

# VE281

## Data Structures and Algorithms

### **Minimum Spanning Tree**

#### **Learning Objectives:**

- Know what a minimum spanning tree (MST) is
- Know the Prim's algorithm for finding the MST
- Know how the various choices of the supporting data structures affect the runtime of the Prim's algorithm

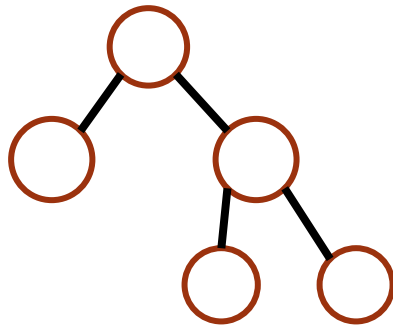
# Outline

- Minimum Spanning Tree
  - Problem
  - Prim's Algorithm

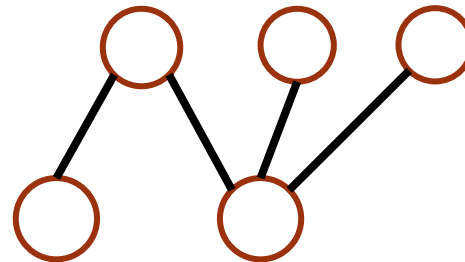
# Tree and Graph

- A **tree** is an **acyclic, connected undirected** graph.

The tree we see before



However, this is also a tree

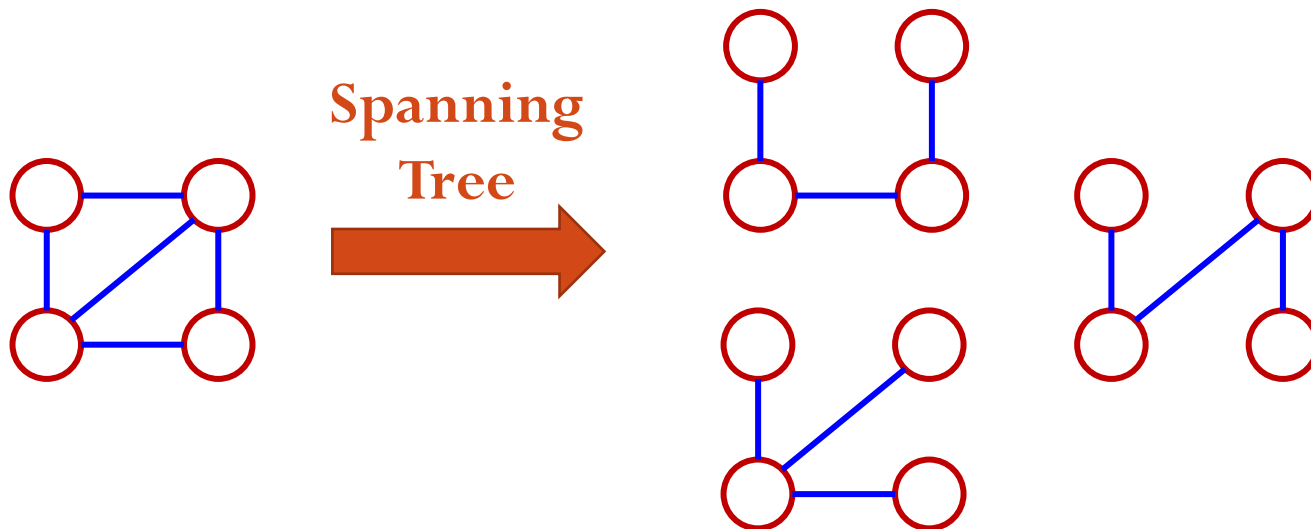


Any node can be the root of the tree.

- For a tree,  $|E| = |V| - 1$ .
- Claim: Any **connected** graph with  $N$  nodes and  $N - 1$  edges is a tree.

# Subgraph and Spanning Tree

- $G' = (V', E')$  is a **subgraph** of  $G = (V, E)$  if and only if  $V' \subseteq V$  and  $E' \subseteq E$ .
- A **spanning tree** of a **connected undirected** graph  $G$  is a subgraph of  $G$  that
  1. contains all the nodes of  $G$ ;
  2. is a tree, i.e., connected and acyclic.



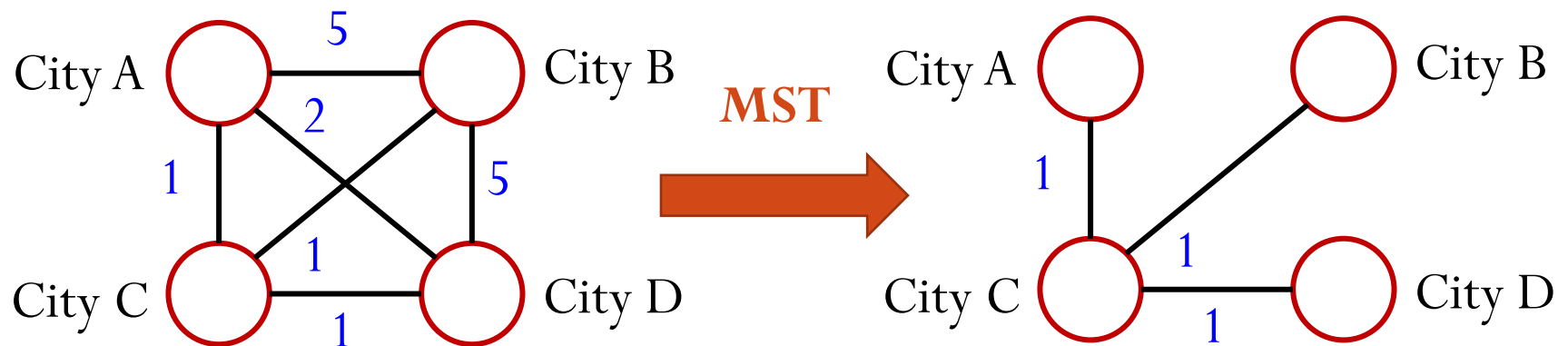
# Minimum Spanning Tree (MST)

- Given a weighted, connected, undirected graph  $G = (V, E)$ , a **minimum spanning tree**  $T$  of  $G$  is a spanning tree of  $G$  whose sum of all edge weights is the minimal.



# Application of MST

- A government planning a freeway system to connect all the cities.



- A power company planning where to lay down high-voltage power lines.

# Minimum Spanning Tree

## Algorithms

- Main idea: greedily select edges one by one and add to a growing sub-graph.
- Two standard algorithms:
  - Prim's algorithm
  - Kruskal's algorithm

# Outline

- Minimum Spanning Tree
  - Problem
  - Prim's Algorithm



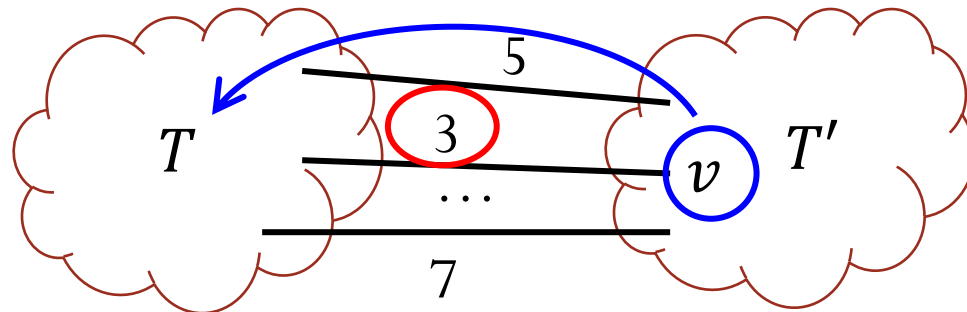
# Prim's Algorithm

- Separate  $V$  into two sets:
  - $T$ : the set of nodes that have been added to the MST.
  - $T'$ : those nodes that have not been added to the MST, i.e.,  $T' = V - T$ .
- Prim's algorithm initially sets  $T = \{s\}$ , where  $s$  is an **arbitrarily** picked **node**, and  $T' = V - \{s\}$ . The algorithm moves one node from  $T'$  to  $T$  in each iteration. After the last iteration,  $T = V$  and we have constructed the MST.

# Prim's Algorithm

## Basic Version

1. Arbitrarily pick one node  $s$ ; set  $T = \{s\}$  and  $T' = V - \{s\}$ .
2. While  $T' \neq \emptyset$ 
  - Select an edge with the **smallest weight** that connects between a node in  $T$  and a node in  $T'$ . Suppose the edge connects with node  $v$  in  $T'$ . Move  $v$  from  $T'$  to  $T$ .



# Selecting the Smallest Edge and Node

- For each node  $v \in T'$ , keep a measure  $D(v)$ , storing the “**current**” **smallest weight** over all edges that connect  $v$  to a node in  $T$ .
  - Will be updated later.
- To choose the edge with the smallest weight that connects between a node in  $T$  and a node in  $T'$ , we pick the node  $v \in T'$  with **the smallest**  $D(v)$ .
  - If edge  $(u, v)$  gives **the smallest**  $D(v)$ , then  $(u, v)$  is the edge with the smallest weight **across** set  $T$  and  $T'$ .

# Updating $v$ 's Neighbor

- If we move a node  $v$  from  $T'$  to  $T$ , then for each of  $v$ 's neighbor  $u$  that is **still** in  $T'$ , we update its  $D(u)$  as follows:
  - If  $D(u) > w(v, u)$ , then let  $D(u) = w(v, u)$ .
  - I.e., update  $D(u)$  if the weight of edge  $(v, u)$  is smaller than the weight of any other edge that connects a node in  $T$  to  $u$ .

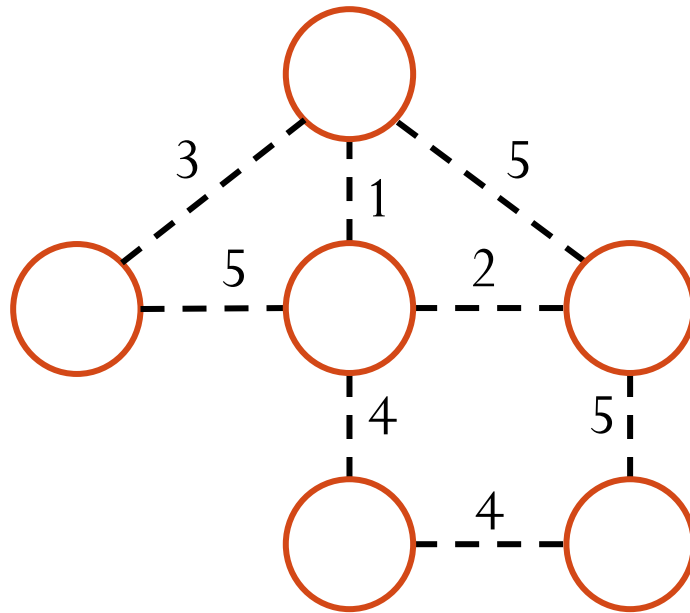
# Prim's Algorithm

## Full Version

- We keep  $P(v)$  for each node  $v$ :  $(P(v), v)$  is the edge chosen in the MST.
- 1. Arbitrarily pick one node  $s$ . Set  $D(s) = 0$ . For any other node  $v$ , set  $D(v)$  as infinite and  $P(v)$  as unknown.
- 2. Set  $T' = V$ .
- 3. While  $T' \neq \emptyset$ 
  1. Choose node  $v$  in  $T'$  such that  $D(v)$  is the smallest. Remove  $v$  from the set  $T'$ .
  2. For each of  $v$ 's **neighbors**  $u$  that is **still** in  $T'$ , if  $D(u) > w(v, u)$ , then update  $D(u)$  as  $w(v, u)$  and  $P(u)$  as  $v$ .

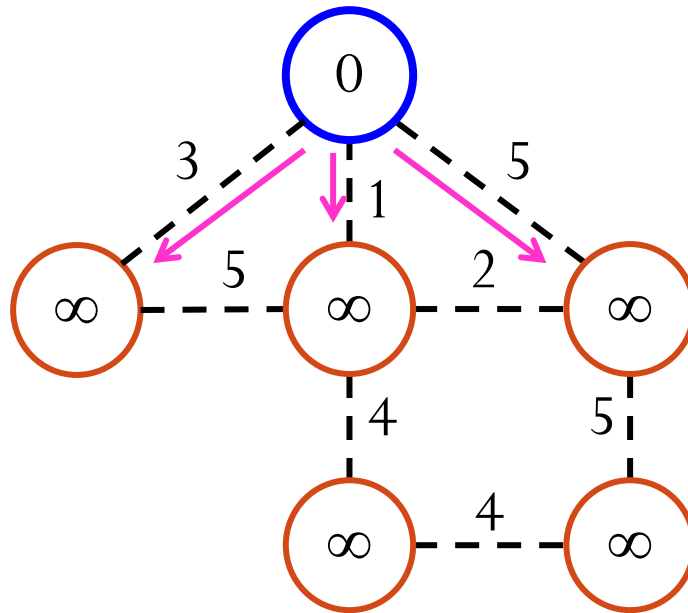
# Prim's Algorithm

## Example



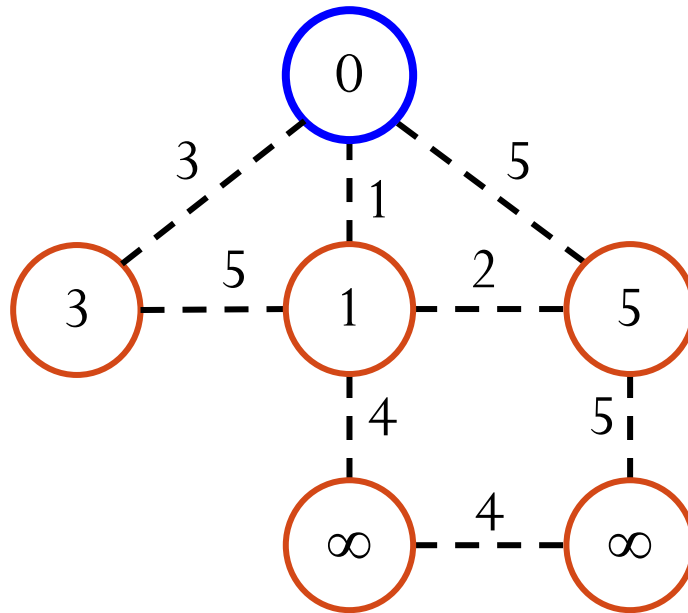
# Prim's Algorithm

## Example



# Prim's Algorithm

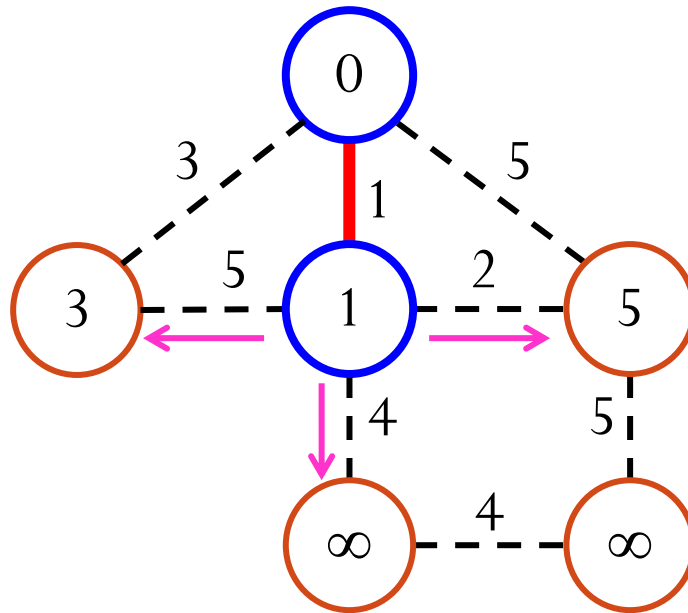
## Example





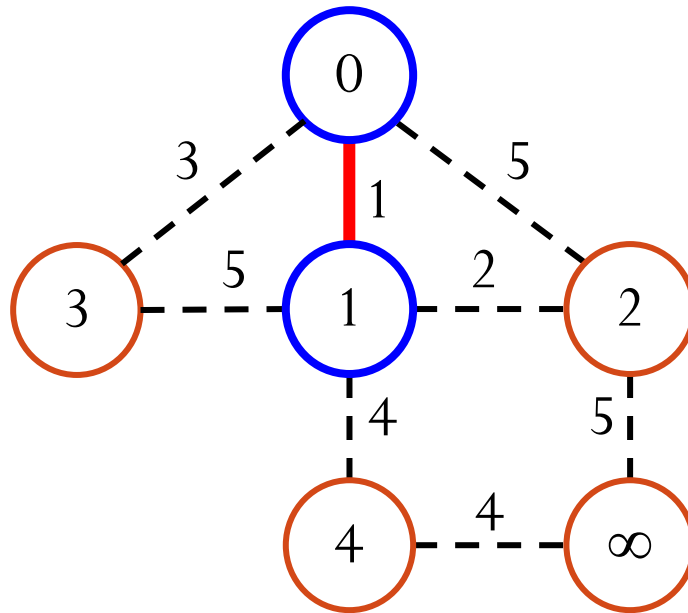
# Prim's Algorithm

## Example



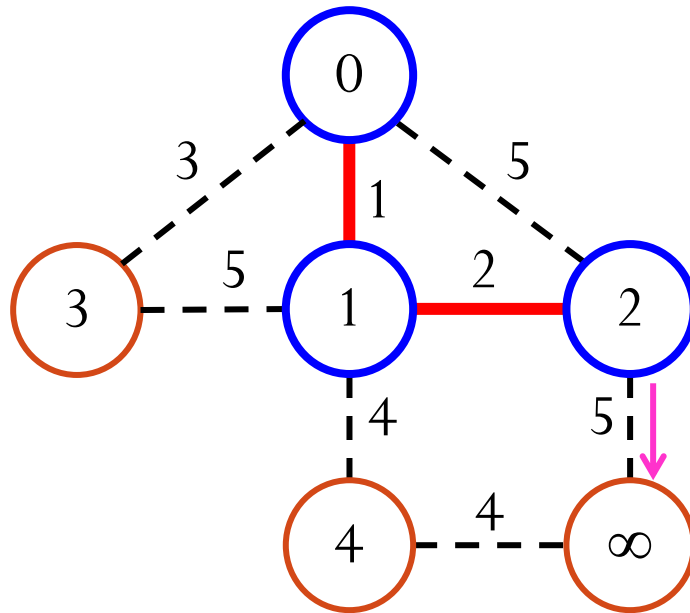
# Prim's Algorithm

## Example



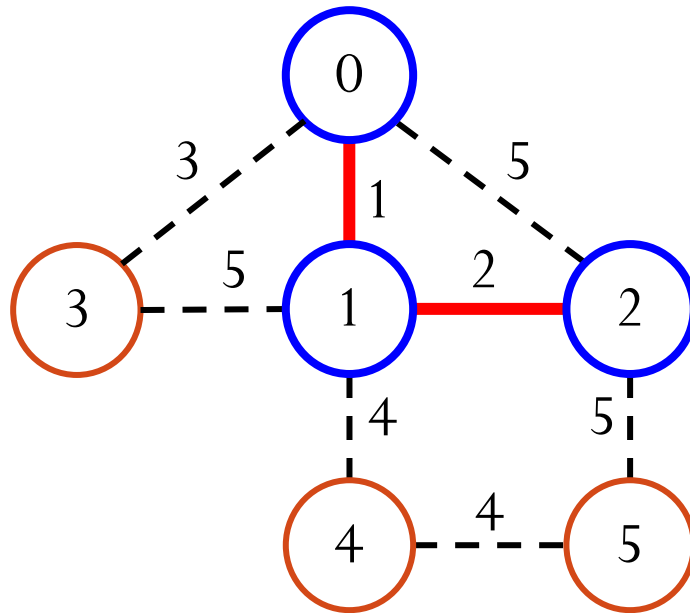
# Prim's Algorithm

## Example



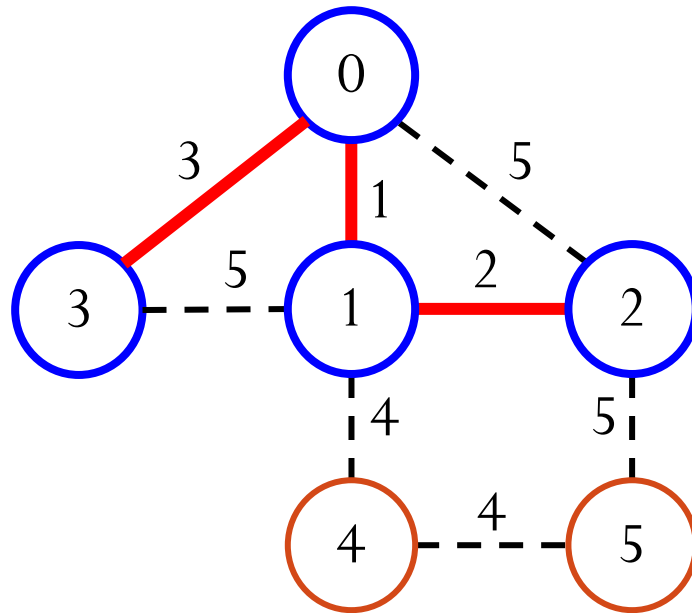
# Prim's Algorithm

## Example



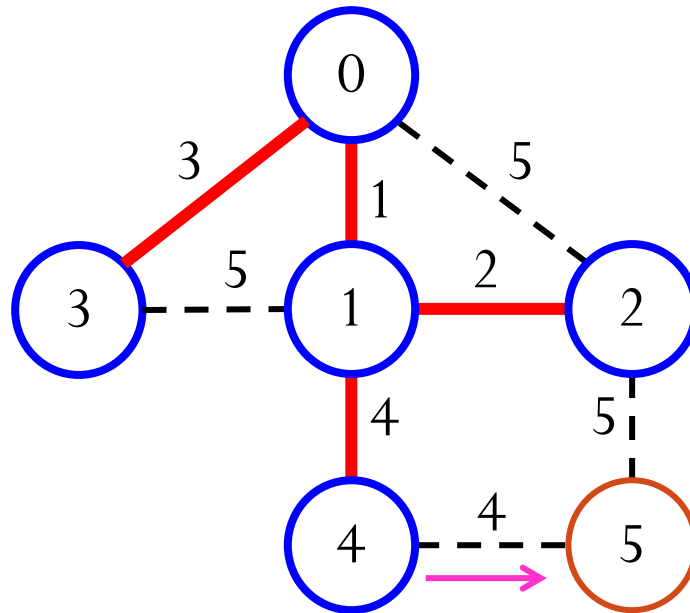
# Prim's Algorithm

## Example



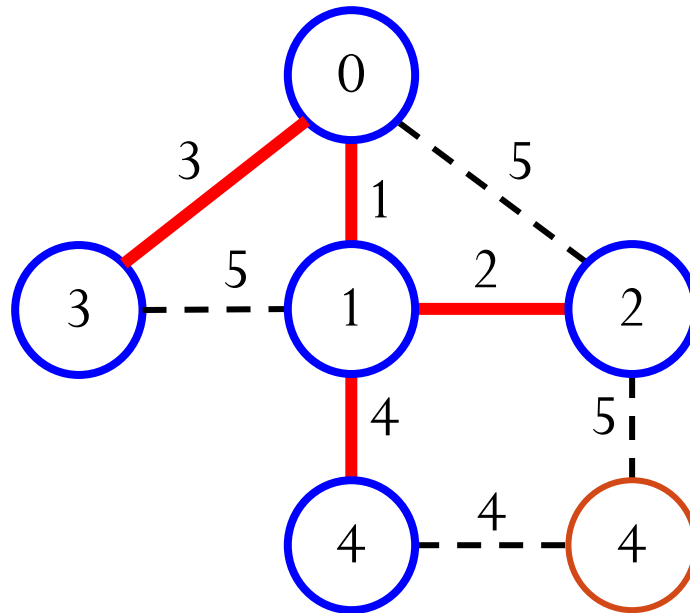
# Prim's Algorithm

## Example



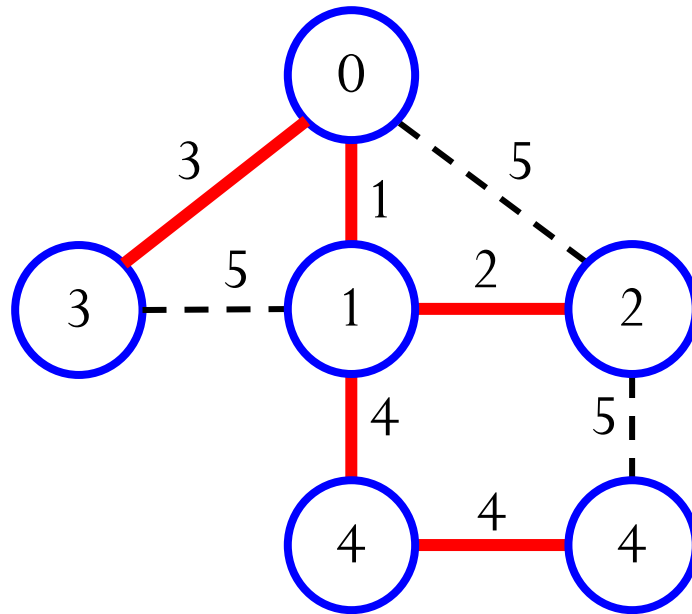
# Prim's Algorithm

## Example



# Prim's Algorithm

## Example

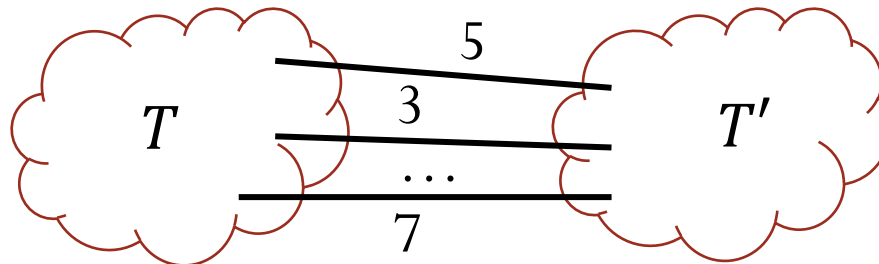




# Prim's Algorithm

## Justification

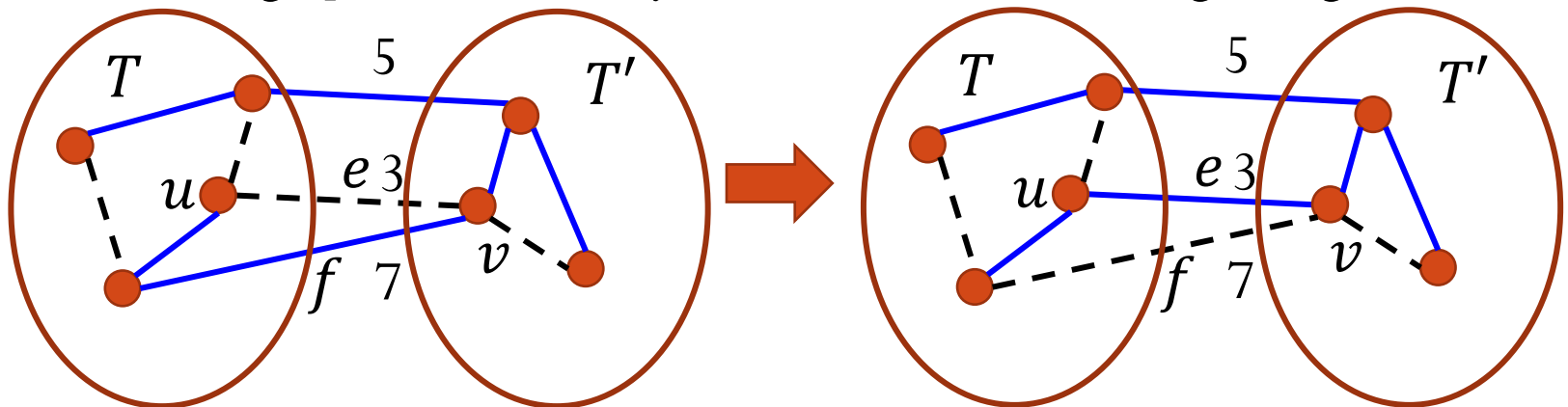
- Claim: the obtained subgraph is a tree
- Proof:
  - The nodes in set  $T$  are connected (can be shown by induction)
  - Furthermore,  $|V| = |E| + 1$
  - Claim: Any **connected** graph with  $N$  nodes and  $N - 1$  edges is a tree



# Prim's Algorithm

## Justification

- Claim: the obtained subgraph is an MST
- Proof by contradiction:
  - Assume the MST does not contain the cheapest edge  $e$  between  $T$  and  $T'$
  - Assume  $e = (u, v)$ . Its weight is  $w$
  - In the MST, there exists a unique path between  $u$  and  $v$ . On this path, there is an edge  $f$  across  $T$  and  $T'$ . Its weight  $> w$
  - We replace  $f$  by  $e$  in original MST.
  - The new graph is a tree (Why?) with smaller sum of edge weights



# Prim's Algorithm

## Time Complexity

1. Arbitrarily pick one node  $s$ . Set  $D(s) = 0$ . For any other node  $v$ , set  $D(v)$  as infinite and  $P(v)$  as unknown.
2. Set  $T' = V$ .
3. While  $T' \neq \emptyset$ 
  1. Choose node  $v$  in  $T'$  such that  $D(v)$  is the smallest. Remove  $v$  from the set  $T'$ .
  2. For each of  $v$ 's **neighbors**  $u$  that is **still** in  $T'$ , if  $D(u) > w(v, u)$ , then update  $D(u)$  as  $w(v, u)$  and  $P(u)$  as  $v$ .

What is the time complexity of Prim's algorithm?

# Prim's Algorithm

## Time Complexity

- Method 1: linear scan the set  $T'$  to find the smallest  $D(v)$ .
- Number of times to find the smallest  $D(v)$ :  $|V|$ .
  - Each cost:  $O(|V|)$ .
- **Maximal** number of times to update the neighbors:  $|E|$ .
  - Since each neighbor of each node could be **potentially** updated.
  - Each cost:  $O(1)$ .
- Total running time is  $O(|E| + |V|^2) = O(|V|^2)$ .

# Prim's Algorithm

## Time Complexity

- Method 2: use a binary heap to store  $D(v)$ 's.
- Number of times to extract the smallest  $D(v)$ :  $|V|$ .
  - Each cost:  $O(\log |V|)$ .
- **Maximal** number of times to update the neighbors:  $|E|$ .
  - Each cost is  $O(\log |V|)$ , since after updating  $D(v)$ , we should percolate up new  $D(v)$  into right location of binary heap.
- Total running time is  $O(|V| \log |V| + |E| \log |V|)$   
 $= O((|V| + |E|) \log |V|)$ .

# Prim's Algorithm

## Time Complexity

- Method 3: use a Fibonacci heap to store  $D(v)$ 's.
- Number of times to extract the smallest  $D(v)$ :  $|V|$ .
  - Each cost:  $O(\log |V|)$ .
- **Maximal** number of times to update the neighbors:  $|E|$ .
  - Each cost is  $O(1)$  (decreaseKey operation; amortized time).
- Total running time is  $O(|V| \log |V| + |E|)$ .

# Prim's Algorithm

## Time Complexity

- Method 1: linear scan the set  $T'$  to find the smallest  $D(v)$ 
  - Total runtime:  $O(|V|^2)$
- Method 2: use a binary heap to store  $D(v)$ 's
  - Total runtime:  $O((|V| + |E|) \log |V|)$
- Method 3: use a Fibonacci heap to store  $D(v)$ 's
  - Total runtime:  $O(|V| \log |V| + |E|)$
- Which one is the best?
  - Answer: Fibonacci heap.
  - For sparse graphs, i.e.,  $|E| \approx \Theta(|V|)$ , using binary heap has same runtime complexity as Fibonacci heap. The runtime complexity is  $O(|V| \log |V|)$