

# COL106

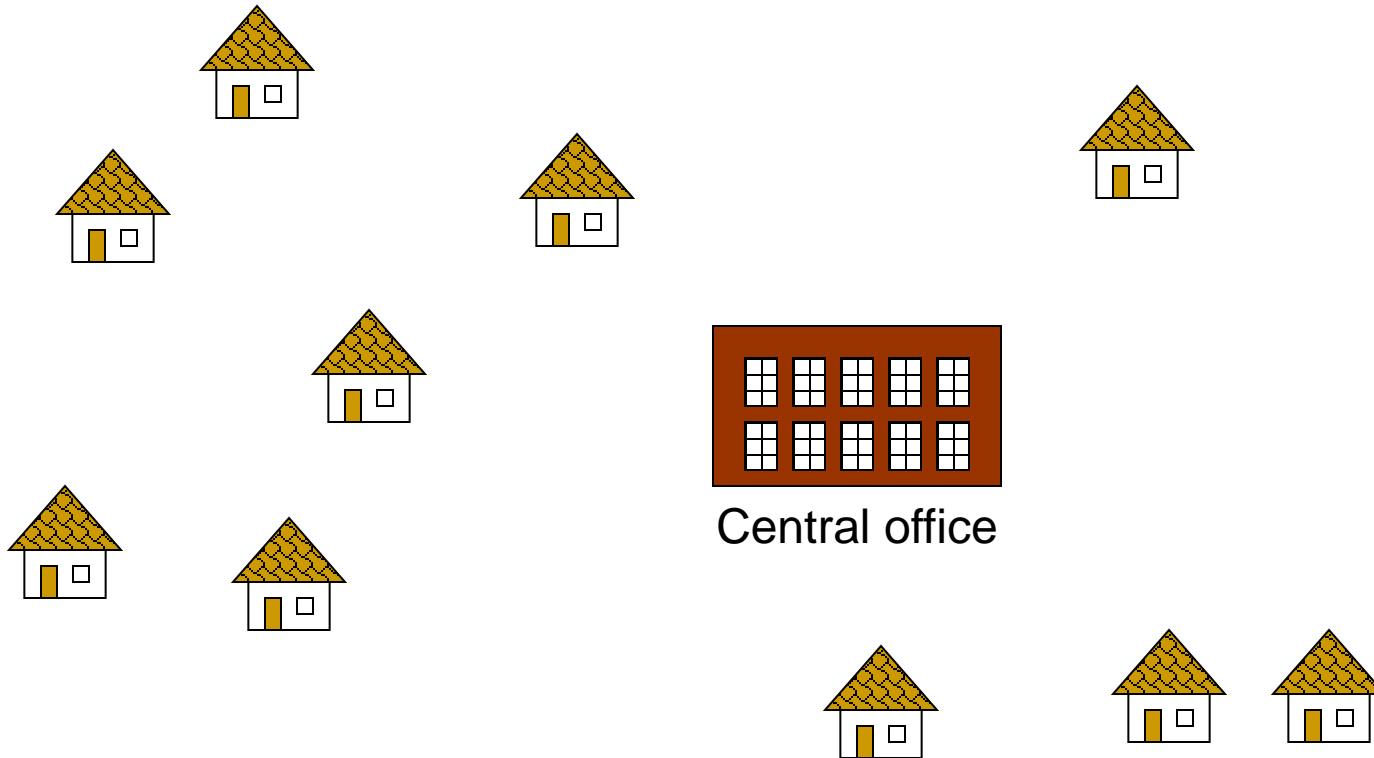
# Data Structures and Algorithms

Subodh Sharma and Rahul Garg

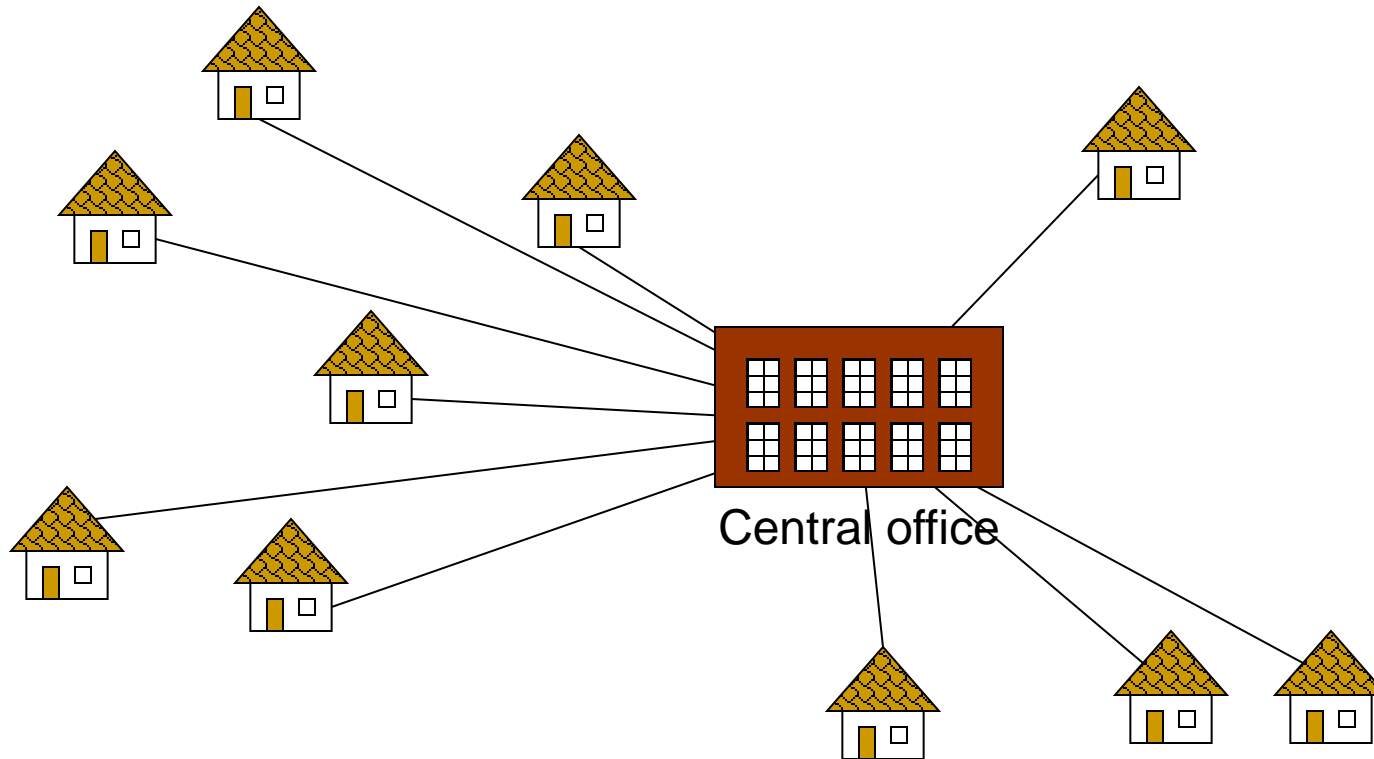
# Minimum Spanning Trees

Based on slides by: Guy Kortsarz, Rutgers University, Rose Hoberman, CMU, Longin Jan Latecki, Temple University

# Problem: Laying Telephone Wire

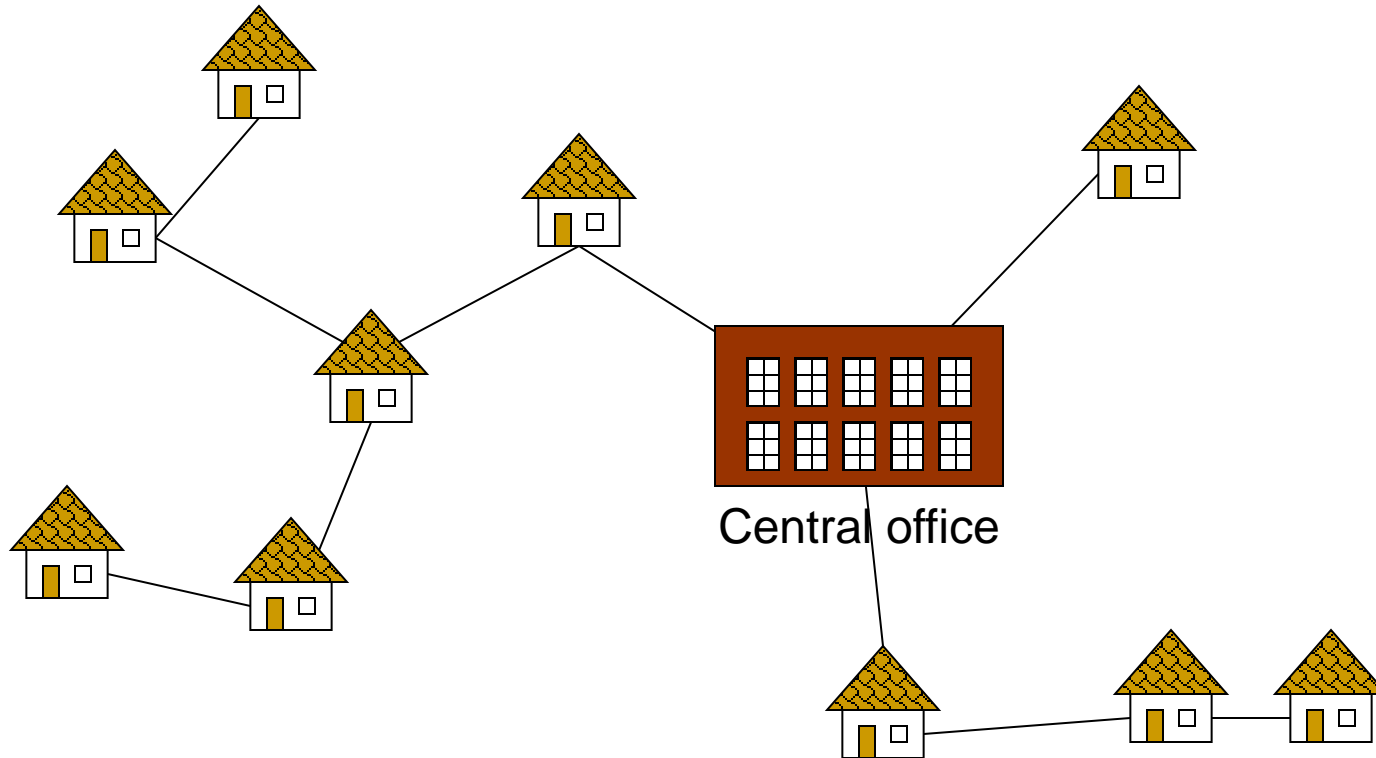


# Wiring: Naive Approach



**Expensive!**

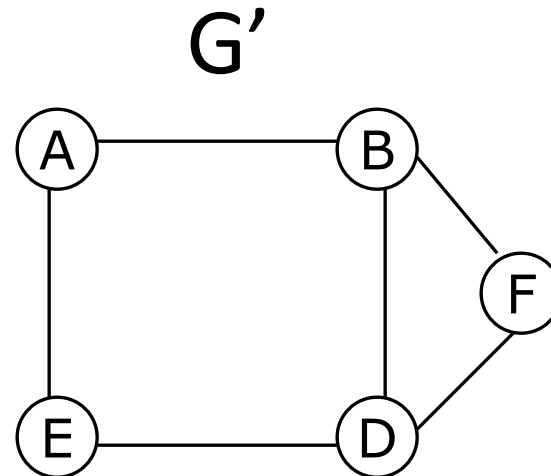
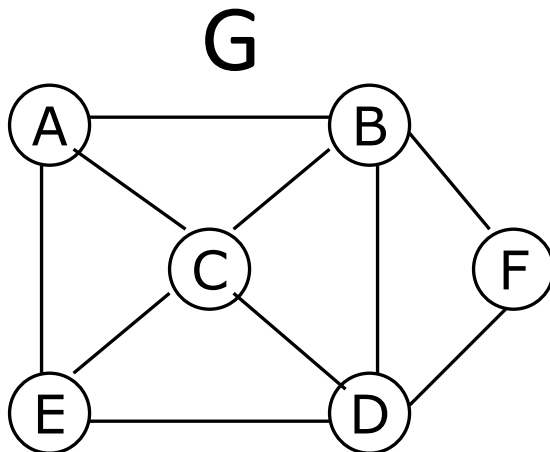
# Wiring: Better Approach



Minimize the total length of wire connecting the customers

# Definitions

- Given a graph  $G = (V, E)$
- A **subgraph** of  $G$  is a graph  $G'$  with a subset of vertices and a subset of edges
- $G' = (V', E')$  where  $V' \subseteq V, E' \subseteq E$
- Examples:



# Spanning Tree of a Graph

- T is a **spanning tree** of a graph  $G = (V, E)$  if
  - It is a **subgraph** of G
  - It has **all** the vertices V of G
  - All the vertices are **connected**
  - It has **no cycles**

*Claim:* If T is a spanning tree of G, then it has exactly  $|V| - 1$  edges

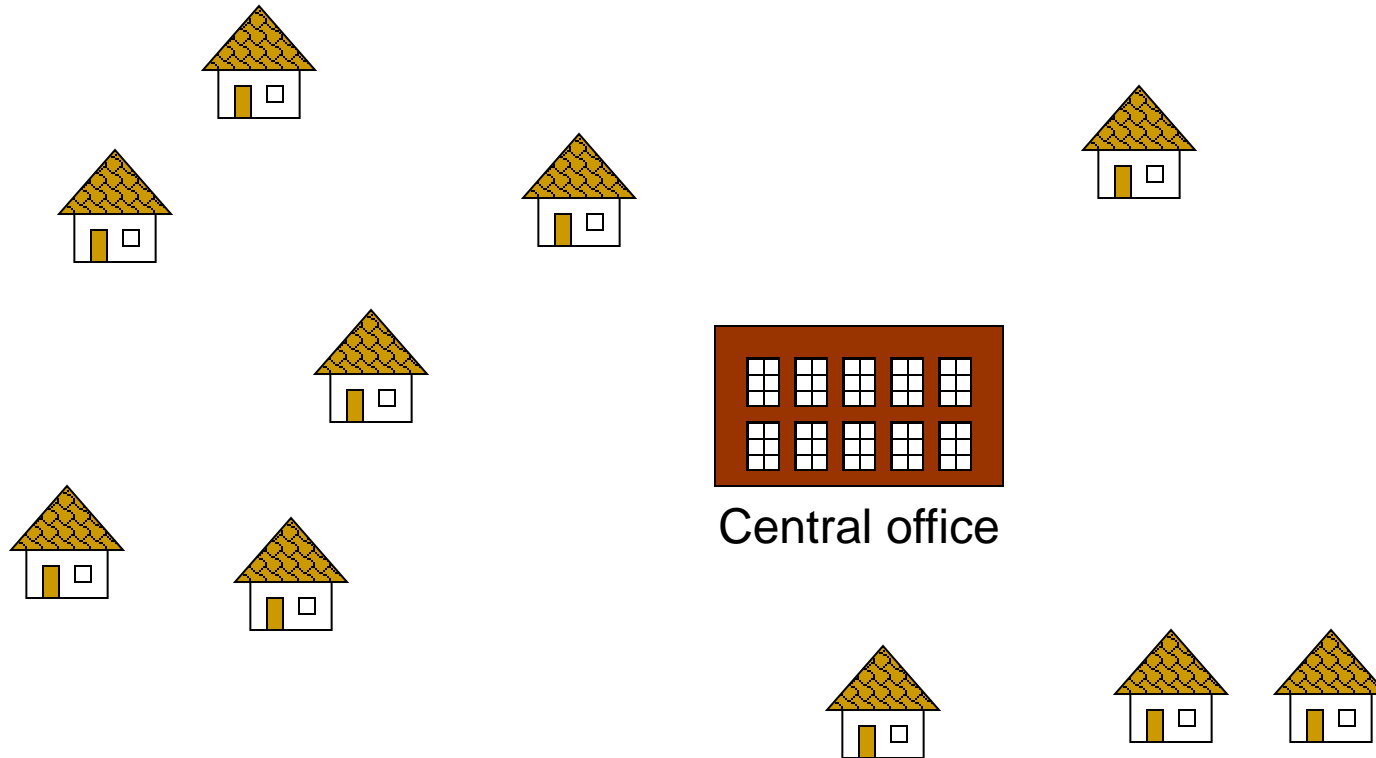
*Proof:* A tree with n vertices has n-1 edges

# Minimum Spanning Tree

- Given a undirected weighted graph  $G = (V, E, W)$  where  $W: E \rightarrow R^+$
- Cost of a spanning tree  $T = (V, E')$  is given by 
$$W(T) = \sum_{e \in E'} W(e)$$
- $T$  is the minimum cost spanning tree if and only iff  $T$  is a spanning tree of  $G$  and has the minimum cost

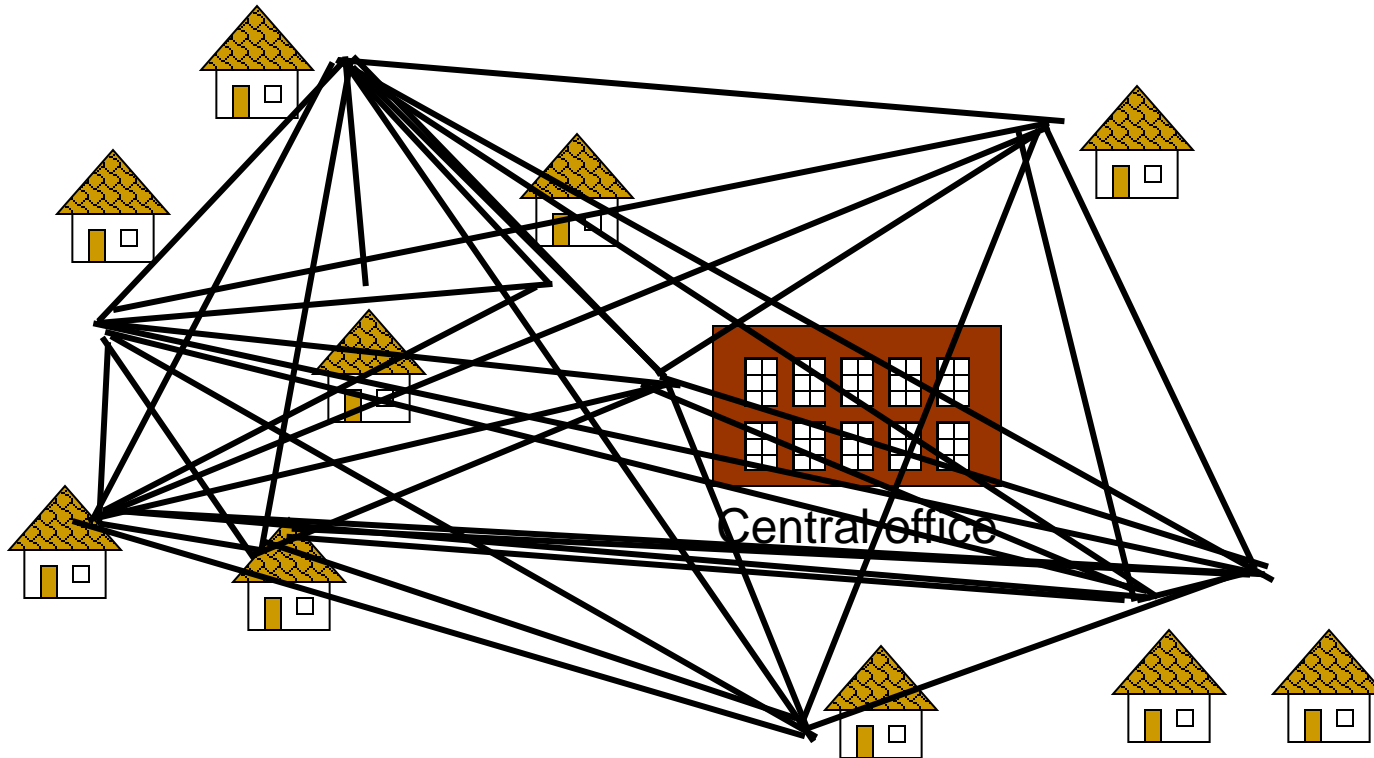


# Problem: Laying Telephone Wire



Can it be formulated as a MST problem?  
What will be the corresponding graph?

# Problem: Laying Telephone Wire



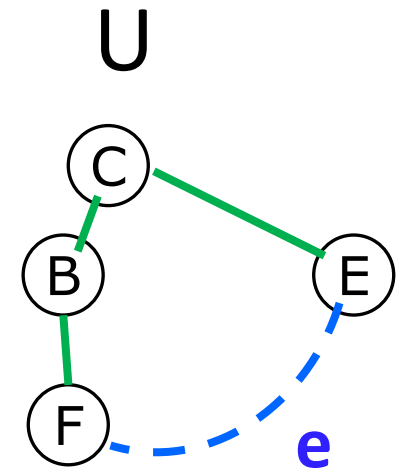
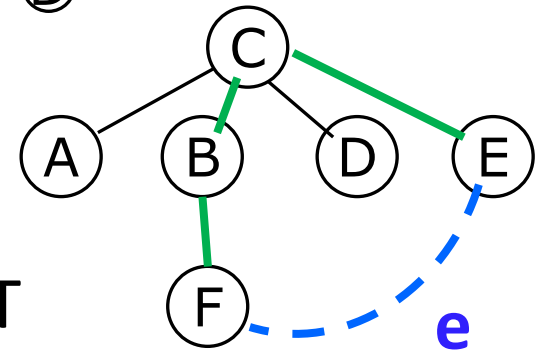
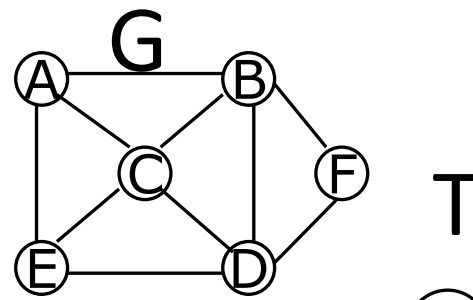
Can it be formulated as a MST problem?  
What will be the corresponding graph?

# How to Find a MST?

- Prims Algorithm for MST:  $O(|E| + |V| \log |V|)$
- Today, we will cover Kruskal's algorithm for finding the MST
- Let us revisit their properties

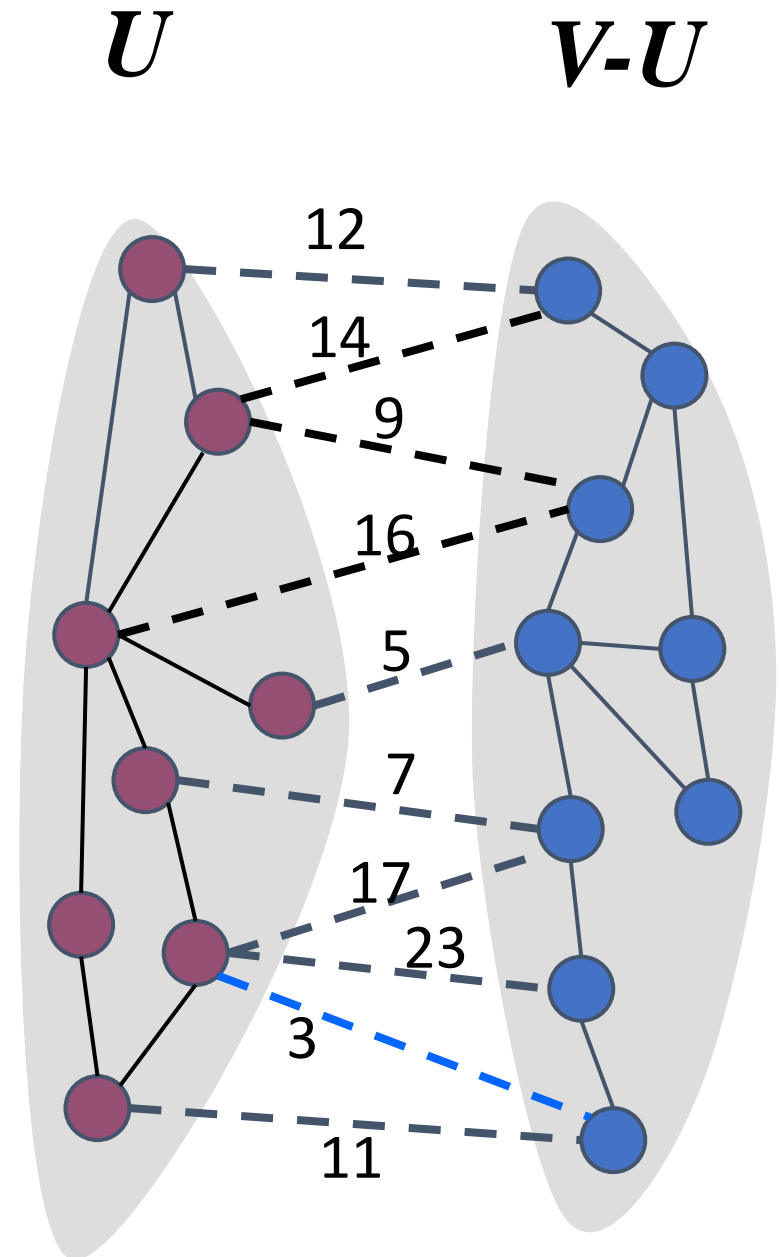
# Cycle Property

- Let **T** be a MST of a weighted undirected weighted graph **G**
- Let **e** be an edge of **G** that is not in **T**
- Let **U** be the cycle formed by adding **e** in **T**
- For every edge **f** of **U**,  $W(f) \leq W(e)$
- Why?
- Removing any single edge in **U** will make it a spanning tree again



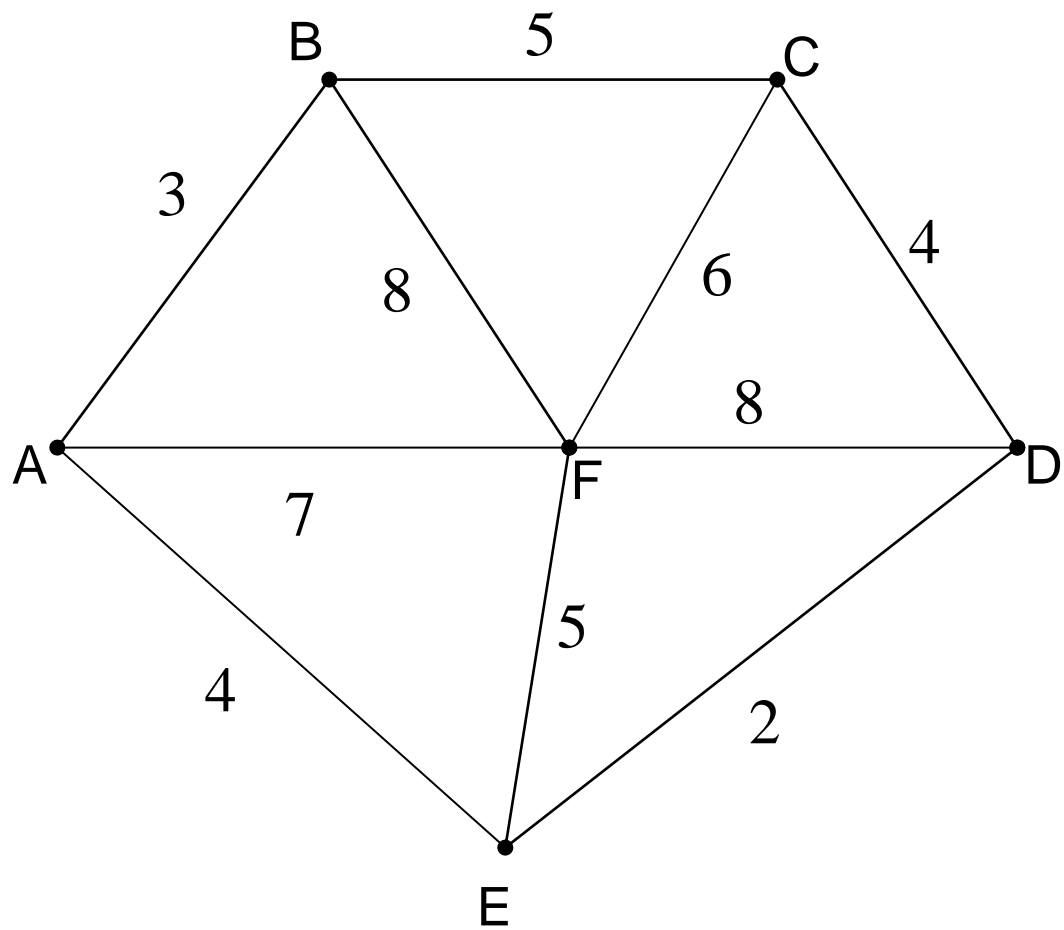
# Cut Property

- Consider a partition of the vertices of  $G$  into subsets  $U$  and  $V-U$
- Let  $e$  be an edge of minimum weight across  $(U, V-U)$
- There is a minimum spanning tree of  $G$  containing edge  $e$

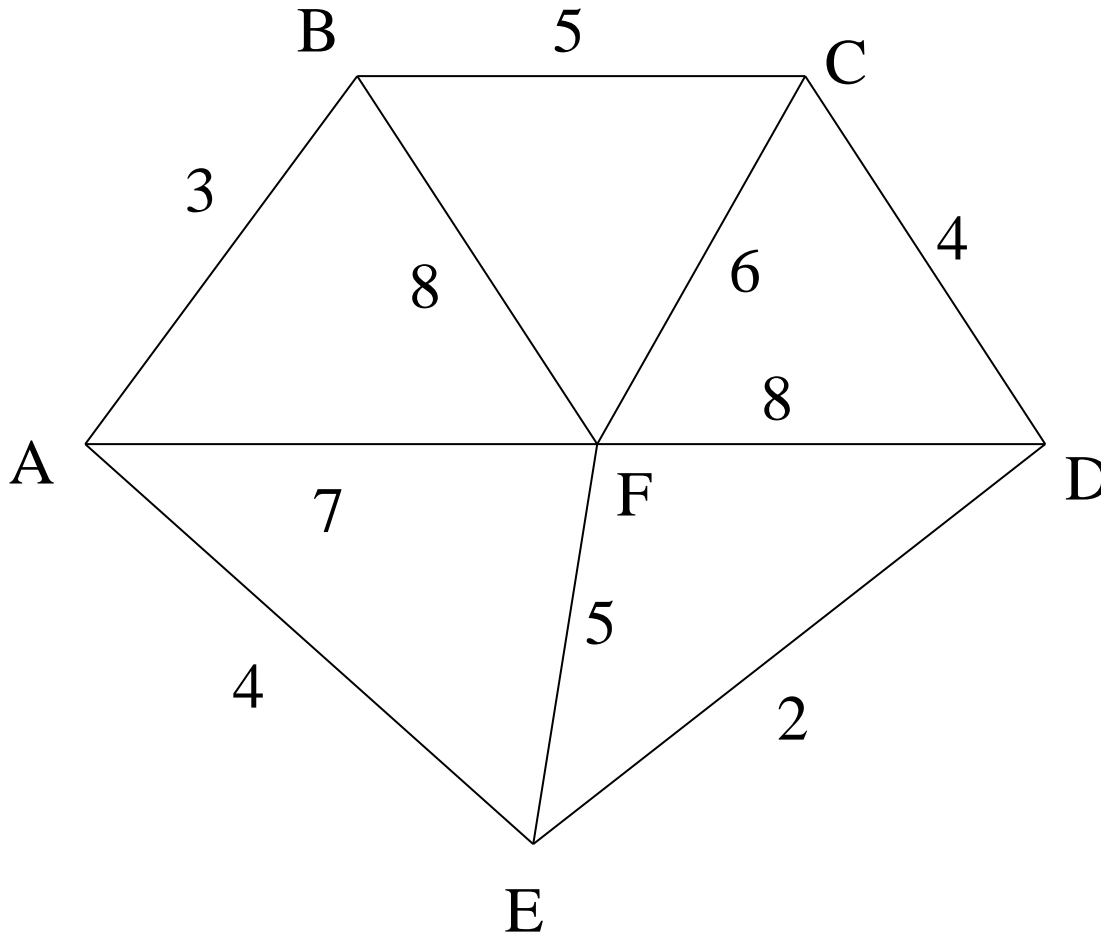


# Outline of Kruskal's Algorithm for Finding MST

- Sort the edges in ascending order of their costs
- Keep on adding the edge to the forest in this order
- As long as they don't make a cycle
- After adding  $|V|-1$  edges we have the MST
- Why?



# Kruskal's Algorithm



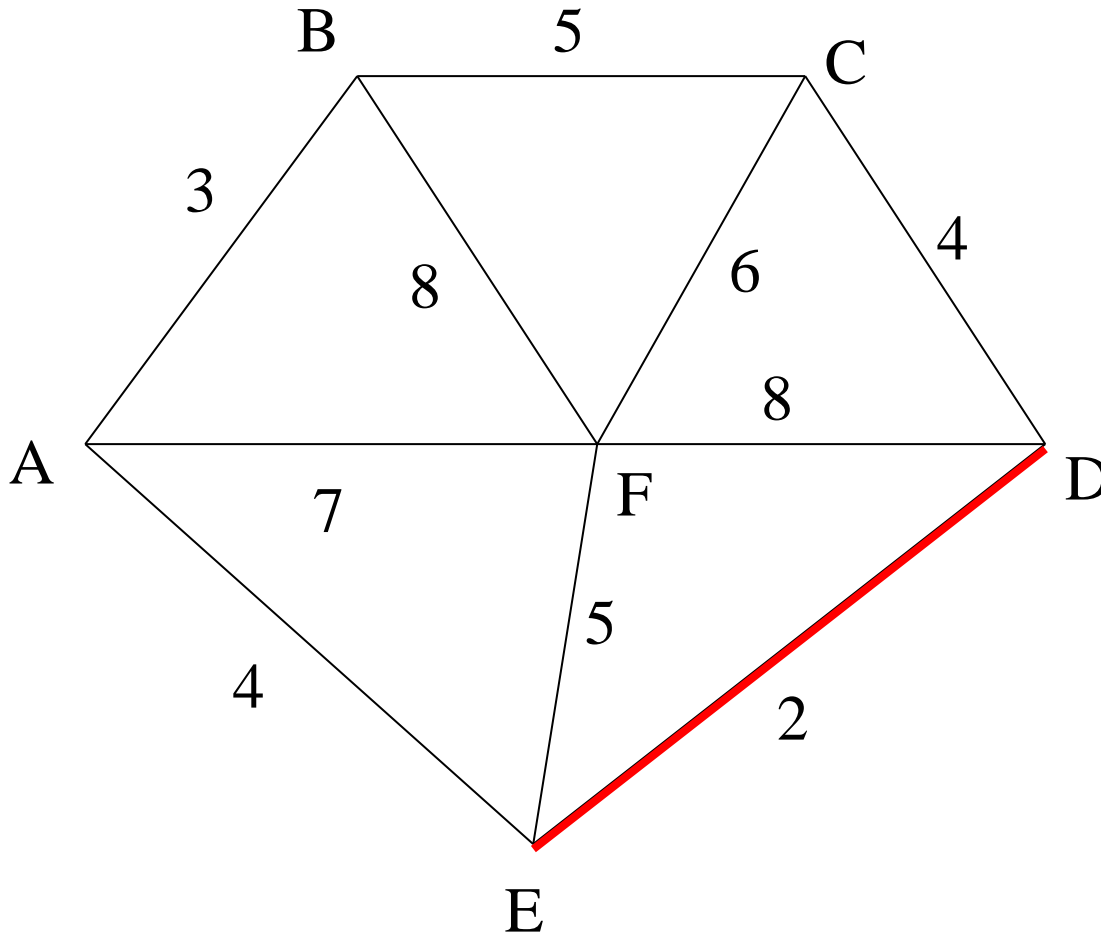
List the edges in increasing costs:

ED	2
AB	3
AE	4
CD	4
BC	5
EF	5
CF	6
AF	7
BF	8
CF	8



# Kruskal's Algorithm

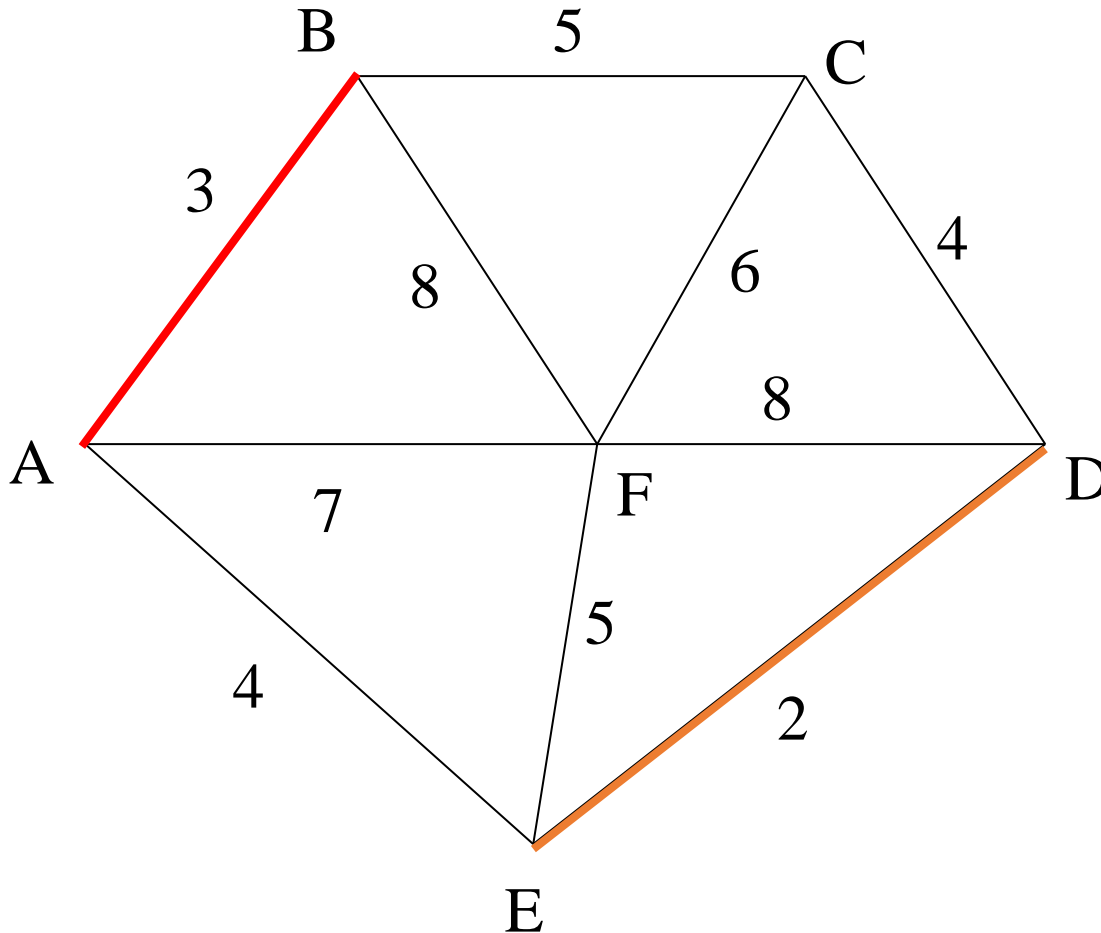
Select the shortest edge in the network



**ED 2**

# Kruskal's Algorithm

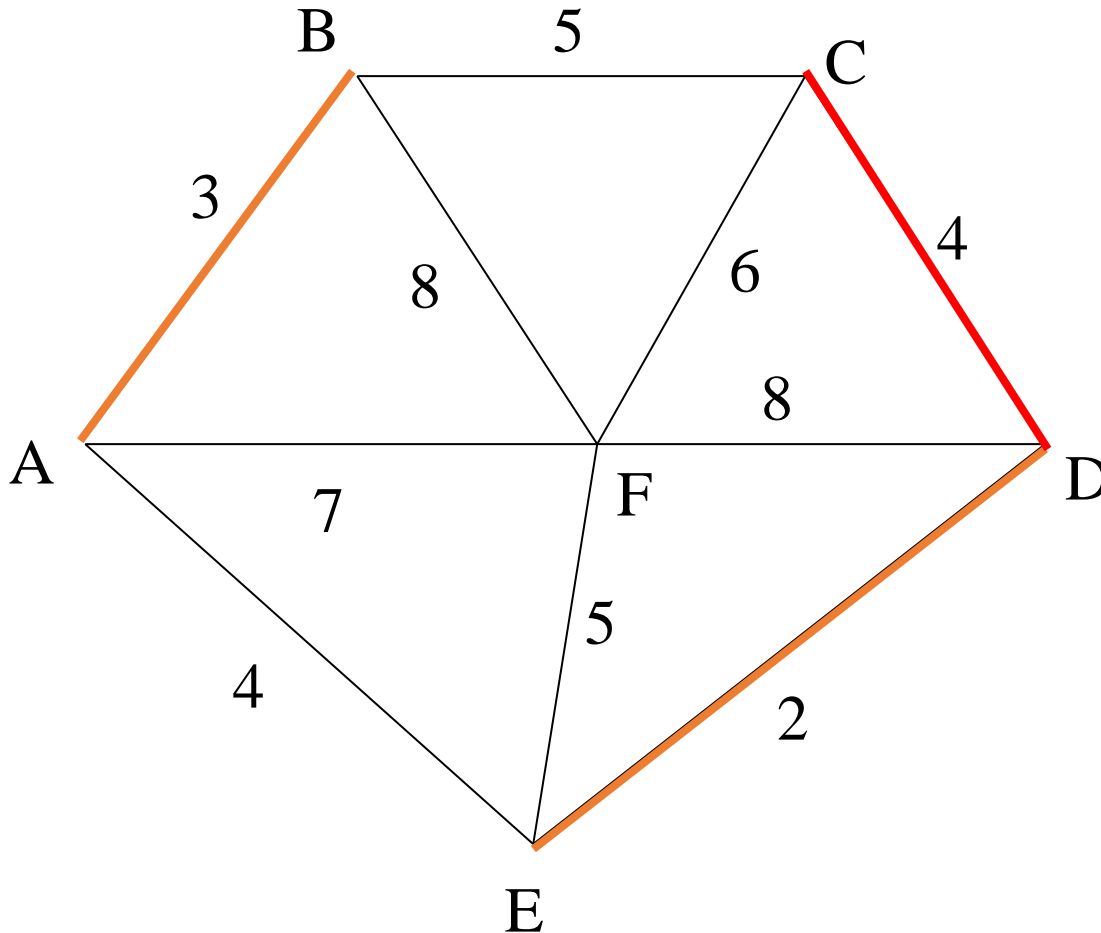
Select the next shortest edge which does not create a cycle



**ED 2**

**AB 3**

# Kruskal's Algorithm



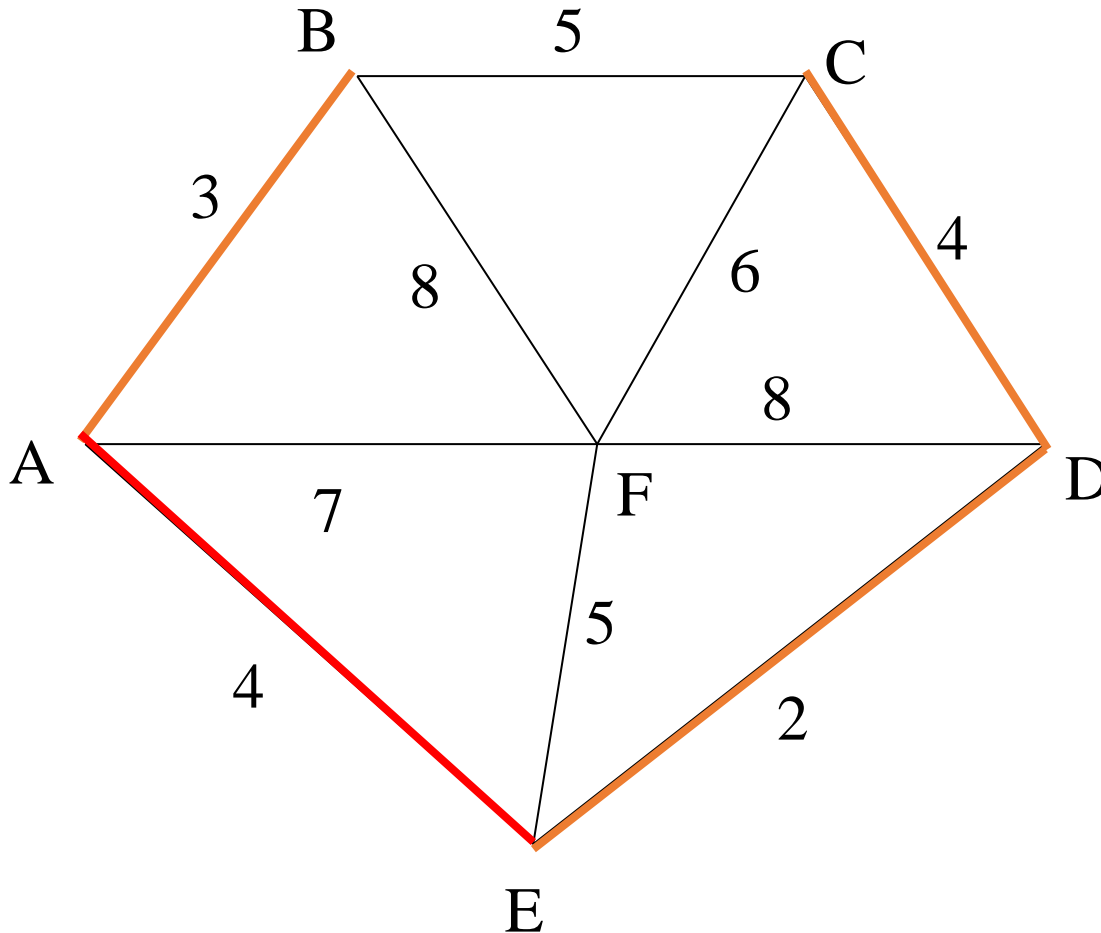
Select the next shortest edge which does not create a cycle

**ED 2**

**AB 3**

**CD 4 (or AE 4)**

# Kruskal's Algorithm



Select the next shortest edge which does not create a cycle

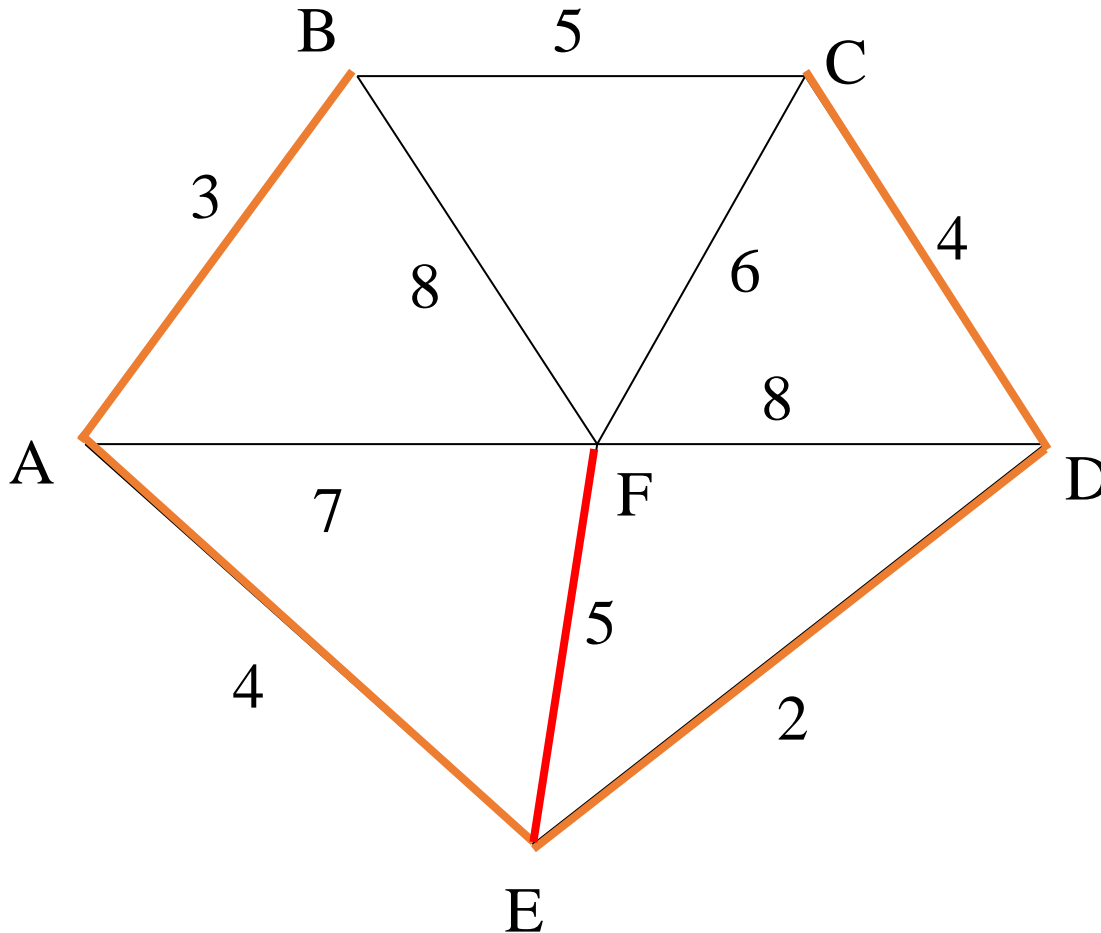
**ED 2**

**AB 3**

**CD 4**

**AE 4**

# Kruskal's Algorithm



Select the next shortest edge which does not create a cycle

**ED 2**

**AB 3**

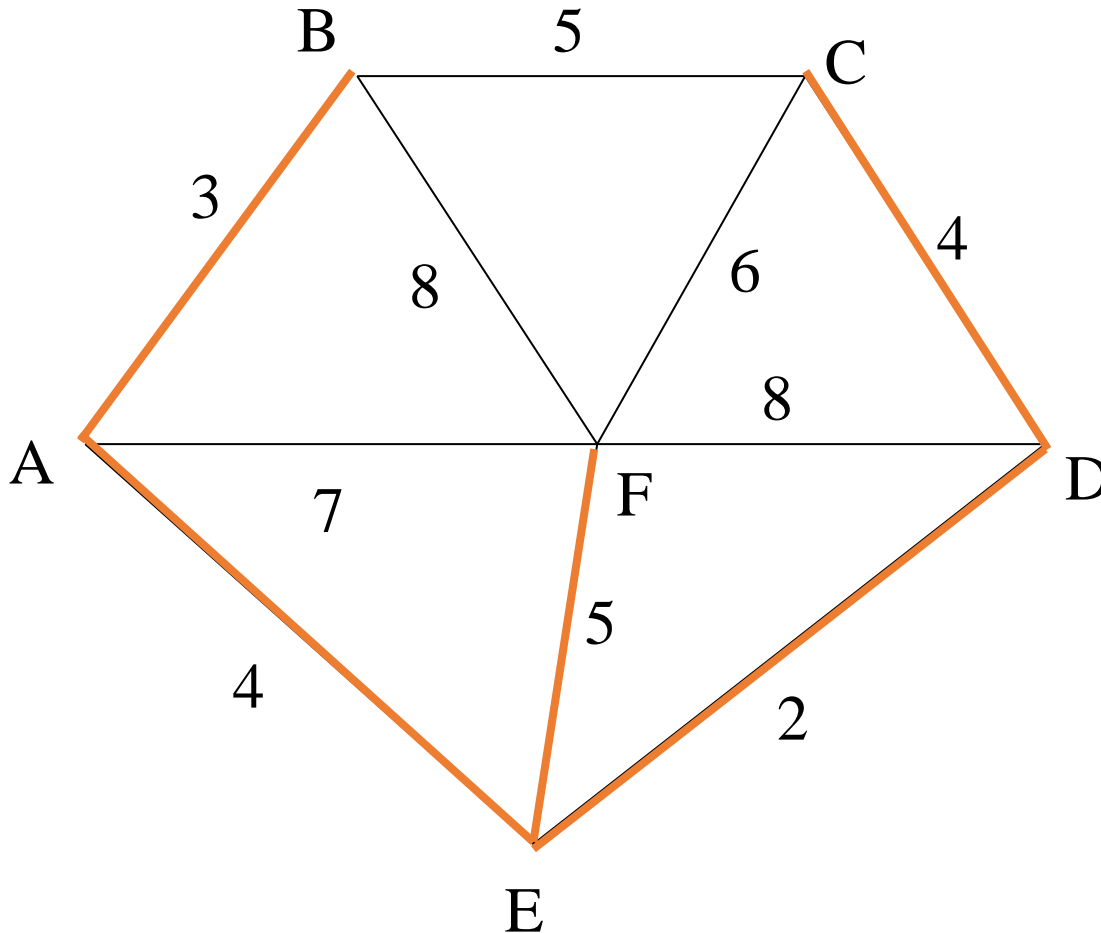
**CD 4**

**AE 4**

**BC 5 – forms a cycle**

**EF 5**

# Kruskal's Algorithm



All vertices have been connected.

The solution is

**ED 2**  
**AB 3**  
**CD 4**  
**AE 4**  
**EF 5**

Total weight of tree: 18

# Kruskal's MST Algorithm

Input: Weighted undirected graph  $G = (V, E, W)$

Output: A MST  $T$  of the graph

1.  $T \leftarrow \emptyset$
2. Sort  $E$  into non-decreasing order by  $w$  }  $O(|E|\log|E|)$
3. **for** each  $(u, v)$  taken from the sorted  $E$  **do**
4.     **if**  $(T \cup \{(u, v)\})$  does not form a cycle) ?
5.          $T \leftarrow T \cup \{(u, v)\}$  }  $O(1) * (|V|-1)$
6.     **end for**
7. **return**  $T$

# How to Check for Cycle in T

Input: Weighted undirected graph  $G = (V, E, W)$

Output: A MST  $T$  of the graph

1.  $T \leftarrow \emptyset$
2. Sort  $E$  into non-decreasing order by  $w$  }  $O(|E|\log|E|)$
3. for each  $(u, v)$  taken from the sorted  $E$  do
4.     if  $(T \cup \{(u, v)\})$  does not form a cycle)  $|E|$  times
5.          $T \leftarrow T \cup \{(u, v)\}$  }  $O(1) * (|V|-1)$
6. end for
7. return  $T$



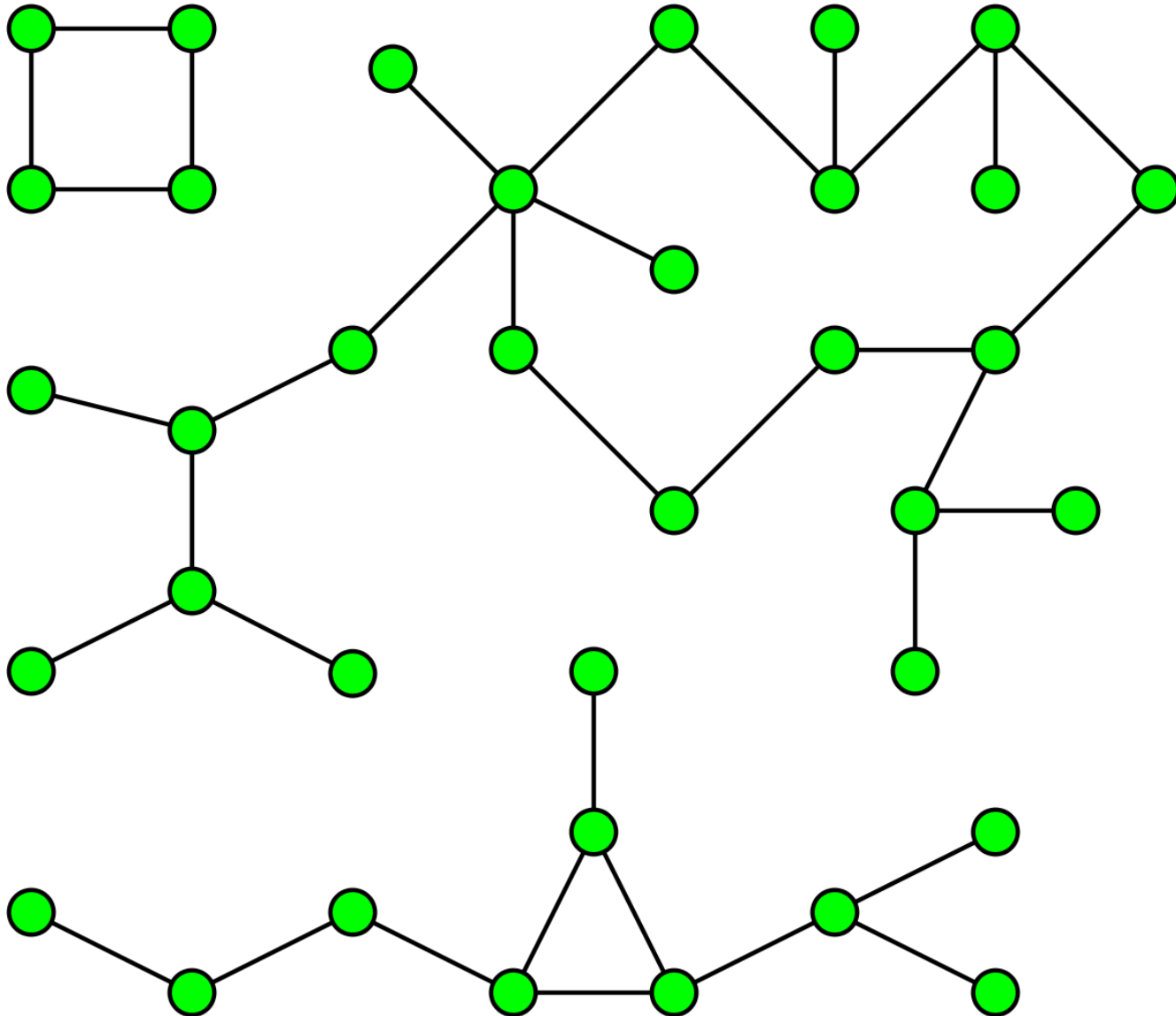
# How to Efficiently Check for Cycle in a Graph?

- Need not start from **scratch** each time
- We need an **incremental** algorithm
- Can maintain some data structures corresponding to  $T$
- At every step,  $T$  **changes** by union of 1 edge
- Update the data structures **efficiently**

# Connected Components in an Undirected Graph

- **Connectivity** induces an **equivalence** relationship on the graph nodes
- **Reflexivity**:  $v$  is connected to  $v$  for all  $v$
- **Symmetry**: if  $u$  is connected to  $v$  then  $v$  is connected to  $u$
- **Transitivity**: if  $u$  is connected to  $v$  and  $v$  is connected to  $w$ , then  $u$  is also connected to  $w$
- The set of vertices are partitioned into **connected components**

# How Many Connected Components?



# Can Represent Using Sets

- Operation needed
  - $S = \text{Initialize}(u)$ : Initializes the set with singleton vertex  $u$
  - $S = \text{Find}(u)$ : Returns the set id of the vertex  $u$
  - $S = \text{Union}(S1, S2)$ : Takes the union of the two sets

# Implementing Union Find using Heaps

- $S = \text{Find}(u)$ : Returns the set id of the vertex  $u$
- $S = \text{Union}(S1, S2)$ : Takes the union of the two sets
- $S = \text{Initialize}(u)$ : Initializes the set with vertex  $u$
- Implement each set as a separate heap
- Smallest vertex id at the root = identity of the set
- $\text{find}(u)$ : traverse to the root and return the id of root
  - $O(\log(|V|))$
- $\text{union}(S1, S2)$ : May use binomial heaps
  - Or make smaller height tree the child of the larger height tree
  - $O(1)$

# Kruskal's MST Algorithm

1.  $T \leftarrow \emptyset$
2. **for** each vertex  $v \in V$
3.     **do** initialize( $v$ ) }  $O(V)$
4.     Sort  $E$  into non-decreasing order by  $w$  }  $O(|E| \log |E|)$
5.     **for** each  $(u, v)$  taken from the sorted list }  $\leftarrow O(|E|)$
6.     **do if** find( $u$ )  $\neq$  find( $v$ )
7.         **then**  $T \leftarrow T \cup \{(u, v)\}$  }  $\leftarrow O(\log |V|)$
8.         union(find( $u$ ), find( $v$ ))
9. **return**  $T$

Total runtime:  $O(|E| \log |E| + |E| \log |V|) = O(|E| \log |E|)$

# Kruskal's Algorithm Invariant

- After each iteration, every tree in the forest is a MST of the vertices it connects
- Algorithm terminates when all vertices are connected into one tree

# Correctness of Kruskal's

*Lemma:* If the input graph was connected, then Kruskal's algorithm ends up finding a spanning tree

*Proof:* By contradiction. Assume Kruskal's algo doesn't return a spanning tree.

1. The algorithm starts with singleton vertices (no edges) and never adds an edge that results into a cycle. So in the worst case, the algorithm might have returned a forest.
2. Let a component  $U_1$  be disconnected from a component  $U_2$
3. Since, the original graph was connected, there must be an edge  $(u, v)$  between  $U_1$  and  $U_2$ .
4. This edge must have been considered by Kruskal's algorithm. In the final result,  $u$  and  $v$  were disconnected, they must be disconnected even at the stage when this edge was considered
5. Adding this edge could not have created a cycle.  
Contradiction!!



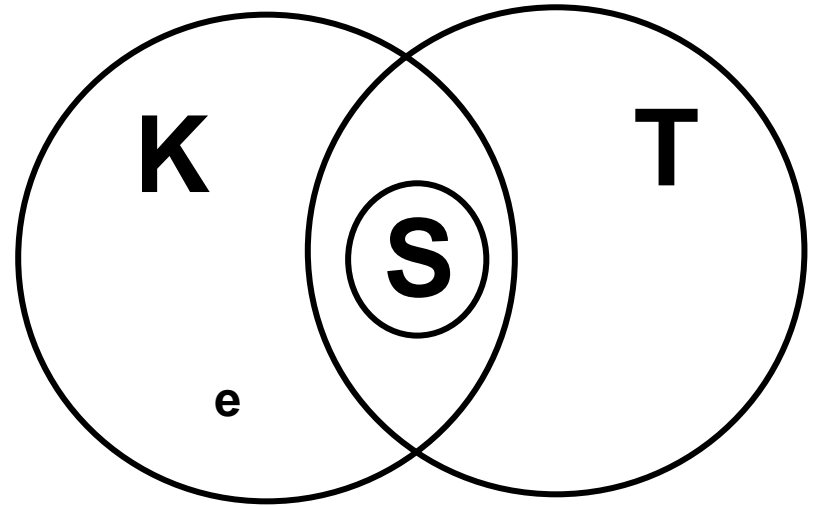
# Correctness of Kruskal's

But is this a *minimum* spanning tree?

Suppose it wasn't.

- There must be point at which it fails, and in particular there must a single edge whose insertion first prevented the spanning tree from being a minimum spanning tree.

# Correctness of Kruskal's



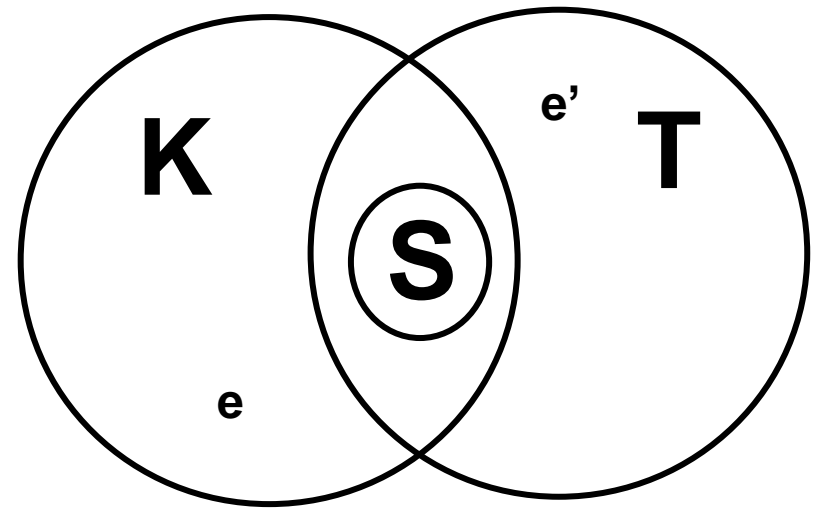
- Let **e** be this first errorful edge.
- Let **K** be the Kruskal spanning tree
- Let **S** be the set of edges chosen by Kruskal's algorithm *before* choosing **e**
- Let **T** be a MST containing all edges in **S**, but not **e**.

# Correctness of Kruskal's

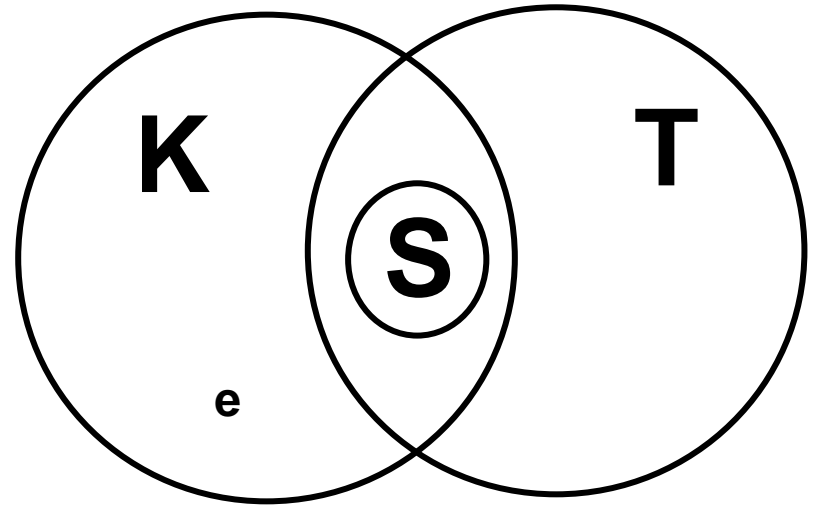
Lemma:  $w(\mathbf{e}') \geq w(\mathbf{e})$  for all edges  $\mathbf{e}'$  in  $\mathbf{T} - \mathbf{S}$

**Proof** (*by contradiction*):

- Assume there exists some edge  $\mathbf{e}'$  in  $\mathbf{T} - \mathbf{S}$ ,  $w(\mathbf{e}') < w(\mathbf{e})$
  - Kruskal's must have considered  $\mathbf{e}'$  before  $\mathbf{e}$
  - $\mathbf{e}'$  cannot be in  $\mathbf{K}$  else it would have been in  $\mathbf{S}$
  - However, since  $\mathbf{e}'$  is not in  $\mathbf{K}$ , it must have been discarded because it caused a cycle with some of the other edges only in  $\mathbf{S}$ .
  - But  $\mathbf{e}' + \mathbf{S}$  is a subgraph of  $\mathbf{T}$ , which means it cannot form a cycle
- ...Contradiction**

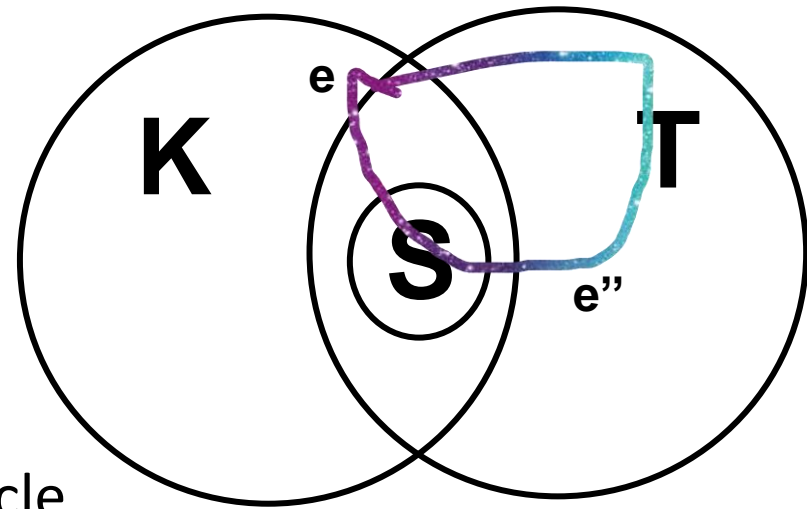


# Correctness of Kruskal's



- Let **e** be this first errorful edge.
- Let **K** be the Kruskal spanning tree
- Let **S** be the set of edges chosen by Kruskal's algorithm *before* choosing **e**
- Let **T** be a MST containing all edges in **S**, but not **e**.

# Correctness of Kruskal's



- Inserting edge  $e$  into  $T$  will create a cycle
- There must be an edge on this cycle which is not in  $K$  (*why??*). Call this edge  $e''$
- $e''$  must be in  $T - S$ , so (by our lemma)  $w(e'') \geq w(e)$
- We could form a new spanning tree  $T'$  by swapping  $e$  for  $e''$  in  $T$  (*prove this is a spanning tree*).
- $w(T')$  is clearly no greater than  $w(T)$
- But that means  $T'$  is a MST
- And yet it contains all the edges in  $S$ , and also  $e$

...Contradiction

Thank You