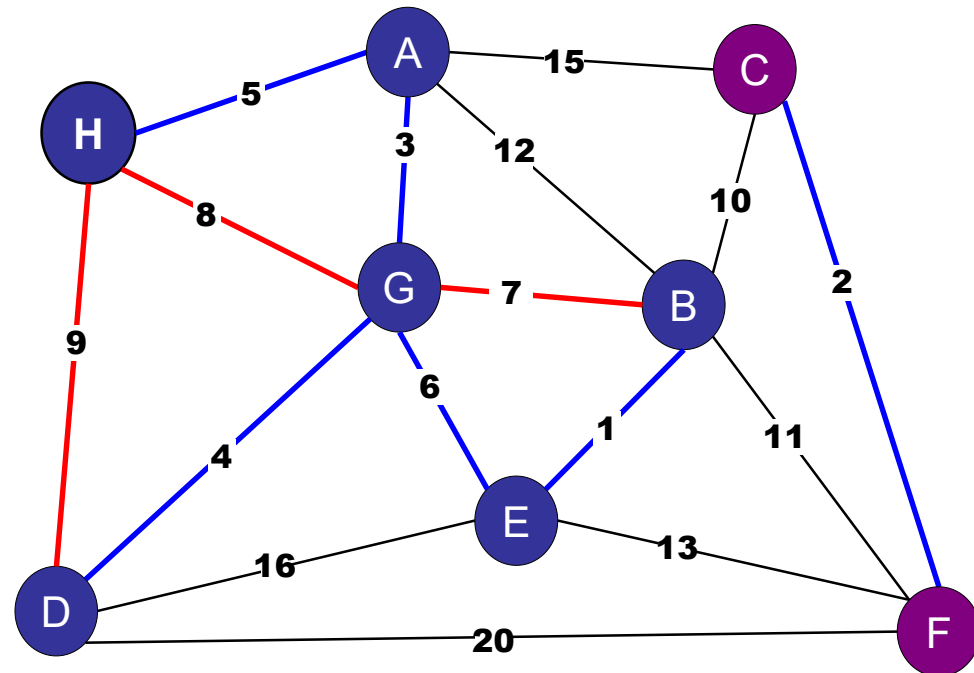
## Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.

Performance:

- Sorting:  $O(E \log E) = O(E \log V)$
- For  $E$  edges:
  - Find:  $O(\alpha)$  or  $O(\log V)$
  - Union:  $O(\alpha)$  or  $O(\log V)$



# Roadmap

---

## Minimum Spanning Trees

- The MST Problem
- Basic Properties of an MST
- Generic MST Algorithm
- Prim's Algorithm
- Kruskal's Algorithm
- **Boruvka's Algorithm**
- Variations

# MST Algorithms

---

## Classic:

- Prim's Algorithm
- Kruskal's Algorithm

## Modern requirements:

- Parallelizable
- Faster in “good” graphs (e.g., planar graphs)
- Flexible

# Boruvka's Algorithm

---

Origin: 1926

- Otakar Boruvka
- Improve the electrical network of Moravia

Based on generic algorithm:

- Repeat: add all “obvious” blue edges.
- Very simple, very flexible.



# Generic MST Algorithm

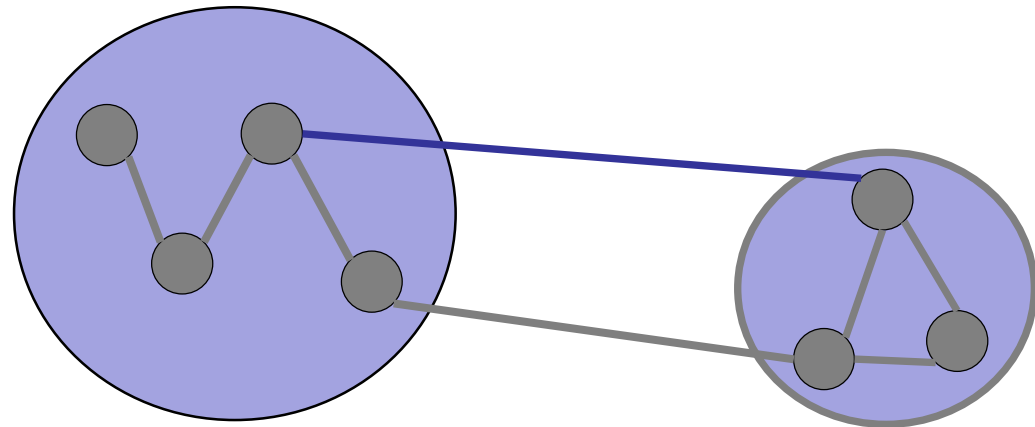
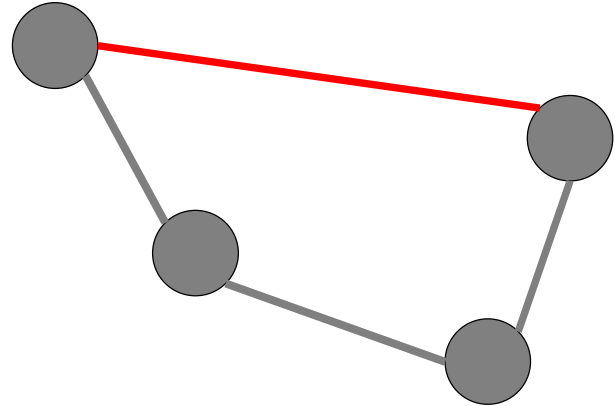
---

## Greedy Algorithm:

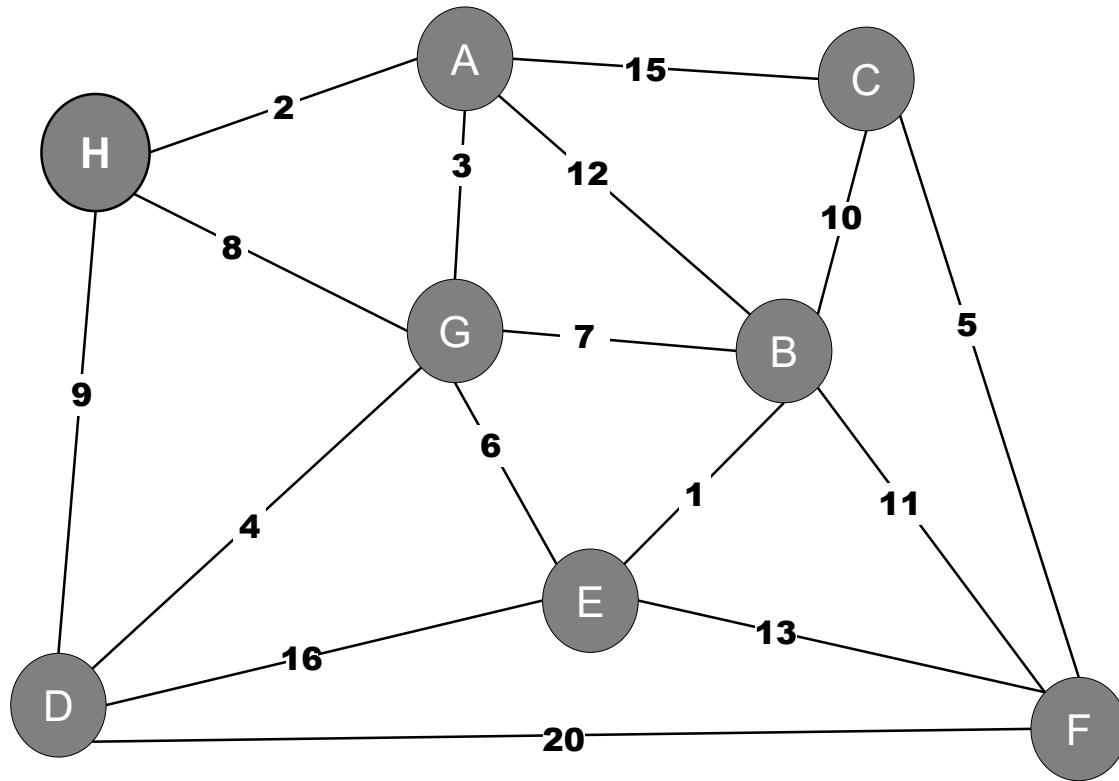
Repeat:

**Apply red rule or  
blue rule to an  
arbitrary edge.**

until no more edges  
can be colored.



# Boruvka's Example

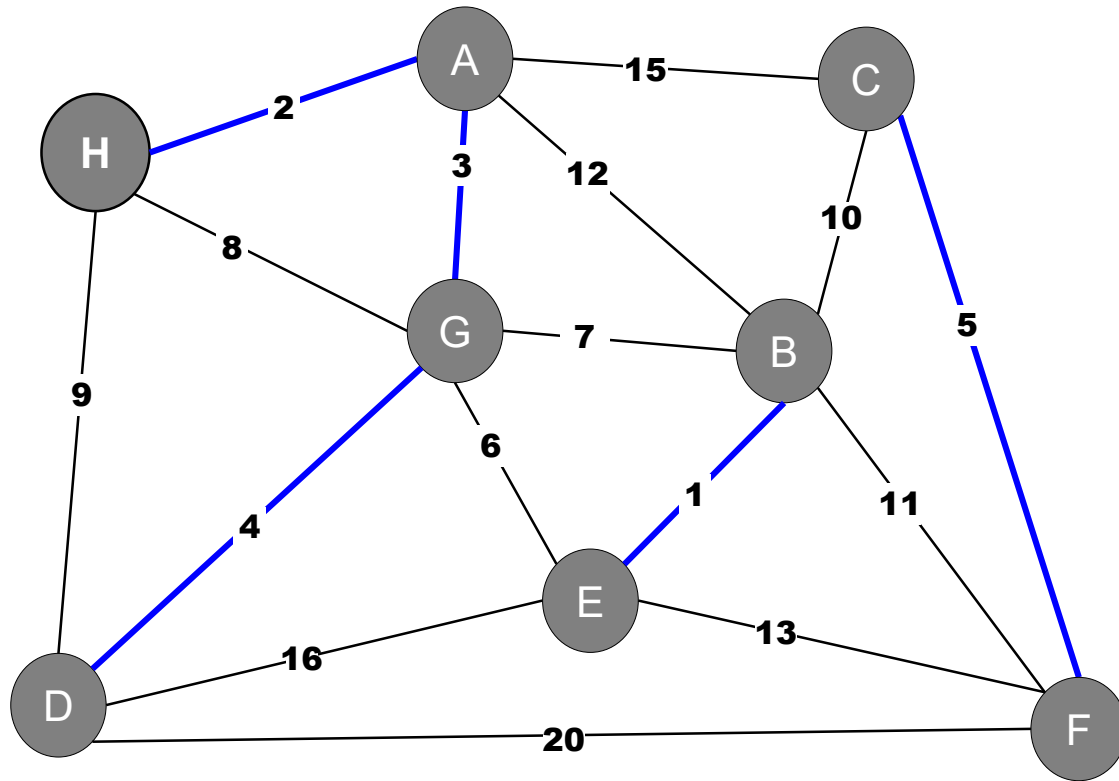


Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Which edges are “obviously” in the MST?

All the min outgoing edges! (Property 4b)

# Boruvka's Example

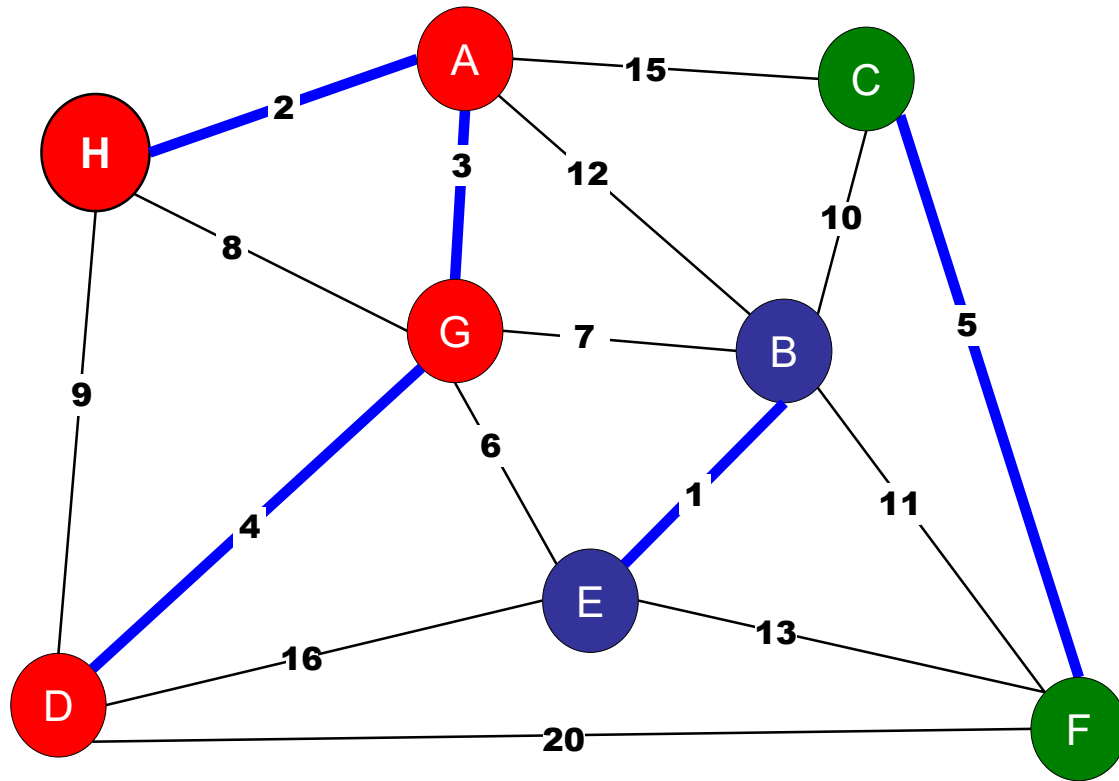


Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

For every node: add minimum adjacent edge.

Add at least  $n/2$  edges.

# Boruvka's Example

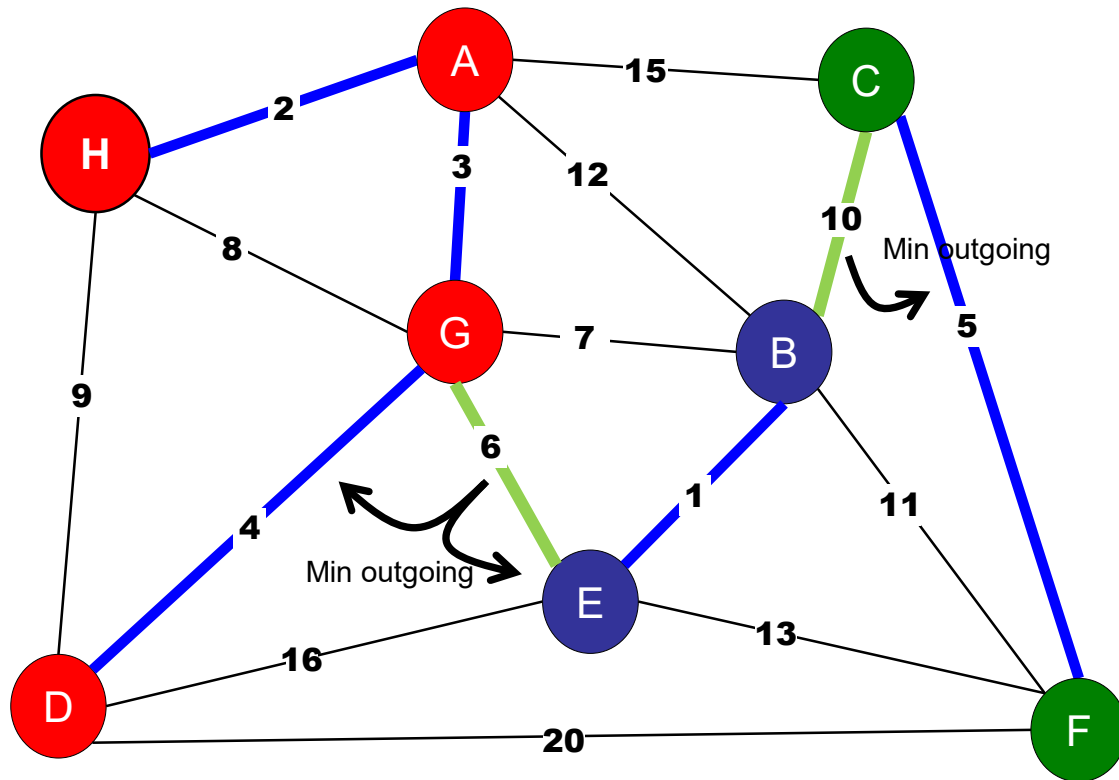


Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

Look at connected components...

At most  $n/2$  connected components.

# Boruvka's Example



Repeat: for every connected components, add minimum outgoing edge.

Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)



Weight	Edge
1	(E,B)
2	(C,F)
3	(A,G)
4	(D,G)
5	(C,F)
6	(E,G)
7	(B,G)
8	(G,H)
9	(D,G)
10	(B,C)
11	(B,F)
12	(A,B)
13	(E,F)
15	(A,C)
16	(D,E)
20	(D,F)

# Boruvka's Algorithm

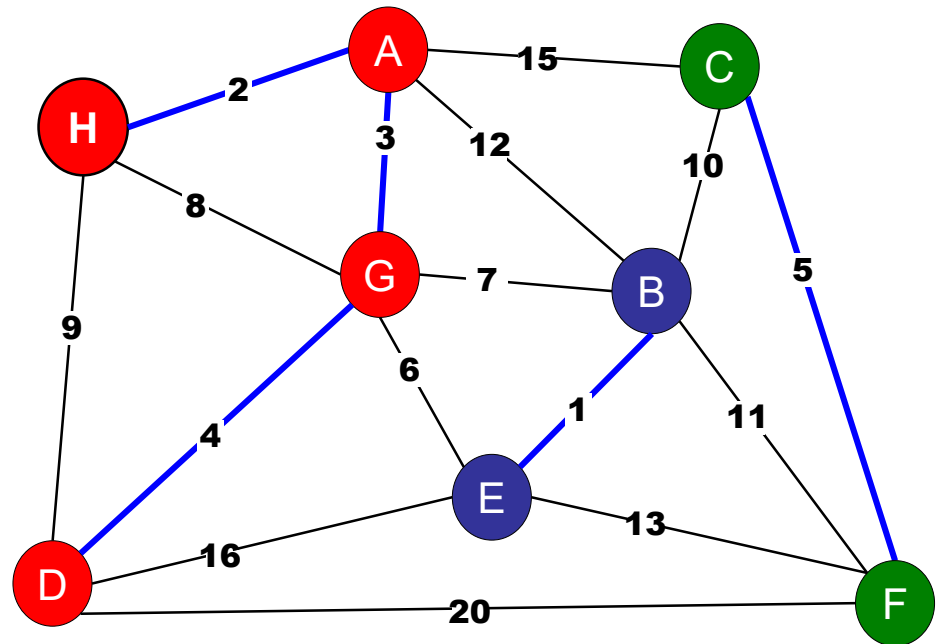
## Boruvka's Algorithm

Initially:

- Create  $n$  connected components, one for each node in the graph.

One “Boruvka” Step:

- For each connected component, search for the minimum weight outgoing edge.
- Add selected edges.
- Merge connected components.



# Boruvka's Algorithm

---

## Boruvka's Algorithm

Initially:

- Create **n** connected components, one for each node in the graph.

For each node: store a component identifier.



One "Boruvka" Step:

- For each connected component, search for the minimum weight outgoing edge.
- Add selected edges.
- Merge connected components.



# Boruvka's Algorithm

## Boruvka's Algorithm

Initially:

- Create **n** connected components, one for each node in the graph.

One "Boruvka" Step:

- For each connected component, search for the minimum weight outgoing edge.
- Add selected edges.
- Merge connected components.

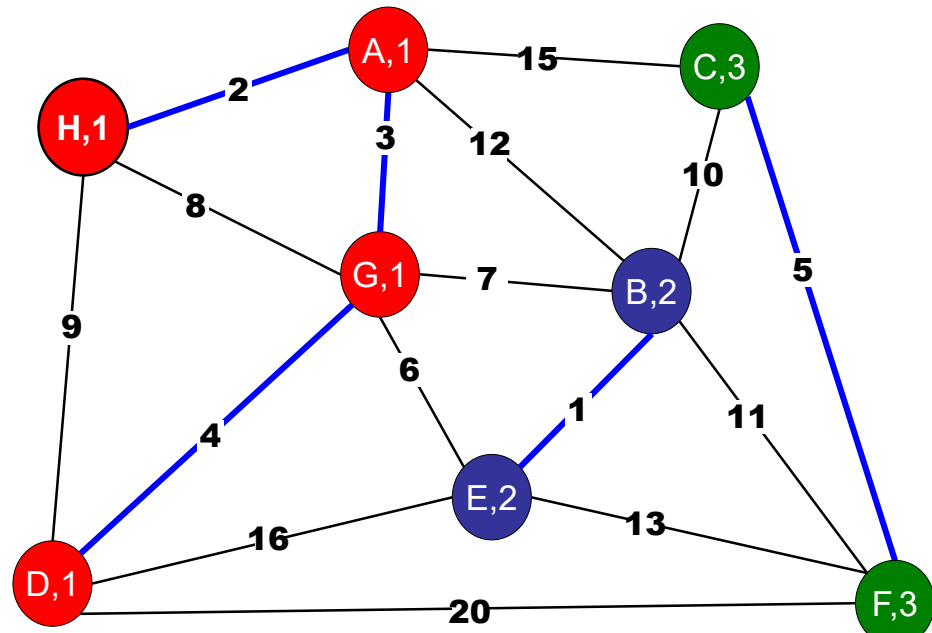
Component	1	2	3
Min cost edge	(G,E), 6	(G,E), 6	(B,C), 10
To be merged	1 and 2	1 and 2	2 and 3

For each node: store a component identifier.

DFS or BFS:

Check if edge connects two components.

Remember minimum cost edge connected to each component.



# Boruvka's Algorithm

## Boruvka's Algorithm

Initially:

- Create **n** connected components, one for each node in the graph.

One "Boruvka" Step:

- For each connected component, search for the minimum weight outgoing edge.
- Add selected edges.
- Merge connected components.

**For each node:** store a component identifier.

**DFS or BFS:**

Check if edge connects two components.

Remember minimum cost edge connected to each component.

**Scan every node:**

Compute new component ids.

Update component ids.

Mark added edges.

Component	1	2	3
Min cost edge	(G,E), 6	(G,E), 6	(B,C), 10
To be merged	1 and 2	1 and 2	2 and 3
New ID:	1	1	1

# Boruvka's Algorithm

---

## Boruvka's Algorithm

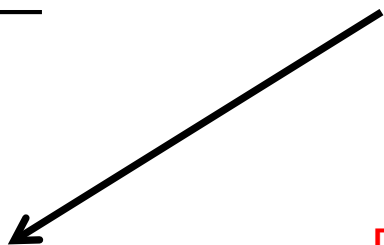
Initially:

- Create  $n$  connected components, one for each node in the graph.

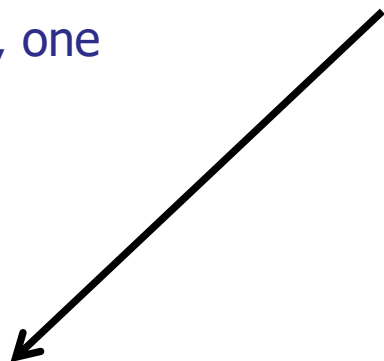
One "Boruvka" Step:  $O(V+E)$

- For each connected component, search for the minimum weight outgoing edge.
- Add selected edges.
- Merge connected components.

For each node:  $O(V)$   
store a component identifier.

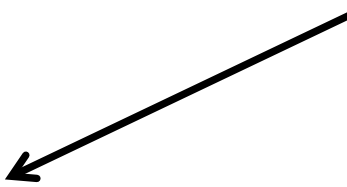


DFS or BFS:  $O(V + E)$   
Check if edge connects two components.



Remember minimum cost edge connected to each component.

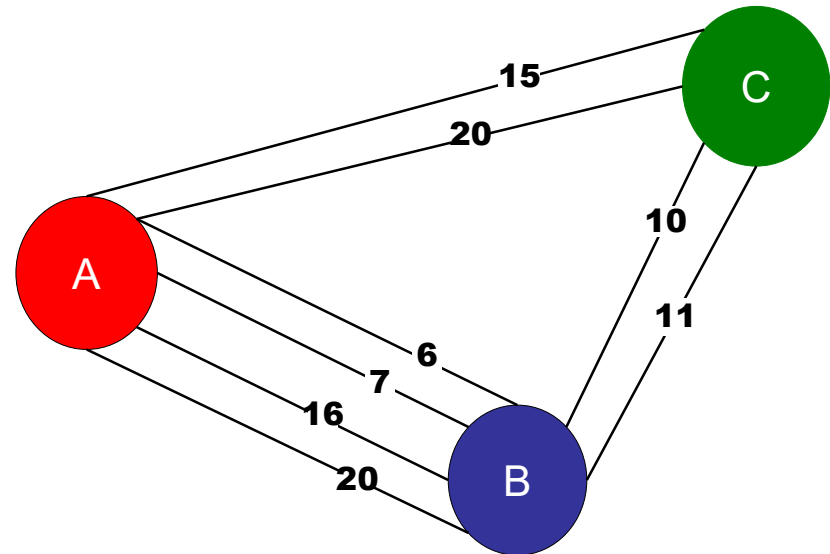
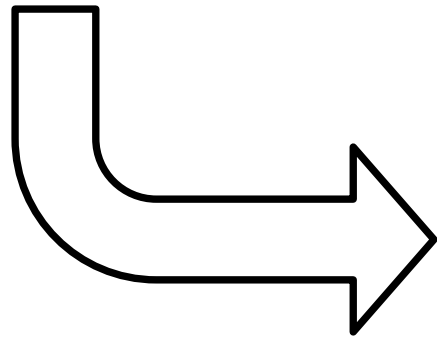
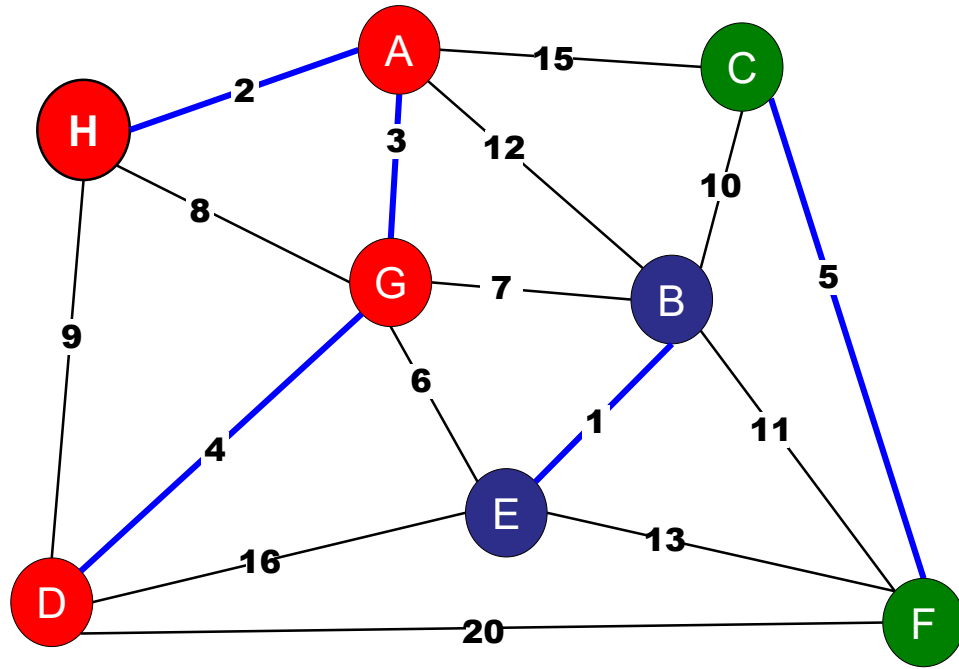
Scan every node:  $O(V)$   
Computer new component ids.



Update component ids.

Mark added edges.

# Boruvka's Example: Contraction



# Boruvka's Algorithm

---

## Boruvka's Algorithm

Initially:

- Create  $n$  connected components, one for each node in the graph.

In each “Boruvka” Step:  $O(V+E)$

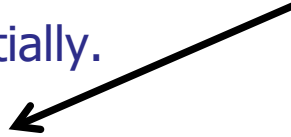
- Assume  $k$  components, initially.
- At least  $k/2$  edges added.

Count edges:

Each component adds one edge.

Some choose same edge.

Each edge is chosen by at most two different components.



# Boruvka's Algorithm

---

## Boruvka's Algorithm

Initially:

- Create  $n$  connected components, one for each node in the graph.

In each “Boruvka” Step:  $O(V+E)$

- Assume  $k$  components, initially.
- At least  $k/2$  edges added.
- At least  $k/2$  components merge.

Merging:

Each edge merges two components

# Boruvka's Algorithm

---

## Boruvka's Algorithm

Initially:

- Create  $n$  connected components, one for each node in the graph.

In each “Boruvka” Step:  $O(V+E)$

- Assume  $k$  components, initially.
- At least  $k/2$  edges added.
- At least  $k/2$  components merge.
- At end, at most  $k/2$  components remain.

# Boruvka's Algorithm

---

## Boruvka's Algorithm

Initially:

$n$  components

At each step:

$k$  components  $\rightarrow k/2$  components.

Termination:

1 component

Conclusion:

At most  $O(\log V)$  Boruvka steps.

Total time:

$O((E+V)\log V) = O(E \log V)$



# Boruvka's Algorithm

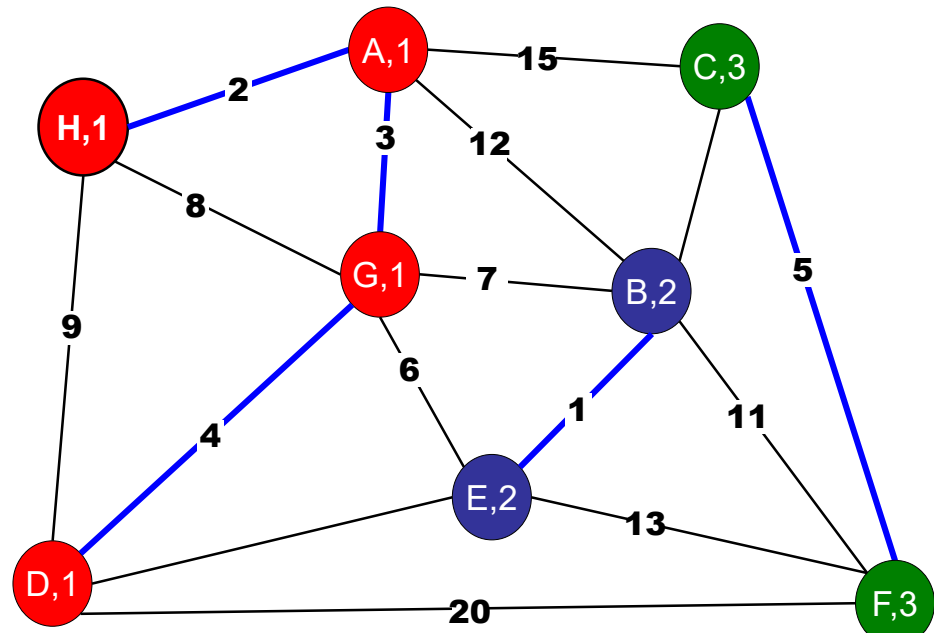
## Boruvka's Algorithm

Initially:

- Create  $n$  connected components, one for each node in the graph.

One "Boruvka" Step:  $O(V+E)$

- For each connected component, search for the minimum weight outgoing edge.
- Add selected edges.
- Merge connected components.



# Websites to have fun with MST

- [https://www-m9.ma.tum.de/graph-algorithms/mst-prim/index\\_en.html](https://www-m9.ma.tum.de/graph-algorithms/mst-prim/index_en.html)



## Prim's Algorithm



[Introduction](#) [Create a graph](#) [Run the algorithm](#) [Description of the algorithm](#) [Exercise 1](#) [Exercise 2](#) [More](#)

Which graph do you want to execute the algorithm on?

Start with an example graph:

Select

Modify it to your desire:

- To create a node, make a double-click in the drawing area.
- To create an edge, first click on the output node and then click on the destination node.
- The edge weight can be changed by double clicking on the edge.
- Right-clicking deletes edges and nodes.

Download the modified graph:

[Download](#)

Upload an existing graph:

No file chosen

What next?

[+ Legend](#)

[SVG Download](#)

# Websites to have fun with MST

- <https://www.cs.usfca.edu/~galles/visualization/Prim.html>

## Prim Minimum Cost Spanning Treeh

Start Vertex:

☒ Small Graph

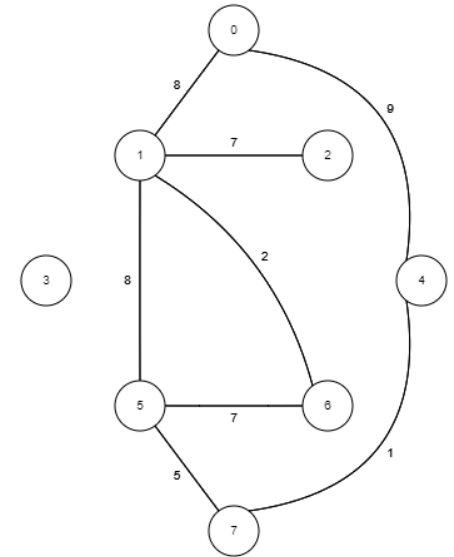
☒ Logical Representation

☐ Large Graph

☐ Adjacency List Representation

☐ Adjacency Matrix Representation

Vertex	Known	Cost	Path
0			
1			
2			
3			
4			
5			
6			
7			



- Kruskal's

- <https://www.cs.usfca.edu/~galles/visualization/Kruskal.html>

# Roadmap

---

So far:

## Minimum Spanning Trees

- Prim's Algorithm
- Kruskal's Algorithm
- Boruvka's Algorithm

# Minimum Spanning Tree Summary

---

Classic greedy algorithms:  $O(E \log V)$

- Prim's (Priority Queue)
- Kruskal's (Union-Find)
- Boruvka's

Best known:  $O(m \alpha(m, n))$

- Chazelle (2000)

Holy grail and major open problem:  $O(m)$

# Minimum Spanning Tree Summary

---

Classic greedy algorithms:  $O(E \log V)$

- Prim's (Priority Queue)
- Kruskal's (Union-Find)
- Boruvka's

Best known:  $O(m \alpha(m, n))$

- Chazelle (2000)

Holy grail and major open problem:  $O(m)$

- Randomized: Karger-Klein-Tarjan (1995)
- Verification: Dixon-Rauch-Tarjan (1992)

# Roadmap

---

Today: Minimum Spanning Trees

- Prim's Algorithm
- Kruskal's Algorithm
- Boruvka's Algorithm

Variations:

- Constant weight edges
- Bounded integer edge weights
- Euclidean
- Directed graphs
- Maximum Spanning Tree
- Steiner Tree

# MST Variants

---

What if all the edges have the same weight?

- Depth-First-Search or Breadth-First-Search



# MST Variants

---

What if all the edges have the same weight?

- Depth-First-Search or Breadth-First-Search
- An MST contains exactly  $(V-1)$  edges.
- Every spanning tree contains  $(V-1)$  edges!
- Thus, any spanning tree you find with DFS/BFS is a minimum spanning tree.

# Kruskal's Variants

---

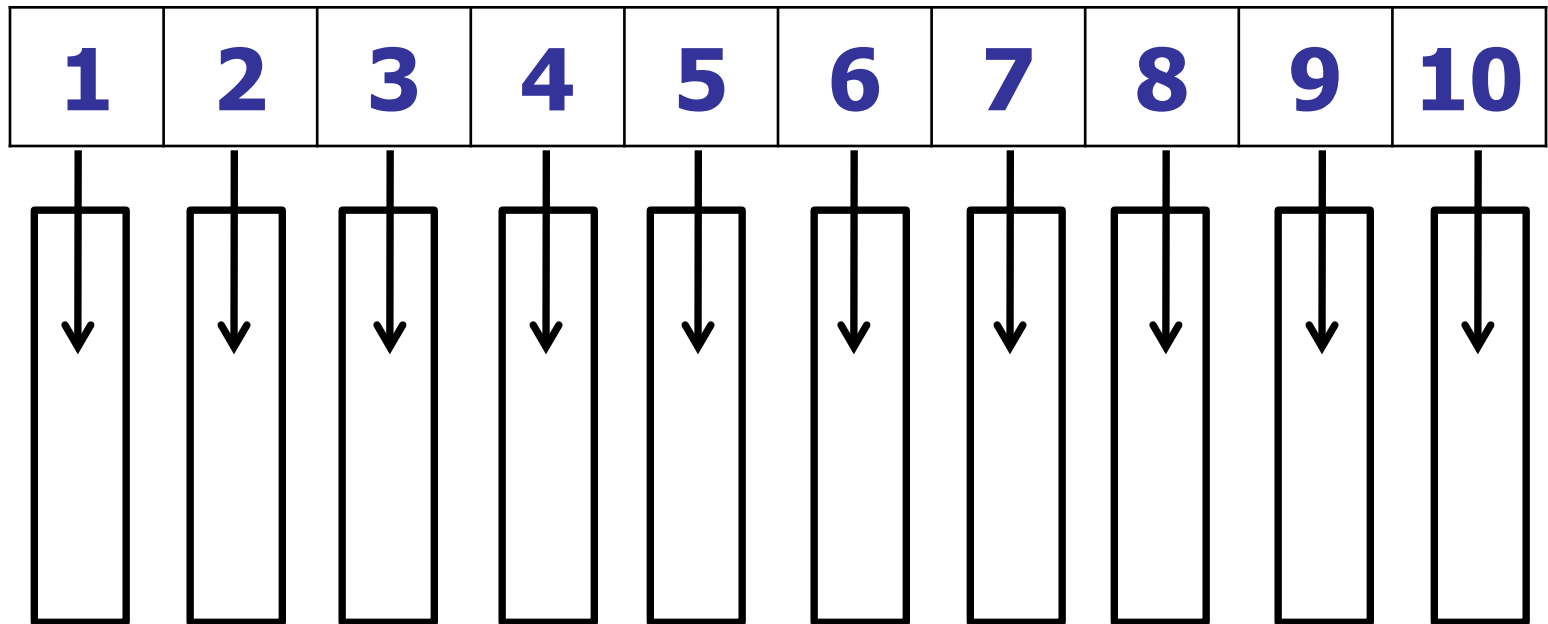
What if all the edges have weights from  $\{1..10\}$ ?

# Kruskal's Variants

---

What if all the edges have weights from  $\{1..10\}$ ?

Idea: Use an array of size 10



slot  $A[j]$  holds a linked list of edges of weight  $j$

# Kruskal's Variants

---

What if all the edges have weights from  $\{1..10\}$ ?

Idea: Use an array of size 10

- Putting edges in array of linked lists:  $O(E)$
- Iterating over all edges in ascending order:  $O(E)$
- Checking whether to add an edge:  $O(\alpha(V))$
- Union two components:  $O(\alpha(V))$

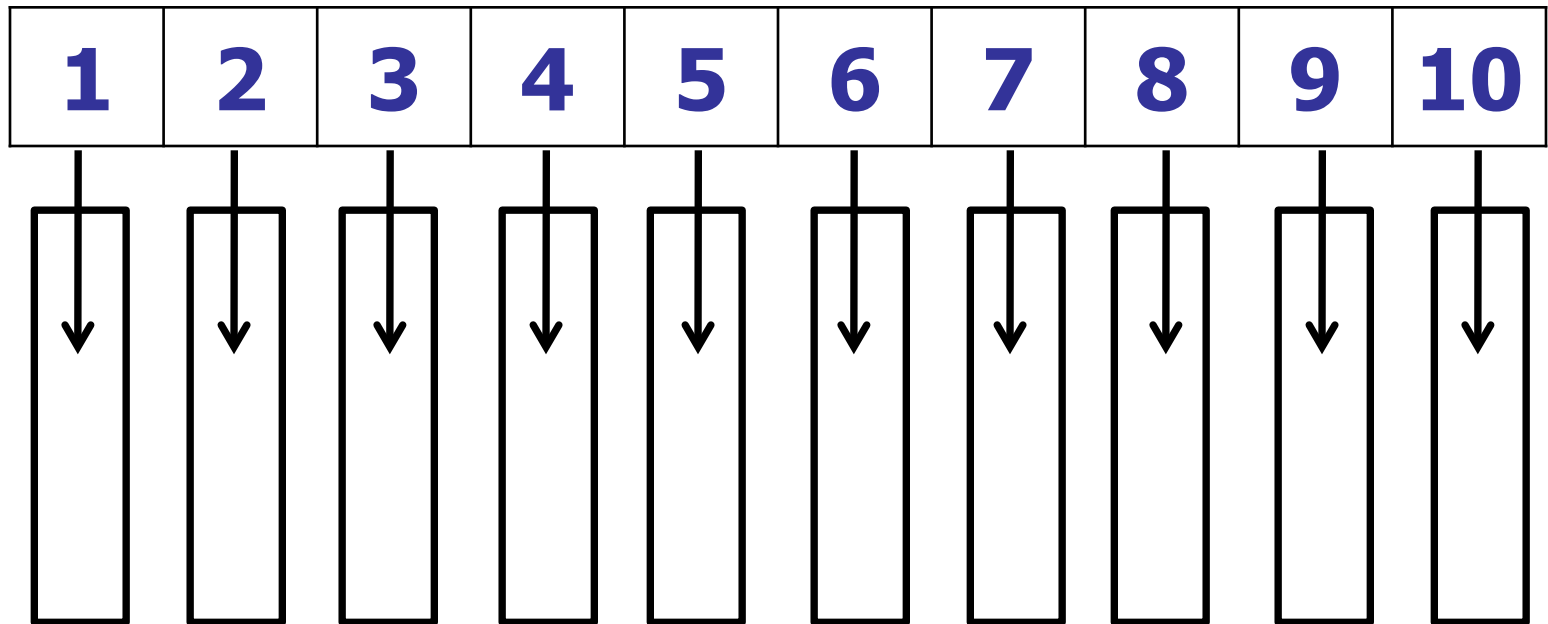
Total:  $O(\alpha(V)E)$

# Prim's Variants

---

What if all the edges have weights from  $\{1..10\}$ ?

Idea: Use an array of size 10 as a Priority Queue



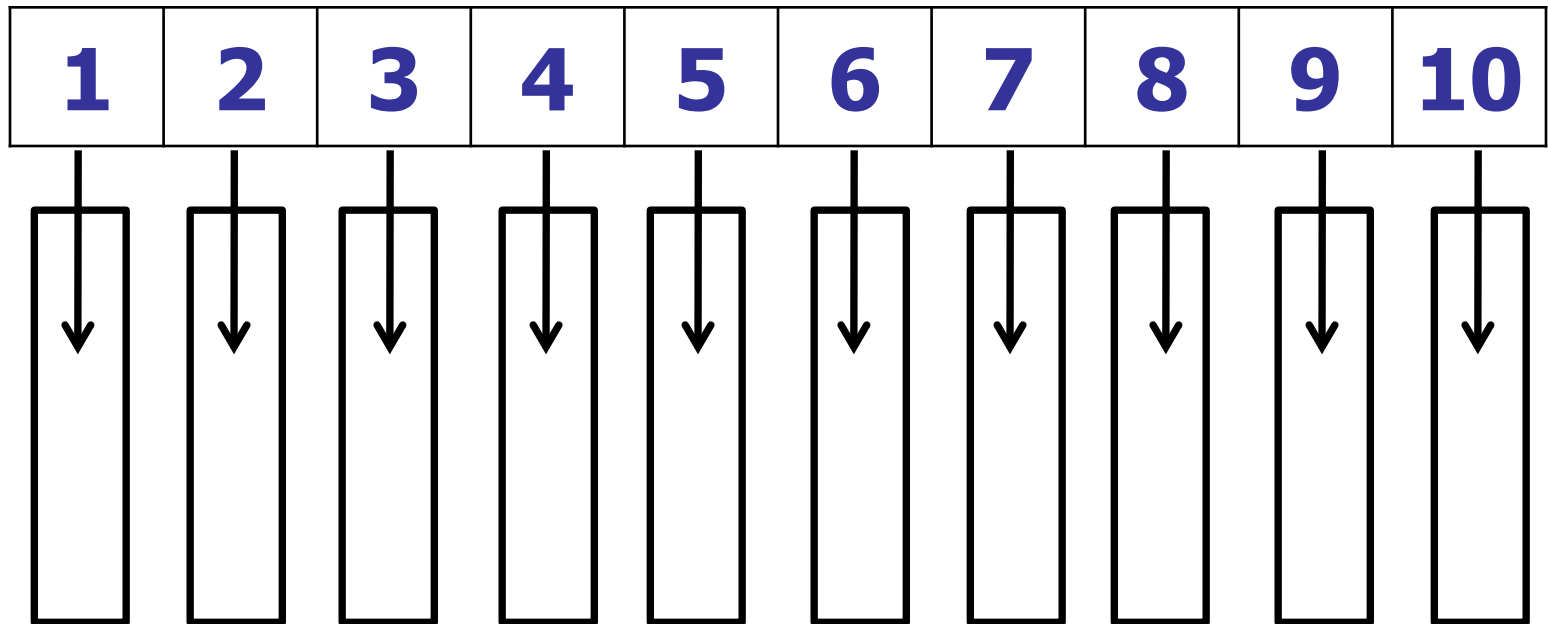
slot  $A[j]$  holds a linked list of **nodes** of weight  $j$

# Prim's Variants

---

What if all the edges have weights from  $\{1..10\}$ ?

Idea: Use an array of size 10 as a Priority Queue



decreaseKey: move node to new linked list

# Prim's Variants

---

What if all the edges have weights from  $\{1..10\}$ ?

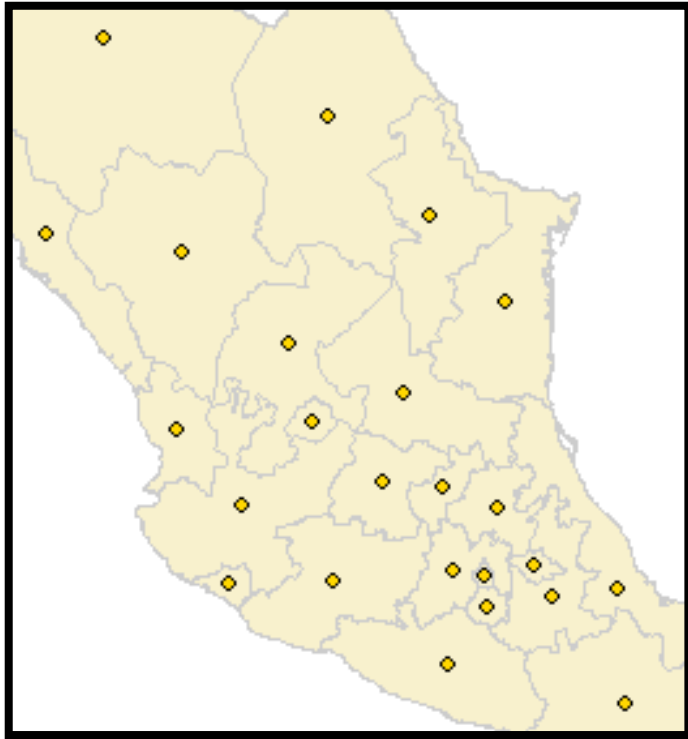
Idea: Use an array of size 10

- Inserting/Removing nodes from PQ:  $O(V)$
- decreaseKey:  $O(E)$

Total:  $O(V + E) = O(E)$

# Euclidean Minimal Spanning Tree

- Given point set  $P$ ,  $EMST(P)$  is the tree that spans  $P$  and the sum of lengths of all edges is minimal





# EMST: Naïve solution

---

- Compute a complete graph of  $P$  with each edge equal to the Euclidean distance
  - $O(n^2)$
- Then run MST
  - $O(n^2 \log n)$
- Any better solution?
  - $O(n \log n)$  by Delaunay Triangulation
  - Come to my computational Geometry Class

# Roadmap

---

Today: Minimum Spanning Trees

- Prim's Algorithm
- Kruskal's Algorithm
- Boruvka's Algorithm

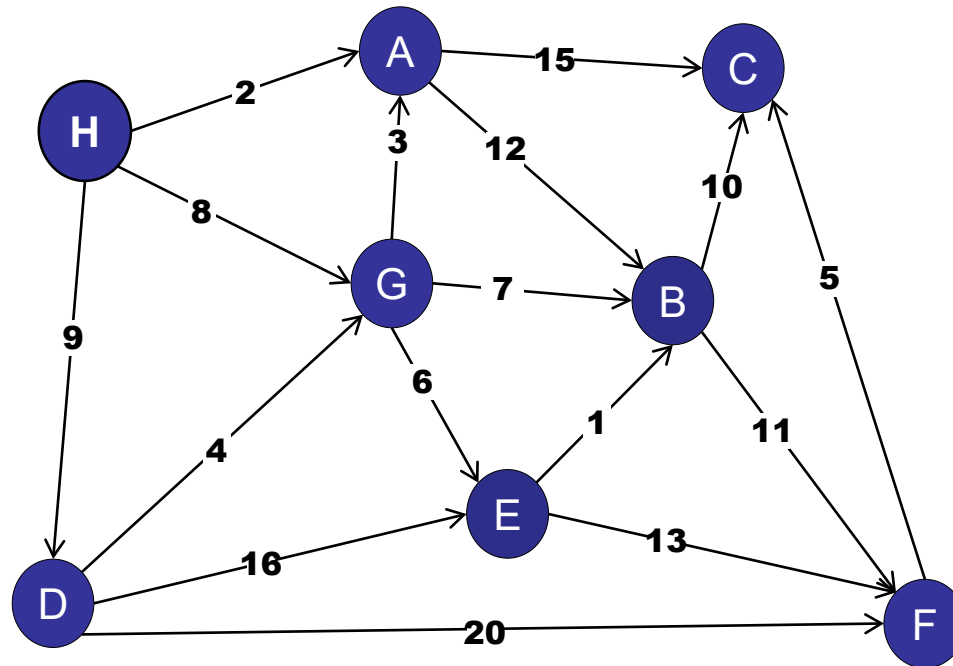
Variations:

- Constant weight edges
- Bounded integer edge weights
- Euclidean
- Directed graphs
- Maximum Spanning Tree
- Steiner Tree

# Directed Minimum Spanning Tree

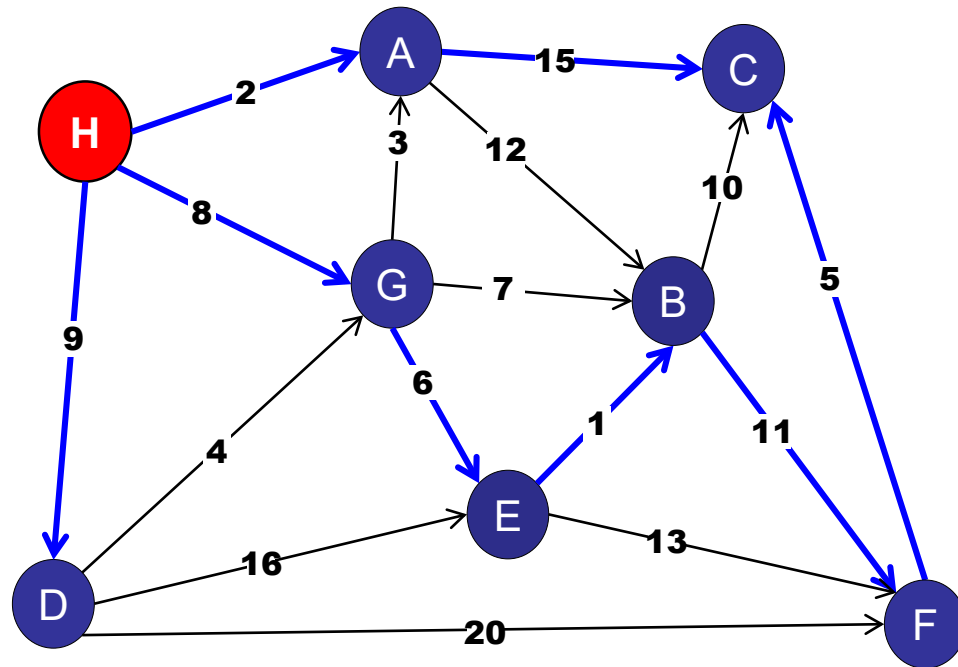
---

What if the edges are directed?



# Directed Minimum Spanning Tree

A rooted spanning tree:



Every node is reachable on a path from the root.

No cycles.

# Directed Minimum Spanning Tree

---

## Harder problem:

- Cut property does not hold.
- Cycle property does not hold.
- Generic MST algorithm does not work.

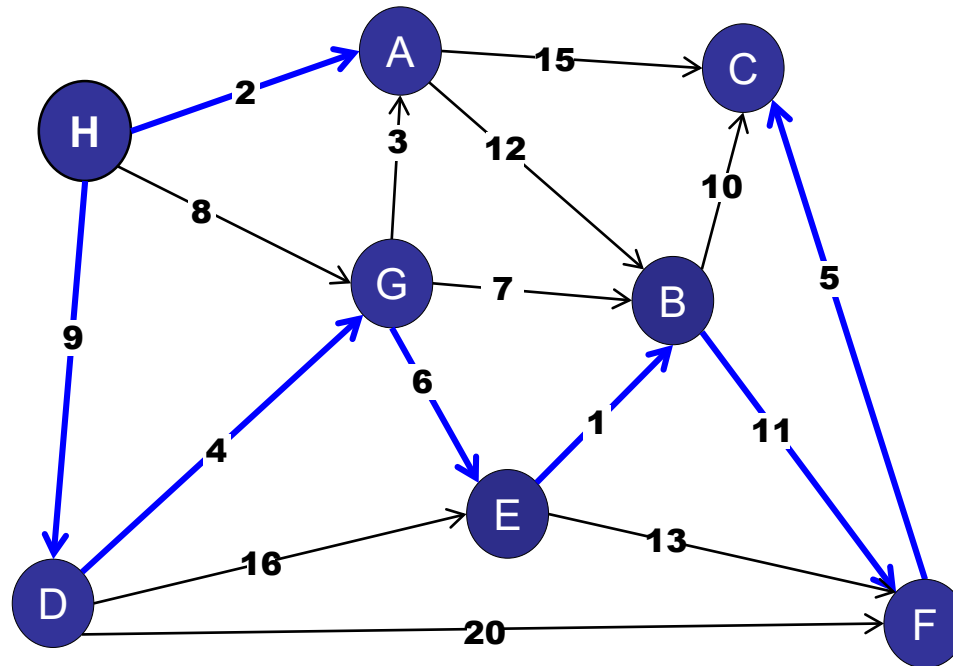
Prim's, Kruskal's, Boruvka's do not work.

See CS3230 / CS5234 for more details...

# Directed Minimum Spanning Tree

For a directed acyclic graph with one "root":

For every node except the root: add minimum weight incoming edge.



# Directed Minimum Spanning Tree

---

For a directed acyclic graph with one “root”:

For every node except the root: add minimum weight incoming edge.

## Observations:

- No cycles (since acyclic graph).
- Each edge is chosen only once.

Tree:

$V$  nodes

$V - 1$  edges

No cycles



# Directed Minimum Spanning Tree

---

For a directed acyclic graph with one "root":

For every node except the root: add minimum weight incoming edge.

## Observations:

- No cycles (since acyclic graph).
- Each edge is chosen only once.

Tree:

$V$  nodes  
 $V - 1$  edges  
No cycles



- Every node has to have at least one incoming edge in the MST, so this is the minimum spanning tree.



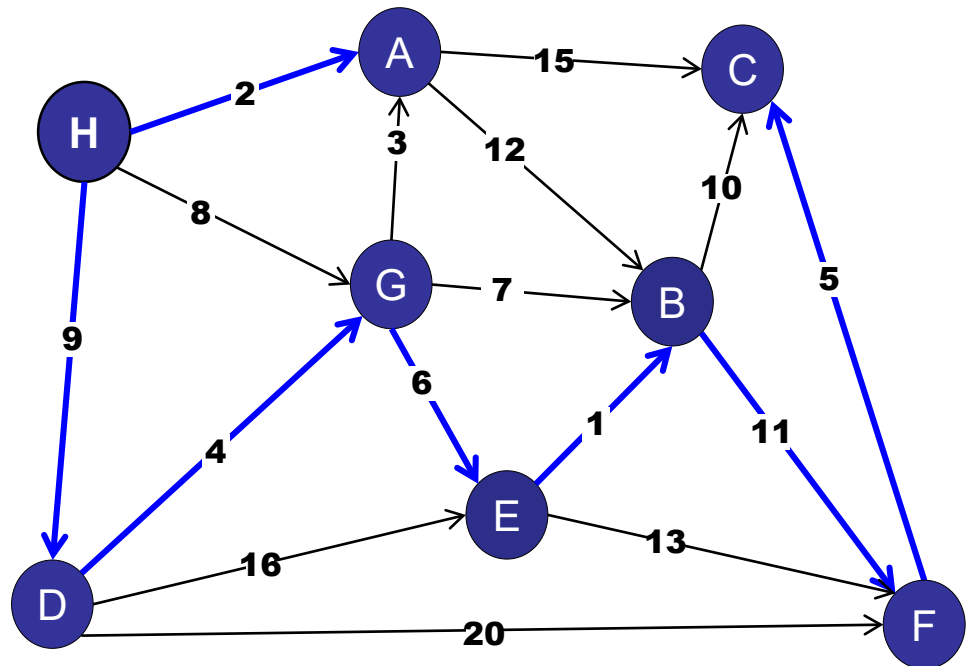
# Directed Minimum Spanning Tree

For a directed acyclic graph with one "root":

For every node except the root: add minimum weight incoming edge.

Conclusion: Minimum Spanning Tree

$O(E)$  time



# Roadmap

---

Today: Minimum Spanning Trees

- Prim's Algorithm
- Kruskal's Algorithm
- Boruvka's Algorithm

Variations:

- Constant weight edges
- Bounded integer edge weights
- Euclidean
- Directed graphs
- Maximum Spanning Tree
- Steiner Tree

# Maximum Spanning Tree

---

A MaxST is a spanning tree of maximum weight.

How do you find a MaxST?

# Maximum Spanning Tree

---

Reweighting a spanning tree:

- What happens if you add a constant  $k$  to the weight of every edge?

# Kruskal's Algorithm

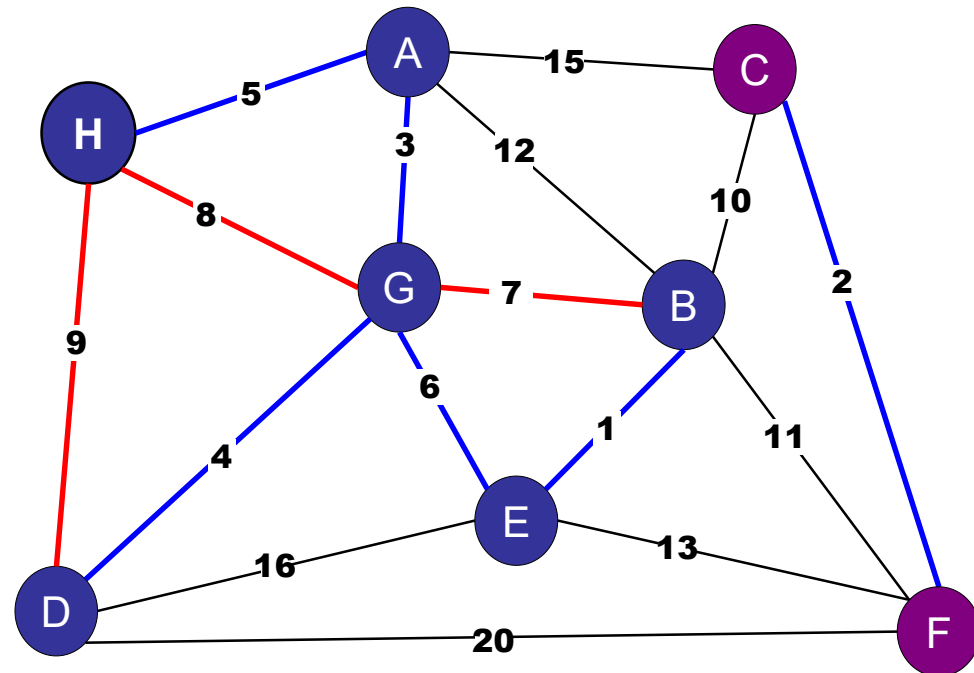
## Kruskal's Algorithm. (Kruskal 1956)

Basic idea:

- Sort edges by weight.
- Consider edges in ascending order:
  - If both endpoints are in the **same** blue tree, then color the edge red.
  - Otherwise, color the edge blue.

What matters?

- Relative edge weights.
- Absolute edge weights have no impact.

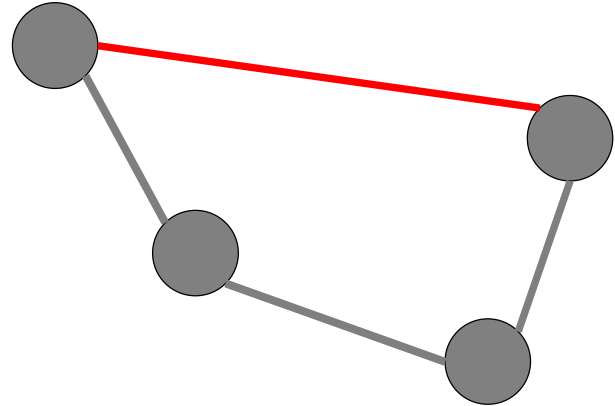


# Generic MST Algorithm

---

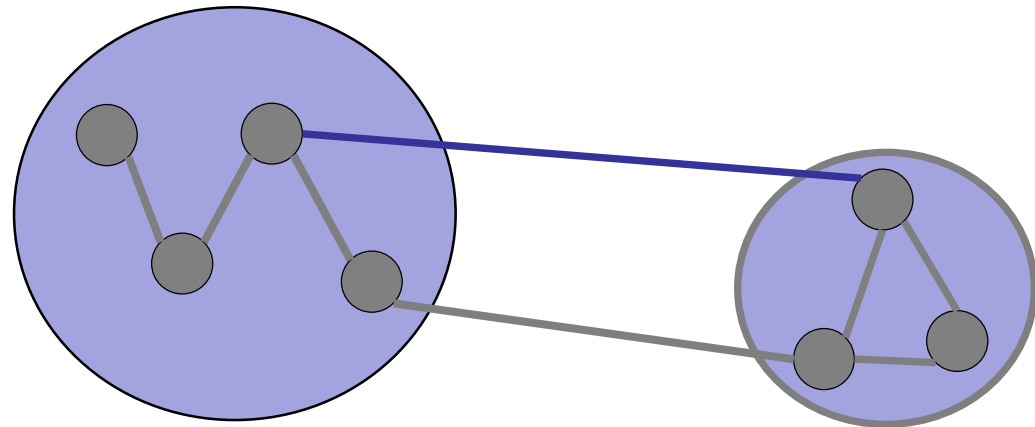
## Red rule:

If  $C$  is a cycle with no red arcs, then color the max-weight edge in  $C$  red.



## Blue rule:

If  $D$  is a cut with no blue arcs, then color the min-weight edge in  $D$  blue.



# Maximum Spanning Tree

---

Reweighting a spanning tree:

- What happens if you add a constant  $k$  to the weight of every edge?

No change!

We can add or subtract weights without effecting the MST.

# Maximum Spanning Tree

---

MST with negative weights?



# Maximum Spanning Tree

---

MST with negative weights?

No problem!

1. Reweight MST by adding a big enough value to each edge so that it is positive.
2. Actually, no need to reweight. Only relative edge weights matter, so negative weights have no bad impact.

# Maximum Spanning Tree

---

A MaxST is a spanning tree of maximum weight.

How do you find a MaxST?

Easy!

1. Multiply each edge weight by -1.
2. Run MST algorithm.
3. MST that is “most negative” is the maximum.