

# CSE 332: Data Structures & Parallelism

## Lecture 24: Minimum Spanning Trees

Ruth Anderson  
Autumn 2025

# Administrative

- EX09 – On Fork Join, Due TONIGHT, Fri Nov 21
- EX10 – Concurrency, Due **Mon** Nov 24
- EX11 – MSTs, programming, coming soon!
- Resources!
  - **Conceptual Office Hours:** 11:30 Tues (Connor) and 11:30 Wed (Samarth) both in CSE1 006. A space to ask about **course content and topics only** *as opposed to direct help with exercises*.
  - [1-on-1 Meeting Requests](#) - Request a meeting with a staff member if you cannot make it to regularly scheduled office hours, or feel like you have an issue that requires a more in depth discussion.

# Trees as graphs

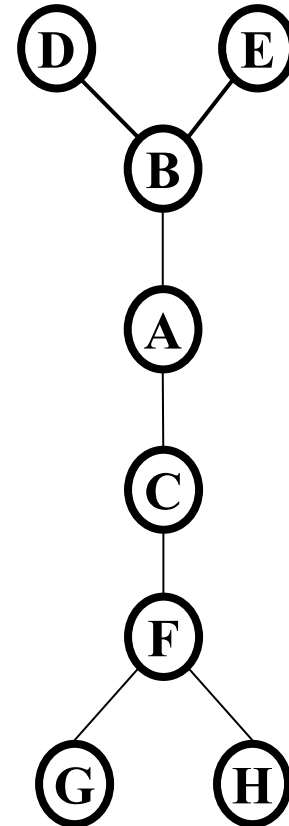
When talking about graphs,  
we say a **tree** is a graph that is:

- undirected
- acyclic
- connected

Q: How many edges will  
there be in a tree?  
(in terms of  $V$ )

$$V - 1$$

Example:

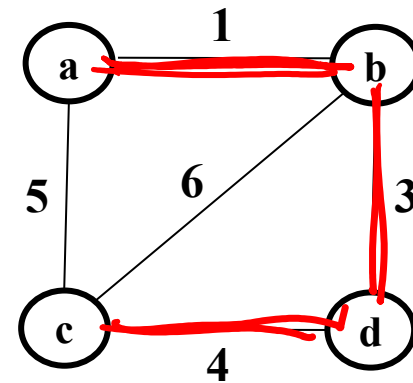


# Minimum Spanning Trees

Given an undirected graph  $G=(V, E)$ , find a graph  $G'=(V, E')$  such that:

- $E'$  is a subset of  $E$
- $G'$  is connected
- $G'$  has no cycles
- $|E'| = |V| - 1$
- $\sum_{(u,v) \in E'} c_{uv}$  is minimal

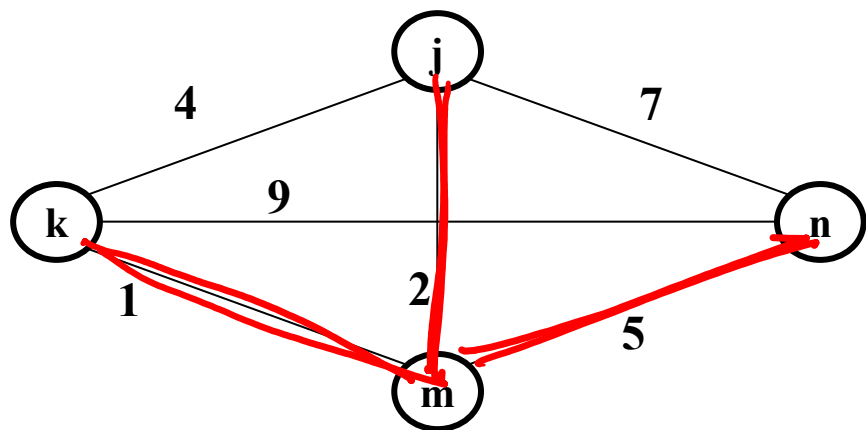
$G'$  is a **minimum spanning tree**.



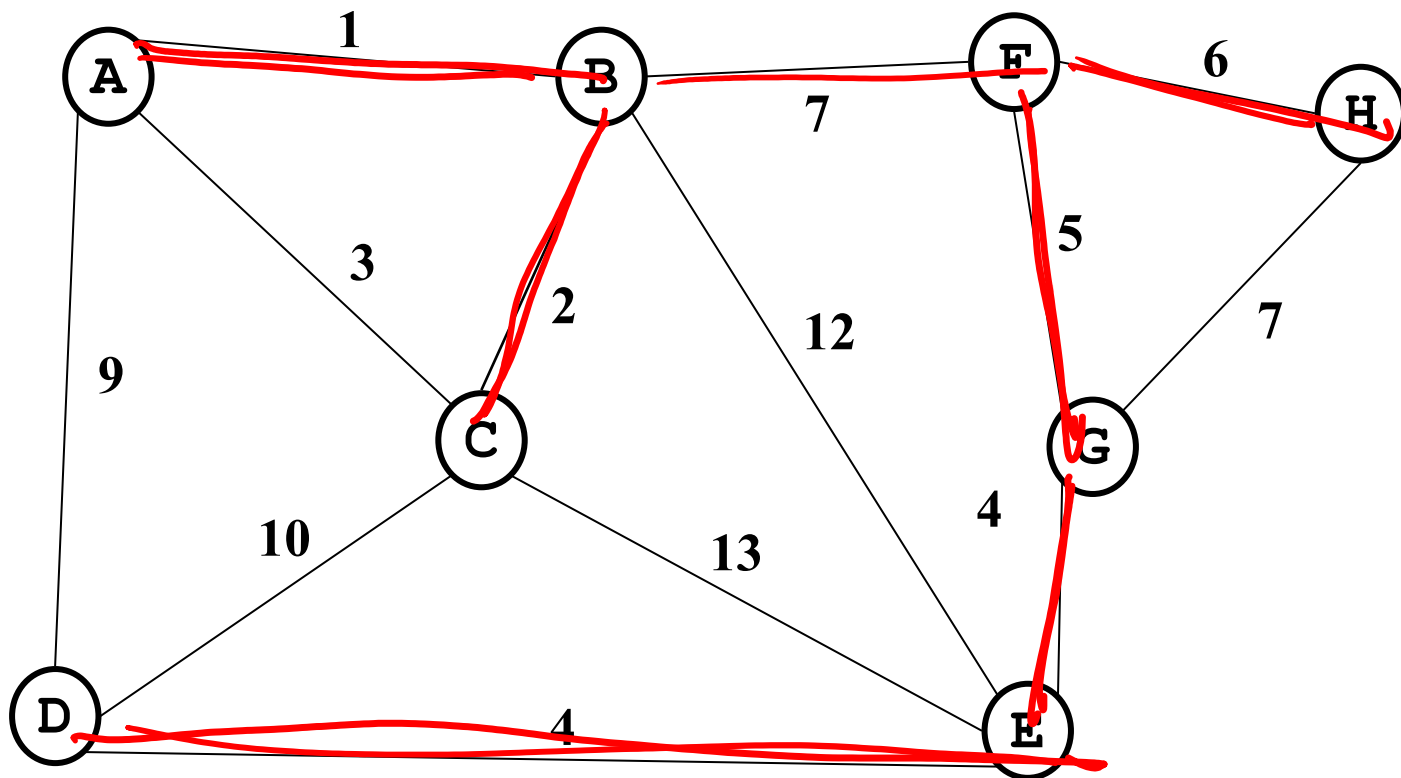
## Applications:

- Example: Electrical wiring for a house or clock wires on a chip
- Example: A road network if you cared about asphalt cost rather than travel time

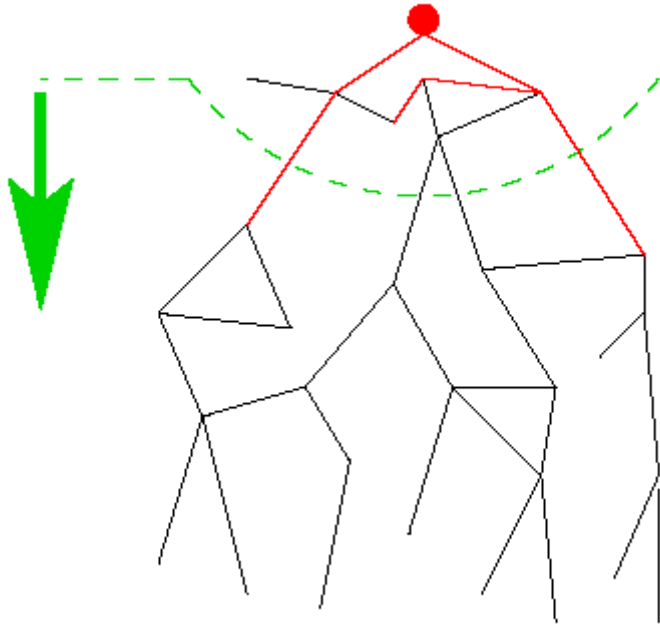
*Find a MST*



total cost: 29

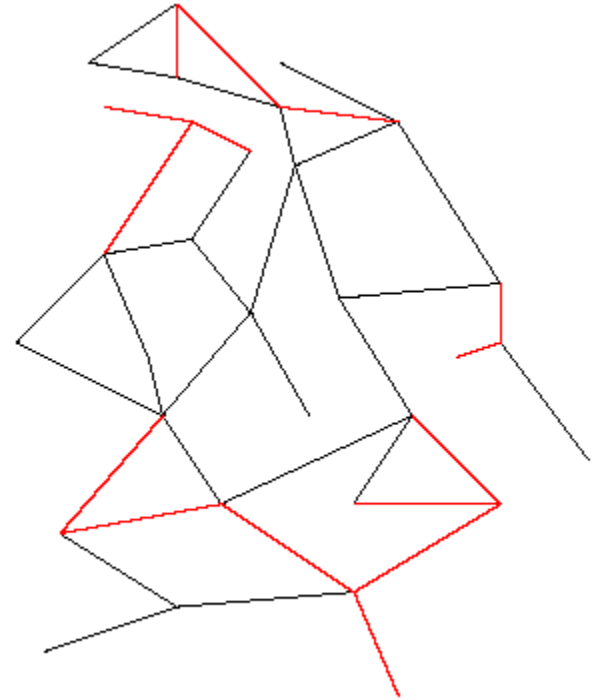


# *Two Different Approaches*



**Prim's Algorithm**

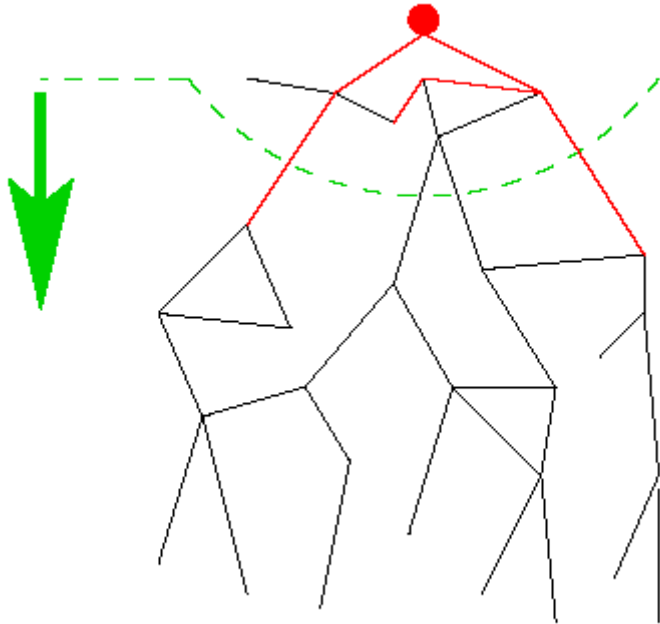
**Almost identical to Dijkstra's**



**Kruskals's Algorithm**

**Completely different!**

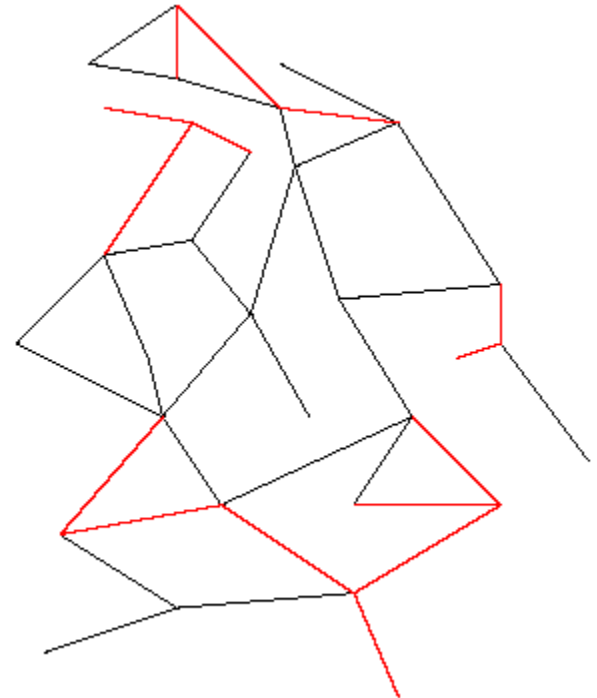
# Two Different Approaches



## Prim's Algorithm

Almost identical to Dijkstra's

**One node, grow greedily**



## Kruskals's Algorithm

Completely different!

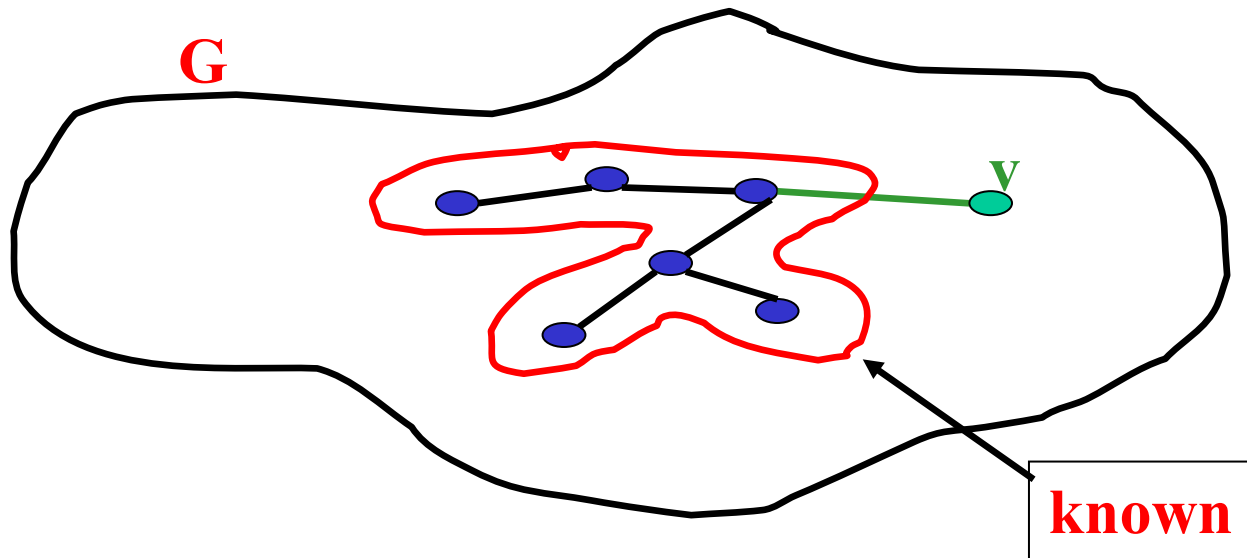
**Forest of MSTs,  
*Union* them together.  
(Need a new data structure for this)**

# Prim's algorithm

**Idea:** Grow a tree by picking a vertex from the unknown set that has the smallest cost. Here cost = cost of the edge that connects that vertex to the known set. *Pick the vertex with the smallest cost that connects “known” to “unknown.”*

**A node-based greedy algorithm**

**Builds MST by greedily adding nodes**





# *Prim's Algorithm vs. Dijkstra's*

Recall:

**Dijkstra** picked the unknown vertex with smallest cost where  
cost = *distance to the source*.

**Prim's** pick the unknown vertex with smallest cost where  
cost = *distance from this vertex to the known set* (in other words,  
the cost of the smallest edge connecting this vertex to the known  
set)

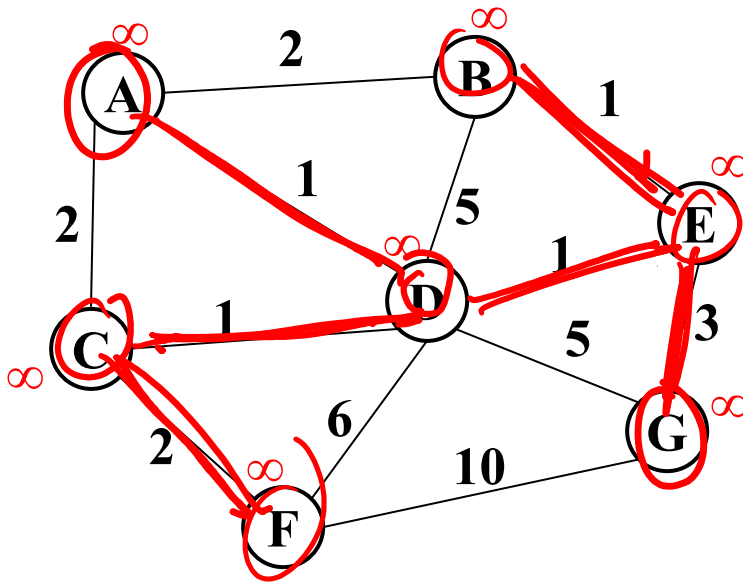
- Otherwise identical
- Compare to slides in Dijkstra lecture!

# *Prim's Algorithm for MST*

1. For each node  $v$ , set  $v.cost = \infty$  and  $v.known = false$
2. Choose any node  $v$ . (this is like your "start" vertex in Dijkstra)
  - a) Mark  $v$  as known
  - b) For each edge  $(v, u)$  with weight  $w$ :  
set  $u.cost = w$  and  $u.prev = v$
3. While there are unknown nodes in the graph
  - a) Select the unknown node  $v$  with lowest **cost**
  - b) Mark  $v$  as known and add  $(v, v.prev)$  to output (the MST)
  - c) For each edge  $(v, u)$  with weight  $w$ , where  $u$  is unknown:  

```
if ( $w < u.cost$ ) {  
     $u.cost = w$ ;  
     $u.prev = v$ ;  
}
```

# Example: Find MST using Prim's



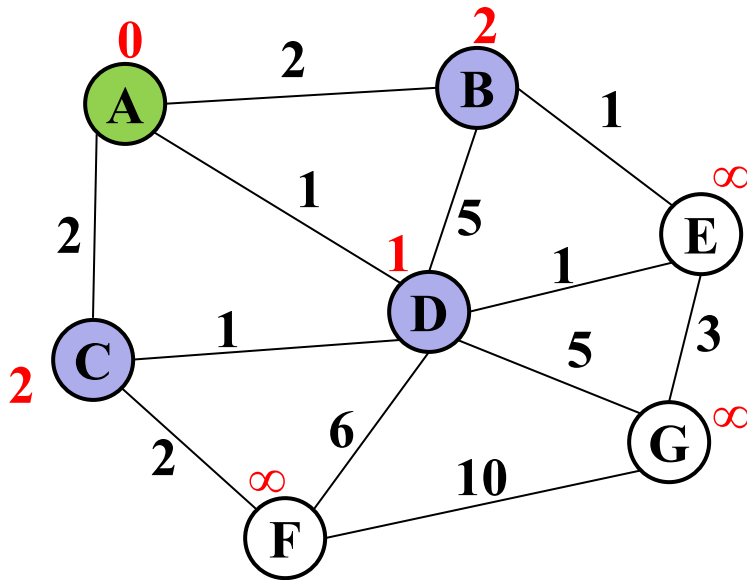
Order added to known set:

A, D, C, E, B, F, G

vertex	known?	cost	prev
A	T	0	
B	T	1	E
C	T	1	D
D	T	1	A
E	T	1	D
F	T	2	C
G	T	3	E

weight of edge  
add this edge  
to known set

## Example: Find MST using Prim's

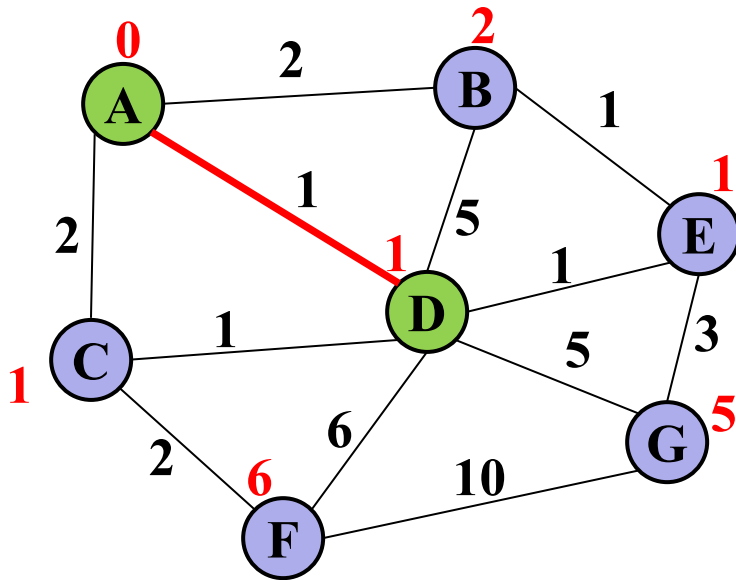


Order added to known set:

A

vertex	known?	cost	prev
A	Y	0	
B		2	A
C		2	A
D		1	A
E		??	
F		??	
G		??	

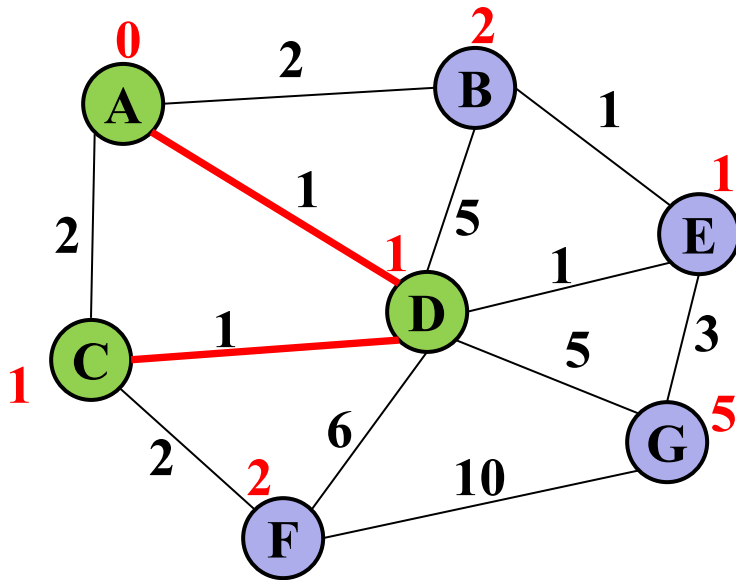
## Example: Find MST using Prim's



Order added to known set:  
A, D

vertex	known?	cost	prev
A	Y	0	
B		2	A
C		1	D
D	Y	1	A
E		1	D
F		6	D
G		5	D

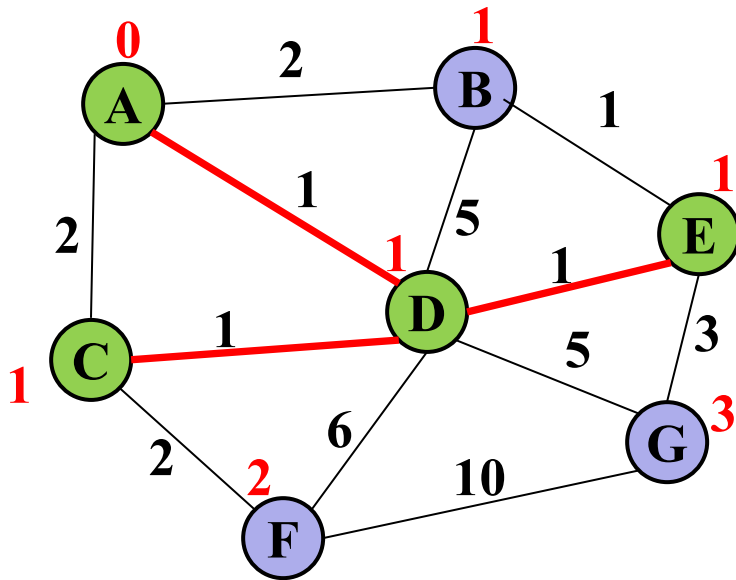
## Example: Find MST using Prim's



Order added to known set:  
A, D, C

vertex	known?	cost	prev
A	Y	0	
B		2	A
C	Y	1	D
D	Y	1	A
E		1	D
F		2	C
G		5	D

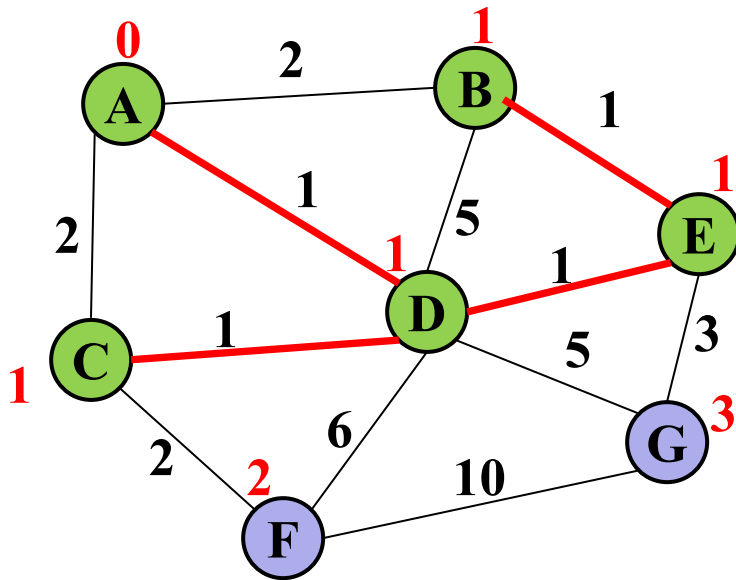
## Example: Find MST using Prim's



Order added to known set:  
A, D, C, E

vertex	known?	cost	prev
A	Y	0	
B		1	E
C	Y	1	D
D	Y	1	A
E	Y	1	D
F		2	C
G		3	E

## Example: Find MST using Prim's

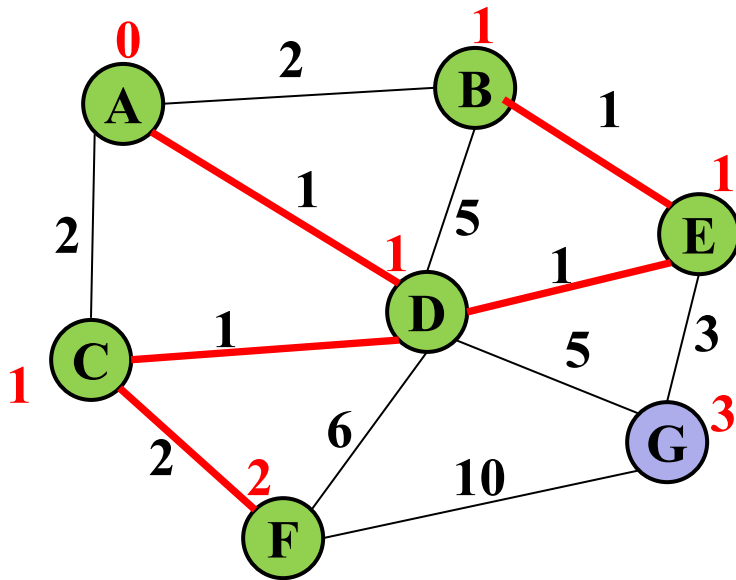


Order added to known set:  
A, D, C, E, B

vertex	known?	cost	prev
A	Y	0	
B	Y	1	E
C	Y	1	D
D	Y	1	A
E	Y	1	D
F		2	C
G		3	E



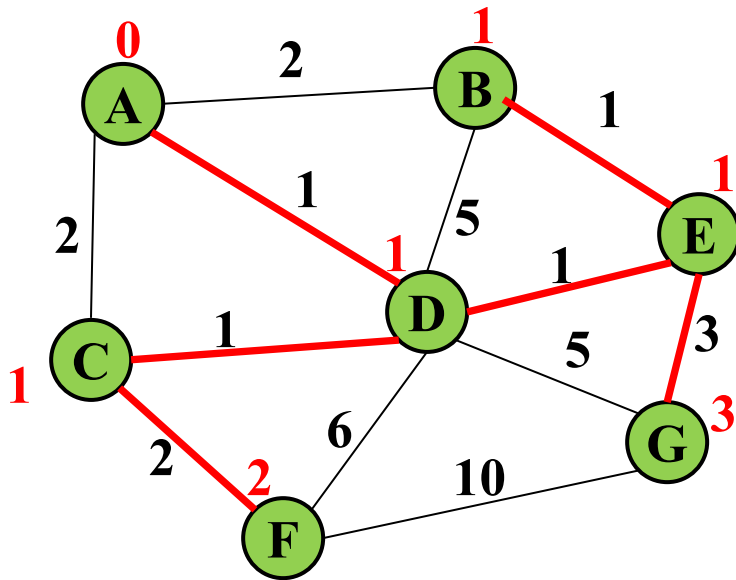
## Example: Find MST using Prim's



Order added to known set:  
A, D, C, E, B, F

vertex	known?	cost	prev
A	Y	0	
B	Y	1	E
C	Y	1	D
D	Y	1	A
E	Y	1	D
F	Y	2	C
G		3	E

## Example: Find MST using Prim's



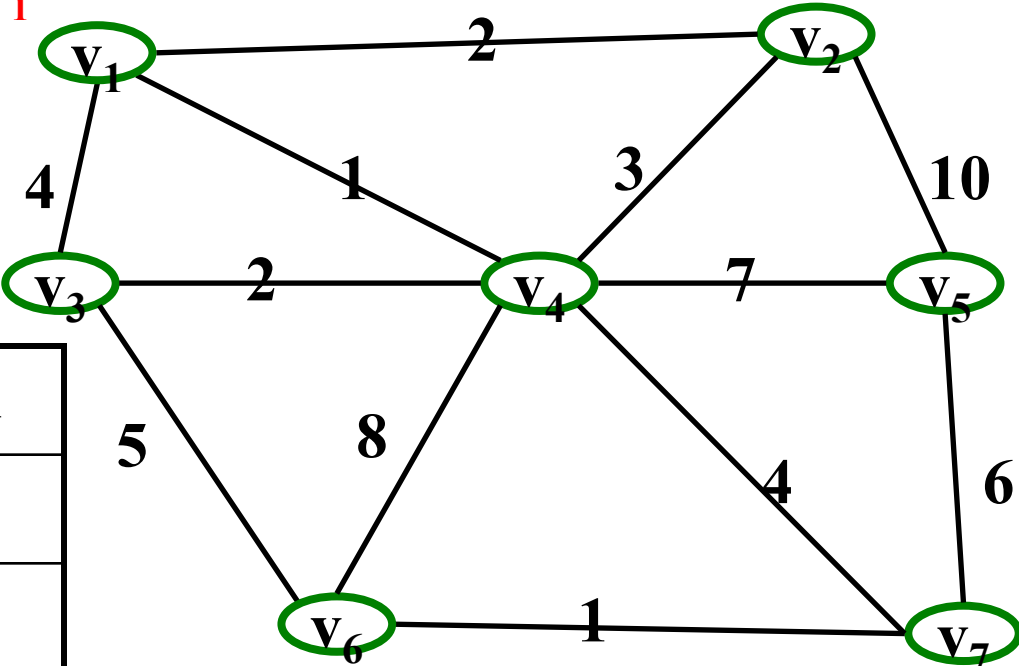
Order added to known set:

A, D, C, E, B, F, G

vertex	known?	cost	prev
A	Y	0	
B	Y	1	E
C	Y	1	D
D	Y	1	A
E	Y	1	D
F	Y	2	C
G	Y	3	E

Find MST using Prim's

V	Kwn	Distance	path
v1			
v2			
v3			
v4			
v5			
v6			
v7			



Order Declared Known:

$V_1$

Total Cost:

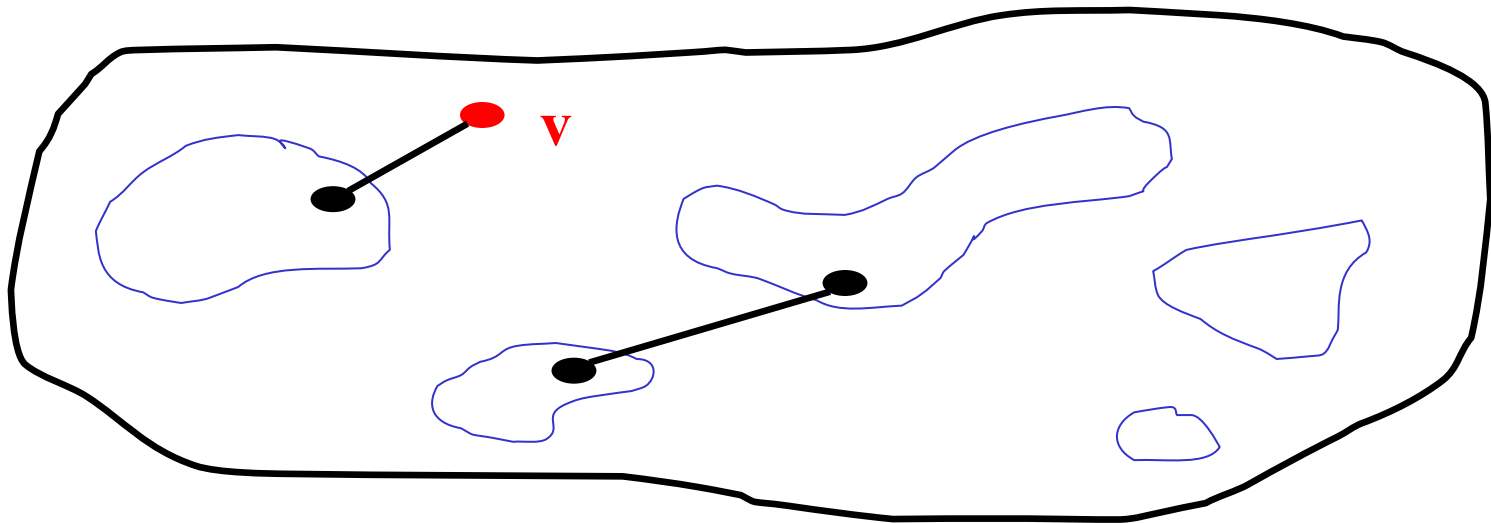
# *Prim's Analysis*

- Correctness
  - Intuitively similar to Dijkstra
- Run-time
  - Same as Dijkstra
  - $O(|E| \log |V|)$  using a priority queue

# Kruskal's MST Algorithm

**Idea:** Grow a **forest** out of edges that do not create a cycle. Pick an edge with the smallest weight.

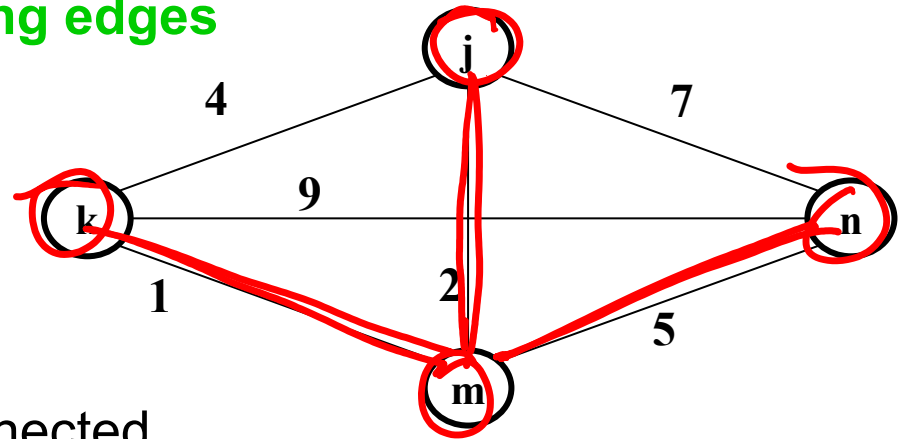
$G=(V, E)$



# Kruskal's Algorithm for MST

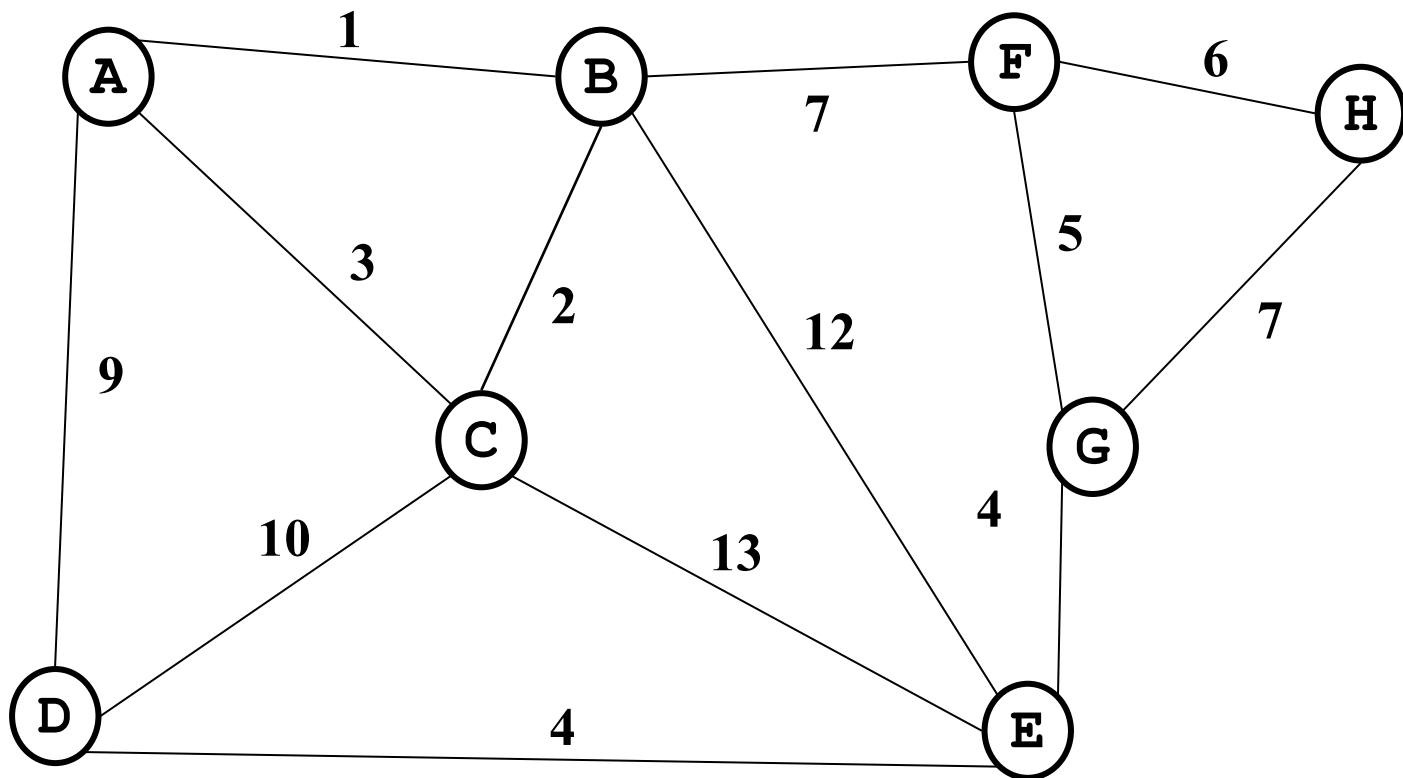
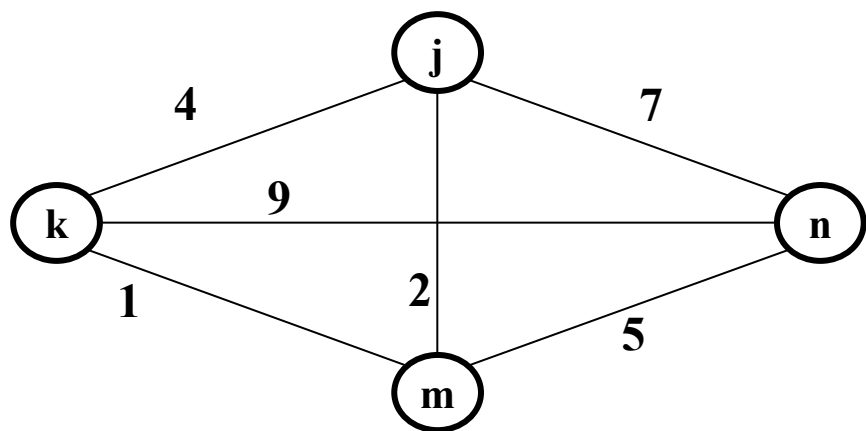
## An edge-based greedy algorithm

Builds MST by greedily adding edges

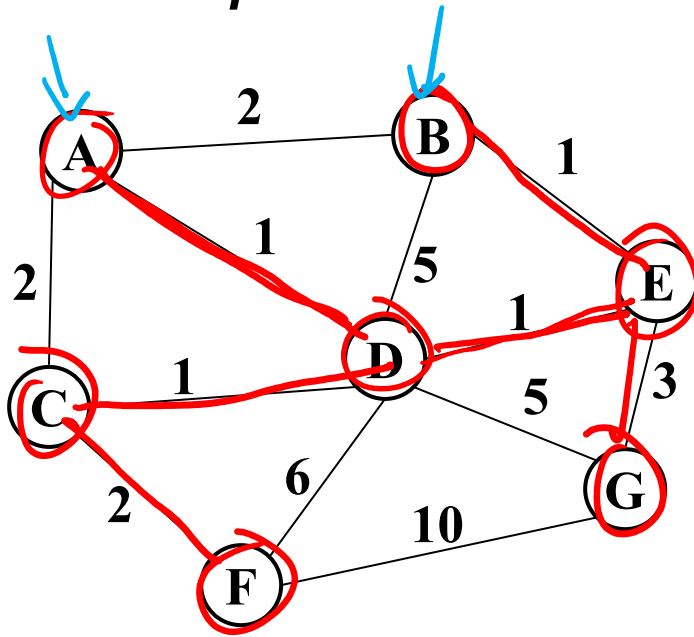


1. Initialize with
  - empty MST
  - all vertices marked unconnected
  - all edges unmarked
2. While all vertices are not connected
  - a. Pick the lowest cost edge  $(u, v)$  and mark it
  - b. If  $u$  and  $v$  are not already connected, add  $(u, v)$  to the MST and mark  $u$  and  $v$  as connected to each other

*Find a MST*



## Example: Find MST using Kruskal's



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

6: (D,F)

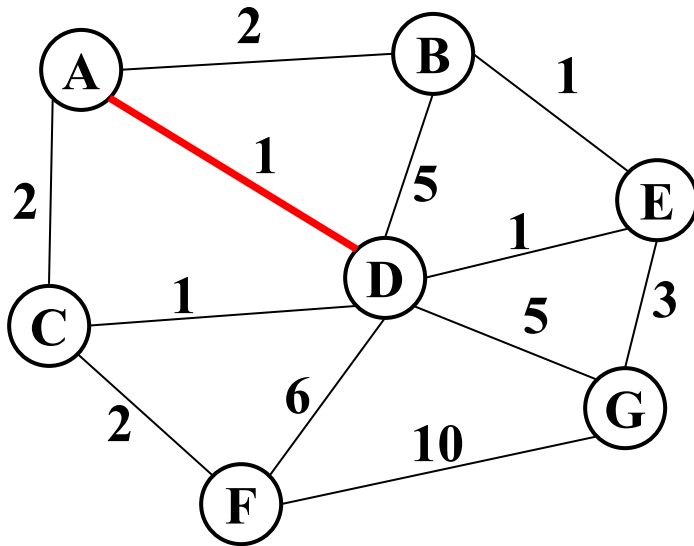
10: (F,G)

Output:

Note: At each step, the union/find sets are the trees in the forest



## Example: Find MST using Kruskal's



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

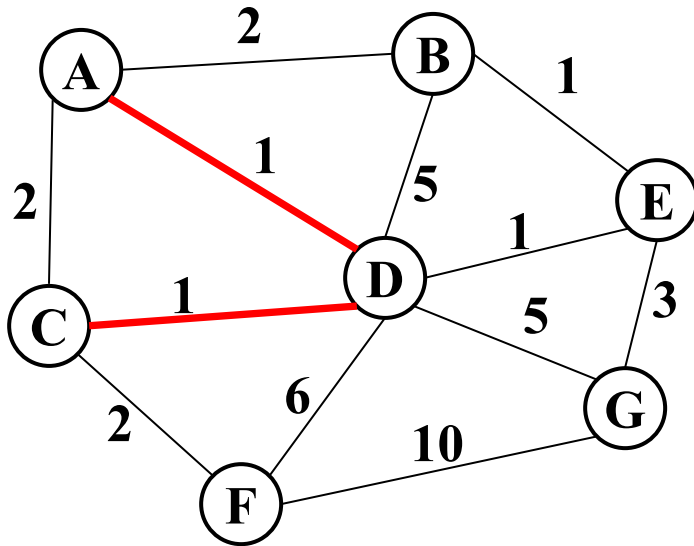
6: (D,F)

10: (F,G)

Output: (A,D)

Note: At each step, the union/find sets are the trees in the forest

## Example: Find MST using Kruskal's



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

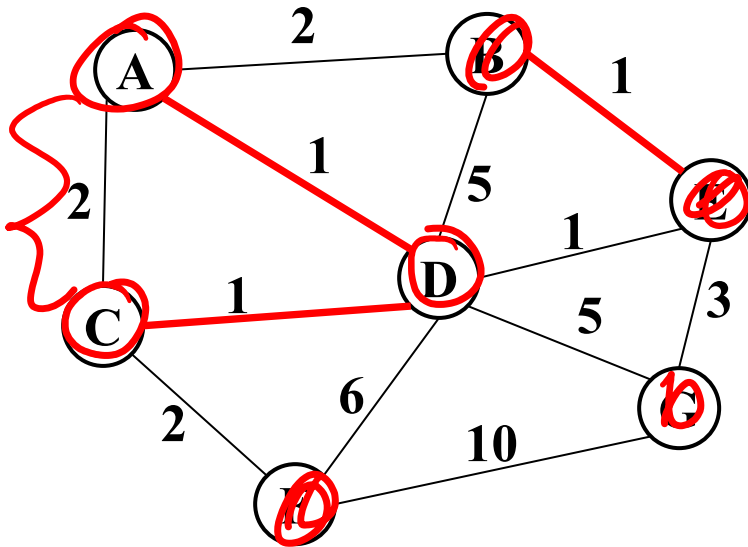
6: (D,F)

10: (F,G)

Output: (A,D), (C,D)

Note: At each step, the union/find sets are the trees in the forest

## Example: Find MST using Kruskal's



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

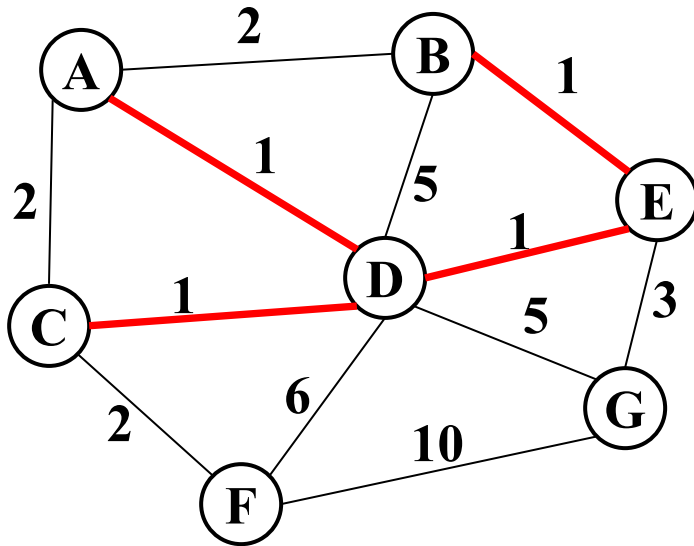
6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E)

Note: At each step, the union/find sets are the trees in the forest

## Example: Find MST using Kruskal's



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

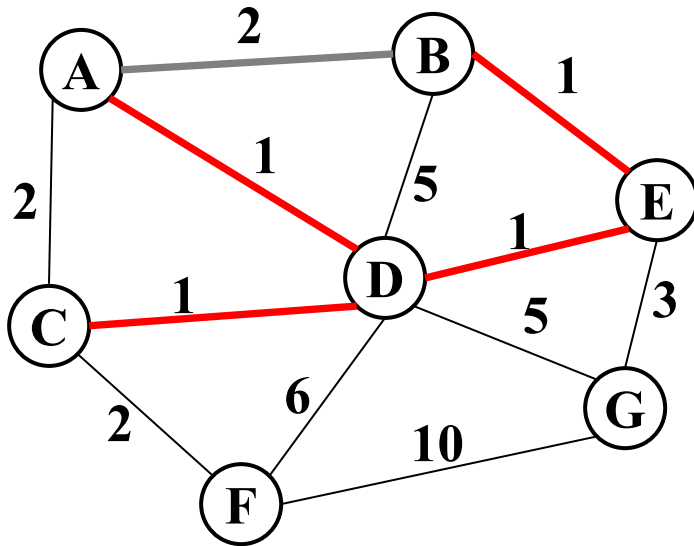
6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E)

Note: At each step, the union/find sets are the trees in the forest

## Example: Find MST using Kruskal's



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

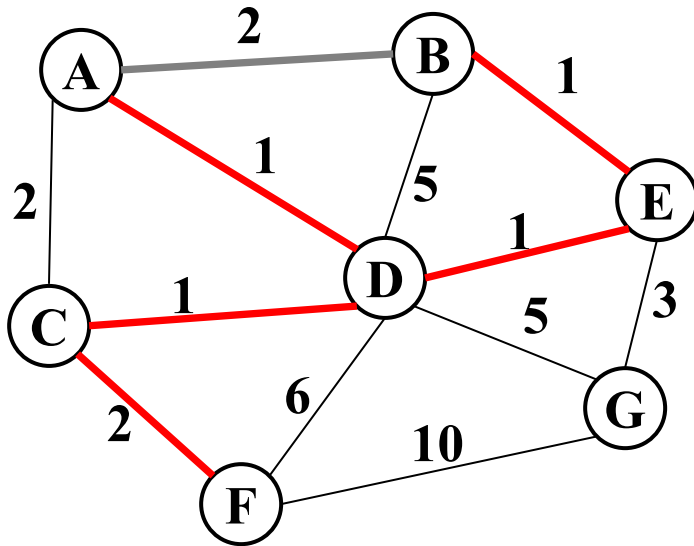
6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E)

Note: At each step, the union/find sets are the trees in the forest

## Example: Find MST using Kruskal's



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

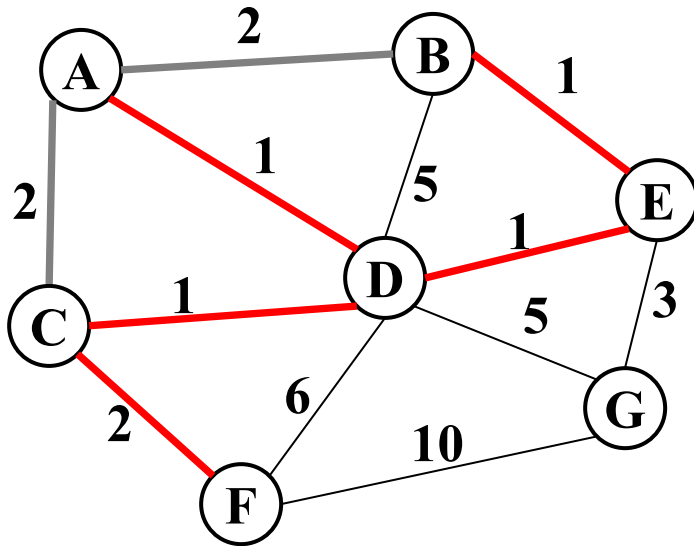
6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E), (C,F)

Note: At each step, the union/find sets are the trees in the forest

## Example: Find MST using Kruskal's



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

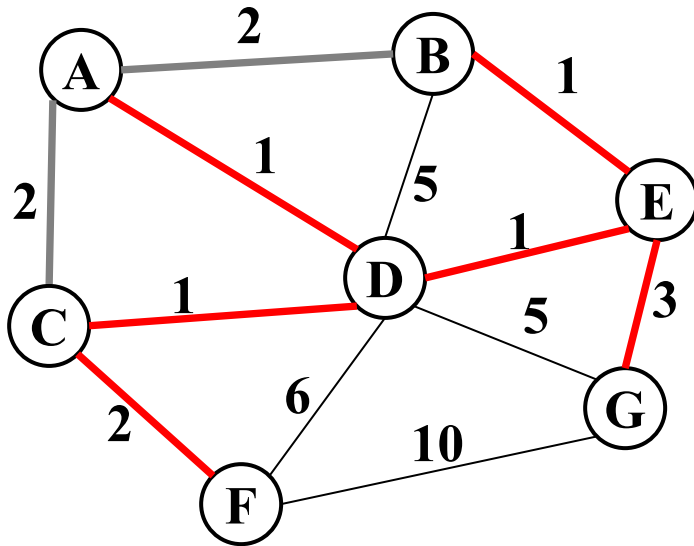
6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E), (C,F)

Note: At each step, the union/find sets are the trees in the forest

## Example: Find MST using Kruskal's



Edges in sorted order:

1: (A,D), (C,D), (B,E), (D,E)

2: (A,B), (C,F), (A,C)

3: (E,G)

5: (D,G), (B,D)

6: (D,F)

10: (F,G)

Output: (A,D), (C,D), (B,E), (D,E), (C,F), (E,G)

Note: At each step, the union/find sets are the trees in the forest



## Aside: Union-Find aka Disjoint Set ADT

- **Union(x,y)** – take the union of two sets named x and y

– Given sets: {3, 5, 7}, {4, 2, 8}, {9}, {1, 6}

– **Union(5,1)**

Result: {3, 5, 7, 1, 6}, {4, 2, 8}, {9},

To perform the union operation, we replace sets x and y by  $(x \cup y)$

- **Find(x)** – return the name of the set containing x.

– Given sets: {3, 5, 7, 1, 6}, {4, 2, 8}, {9},

– **Find(1)** returns 5

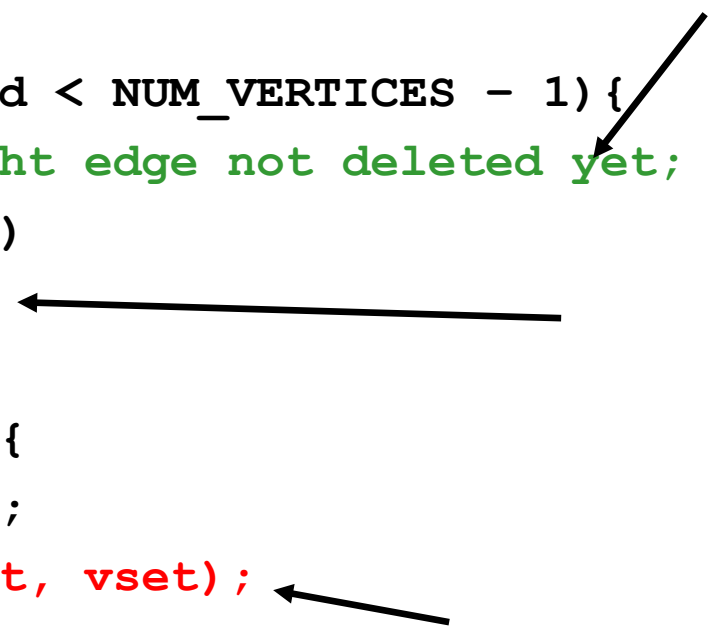
– **Find(4)** returns 8

- We can do **Union** in constant time.

- We can get **Find** to be amortized constant time  
(worst case  $O(\log n)$  for an individual **Find** operation).

# Kruskal's pseudo code

```
void Graph::kruskal() {  
    int edgesAccepted = 0;  
    DisjSet s(NUM_VERTICES);  
  
    while (edgesAccepted < NUM_VERTICES - 1) {  
        e = smallest weight edge not deleted yet;  
        // edge e = (u, v)  
        uset = s.find(u);  
        vset = s.find(v);  
        if (uset != vset) {  
            edgesAccepted++;  
            s.unionSets(uset, vset);  
        }  
    }  
}
```



# Kruskal's pseudo code

```
void Graph::kruskal() {  
    int edgesAccepted = 0;  
    DisjSet s(NUM_VERTICES);
```

```
    while (edgesAccepted < NUM_VERTICES - 1) {  
        e = smallest weight edge not deleted yet;
```

**|E| heap ops**

```
        // edge e = (u, v)
```

```
        uset = s.find(u);
```

**2|E| finds**

```
        vset = s.find(v);
```

```
        if (uset != vset) {
```

```
            edgesAccepted++;
```

```
            s.unionSets(uset, vset);
```

**|V| unions**

```
        }
```

```
    }
```

```
}
```

# Kruskal's pseudo code

```
void Graph::kruskal() {  
    int edgesAccepted = 0;  
    DisjSet s(NUM_VERTICES);
```

On heap of  
edges  
Deletemin =  
 $\log |E|$

**|E| heap ops**

```
while (edgesAccepted < NUM_VERTICES - 1) {  
    e = smallest weight edge not deleted yet;
```

```
// edge e = (u, v)
```

```
uset = s.find(u);
```

```
vset = s.find(v);
```

```
if (uset != vset) {
```

```
    edgesAccepted++;
```

```
    s.unionSets(uset, vset);
```

```
}
```

```
}  
}
```

**2|E| finds**

One for each  
vertex in the  
edge wc for  
Find =  $\log |V|$

**|V| unions**

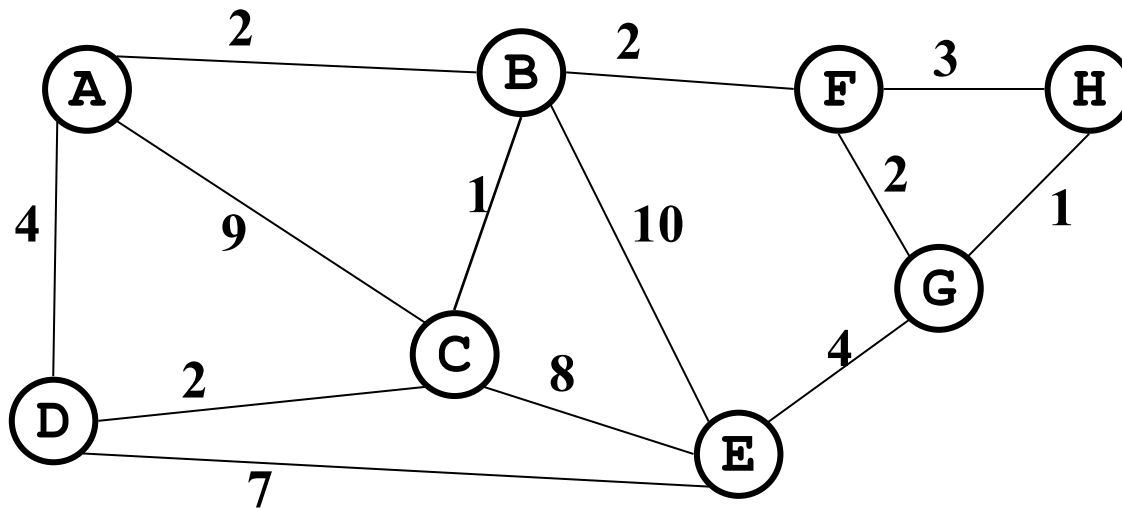
Union =  $O(1)$

$|E| \log |E| + \underline{2|E|\log|V|} + |V|$

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

b/c  $\log |E| < \log |V|^2 = 2\log |V|$

*Find MST using Kruskal's*



**Total Cost:**

- **Now find the MST using Prim's method.**
- **Under what conditions will these methods give the same result?**

# Correctness

Kruskal's algorithm is clever, simple, and efficient

- But does it generate a minimum spanning tree?
- How can we prove it?

First: it generates a spanning tree

- Intuition: Graph started connected and we added every edge that did not create a cycle
- Proof by contradiction: Suppose  $u$  and  $v$  are disconnected in Kruskal's result. Then there's a path from  $u$  to  $v$  in the initial graph with an edge we could add without creating a cycle. But Kruskal would have added that edge. Contradiction.

Second: There is no spanning tree with lower total cost...

## *The inductive proof set-up*

Let **F** (stands for “forest”) be the set of edges Kruskal has added at some point during its execution.

Claim: **F** is a subset of *one or more* MSTs for the graph  
(Therefore, once  $|\mathbf{F}|=|\mathbf{V}|-1$ , we have an MST.)

Proof: By induction on  $|\mathbf{F}|$

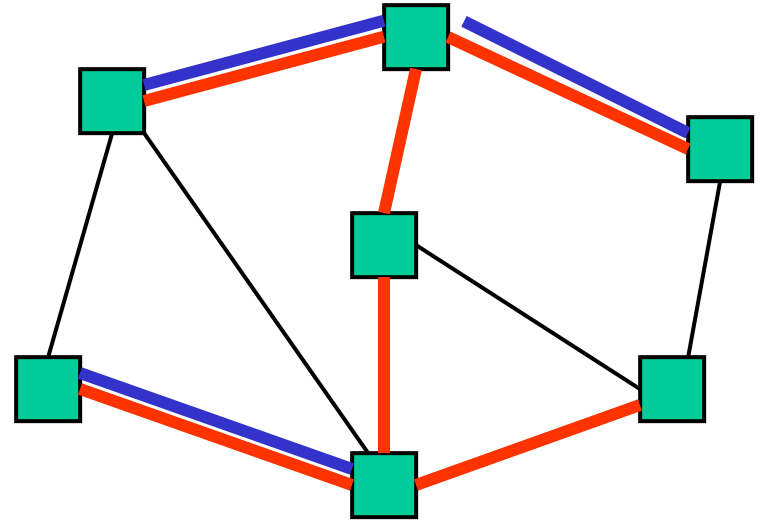
Base case:  $|\mathbf{F}|=0$ : The empty set is a subset of all MSTs

Inductive case:  $|\mathbf{F}|=k+1$ : By induction, before adding the  $(k+1)^{\text{th}}$  edge (call it **e**), there was some MST **T** such that  $\mathbf{F}-\{\mathbf{e}\} \subseteq \mathbf{T} \dots$

# Staying a subset of **some** MST

Claim: **F** is a subset of *one or more* MSTs for the graph

So far: **F** - {**e**}  $\subseteq$  **T**:



Two disjoint cases:

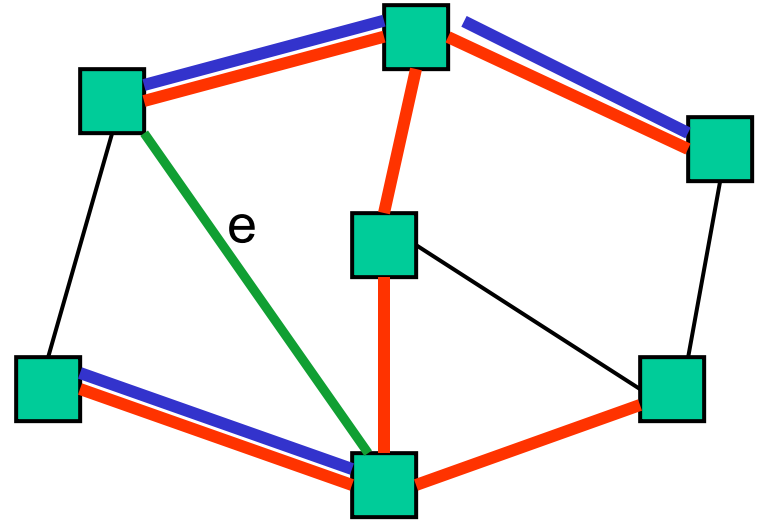
- If {**e**}  $\subseteq$  **T**: Then **F**  $\subseteq$  **T** and we're done
- Else **e** forms a cycle with some simple path (call it **p**) in **T**
  - Must be since **T** is a spanning tree



# Staying a subset of **some** MST

Claim: **F** is a subset of *one or more* MSTs for the graph

So far:  $\mathbf{F} - \{\mathbf{e}\} \subseteq \mathbf{T}$  and  
 $\mathbf{e}$  forms a cycle with  $\mathbf{p} \subseteq \mathbf{T}$



- There must be an edge **e2** on **p** such that **e2** is not in **F**
  - Else Kruskal would not have added **e**
- Claim: **e2.weight == e.weight**

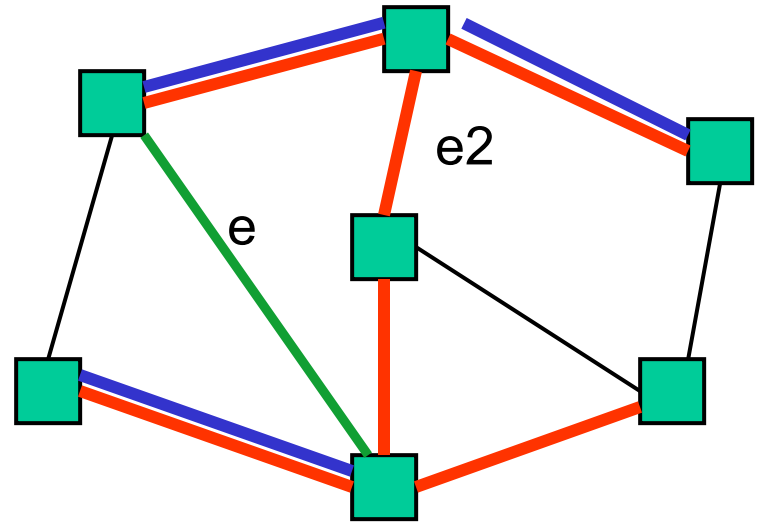
# Staying a subset of **some** MST

Claim: **F** is a subset of *one or more* MSTs for the graph

So far:  $\mathbf{F} - \{\mathbf{e}\} \subseteq \mathbf{T}$

$\mathbf{e}$  forms a cycle with  $\mathbf{p} \subseteq \mathbf{T}$

$\mathbf{e2}$  on  $\mathbf{p}$  is not in  $\mathbf{F}$



- Claim:  $\mathbf{e2.weight} == \mathbf{e.weight}$ 
  - If  $\mathbf{e2.weight} > \mathbf{e.weight}$ , then  $\mathbf{T}$  is not an MST because  $\mathbf{T} - \{\mathbf{e2}\} + \{\mathbf{e}\}$  is a spanning tree with lower cost: contradiction
  - If  $\mathbf{e2.weight} < \mathbf{e.weight}$ , then Kruskal would have already considered  $\mathbf{e2}$ . It would have added it since  $\mathbf{T}$  has no cycles and  $\mathbf{F} - \{\mathbf{e}\} \subseteq \mathbf{T}$ . But  $\mathbf{e2}$  is not in  $\mathbf{F}$ : contradiction

# Staying a subset of **some** MST

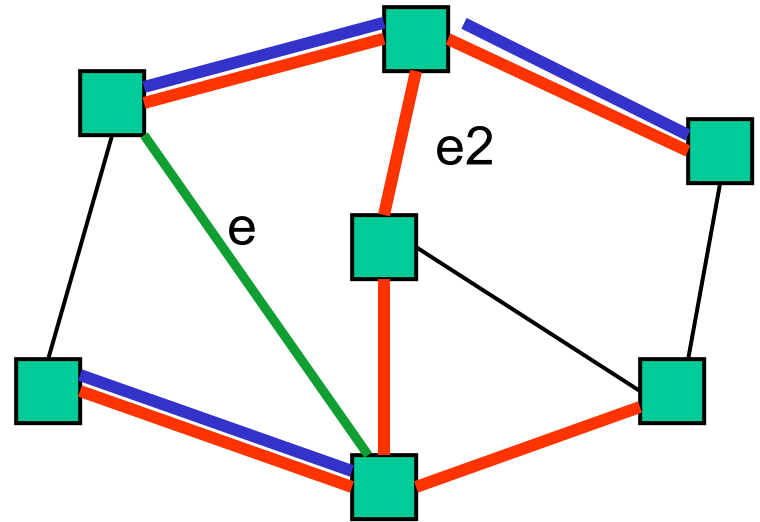
Claim: **F** is a subset of *one or more* MSTs for the graph

So far: **F** - {**e**}  $\subseteq$  **T**

**e** forms a cycle with **p**  $\subseteq$  **T**

**e2** on **p** is not in **F**

**e2.weight == e.weight**



- Claim: **T** - {**e2**} + {**e**} is an MST
    - It's a spanning tree because **p** - {**e2**} + {**e**} connects the same nodes as **p**
    - It's minimal because its cost equals cost of **T**, an MST
  - Since **F**  $\subseteq$  **T** - {**e2**} + {**e**}, **F** is a subset of one or more MSTs
- Done.