# CSC 411
# Design and Analysis of Algorithms

---

## Chapter 9 Algorithm Design

## using Greedy Techniques

## Part 1

Instructor: Minhee Jun

# Greedy Technique

- Constructs a solution to an *optimization problem* piece by piece through a sequence of choices
- On each step, it suggests a “greedy” grab of the best alternative available in the hope of finding a global optimum
- For *some* problems, yields an optimal solution for every instance.
- For most, does not yield an optimal solution but can be useful for fast approximations.

# Change-Making Problem

- Given unlimited amounts of coins of denominations
  - $d_1 > \ldots > d_m$ ,
- give change for amount $n$ with the least number of coins
- Example:
  - $d_1 = 25c$, $d_2 = 10c$, $d_3 = 5c$, $d_4 = 1c$ and $n = 48c$
  - 1 quarter, 2 dimes, and 3 pennies
- Greedy technique yield an optimal solution.
  - Question: Does Greedy method also yield an optimal solution for another set of coin denominations?

# Change-Making Problem

- Example:
  - $d_1 = 7$c, $d_2 = 5$c, $d_3 = 1$c and $n = 25$c
- Greedy solution:
  - 3 $d_1$ and 4 $d_3$ $\rightarrow$ 7 coins
- Optimal solution:
  - $5\ d_2$ $\rightarrow$ 5 coins
- Greedy technique does not give optimal solution

# Change-Making Problem

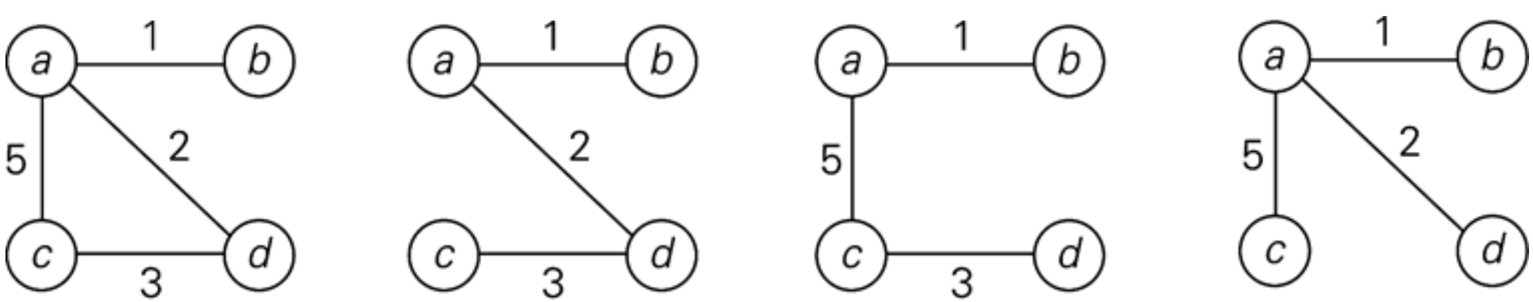
- Greedy solution:
  - Greedy solution is optimal for any amount and "normal" set of denominations
  - may not be optimal for arbitrary coin denominations

# Applications of the Greedy Strategy

- Optimal solutions:
  - Change making for “normal” coin denominations
  - Minimum spanning tree (MST)
    - Prim’s algorithm
    - Kruskal’s algorithm
  - Single-source shortest paths in a weighted graph
    - Dijkstra’s algorithm
  - Data compression method
    - Huffman codes
  - Approximations:
    - traveling salesman problem (TSP)
    - knapsack problem
    - other combinatorial optimization problems

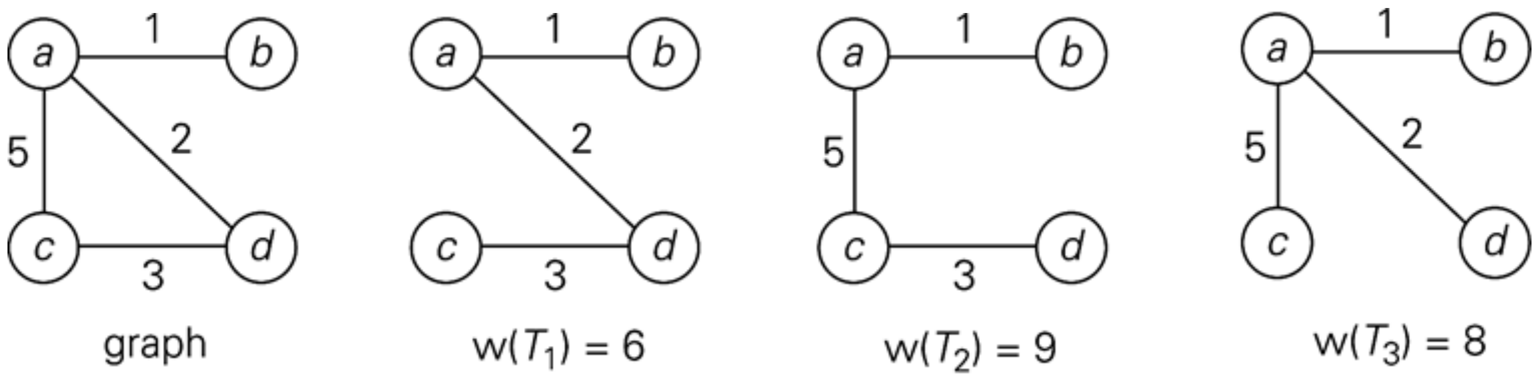
# Minimum Spanning Tree (MST)

- *<u>Spanning tree</u>* of a connected graph *G*
  - a <u>connected acyclic</u> subgraph of *G* that includes all of *G*'s vertices
- *<u>Minimum spanning tree</u>* of a weighted, connected graph *G*
  - a spanning tree of *G* of <u>minimum total weight</u>
- Example:



graph    $w(T_1) = 6$    $w(T_2) = 9$    $w(T_3) = 8$

# Minimum spanning tree: Exhaustive search




- Serious obstacles
  - # spanning trees grows exponentially with the graph size
  - Generating all spanning trees for a given graph is not easy
    - Not efficient
- Proposed efficient algorithms
  - Prim's algorithm
  - Kruskal's algorithm

# 1. Prim's algorithm

- Start with tree $T_1$ consisting of one (any) vertex and "grow" tree one vertex at a time to produce MST through a series of expanding subtrees $T_1, T_2, \ldots, T_n$
- On each iteration, construct $T_{i+1}$ from $T_i$ by adding vertex not in $T_i$ that is <u>closest</u> to those already in $T_i$ (this is a "greedy" step!)
- Stop when all vertices are included

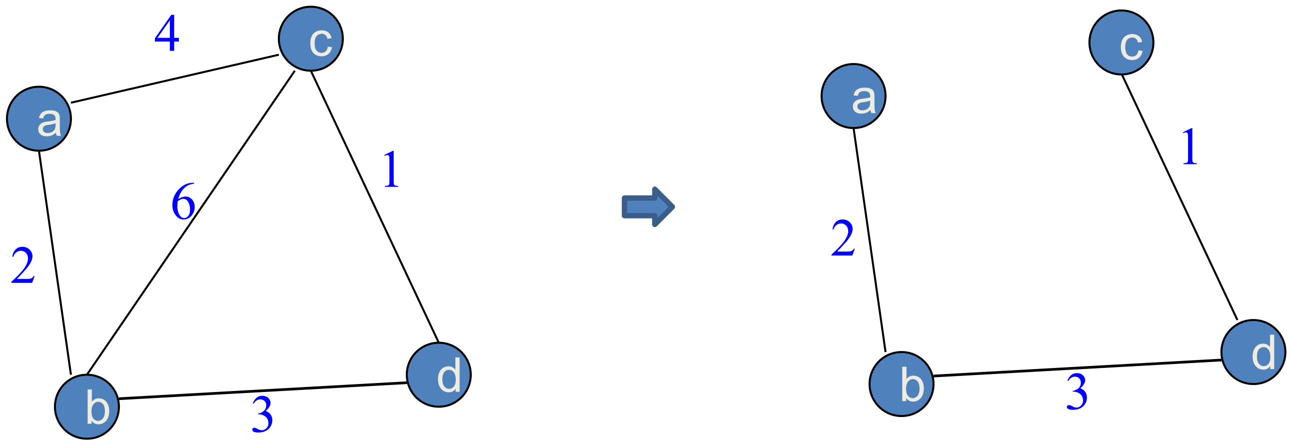
# Prim's algorithm: Pseudocode

**ALGORITHM** *Prim*$(G)$

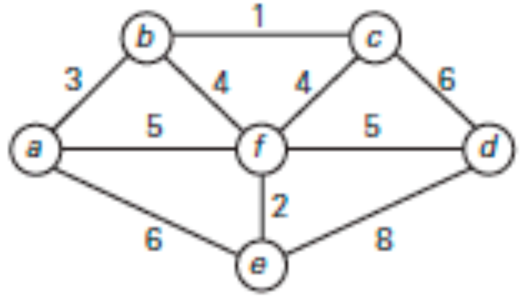
//Prim's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E \rangle$
//Output: $E_T$, the set of edges composing a minimum spanning tree of $G$
$V_T \leftarrow \{v_0\}$ //the set of tree vertices can be initialized with any vertex
$E_T \leftarrow \varnothing$
**for** $i \leftarrow 1$ **to** $|V| - 1$ **do**
    find a minimum-weight edge $e^* = (v^*, u^*)$ among all the edges $(v, u)$
    such that $v$ is in $V_T$ and $u$ is in $V - V_T$
    $V_T \leftarrow V_T \cup \{u^*\}$
    $E_T \leftarrow E_T \cup \{e^*\}$
**return** $E_T$

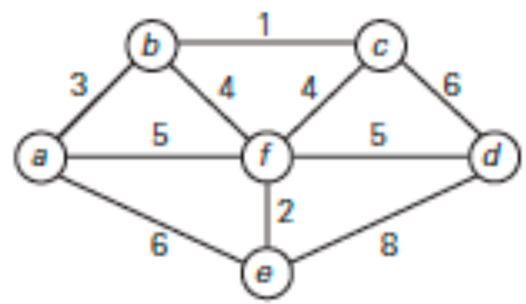
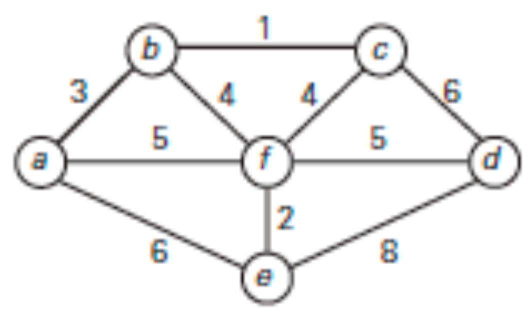
# Prim's algorithm: Example 1

- Find the MST using Prim's algorithm



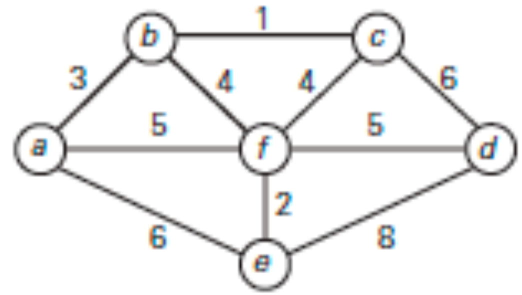
# Prim’s algorithm: Example 2



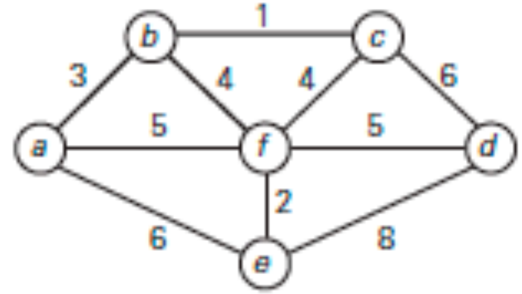

| Tree vertices | Remaining vertices | Illustration |
|---|---|---|
| a(–, –) | **b(a, 3)** c(–, ∞) d(–, ∞) e(a, 6) f(a, 5) |  |
| b(a, 3) | **c(b, 1)** d(–, ∞) e(a, 6) f(b, 4) |  |

# Prim's algorithm: Example 2

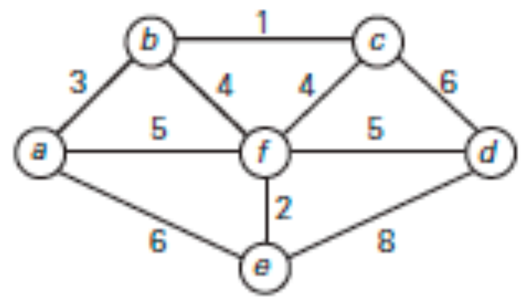
c(b, 1)    d(c, 6)  e(a, 6)  **f(b, 4)**




f(b, 4)    d(f, 5)  **e(f, 2)**




e(f, 2)    **d(f, 5)**




d(f, 5)

# Does Prim's give you a unique MST?

- If the edge weights in the graph are <u>all different</u> from each other, then the graph has a **unique** minimum spanning tree

- If the edge weights in the graph are <u>not all different</u>, then you may have **more than one** MST.

# Analysis of Prim's algorithm

- Efficiency
  - weight matrix representation of graph and
    array implementation of priority queue: $\Theta(|V|^2)$
  - adjacency list representation of graph and
    heap implementation of priority queue: $\Theta(|E|\log|V|)$

# Notes about Prim's algorithm

- How to prove that this construction actually yields Minimum Spanning Tree (MST)?
  - Proof by induction
- What data structure to use when implementing Prim's algorithm?
  - Needs priority queue for locating closest fringe vertex

# 2. Kruskal's algorithm

- Another greedy algorithm for MST:
  - The sum of the edge weights is the smallest
  - However, not necessarily connected on the intermediate stages of the algorithm
- How?
  - **Sort** the graph's edges in nondecreasing order of their weights
  - "Grow" tree one edge at a time to produce MST through a series of expanding forests $F_1$, $F_2$, …, $F_{n-1}$
    - Star with the empty subgraph
    - On each iteration, add the next edge on the **sorted list** if such an inclusion does not create a cycle.
      (otherwise, skip the edge.)

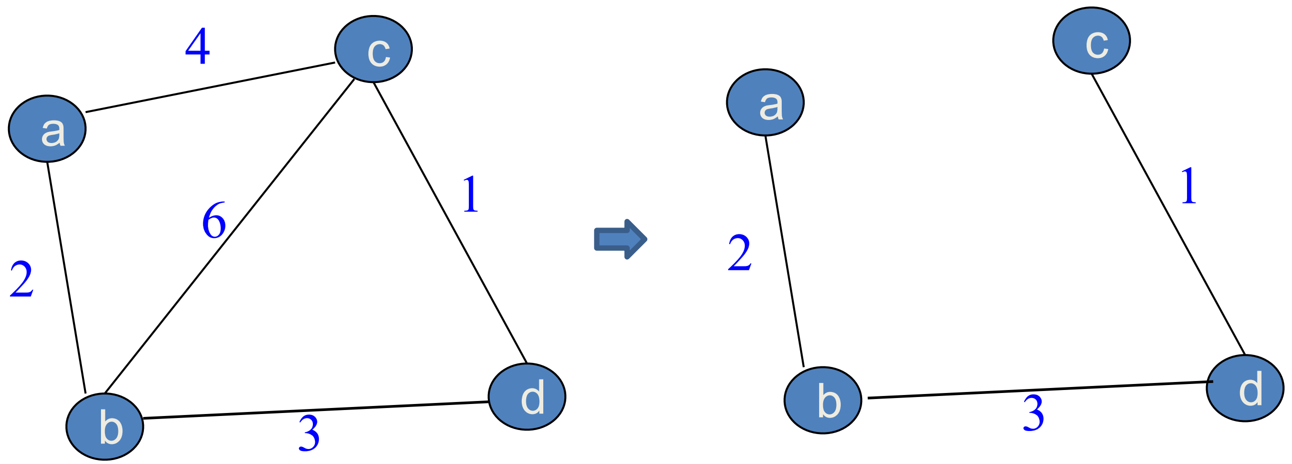
# Kruskal's Algorithm: Pseudocode

**ALGORITHM** $Kruskal(G)$

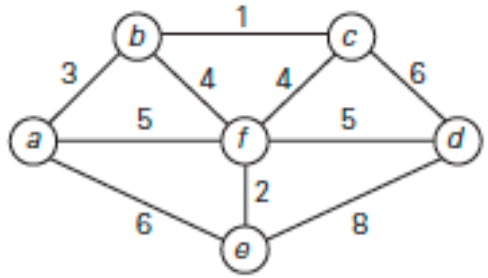
//Kruskal's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E \rangle$
//Output: $E_T$, the set of edges composing a minimum spanning tree of $G$
sort $E$ in nondecreasing order of the edge weights $w(e_{i_1}) \leq \ldots \leq w(e_{i_{|E|}})$
$E_T \leftarrow \varnothing$; $\quad ecounter \leftarrow 0$ $\quad$ //initialize the set of tree edges and its size
$k \leftarrow 0$ $\quad$ //initialize the number of processed edges
**while** $ecounter < |V| - 1$ **do**
$\quad k \leftarrow k + 1$
$\quad$ **if** $E_T \cup \{e_{i_k}\}$ is acyclic
$\quad\quad E_T \leftarrow E_T \cup \{e_{i_k}\}$; $\quad ecounter \leftarrow ecounter + 1$
**return** $E_T$

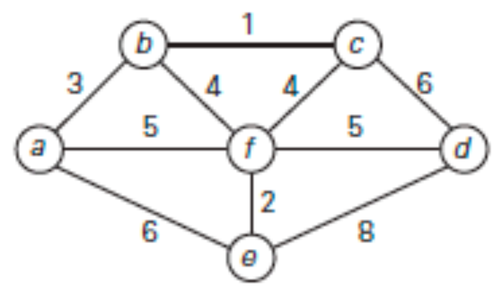
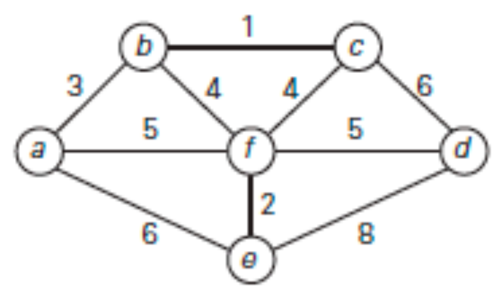
# Kruskal's Algorithm: Example 1

- Find the MST using Kruskal's algorithm



# Kruskal’s Algorithm: Example 2

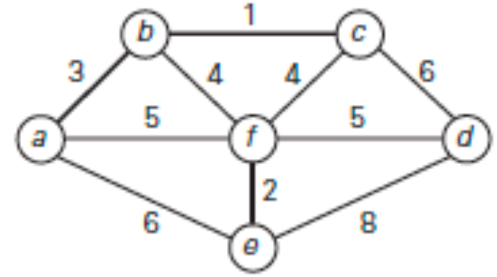



| Tree edges | Sorted list of edges | Illustration |
|---|---|---|
| | **bc** ef ab bf cf af df ae cd de<br>1 2 3 4 4 5 5 6 6 8 |  |
| bc<br>1 | bc **ef** ab bf cf af df ae cd de<br>1 2 3 4 4 5 5 6 6 8 |  |

# Kruskal's Algorithm: Example 2

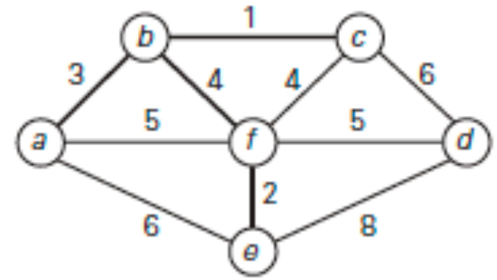
ef
2

| bc | ef | **ab** | bf | cf | af | df | ae | cd | de |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 4 | 5 | 5 | 6 | 6 | 8 |



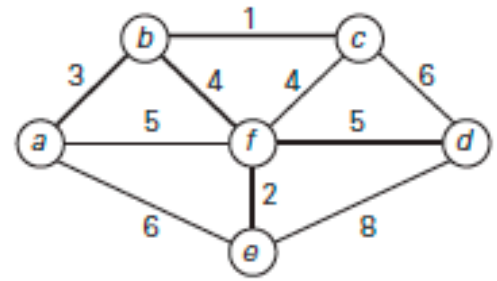

ab
3

| bc | ef | ab | **bf** | cf | af | df | ae | cd | de |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 4 | 5 | 5 | 6 | 6 | 8 |




bf
4

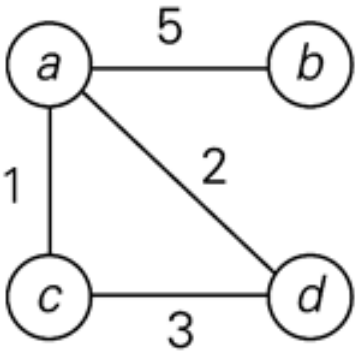
| bc | ef | ab | bf | cf | af | **df** | ae | cd | de |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 4 | 5 | 5 | 6 | 6 | 8 |




df
5

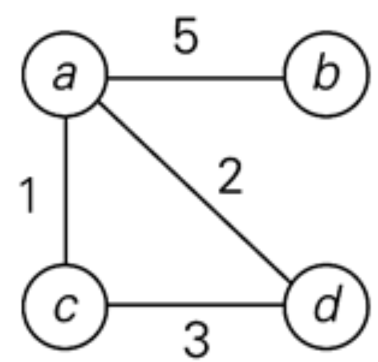
# Analysis of Kruskal’s algorithm

- Efficiency: $O(|E|\log|E|)$



# Notes about Kruskal’s algorithm

- Question: Prim’s or Kruskal’s algorithm,
  Which algorithm is better??
  - Kruskal’s algorithm looks easier than Prim’s algorithm but is ***harder*** to implement because of cycle checking
    - Cycle checking: a cycle is created iff added edge connects vertices in the same connected component

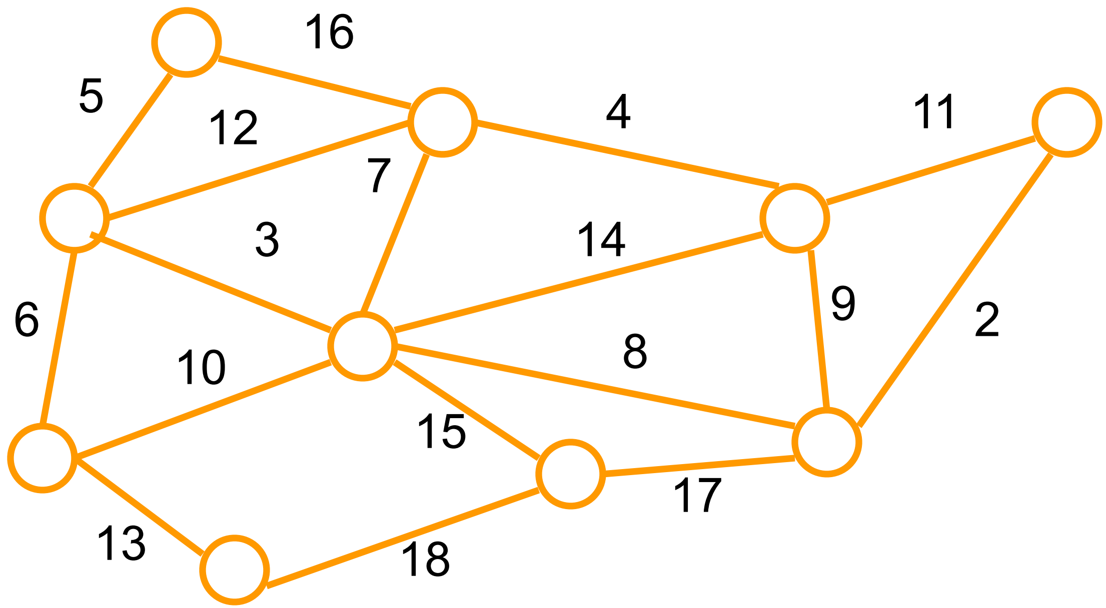


# Notes about Kruskal

- Question: : Does Prim’s and Kruskal give the same MST?
  - If the edge weights in your graph are all <u>different</u> from each other, then your graph has a **unique** minimum spanning tree
    - so Kruskal's and Prim's algorithms are guaranteed to return the same tree.
- If the edge weights in your graph are <u>not</u> all different, then neither algorithm is necessarily deterministic.
    - They both have steps of the form "choose the lowest-weight edge that satisfies some condition" that might yield ambiguous results.

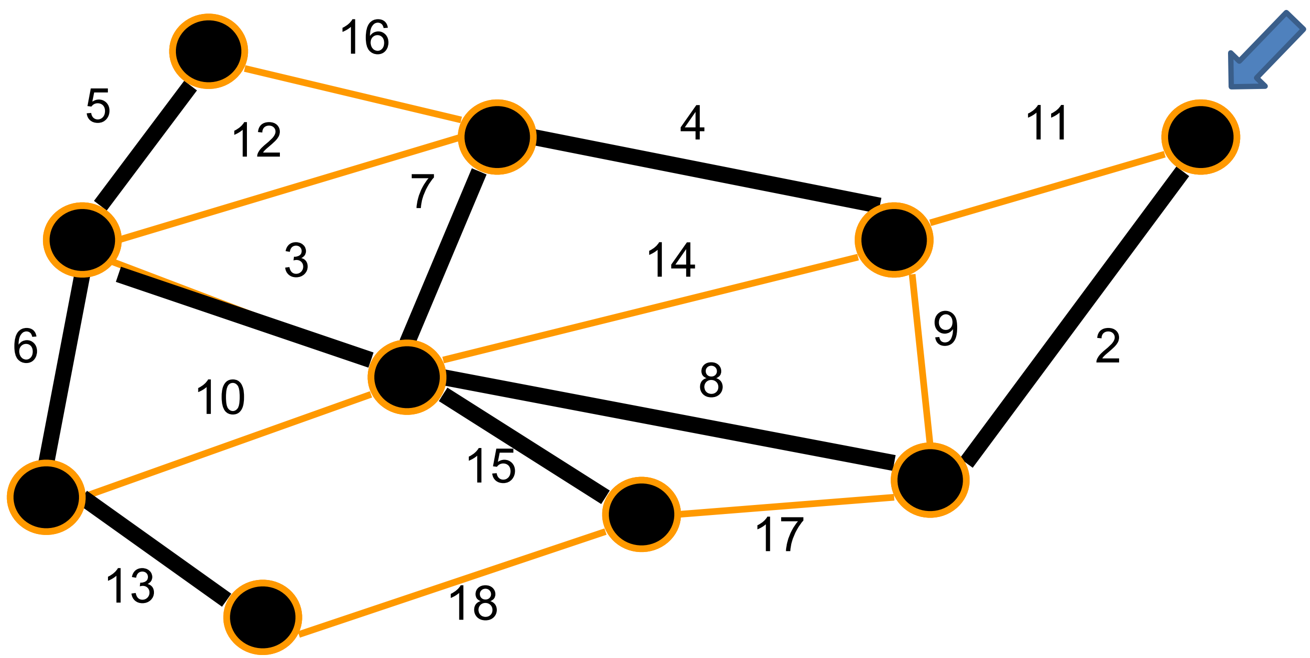
# Example: Minimum Spanning Tree

- Question: Does Prim's and Kruskal give the same MST?



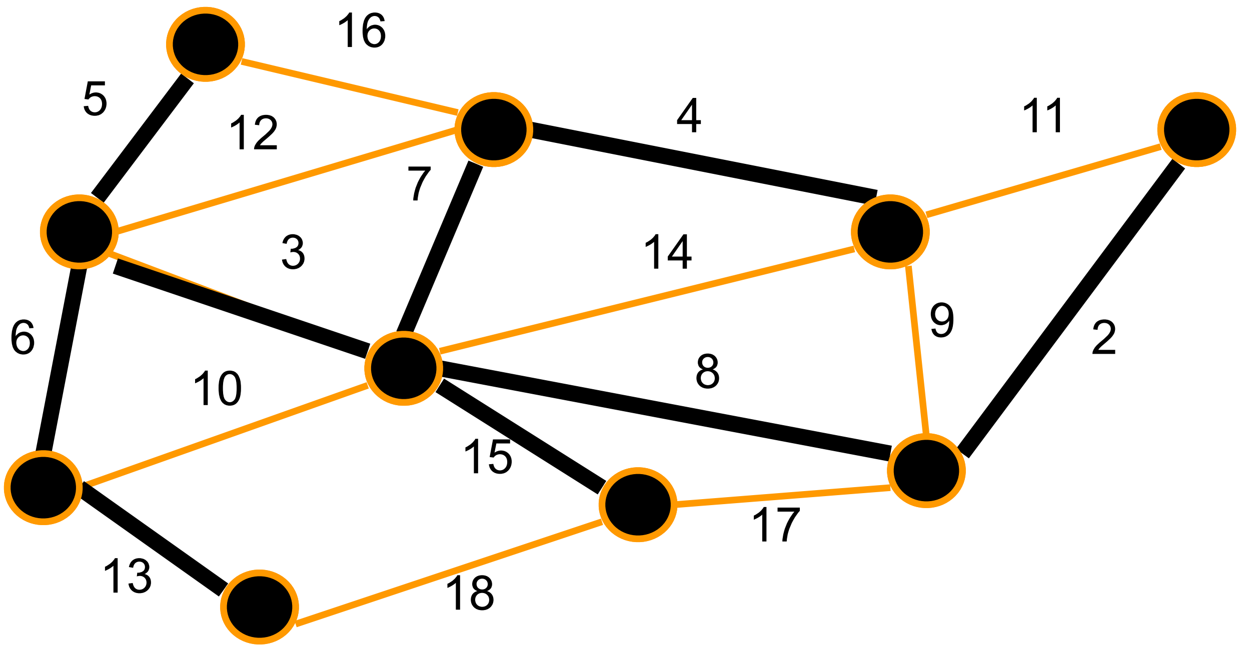

find subset of edges that span all the nodes, create no cycle, and minimize sum of weights

# Example: Prim's MST algorithm




Starting from any node, add an edge that will connect a node and the tree with a minimum weight

# Example: Kruskal's MST algorithm




Sort the edges in increasing order of weight, add in an edge iff it does not cause a cycle