

# COL106

# Data Structures and Algorithms

Subodh Sharma and Rahul Garg

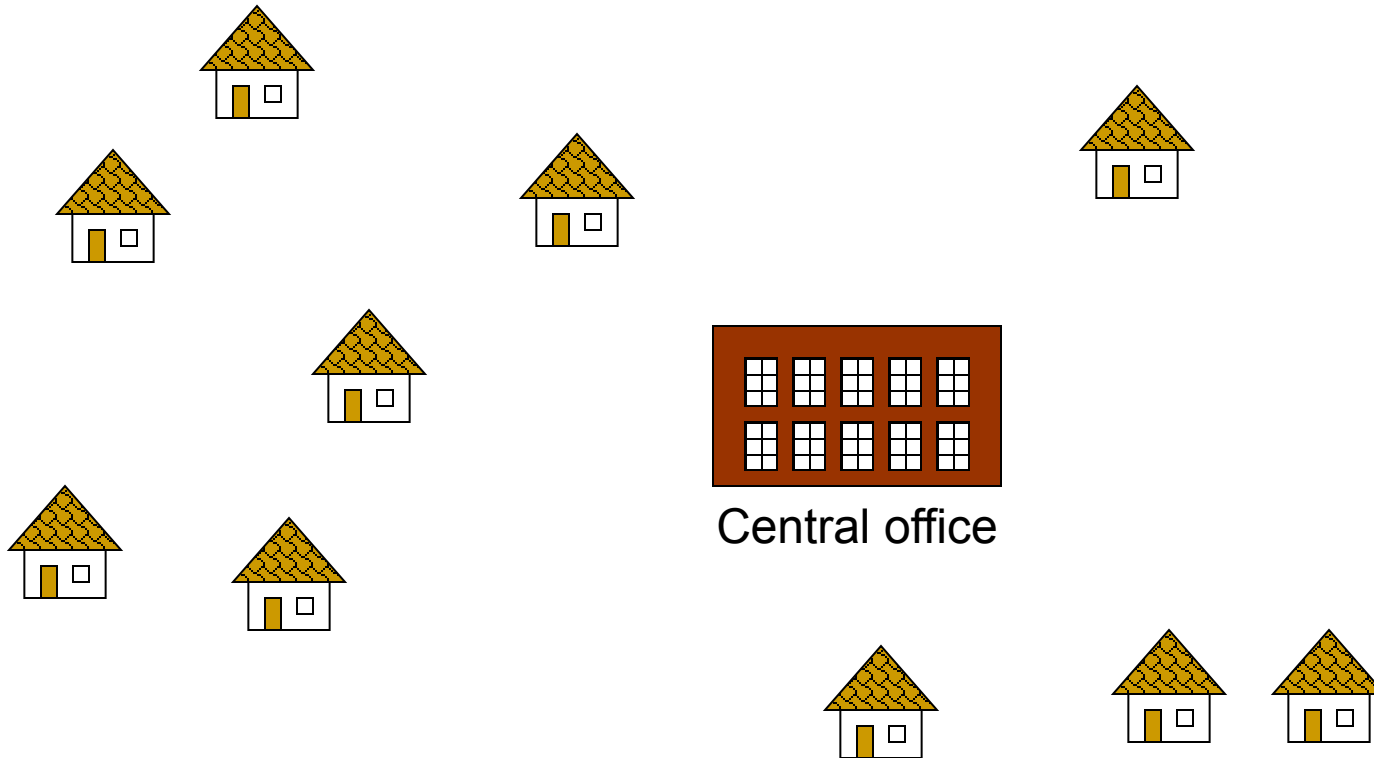
# Announcements

- The lab attendance has gone down significantly
- All TAs will not be coming to the labs now
- Instructors will not be coming to the labs anymore
- Use the labs to discuss doubts and help with the assignments

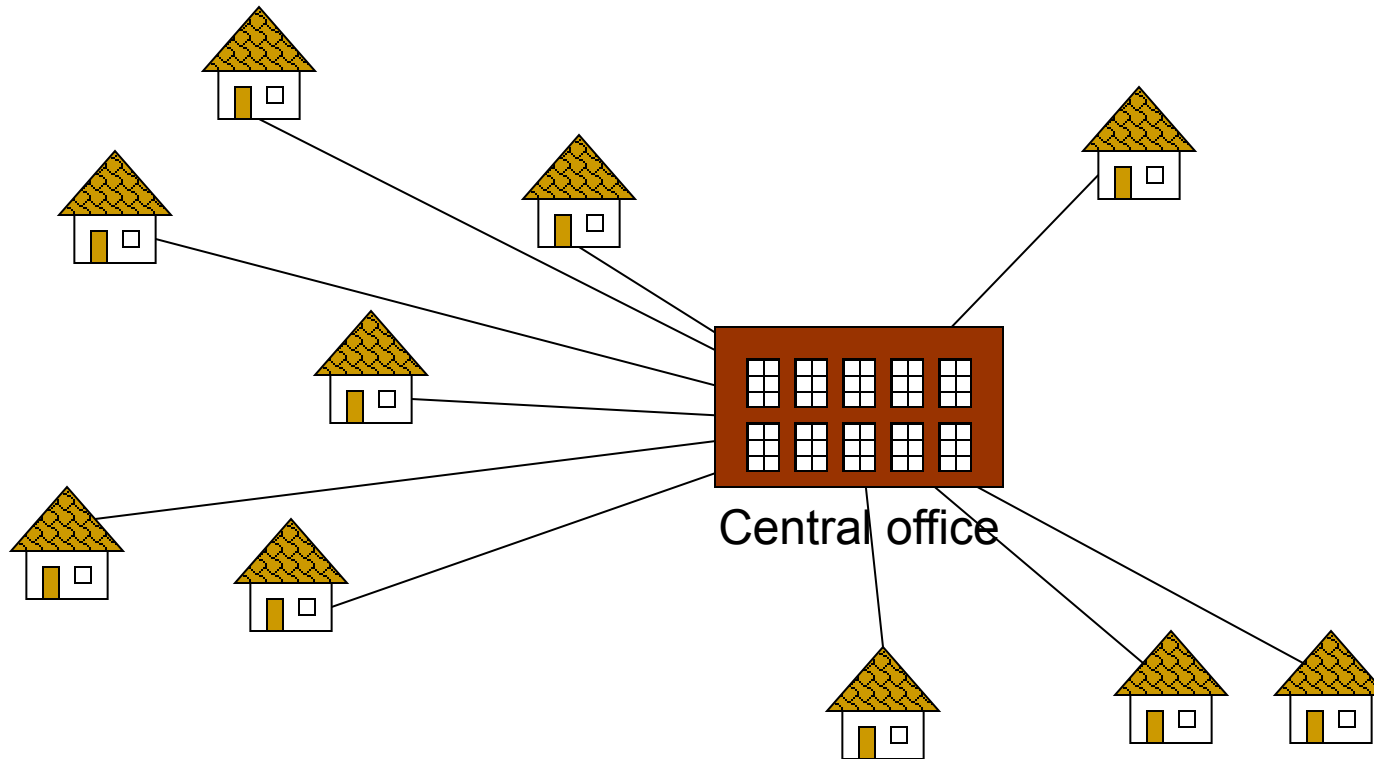
# Minimum Spanning Trees

Based on slides by: Longin Jan Latecki, Temple University,  
George Bebis, University of Nevada, Reno (UNR)

# Problem: Laying Telephone Wire

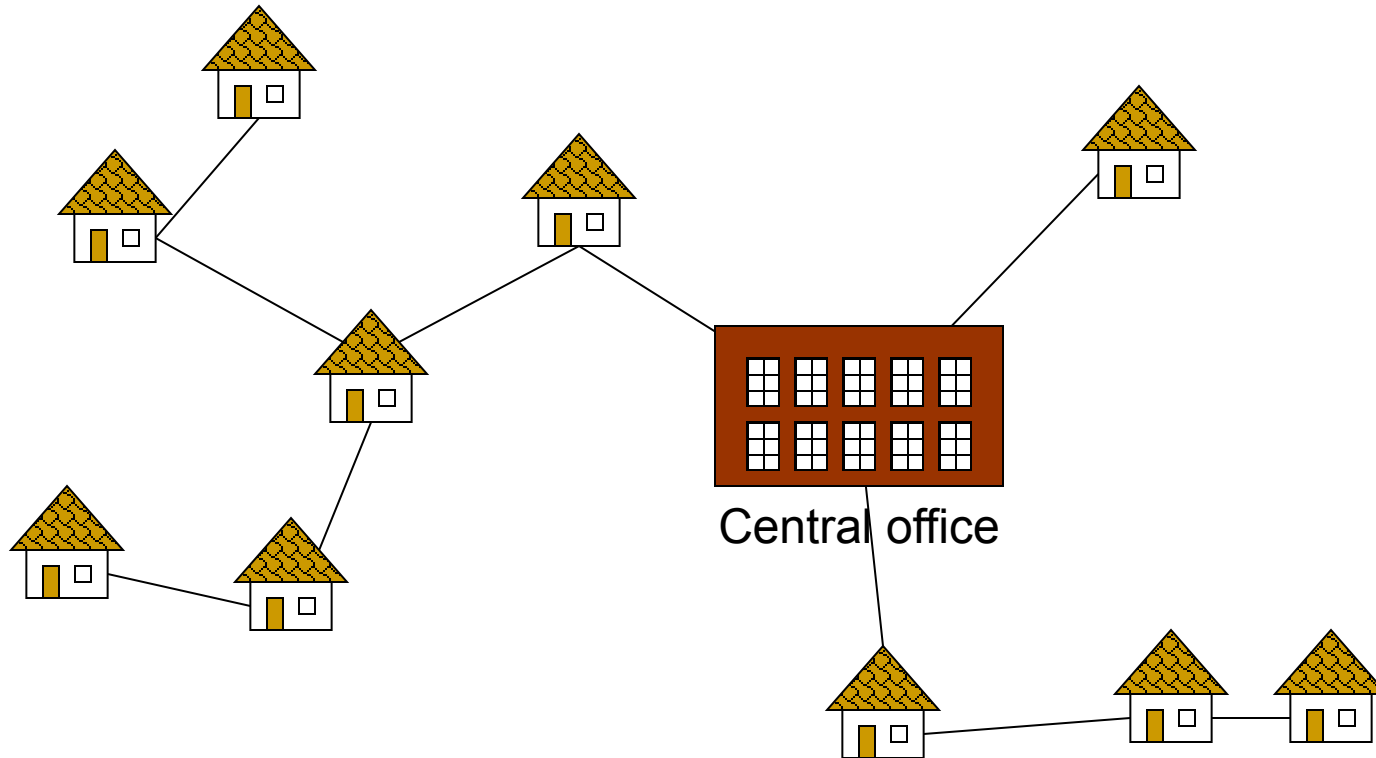


# Wiring: Naive Approach



**Expensive!**

# Wiring: Better Approach



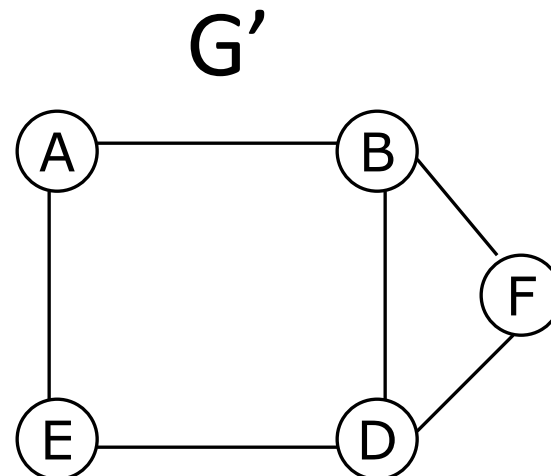
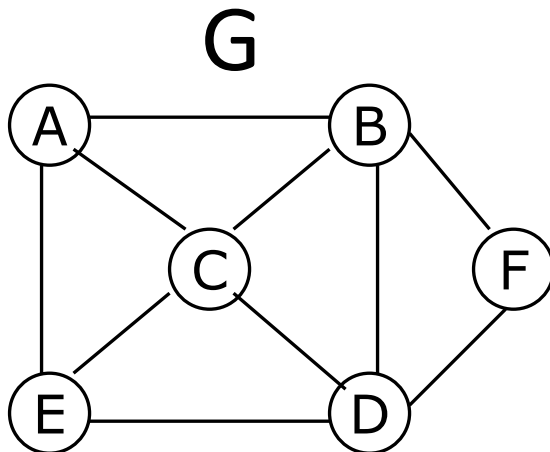
Minimize the total length of wire connecting the customers

# 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  $V' \subseteq V, E' \subseteq E$

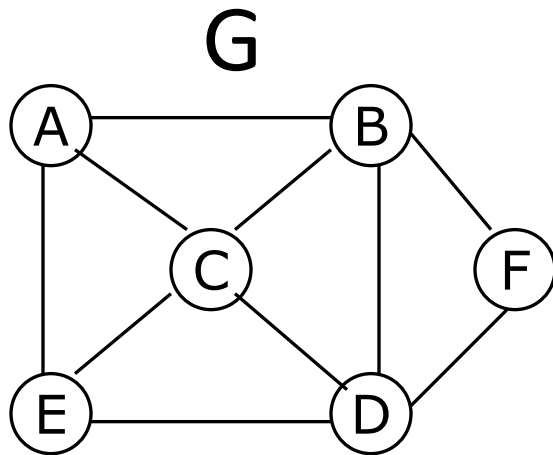
# Definitions

- 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:



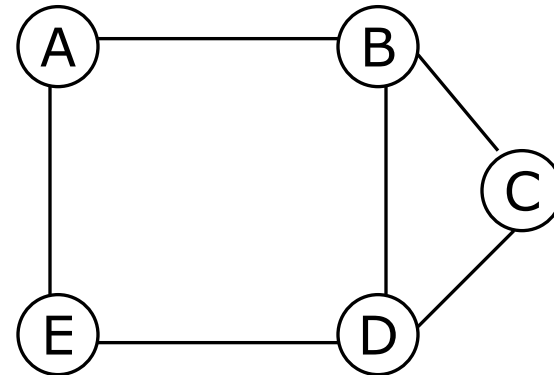


# Which Graphs are Subgraphs of G?

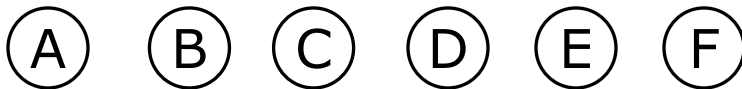


$G' = (V', E')$  is a subgraph of  $G = (V, E)$  if  $V' \subseteq V, E' \subseteq E$

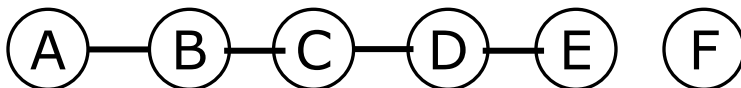
**G1**



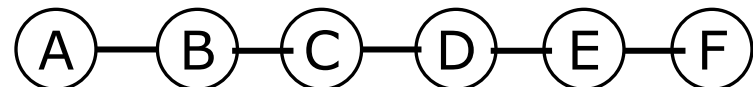
**G2**



**G3**



**G4**



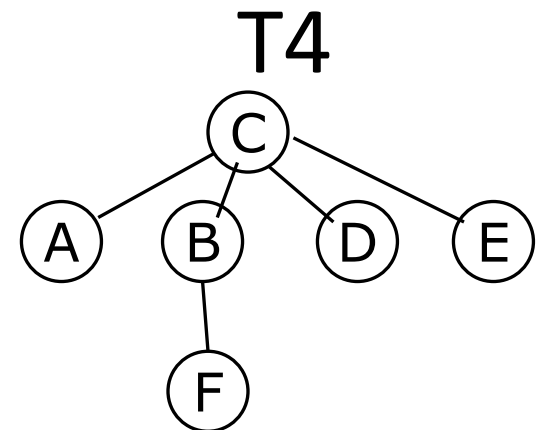
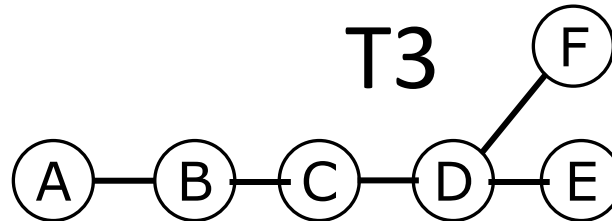
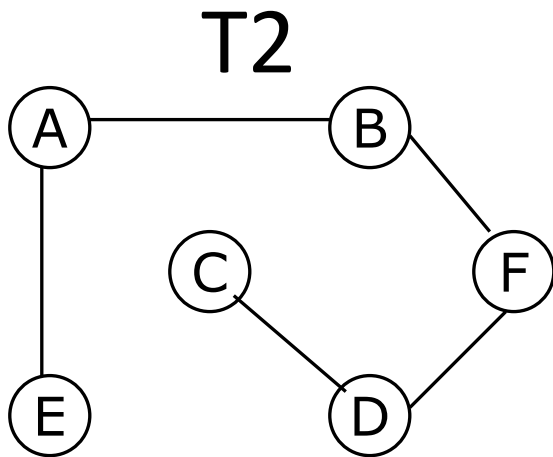
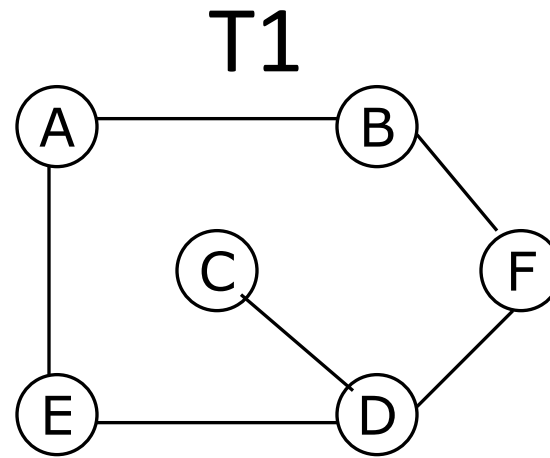
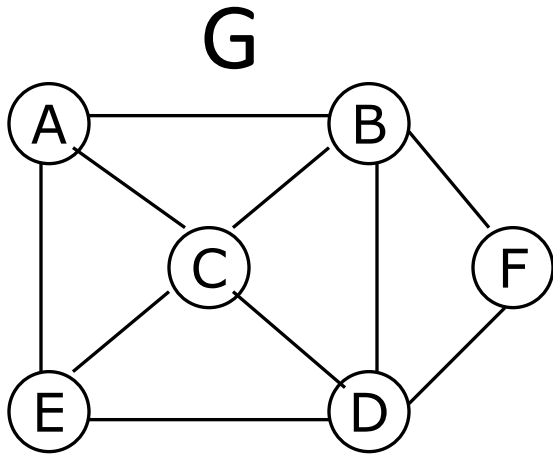
# Spanning Tree of a Graph

- 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

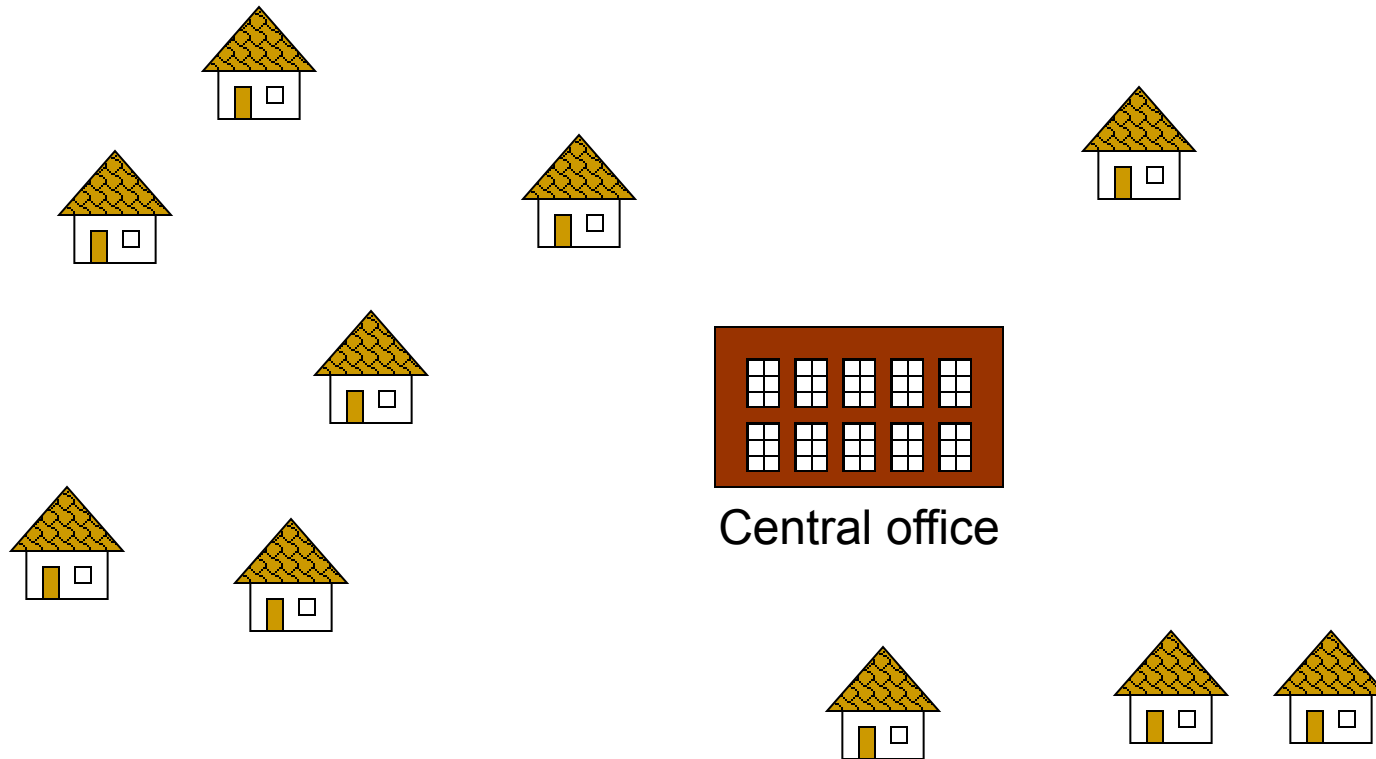
# Which are Spanning Trees of G?



# Minimum Spanning Tree

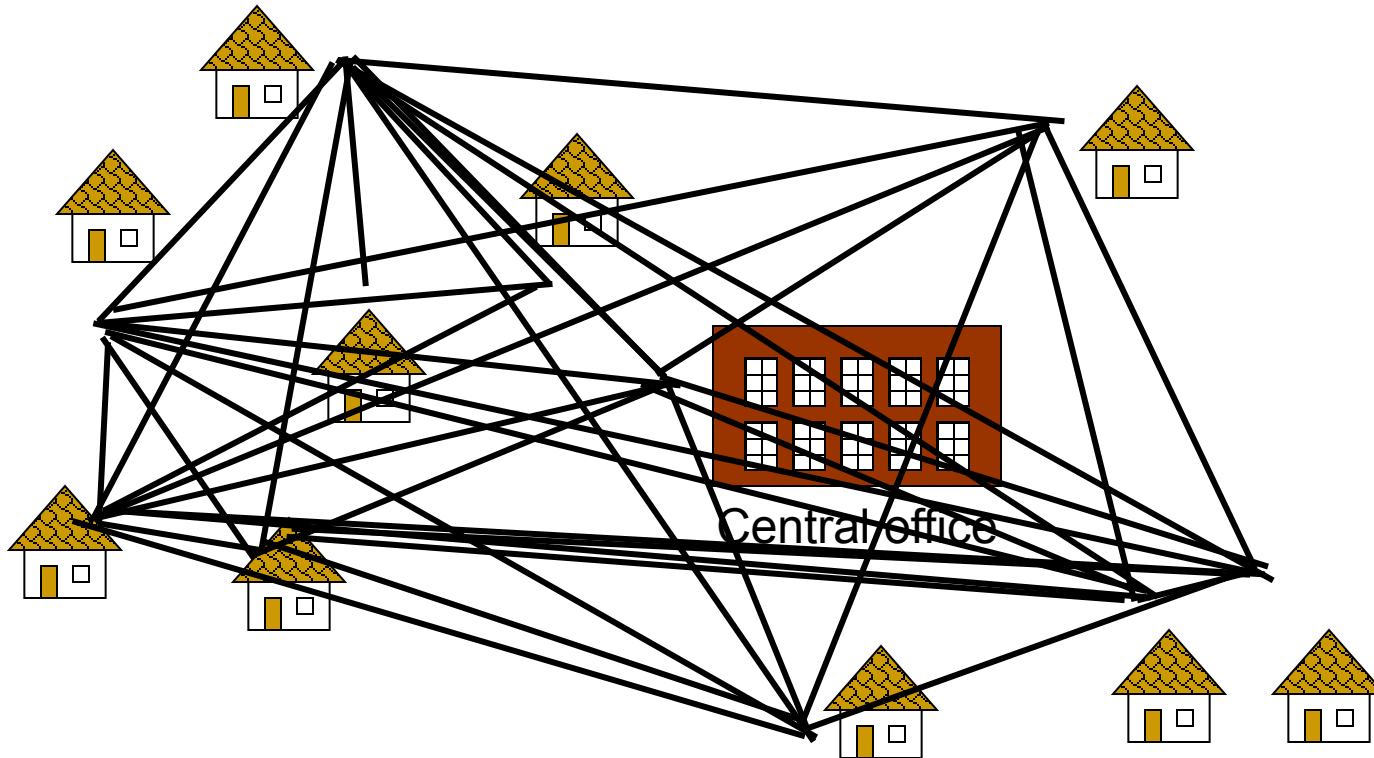
- Given a undirected weighted graph  $G = (V, E, W)$  where  $W: E \rightarrow R^+$
- Cost of a spanning tree  $T = (V, E')$  is given by  $W(T) = \sum_{e \in E'} W(e)$
- $T$  is the minimum cost spanning tree if and only iff  $T$  is a spanning tree of  $G$  and has the minimum cost

# Problem: Laying Telephone Wire



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

# Problem: Laying Telephone Wire



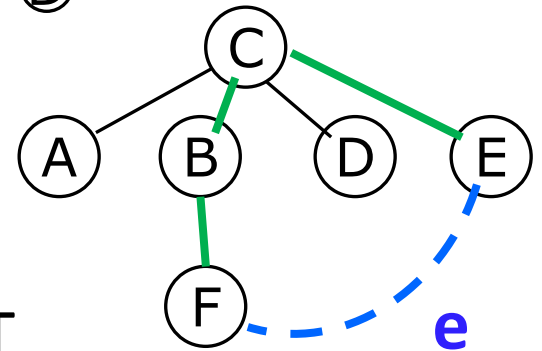
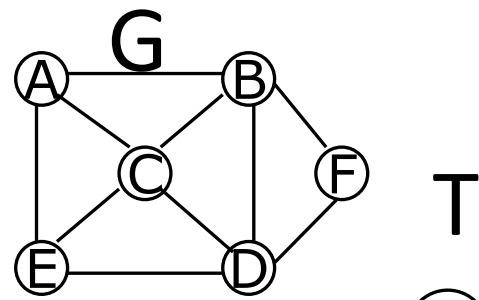
Can it be formulated as a MST problem?  
What will be the corresponding graph?

# How to Find a MST?

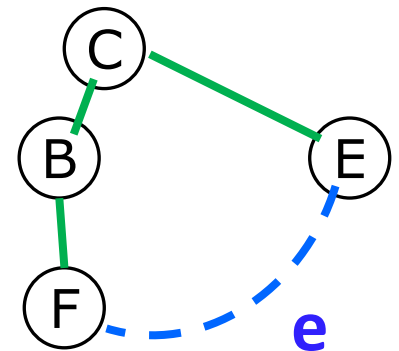
- MST need not be unique
- If the graphs is unweighted, all spanning trees are MSTs
- Why?
- Before trying to find MST let us examine some of their properties

# Cycle Property

- Let **T** be a MST of a weighted undirected 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?



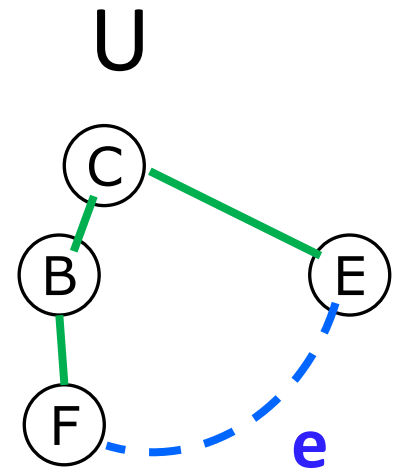
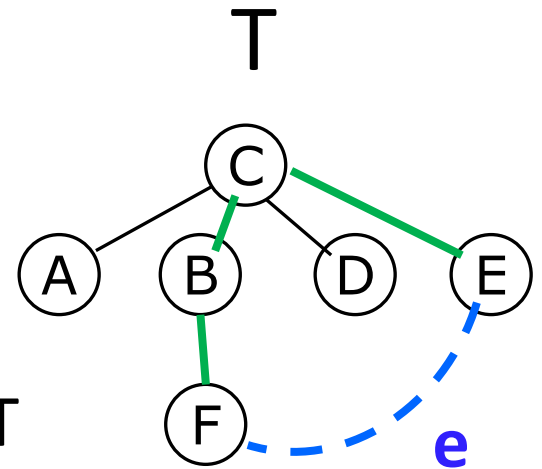
**U**





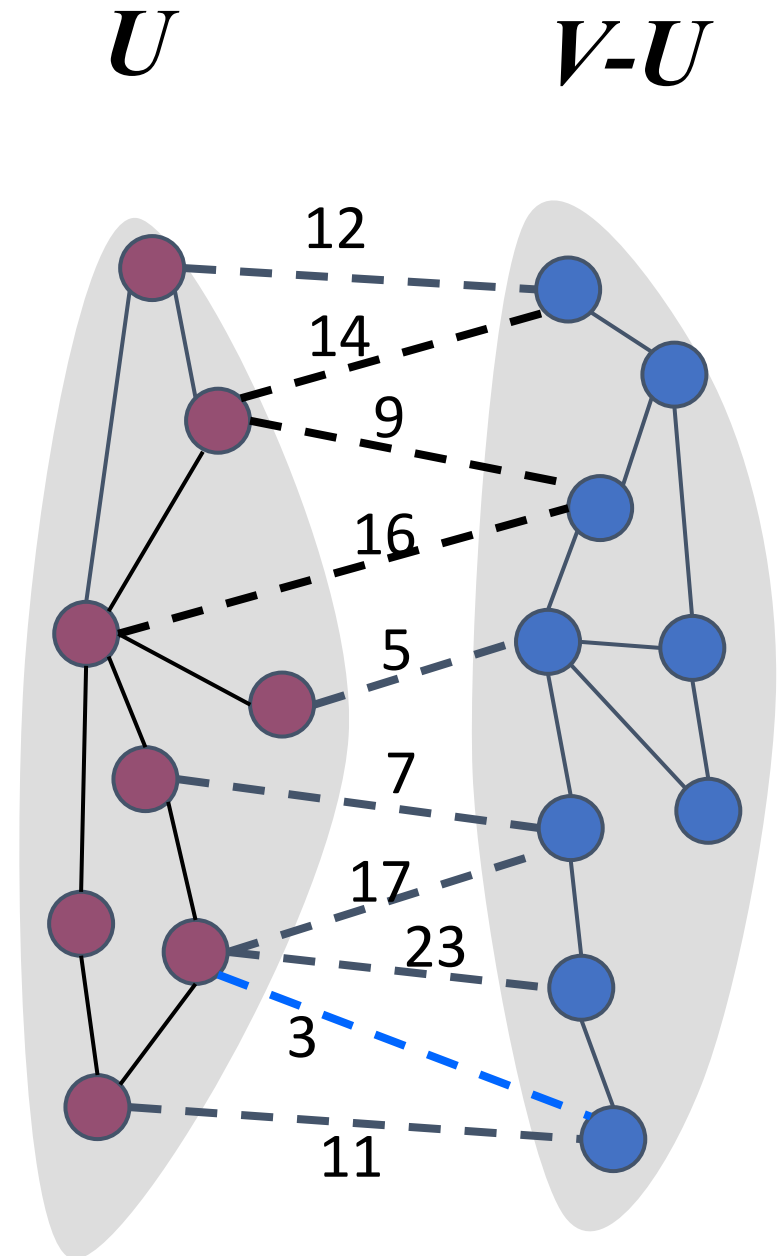
# Cycle Property

- Let **T** be a MST of a weighted undirected 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



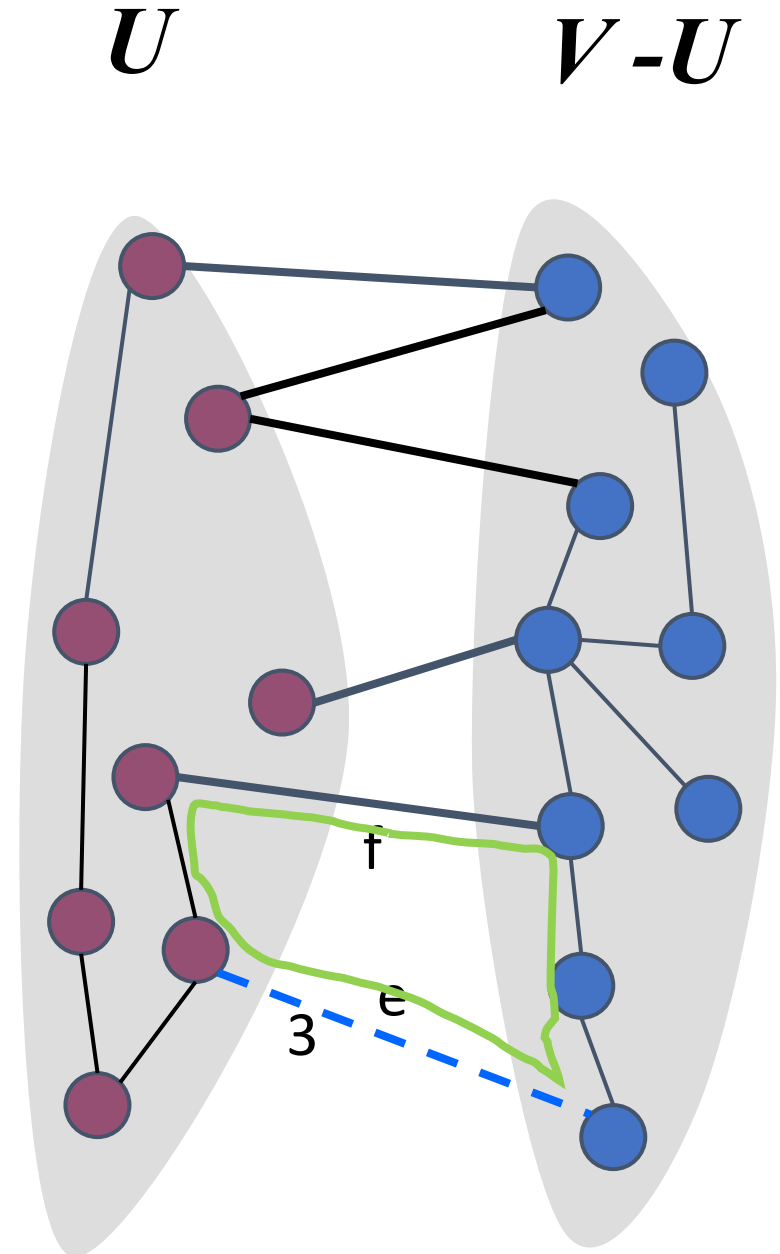
# Cut Property

- 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  $e$



# Cut Property

- $e$ : an edge of **minimum** weight across  $(U, V-U)$  cut
- *Claim:* There is a MST containing edge  $e$
- *Proof:* Let  $T$  be an MST of  $G$  without  $e$
- Consider the cycle  $C$  formed by adding  $e$  to  $T$ .
- Let  $f$  be an edge of  $C$  across the cut
- By the cycle property,  $W(f) \leq W(e)$
- Thus,  $W(f) = W(e)$
- We obtain another MST containing  $e$  by replacing  $f$  with  $e$



# Outline of Prim's Algorithm for MST

- Start with  $U$  = a single vertex  $u$
- Algorithm maintains
  - A connected set of vertices  $U$
  - Using a subset of MST edges
- At every step add the smallest cost edge  $(x, y)$  in the cut  $(U, V-U)$  to the tree
- It cannot induce a cycle
- The newly added edge must be a part of a MST (using the cut property)
- Add the vertex  $y$  to  $U$

# Prim's MST Algorithm $U$ $V-U$

Input:  $G = (V, E, W)$

Output: MST  $T$

Initialize:  $T = \emptyset$ ; Pick an arbitrary  $u \in V$ ;  $U = \{u\}$

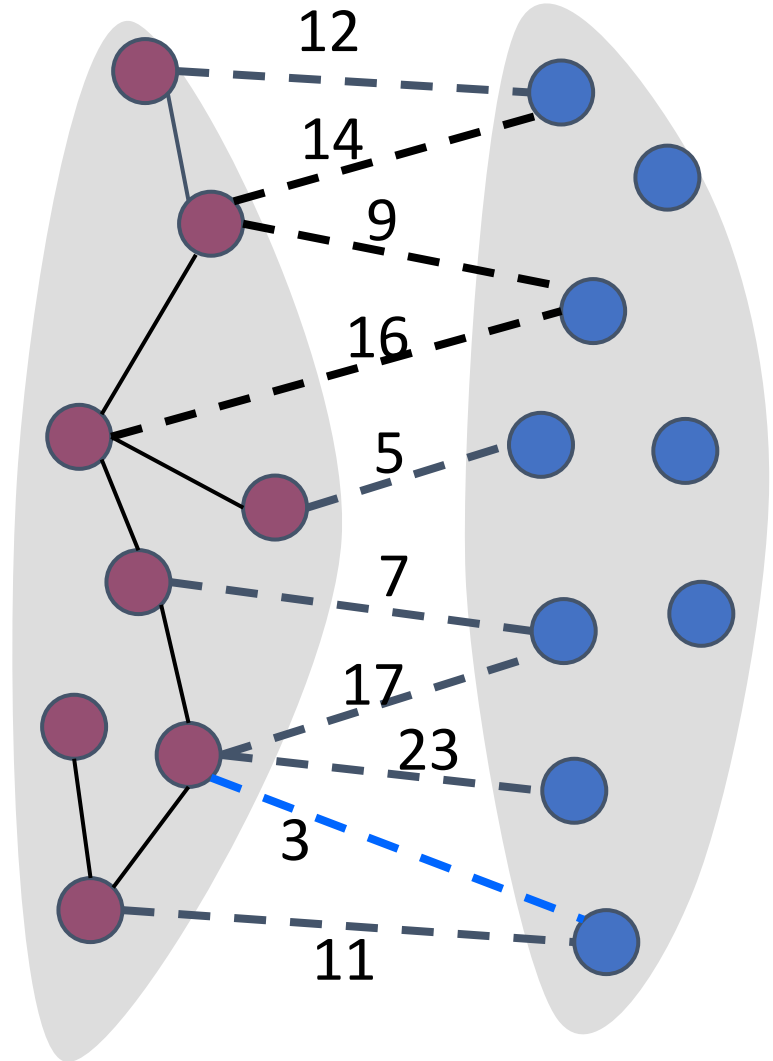
while ( $U \neq V$ ) do

    let  $(x, y)$  be smallest weight edge in  $(U, V-U)$

$U = U \cup \{y\}$

$T = T \cup (x, y)$

end while



# Prim's MST Algorithm: Runtime

Input:  $G = (V, E, W)$

Output: MST  $T$

Initialize:  $T = \emptyset$ ; Pick an arbitrary  $u \in V$ ;  $U = \{u\}$

while ( $U \neq V$ ) do

let  $(x, y)$  be smallest weight edge in  $(U, V-U)$

$U = U \cup \{y\}$

$T = T \cup (x, y)$

end while

$O(|V|)$  Iterations

$O(|E|)$  steps in a naive implementation

$O(|E| |V|)$  time  $= O(n^3)$

Can we do better?

# Prim's MST Algorithm: Improved

*heap making + V (remove min) + E (decrease key)*

Input:  $G = (V, E, W)$

Output: MST  $T$

Initialize:  $T = \emptyset$ ; Pick an arbitrary  $u \in V$ ;  $U = \{u\}$

while ( $U \neq V$ ) do

    let  $(x, y)$  be smallest weight edge in  $(U, V)$

$U = U \cup \{y\}$

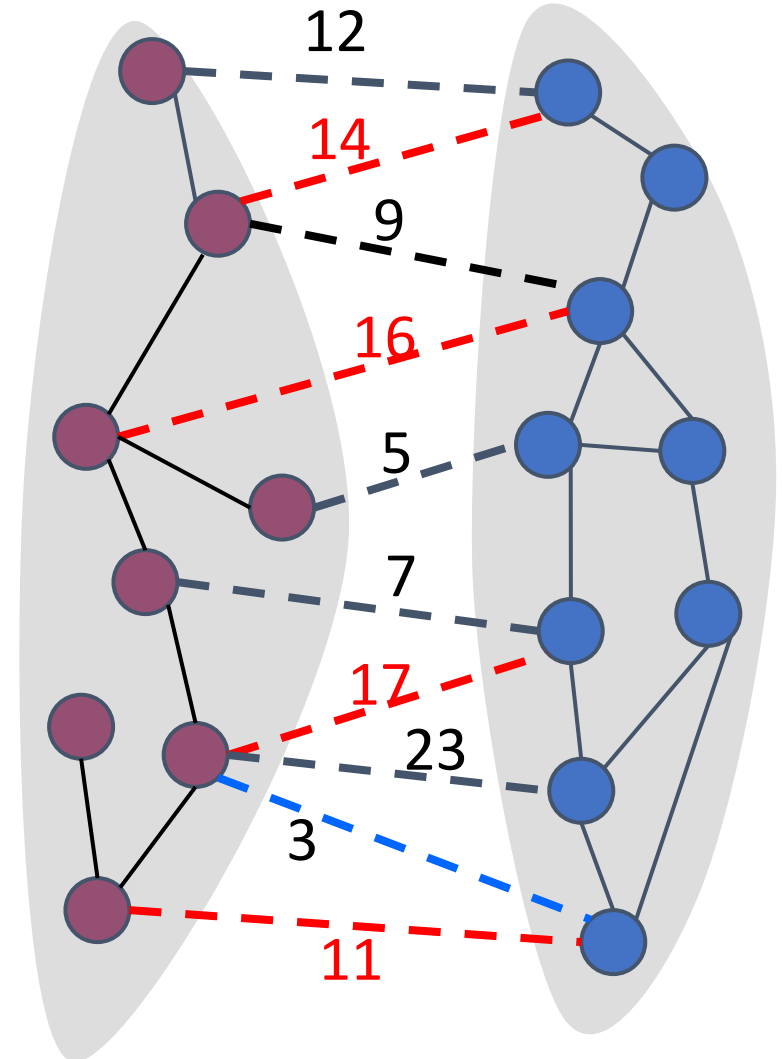
$T = T \cup (x, y)$

end while

- For each vertex in  $V - U$ , store the min-cost edge to  $U$
- Keep all the vertices in  $V - U$  in a priority queue
- Fetch the vertex with lowest edge cost to  $U$
- How to maintain dynamically?

# Prim's MST: Improved $U$ $V-U$

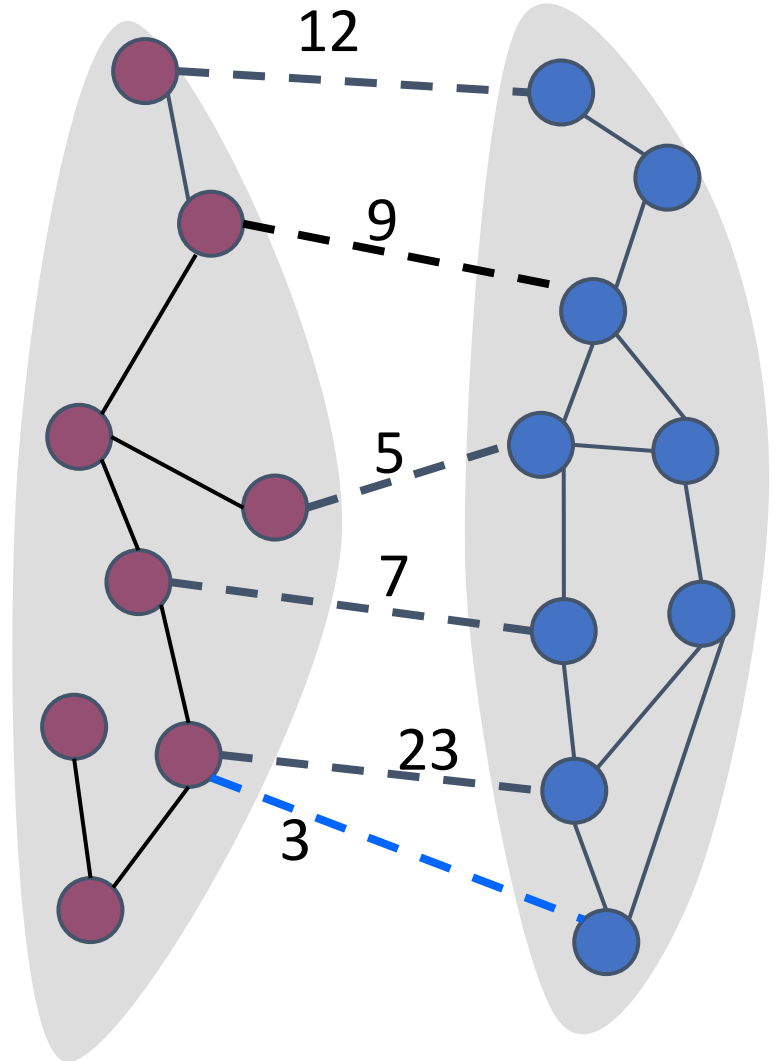
- For each vertex in  $V - U$ , store the min-cost edge to  $U$
- Keep all the vertices in  $V-U$  in a priority queue
- Fetch the vertex with lowest edge cost to  $U$
- How to maintain dynamically?





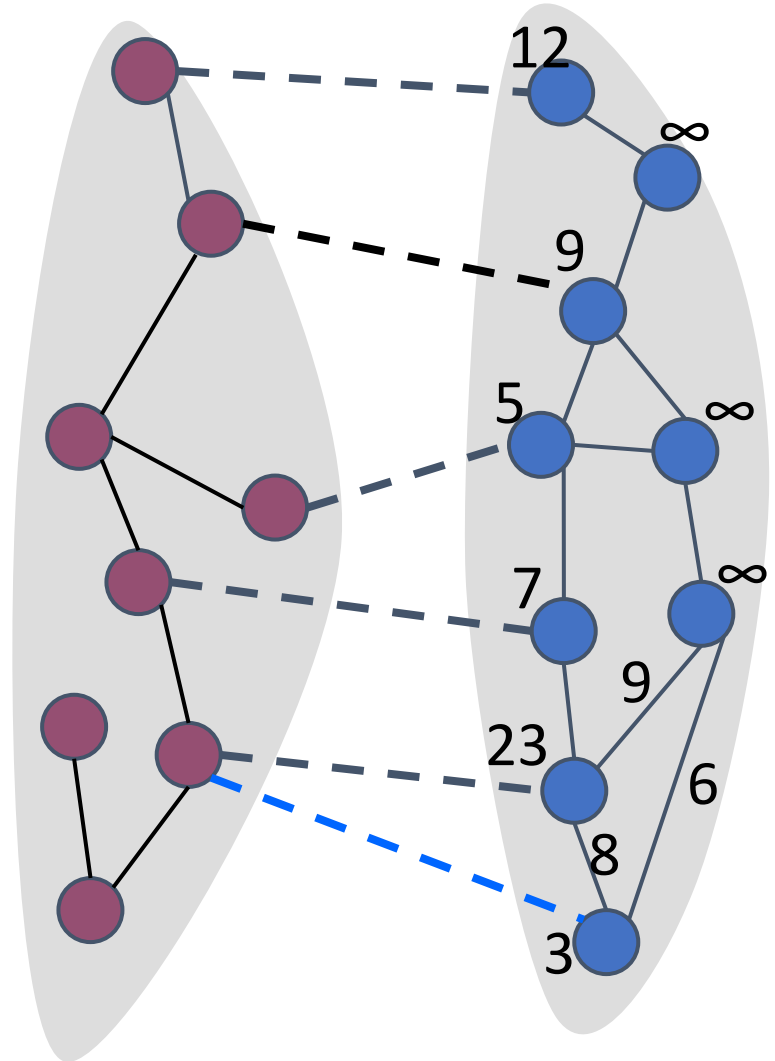
# Prim's MST: Improved $U$ $V-U$

- For each vertex in  $V - U$ , store the min-cost edge to  $U$
- Keep all the vertices in  $V-U$  in a priority queue
- Fetch the vertex with lowest edge cost to  $U$
- How to maintain dynamically?



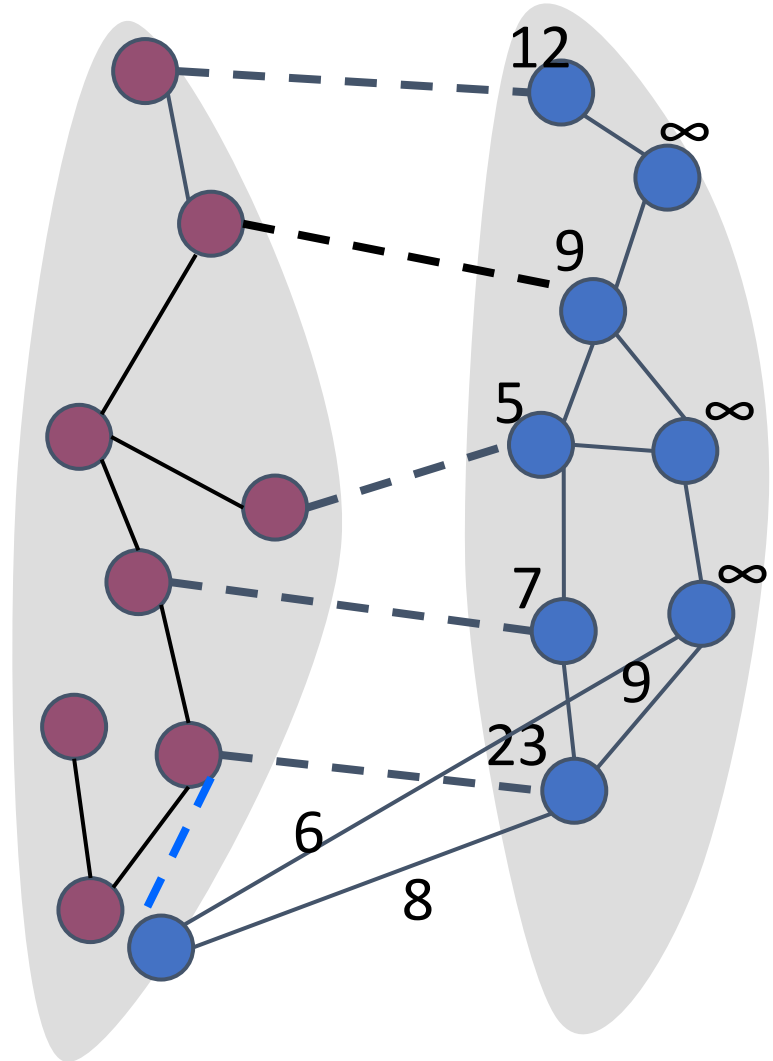
# Prim's MST: Improved $U$ $V-U$

- For each vertex in  $V - U$ , store the min-cost edge to  $U$
- Keep all the vertices in  $V-U$  in a priority queue
- Fetch the vertex with lowest edge cost to  $U$
- How to maintain dynamically?



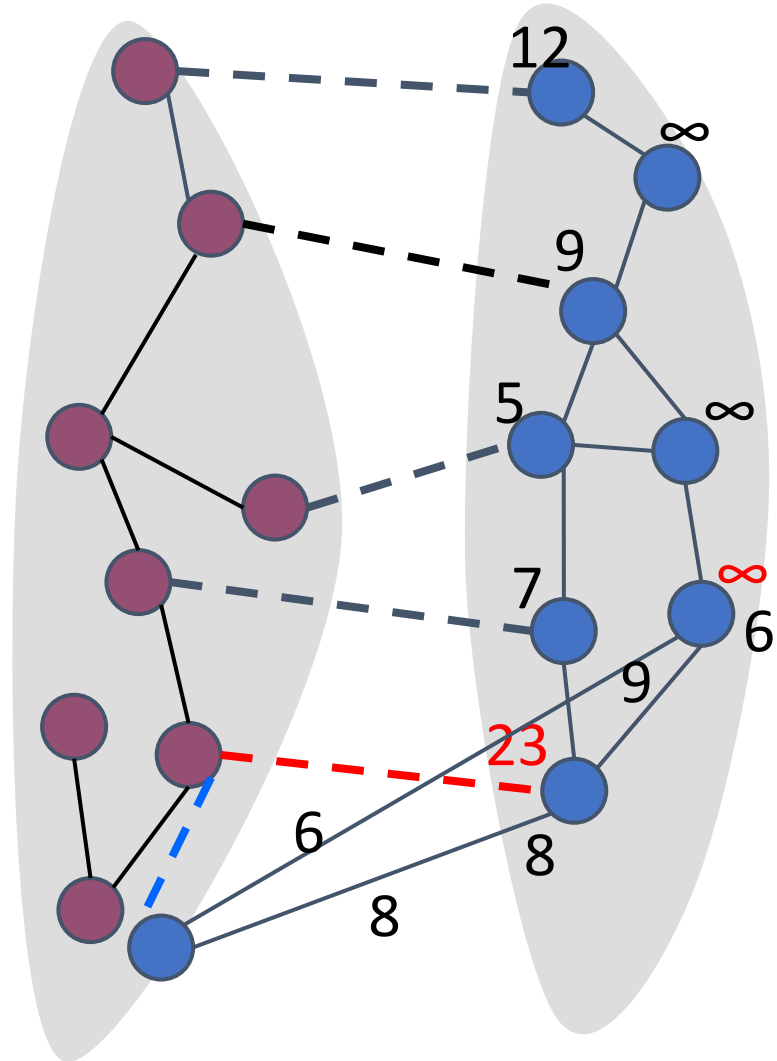
# Prim's MST: Improved $U$ $V-U$

- For each vertex in  $V - U$ , store the min-cost edge to  $U$
- Keep all the vertices in  $V-U$  in a priority queue
- Fetch the vertex with lowest edge cost to  $U$
- How to maintain dynamically?



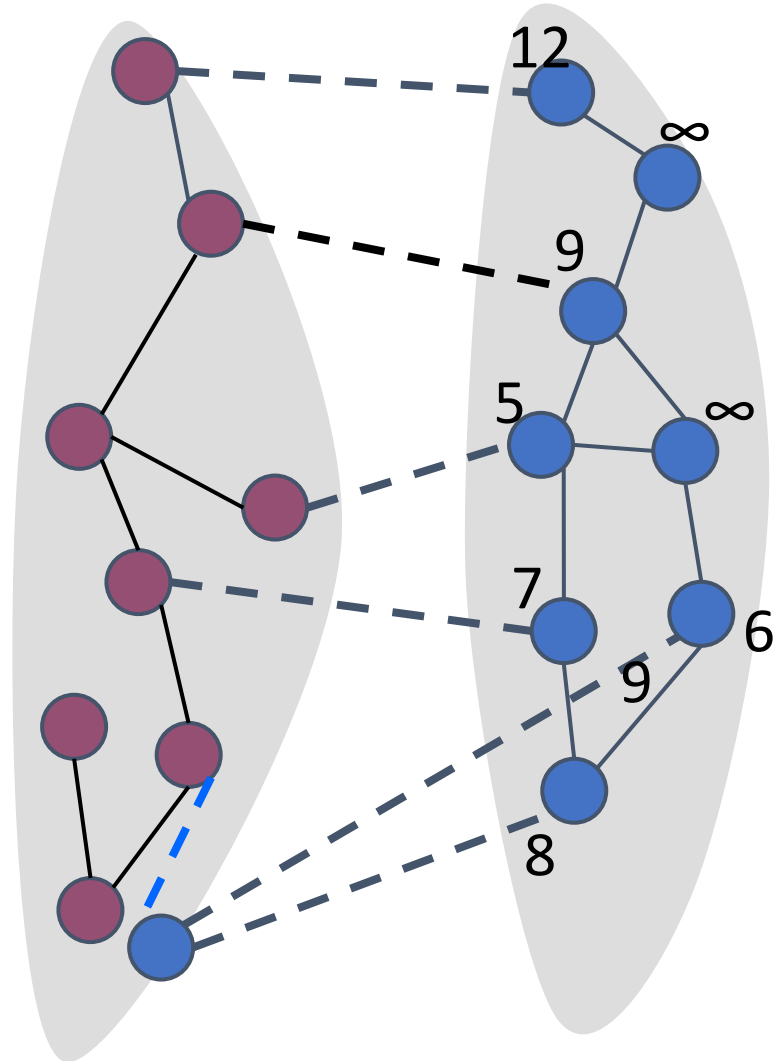
# Prim's MST: Improved $U$ $V-U$

- For each vertex in  $V - U$ , store the min-cost edge to  $U$
- Keep all the vertices in  $V-U$  in a priority queue
- Fetch the vertex with lowest edge cost to  $U$
- How to maintain dynamically?



# Prim's MST: Improved $U$ $V-U$

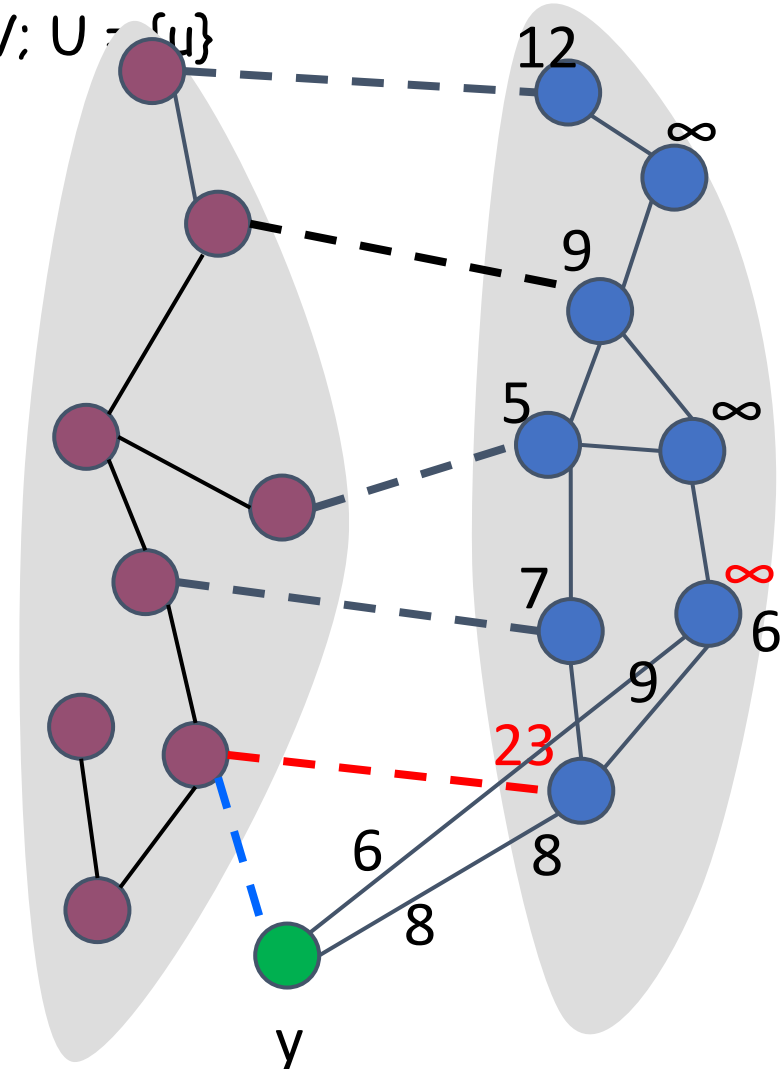
- Fetch the vertex with lowest edge cost to  $U$
- Update the weights of all vertices in  $V-U$
- How to maintain dynamically?
- Need the following in priority queues
  - InsertHeap
  - DeleteMin
  - DecreaseWeight



# Prim's MST Improved $U$ $V-U$

```

Initialize:  $T = \emptyset$ ; Pick an arbitrary  $u \in V$ ;  $U = \{u\}$ 
for all  $v$ :  $(u, v) \in E$  do
     $e = (u, v)$ ;  $D(v) = w(u, v)$ 
    InsertHeap( $w(u, v)$ ,  $e$ )
while ( $U \neq V$ ) do
     $(x, y) = \text{DeleteMin}()$ 
     $U = U \cup \{y\}$ 
     $T = T \cup (x, y)$ 
    forall  $(y, w)$  in  $E$  st  $w$  in  $V-U$  do
        if  $w(y, w) < D(w)$ 
             $D(w) = w(y, w)$ 
            DecreaseWeight( $w$ )
        endif
    endfor
end while
    
```



# Prim's MST Improved: Runtime

Initialize:  $T = \emptyset$ ; Pick an arbitrary  $u \in V$ ;  $U = \{u\}$

for all  $v: (u, v) \in E$  do

$e = (u, v)$ ;  $D(v) = w(u, v)$

    InsertHeap( $w(u, v), e$ )   $O(|V| \log(|V|))$  total

while ( $U \neq V$ ) do

$(x, y) = \text{DeleteMin}()$    $O(|V| \log(|V|))$  total

$U = U \cup \{y\}$

$T = T \cup (x, y)$

    forall  $(y, w) \in E$  st  $w \in V - U$  do   $O(|E|)$  iterations total

        if  $w(y, w) < D(w)$

$D(w) = w(y, w)$

            DecreaseWeight( $w$ )

        endif

    endfor

end while

$O(|E|)$  iterations total  
 $O(\log(V))$  per iteration

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

# Using Fibonacci Heaps

Depending on the heap implementation, running time could be improved!

	<u>EXTRACT-MIN</u>	<u>DECREASE-KEY</u>	<u>Total</u>
binary heap	$O(\lg V)$	$O(\lg V)$	$O(E \lg V)$
Fibonacci heap	$O(\lg V)$	$O(1)$	$O(V \lg V + E)$

*make heap +  $V$  (extract min) +  $E$  (decrease keys)*

From  $O(|E| \log(|V|))$  to  $O(|V| \log |V| + |E|)$



Thank You