CSE332: Data Abstractions

Lecture 23: Minimum Spanning Trees

Ruth Anderson

Autumn 2013

"Scheduling note"

- "We now return to our interrupted program" on graphs
  - Last "graph lecture" was lecture 16
    - Shortest-path problem
    - Dijkstra's algorithm for graphs with non-negative weights

- Why this strange schedule?
  - Needed to do parallelism and concurrency in time for project 3 and homeworks 6 and 7
  - But cannot delay all of graphs because of the CSE312 co-requisite

- So: not the most logical order, but hopefully not a big deal

Minimum Spanning Trees

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

– E' is a subset of E

– |E'| = |V| - 1

– G' is connected

G' is a minimum spanning tree.

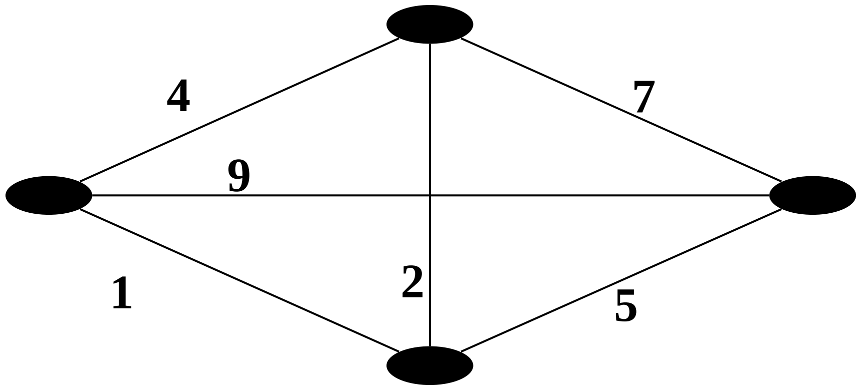
– $$\sum_{(u,v)\in E'} c_{uv}$$ is minimal

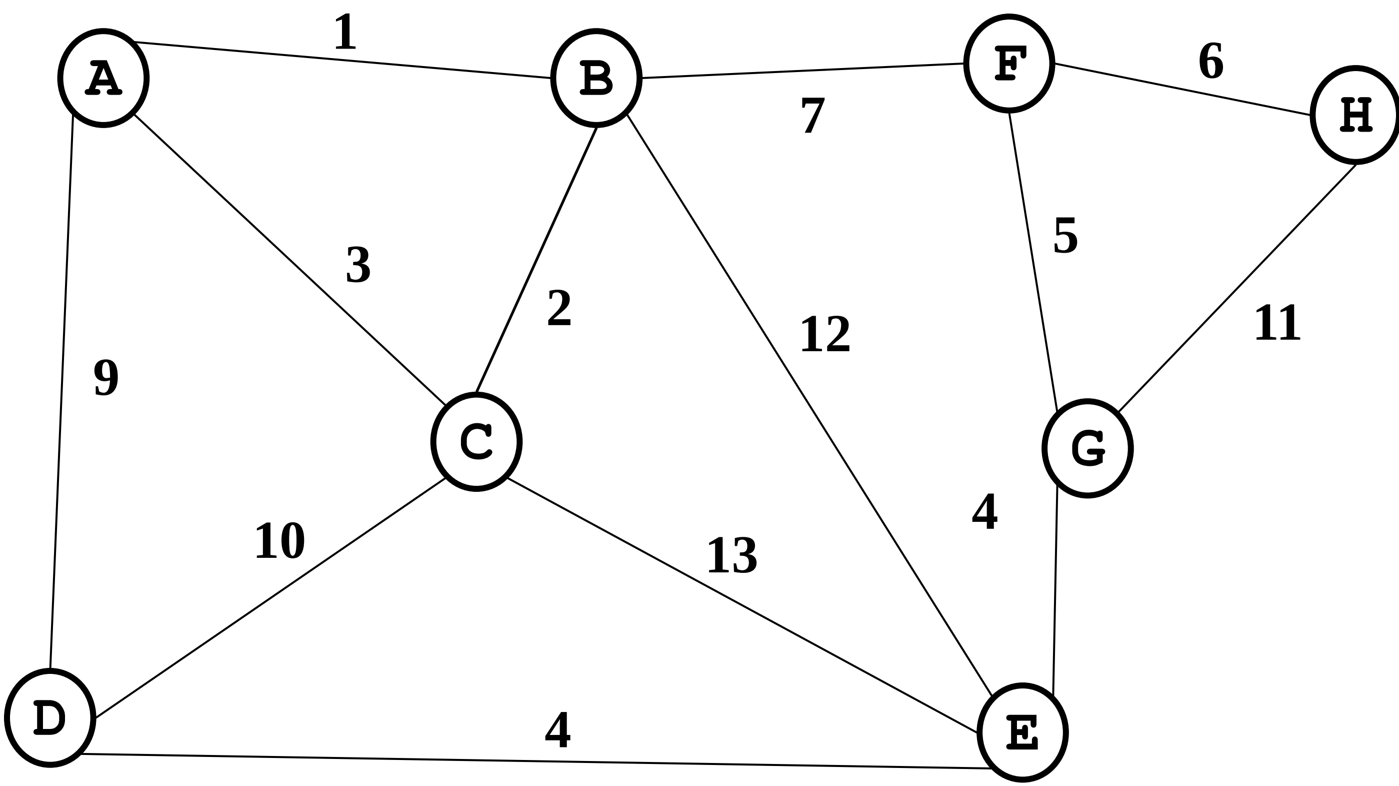
Applications:

- 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

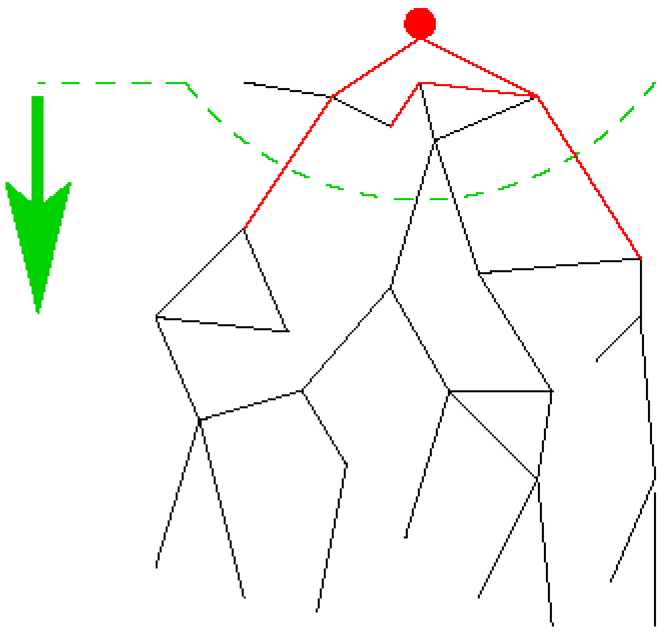
Student Activity

Find the MST



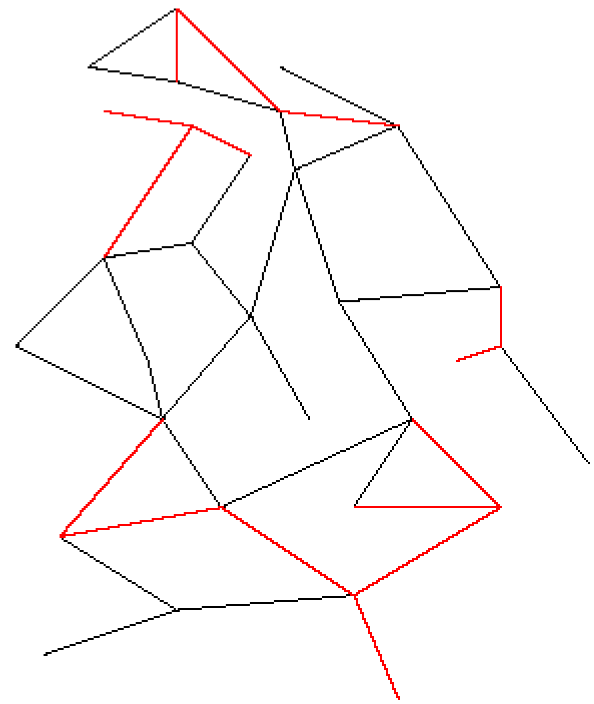


Two Different Approaches



Prim's Algorithm

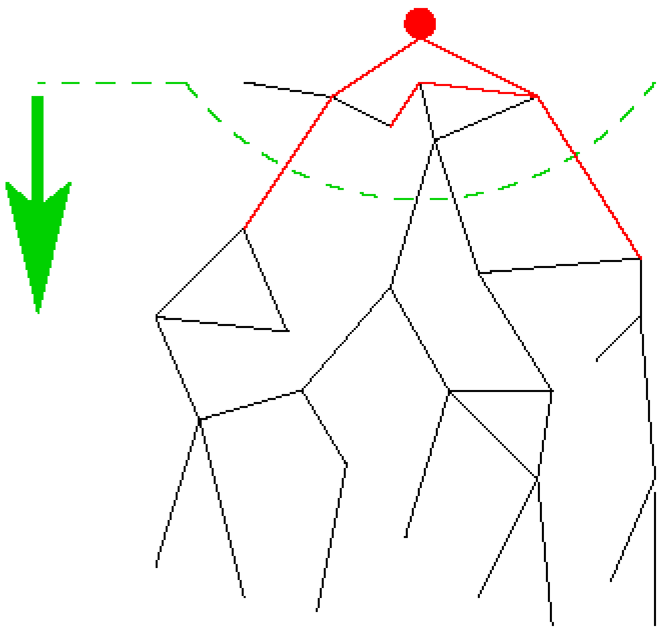
Almost identical to Dijkstra's

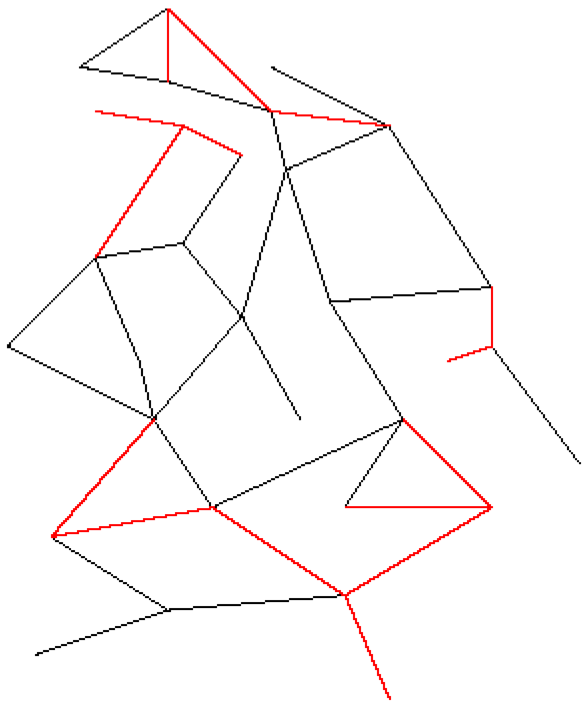


Kruskals's Algorithm

Completely different!

Two Different Approaches





Prim's Algorithm

Almost identical to Dijkstra's

One node, grow greedily

Kruskals's Algorithm

Completely different!

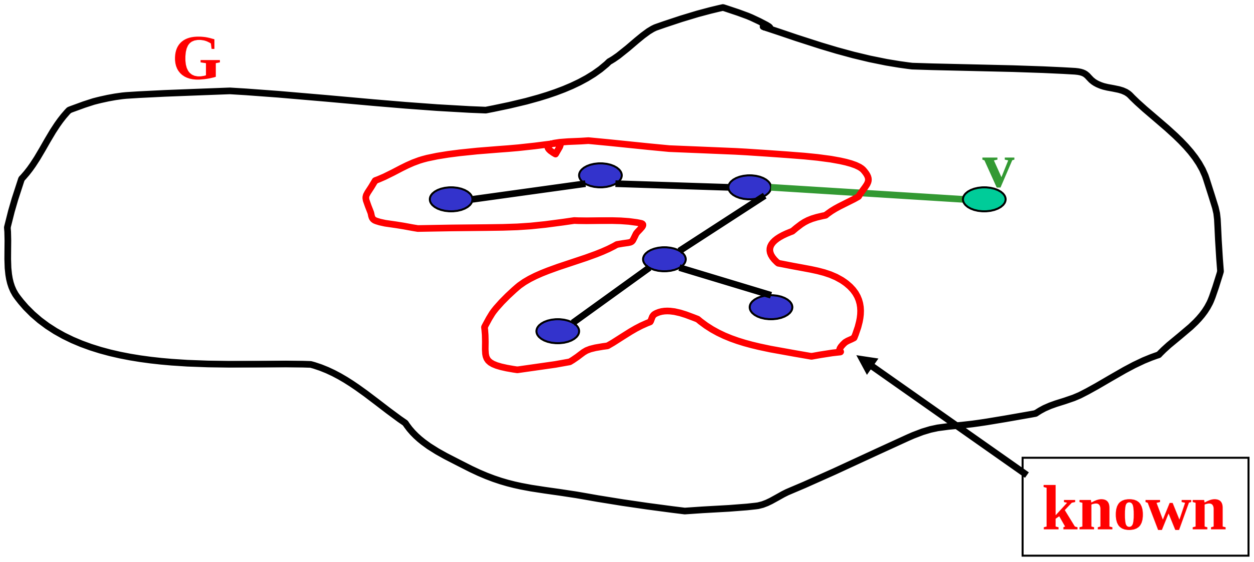
Forest of MSTs,
***Union* them together.**
I wonder how to union…

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)

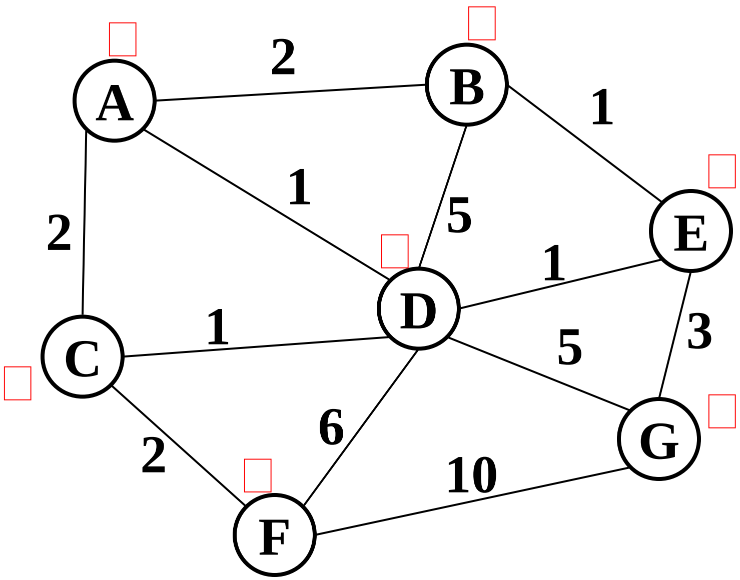
- Otherwise identical
- Compare to slides in lecture 16!

Prim's Algorithm for MST

1. For each node **`v`**, set **`v.cost = ∞`** 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`**,

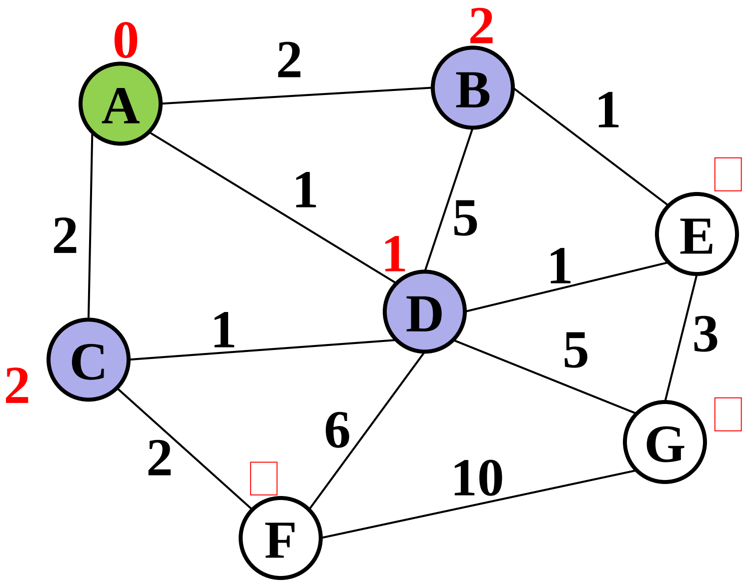
```
if(w < u.cost) {
  u.cost = w;
  u.prev = v;
}
```

Example: Find MST using Prim's



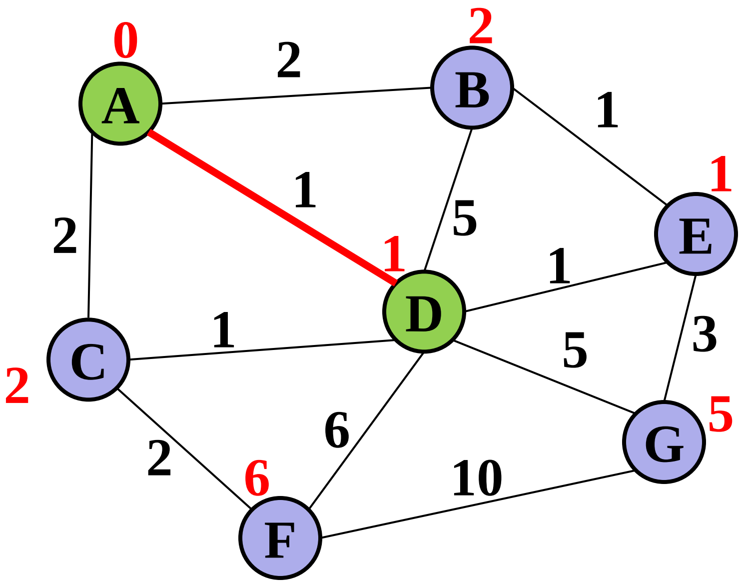
vertex	known?	cost	prev
A		??	
B		??	
C		??	
D		??	
E		??	
F		??	
G		??	

Example: Find MST using Prim's



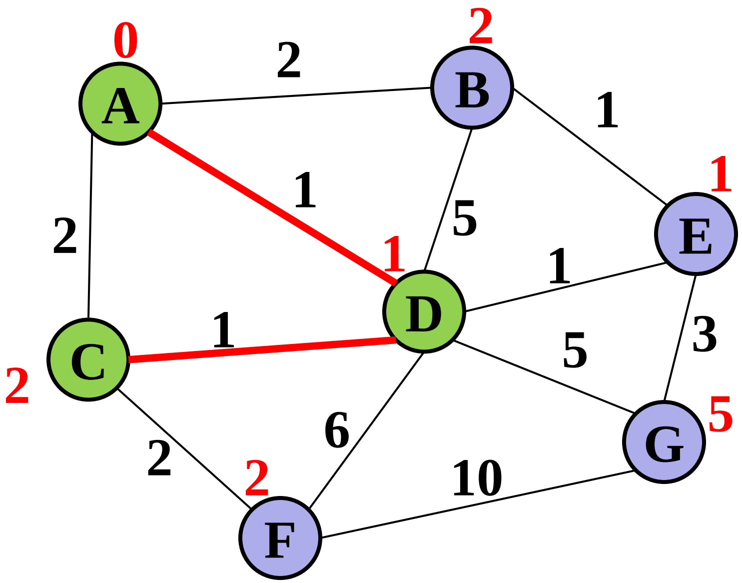
vertex	known?	cost	prev
A	Y	0	
B		2	A
C		2	A
D		1	A
E		??	
F		??	
G		??	

Example: Find MST using Prim's



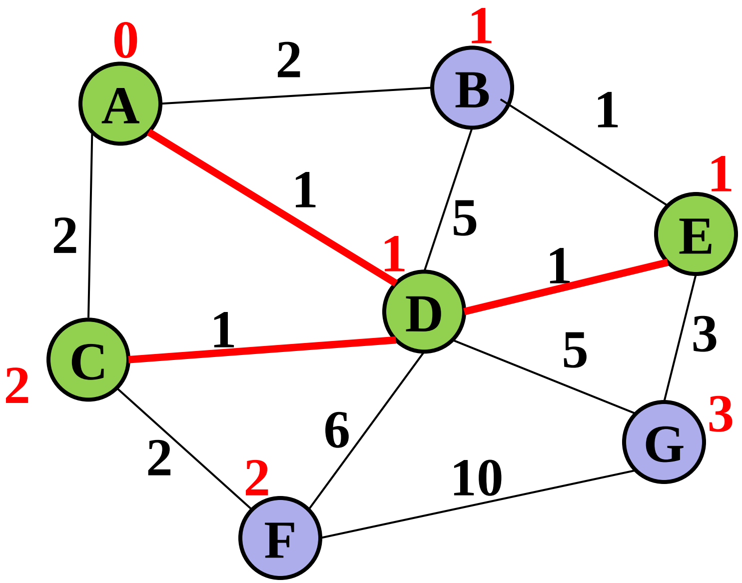
vertex	known?	cost	prev
A	Y	0	
B		2	A
C		1	D
D	Y	1	A
E		1	D
F		6	D
G		5	D

Example: Find MST using Prim's



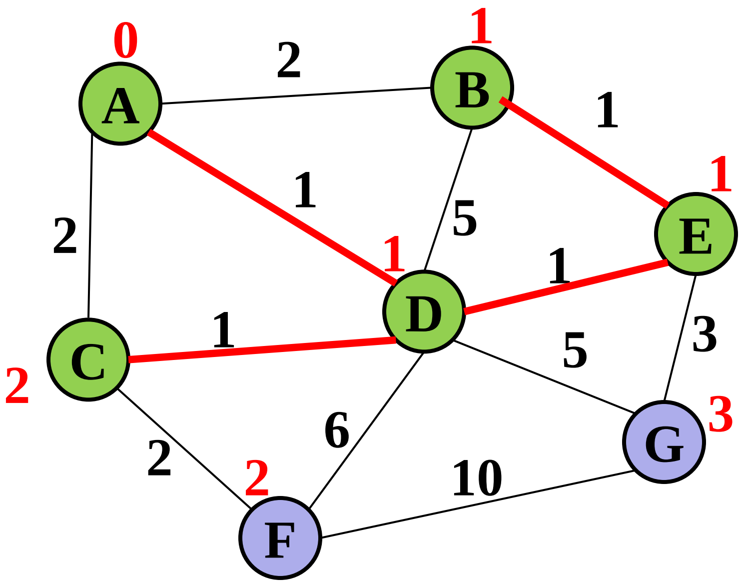
vertex	known?	cost	prev
A	Y	0	
B		2	A
C	Y	1	D
D	Y	1	A
E		1	D
F		2	C
G		5	D

Example: Find MST using Prim's



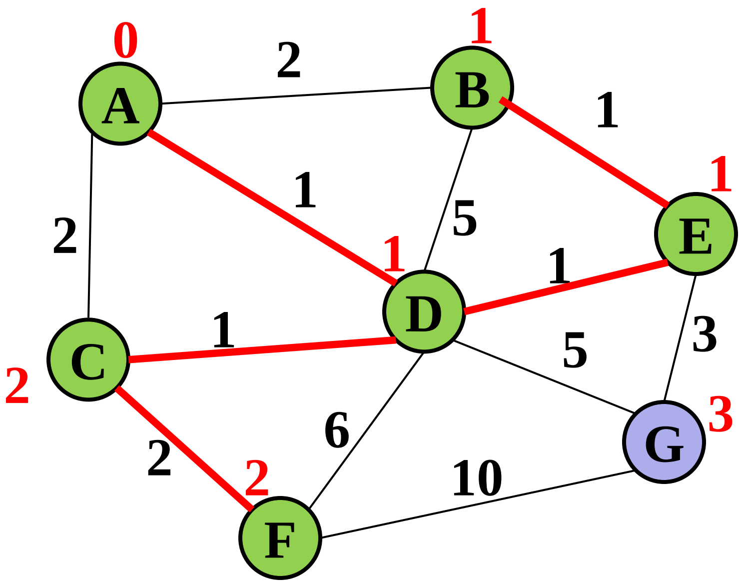
vertex	known?	cost	prev
A	Y	0	
B		1	E
C	Y	1	D
D	Y	1	A
E	Y	1	D
F		2	C
G		3	E

Example: Find MST using Prim's



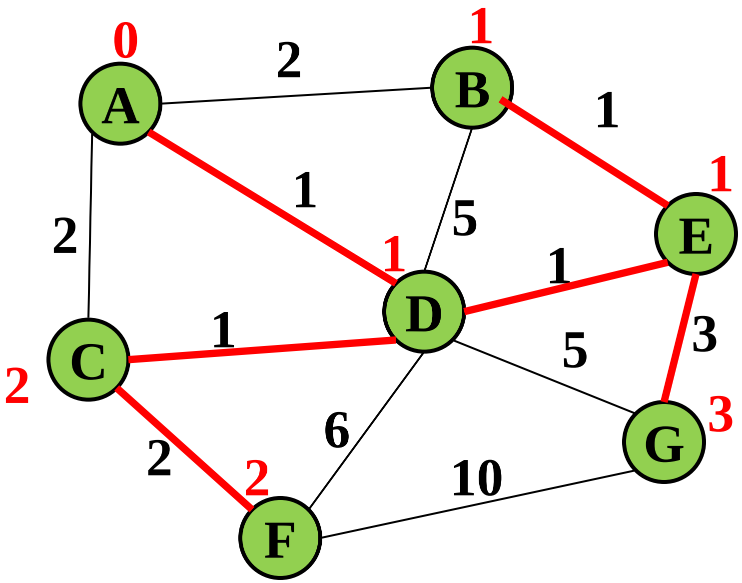
vertex	known?	cost	prev
A	Y	0	
B	Y	1	E
C	Y	1	D
D	Y	1	A
E	Y	1	D
F		2	C
G		3	E

Example: Find MST using Prim's



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

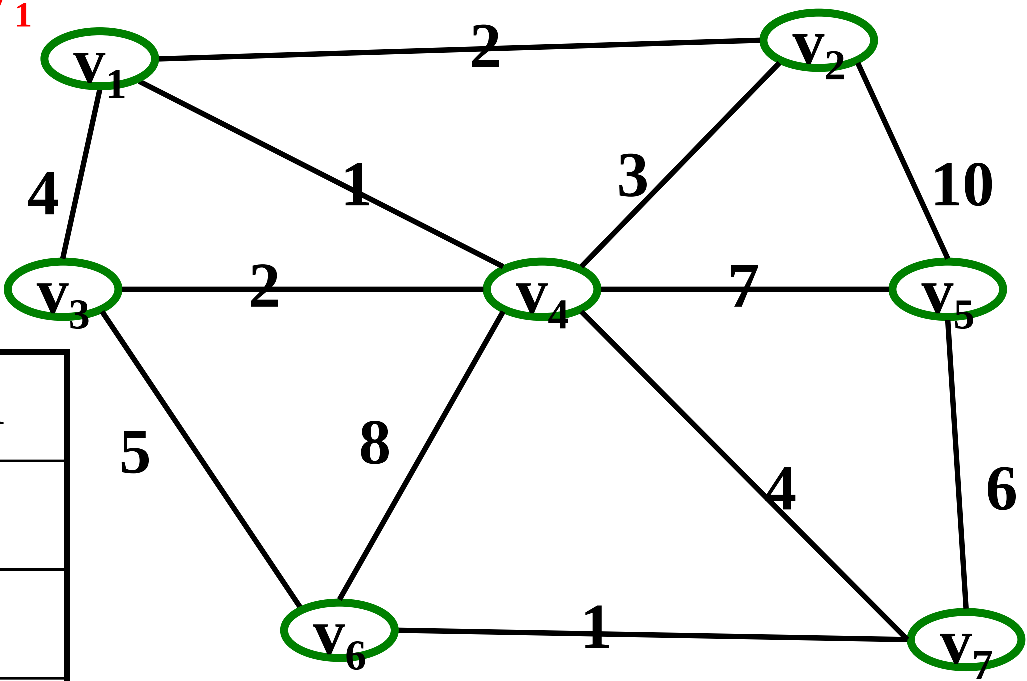
Example: Find MST using Prim's



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

Student Activity

Start with V_1



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

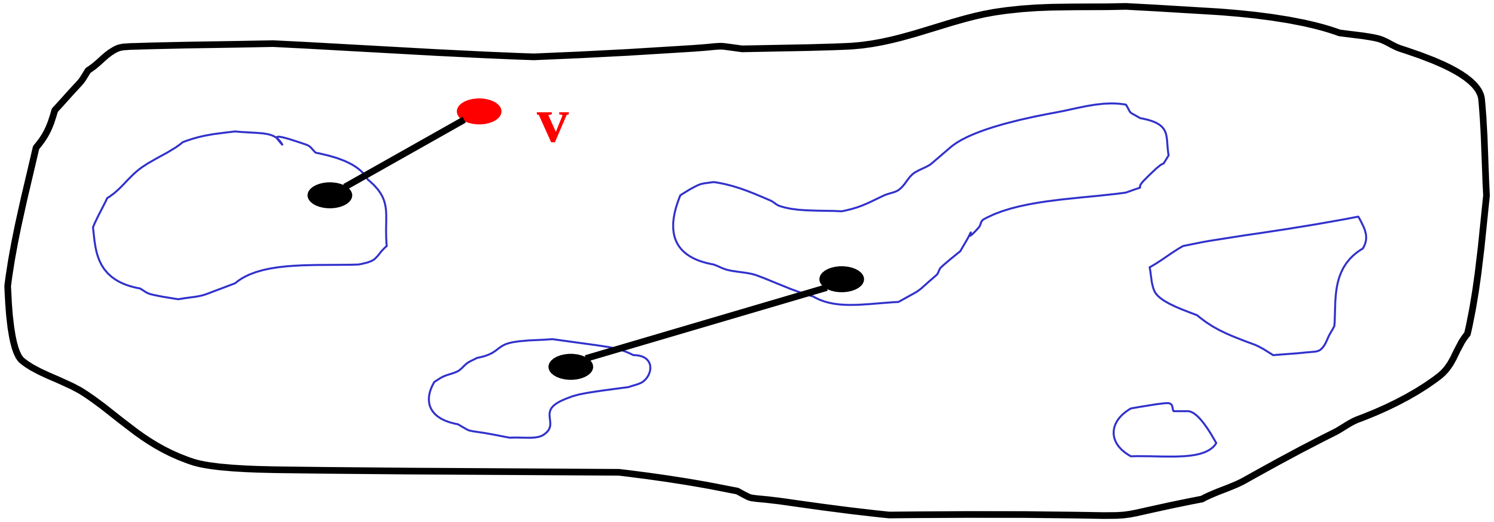
- Correctness ??
  - A bit tricky
  - Intuitively similar to Dijkstra
  - Might return to this time permitting (unlikely)

- 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 there are still unmarked edges
   a. Pick the <u>lowest cost edge</u> **`(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

Aside: Union-Find aka Disjoint Set ADT

- **Union(x,y)** – take the union of two sets named x and y
  - Given sets: {3,<u>5</u>,7} , {4,2,<u>8</u>}, {<u>9</u>}, {<u>1</u>,6}
  - **Union(5,1)**

    Result: {3,<u>5</u>,7,1,6}, {4,2,<u>8</u>}, {<u>9</u>},

  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,<u>5</u>,7,1,6}, {4,2,<u>8</u>}, {<u>9</u>},
  - **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);
```

Sort of ignore this loop in calc run-time…

```
  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);
    }
  }
}
```

|E| heap ops

2|E| finds

|V| unions

Kruskal's pseudo code

On heap of edges
Deletemin = log |E|

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

Sort of ignore this loop in calc run-time…

```
  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);
    }
  }
}
```

|E| heap ops

2|E| finds

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

|V| unions

Union = O(1)

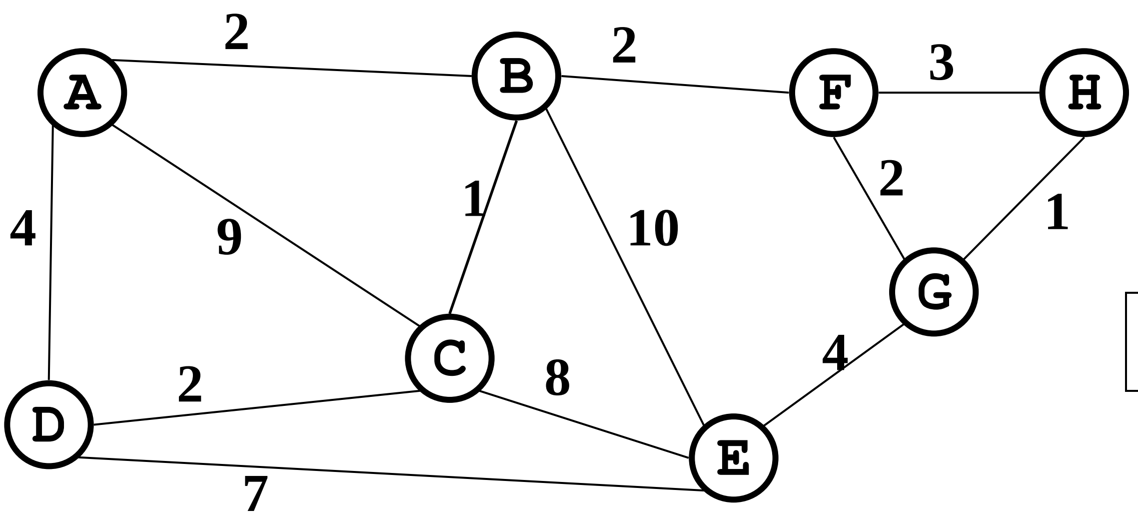
$|E| \log |E| + \underline{2|E|\log|V|+|V|}$

$O(\,|E|\log|E| + |E|\sim O(1)) = O(|E|\log|E|) = O(|E|\log|V|)$

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

Student Activity

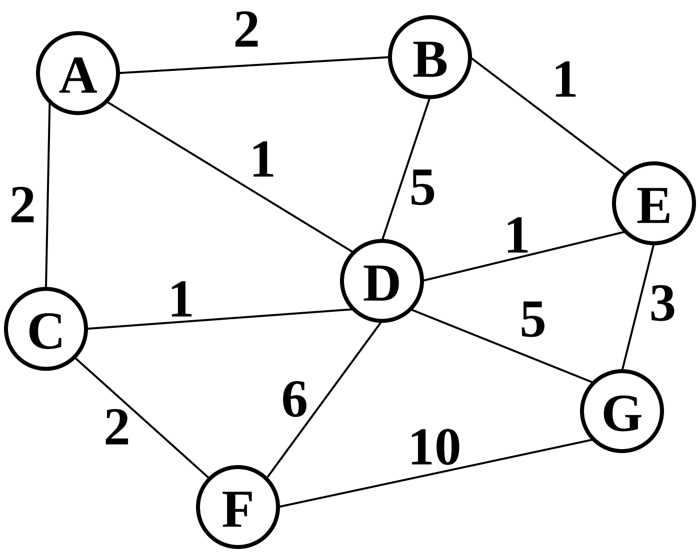
Find MST using Kruskal's



Total Cost:

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

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)

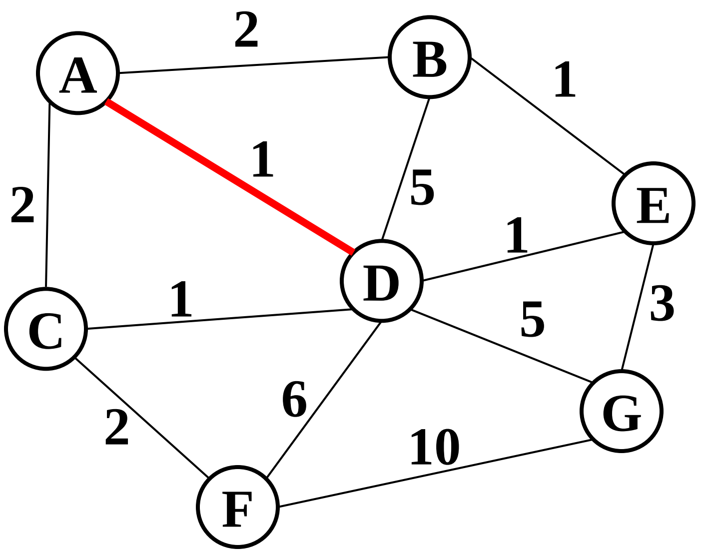
6: (D,F)

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)

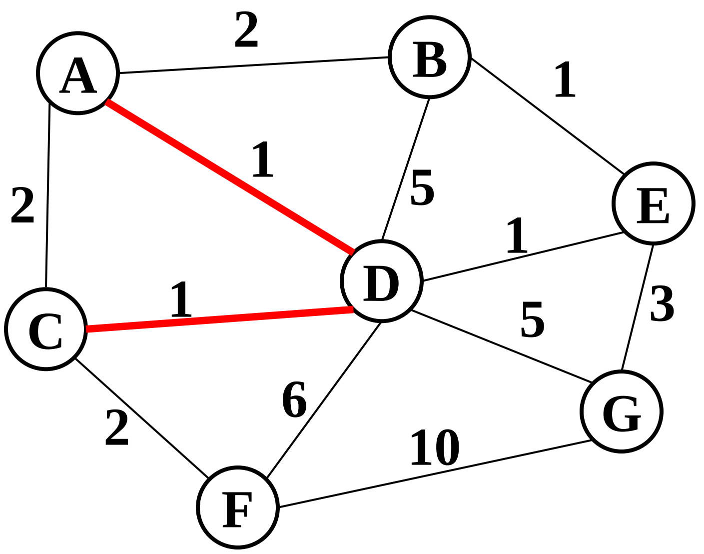
6: (D,F)

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)

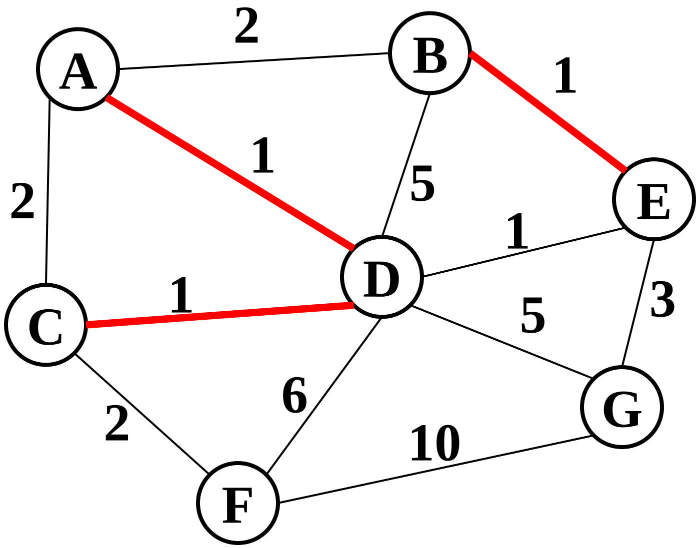
6: (D,F)

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)

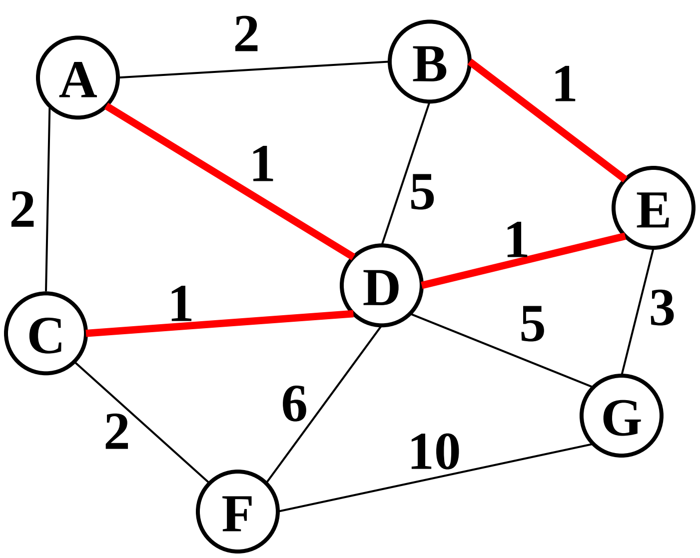
6: (D,F)

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)

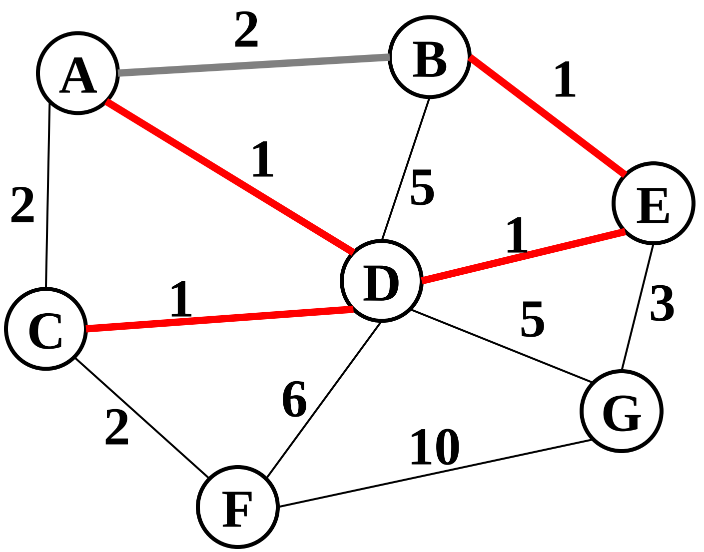
6: (D,F)

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)

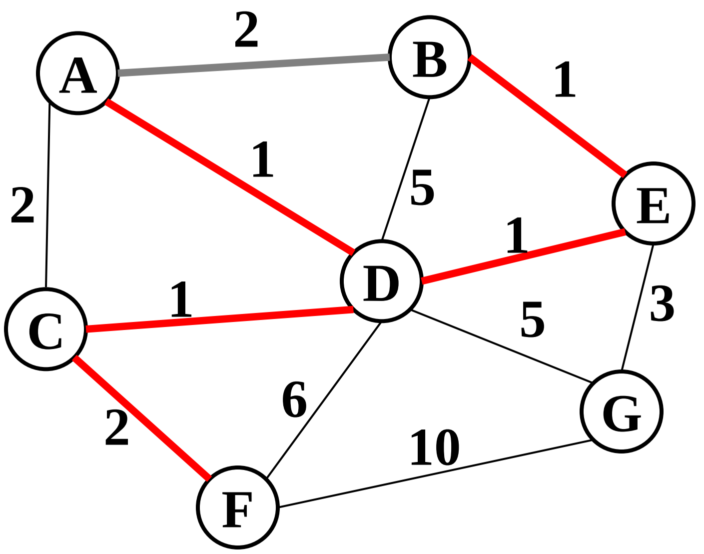
6: (D,F)

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)

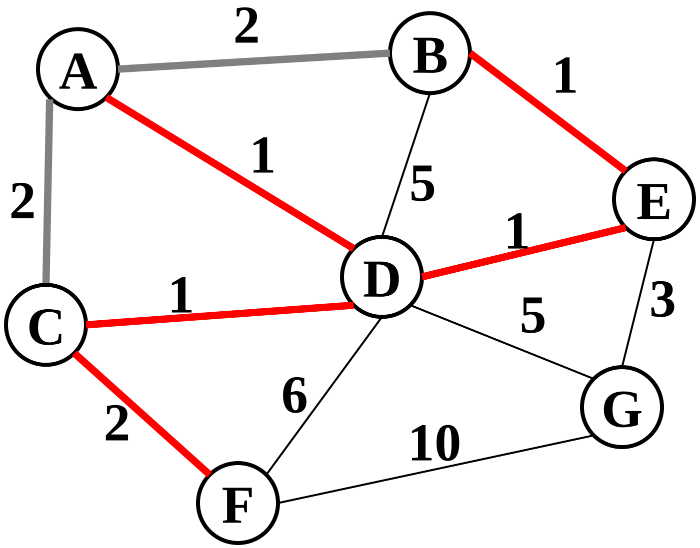
6: (D,F)

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)

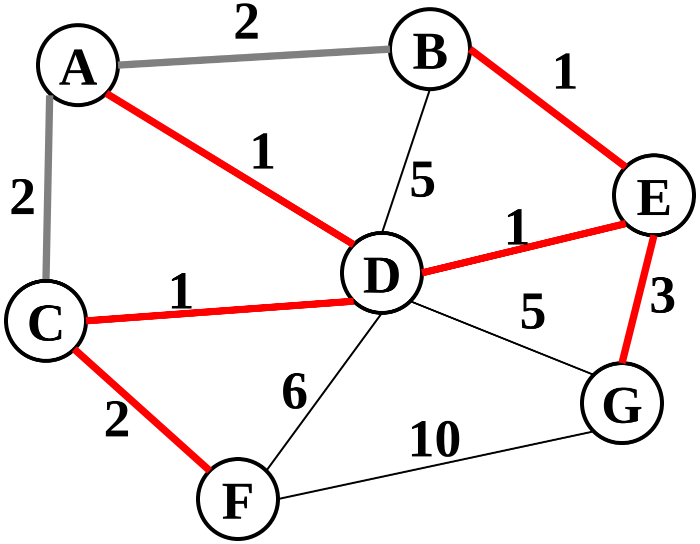
6: (D,F)

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

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 **|F|=|V|-1**, we have an MST.)

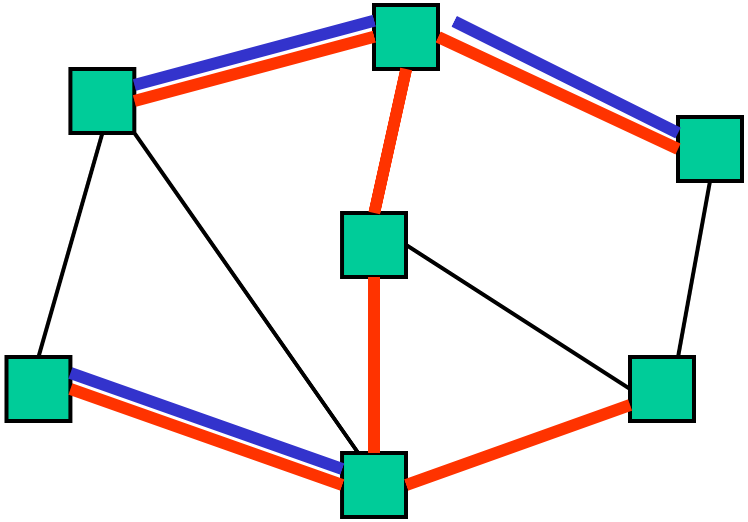
Proof: By induction on **|F|**

Base case: **|F|=0**: The empty set is a subset of all MSTs

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

*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:

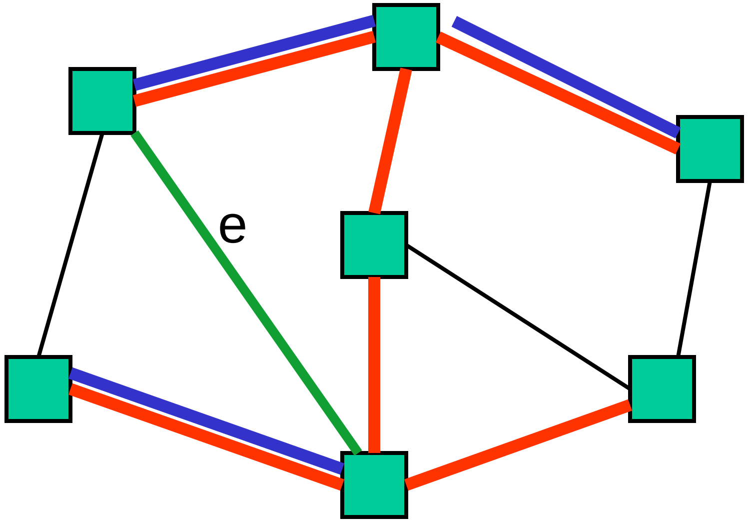
- If **{e}** $\subseteq$ **T**: Then **F** $\subseteq$ **T** and we're done
- Else **e** forms a cycle with some simple path (call it **p**) in **T**
  – Must be since **T** is a spanning tree

*Staying a subset of **some** MST*

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

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

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

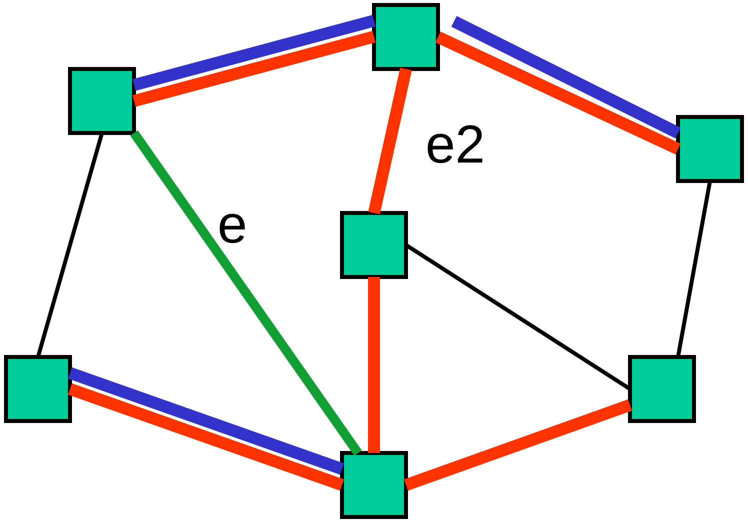
Staying a subset of ***some*** *MST*

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

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

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

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



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

*Staying a subset of **some** MST*

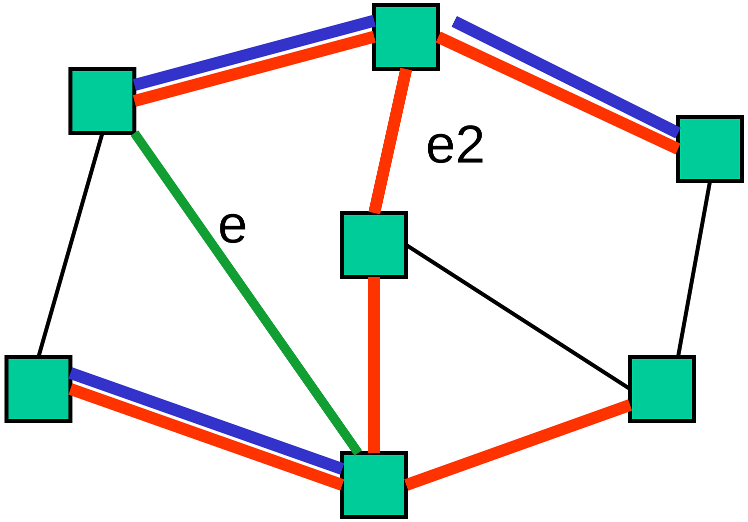
Claim: **F** is a subset of *one or more* MSTs for the graph

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.