Greedy Graph Algorithms: Kruskal’s Algorithm for MSTs

Reminders/Logistics

- Homework feedback process did not go as planned. Debugging meeting this afternoon. Stay tuned…
- Homework 3 due tonight 10 pm
    - See notes in email
- Homework 4 on greedy algorithms will be released tonight
    - "Lighter" than previous problem set so you can maximize reading period
- Mask policy:
    - Everyone is invited to wear a mask in class, but you are not required to do so. Please respect others.
    - Tomorrow I will get clarification about the CS lab space policies
- Announcements?

Today's Plan

Kruskal's Algorithm & the Union-Find Data structure

- Review proofs from activity (or similar variants)
- (Briefly) Review Kruskal's algorithm to motivate
- (Briefly) Review Heaps
- Iterate on data structure designs to arrive at efficient Union-Find

Activity Review: MWSS are Trees

Prove. In a weighted, undirected graph $G = (V, E)$ that has strictly positive edge weights, a minimum weight spanning subgraph must always be a tree.

Proof. (By contradiction)

Suppose G has some MWSS, $S = (V, E')$, that is not a tree.

This means that the set E' connects all vertices in V, and that S contains at least one cycle. Without loss of generality, let the vertices $v_1, \ldots, v_n, v_1$ define some cycle in S.

Suppose we remove edge $e = (v_1, v_n)$ from S.

The resulting graph $S' = (V, E' - e)$ is still connected, (**Why?**) so it is still a spanning subgraph.

However, the weight of S' is less than the weight of S, since all edge weights are positive, including e. This is a contradiction, since S is a *minimum* weight spanning subgraph.

Activity Review: Cut Property

Recall. A cut is a partition of the vertices into two nonempty subsets S and $V - S$. A cut edge of a cut S is an edge with one end point in S and another in $V - S$.

Lemma (Cut Property). For any cut $S \subset V$, let $e = (u, v)$ be the *minimum* weight edge connecting any vertex in S to a vertex in $V - S$, then every minimum spanning tree must include e.

Proof. (By contradiction)

Suppose T is a spanning tree that does not contain $e = (u, v)$.

Main Idea: We will construct another spanning tree $T' = T \cup e - e'$ with weight less than T ($\Rightarrow\!\Leftarrow$)

Question: How to find such an edge e'?

Activity Review: Cut Property

Proof (Cut Property). (By contradiction.)

Suppose T is a spanning tree that does not contain $e = (u, v)$.

- Adding e to T results in a unique cycle C
- Cycle C must "enter" and "leave" cut S, that is, $\exists e' = (u', v') \in C$ s.t. $u' \in S, v' \in V - S$
- $w(e') > w(e)$ (**Why?**)
- $T' = T \cup e - e'$ is a spanning tree (**Why?**)
- $w(T') < w(T)$ ($\Rightarrow\!\Leftarrow$) $\blacksquare$

Kruskal’s Algorithm

CS136 Review: Priority Queue

Priority Queues manage a set S of items and the following operations on S:

- **Insert.** Insert a new element into S
- **Delete.** Delete an element from S
- **Extract.** Retrieve highest priority element in S

Priorities are encoded as a ‘key’ value

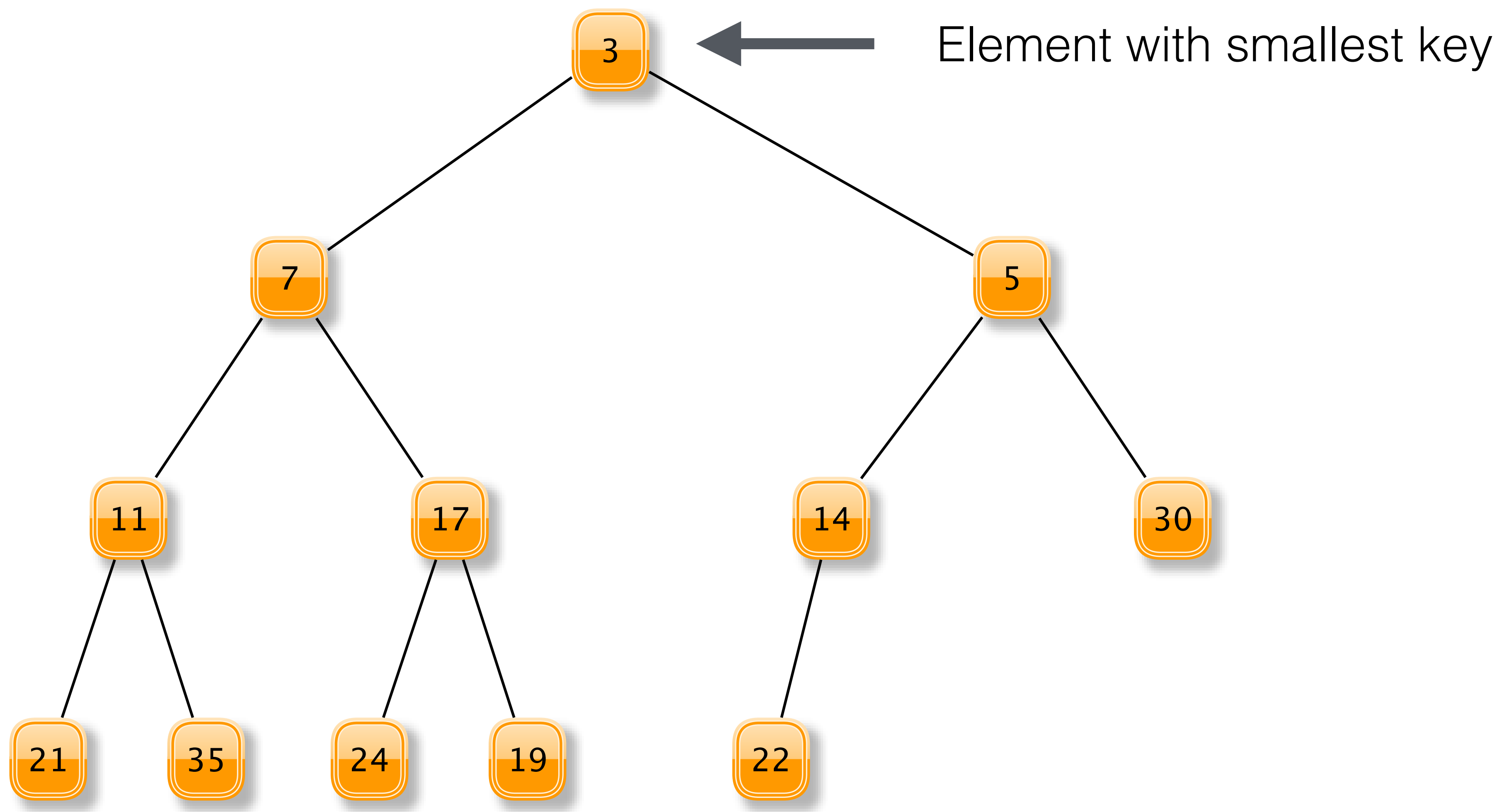
Typically: higher priority <—> lower key value (**Min**Heap)

Heap as Priority Queue. Combines tree structure with array access

- Insert and delete: $O(\log n)$ time (‘tree’ traversal & moves)
- **Extract min.** Delete item with minimum key value: $O(\log n)$

Heap Example

Heap property: For every element v, at node i, the element w at i's parent satisfies key(w) $\leq$ key(v)



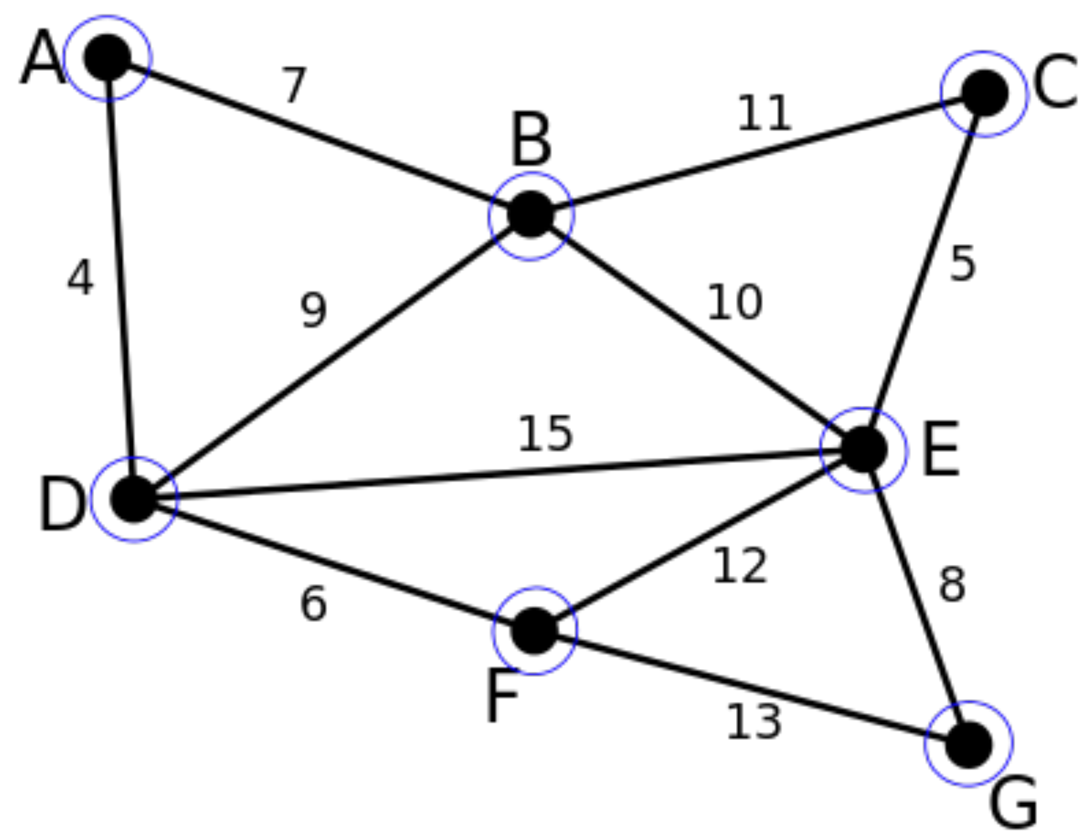
H

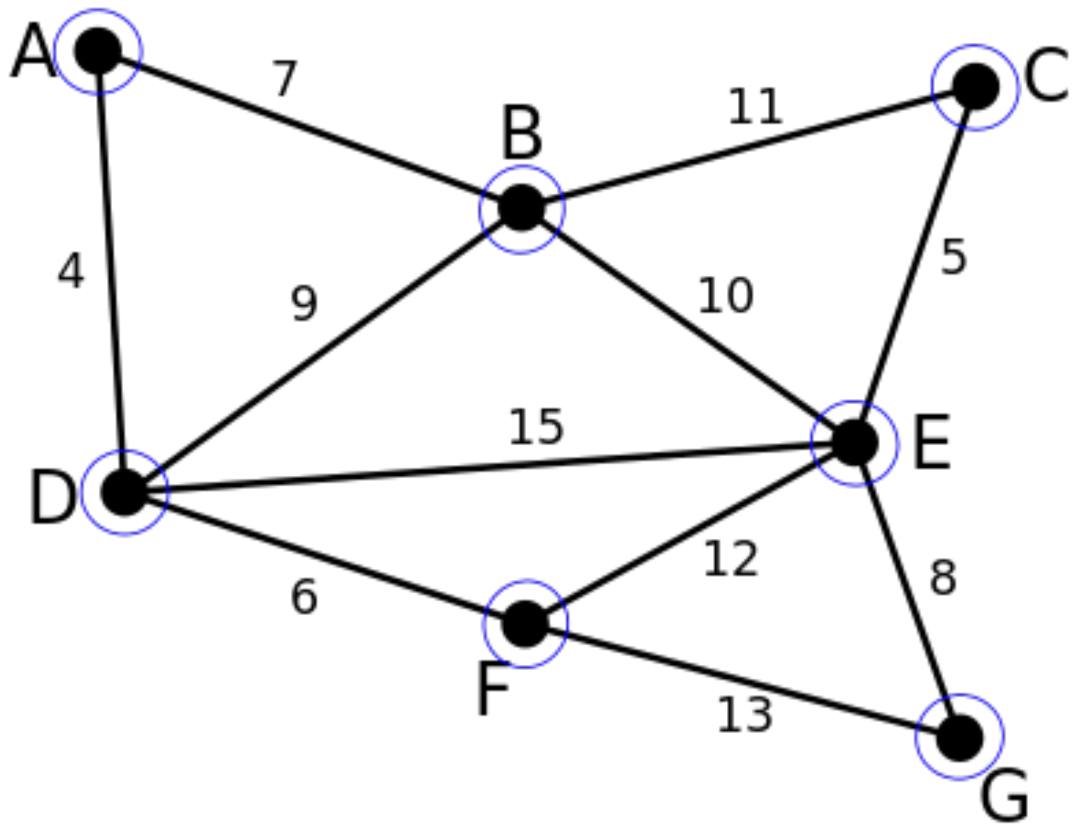
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	7	5	11	17	14	30	21	35	24	19	22	-	-	-	-

Kruskal's Algorithm

Idea: Add the cheapest remaining edge **that does not create a cycle**.

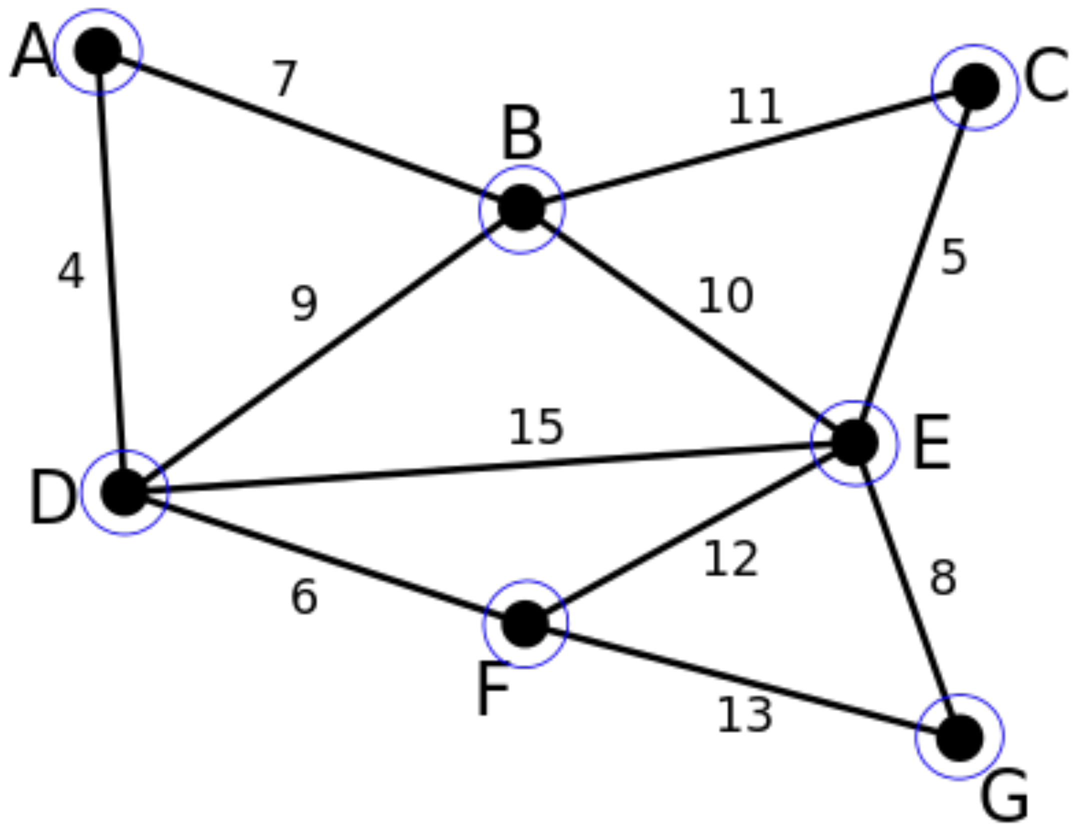
- Initialize $T = \varnothing$, $H \leftarrow E$
- While $|T| < n - 1$:
  - Remove cheapest edge e from H
  - If adding e to T does not create a cycle
    - $T \leftarrow T \cup \{e\}$
  - $H \leftarrow H - \{e\}$





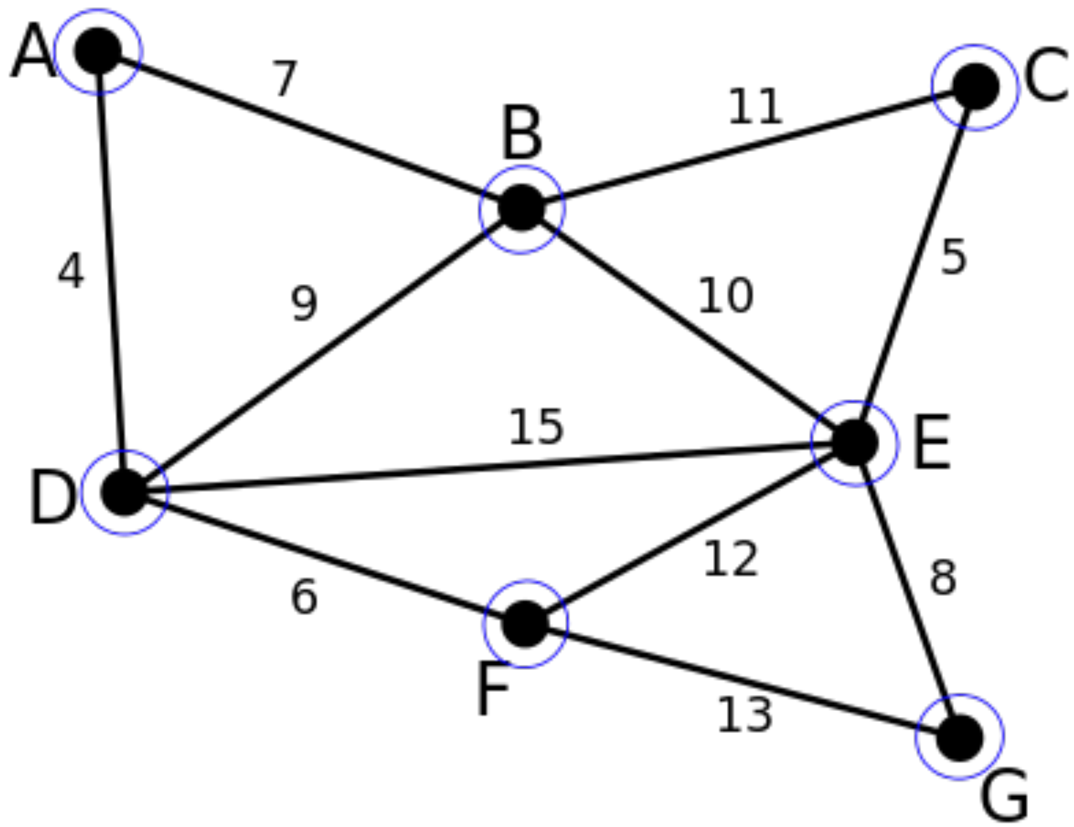
Idea: Add the cheapest remaining edge **that does not create a cycle**.

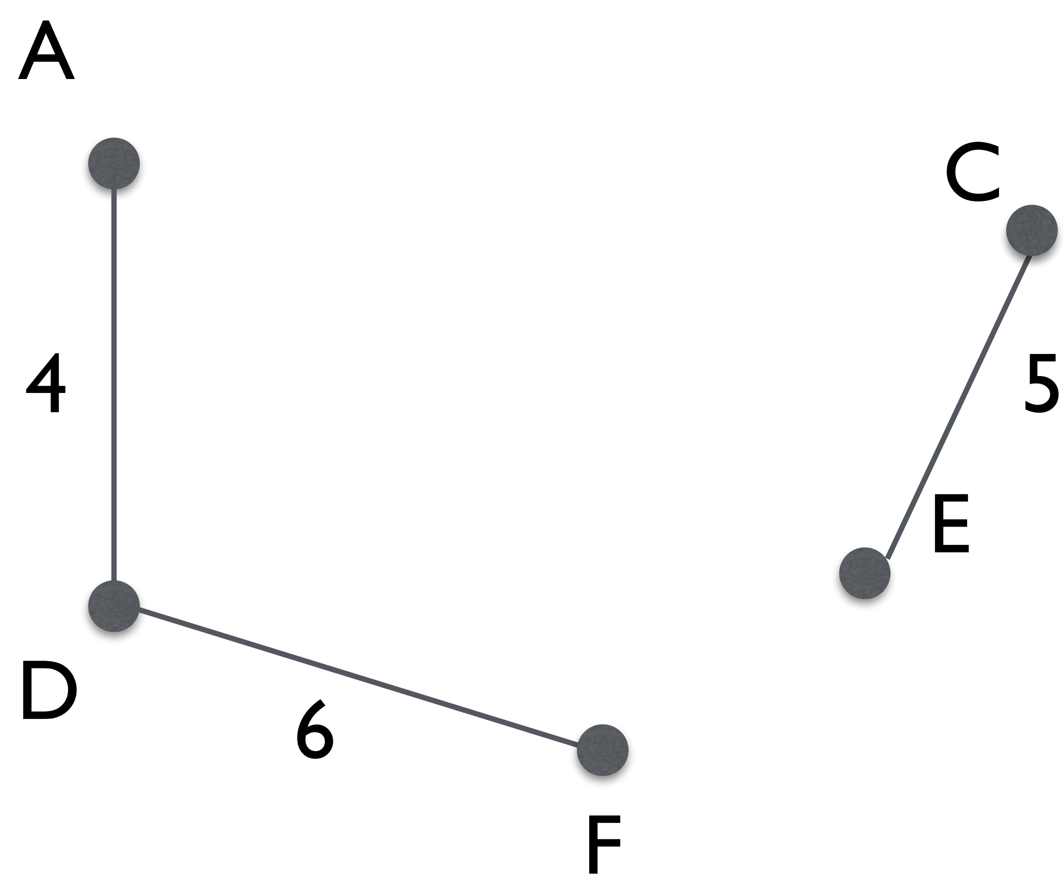
- Initialize $T = \varnothing$, $H \leftarrow E$
- While $|T| < n - 1$:
    - Remove cheapest edge e from H
    - If adding e to T does not create a cycle
        - $T \leftarrow T \cup \{e\}$
    - $H \leftarrow H - \{e\}$



Idea: Add the cheapest remaining edge **that does not create a cycle**.

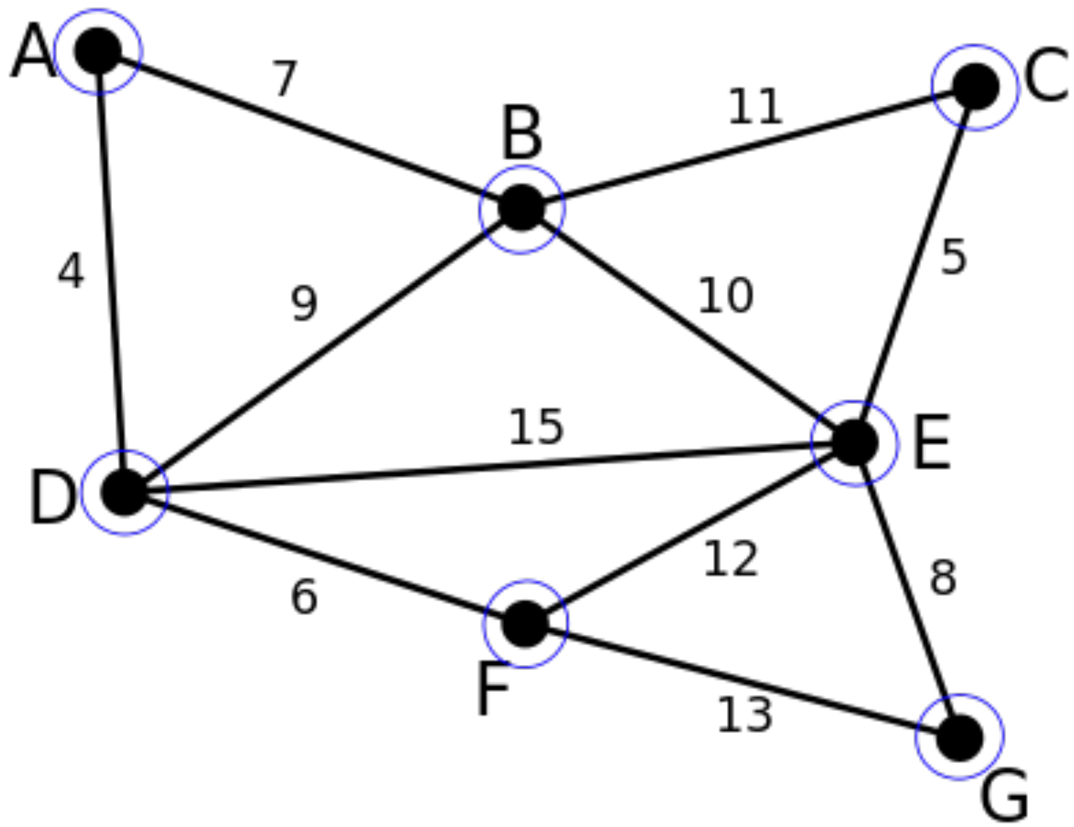
- Initialize $T = \varnothing$, $H \leftarrow E$
- While $|T| < n - 1$:
    - Remove cheapest edge e from H
    - If adding e to T does not create a cycle
        - $T \leftarrow T \cup \{e\}$
    - $H \leftarrow H - \{e\}$





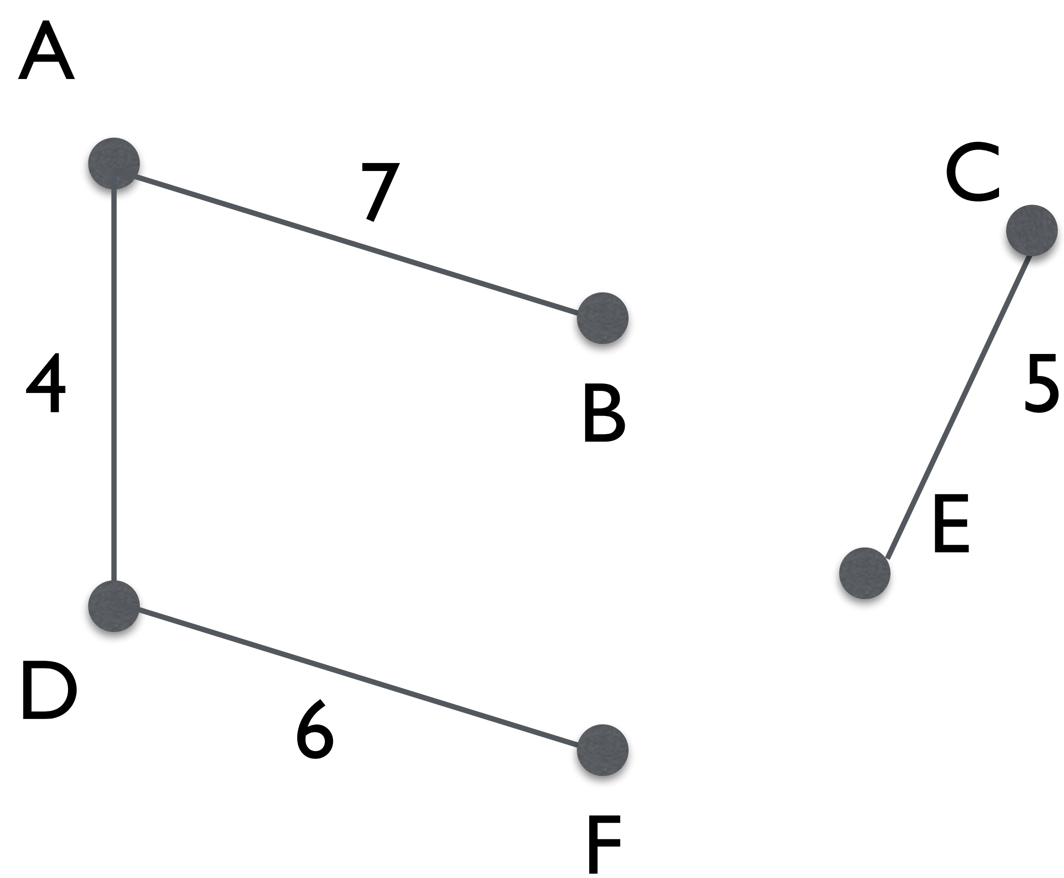
Idea: Add the cheapest remaining edge **that does not create a cycle**.

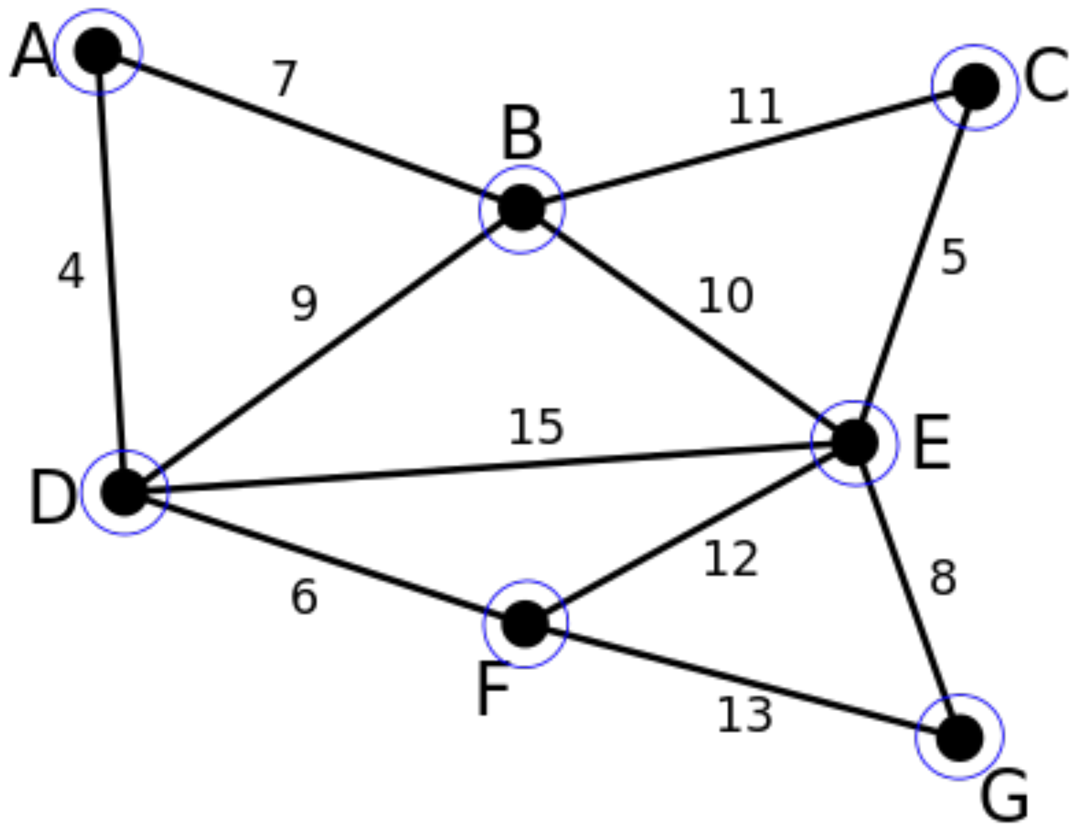
- Initialize $T = \varnothing$, $H \leftarrow E$
- While $|T| < n - 1$:
  - Remove cheapest edge e from H
  - If adding e to T does not create a cycle
    - $T \leftarrow T \cup \{e\}$
  - $H \leftarrow H - \{e\}$



Idea: Add the cheapest remaining edge **that does not create a cycle**.

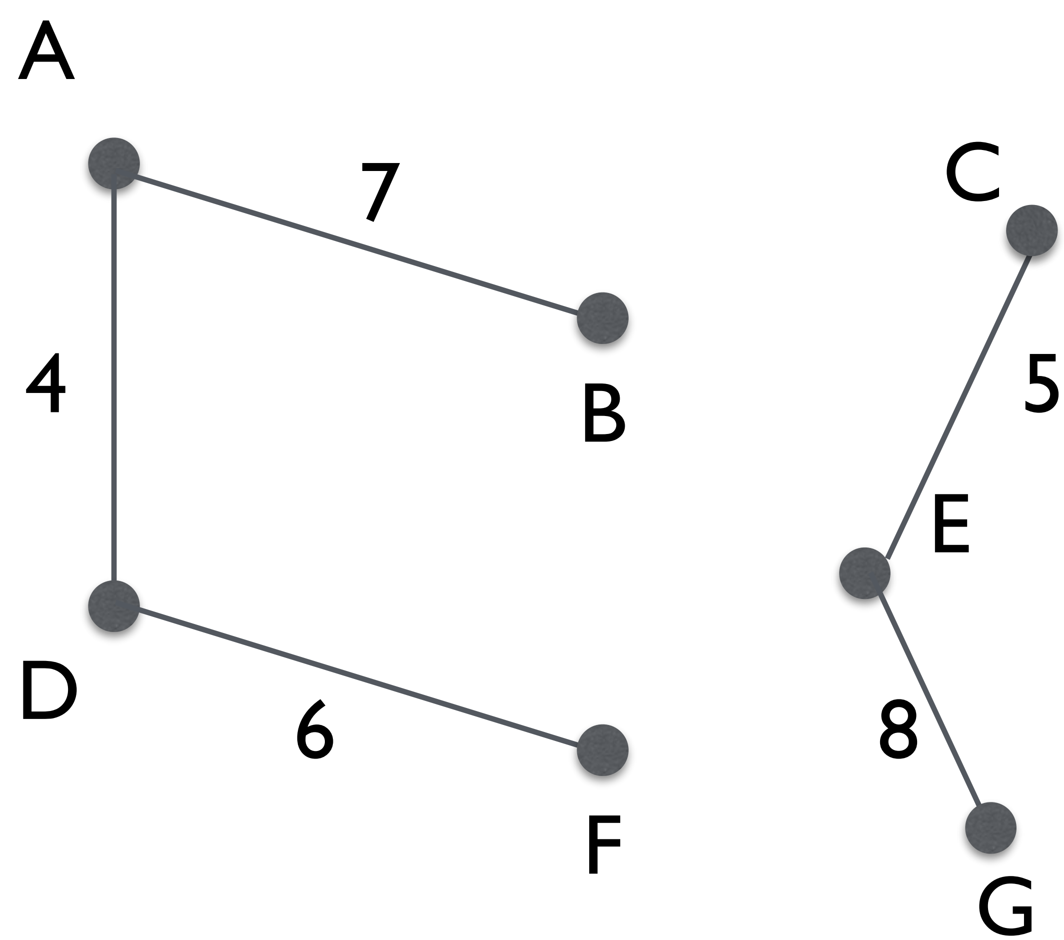
- Initialize $T = \emptyset$, $H \leftarrow E$
- While $|T| < n - 1$:
    - Remove cheapest edge e from H
    - If adding e to T does not create a cycle
        - $T \leftarrow T \cup \{e\}$
    - $H \leftarrow H - \{e\}$

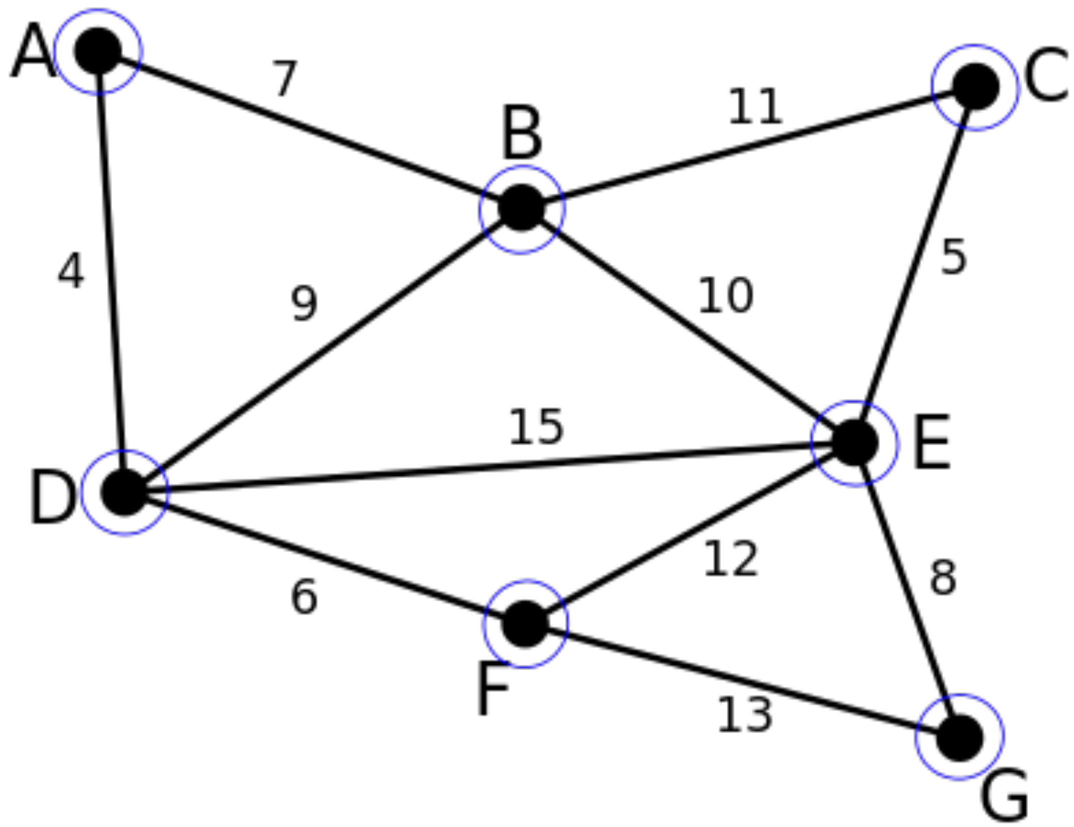




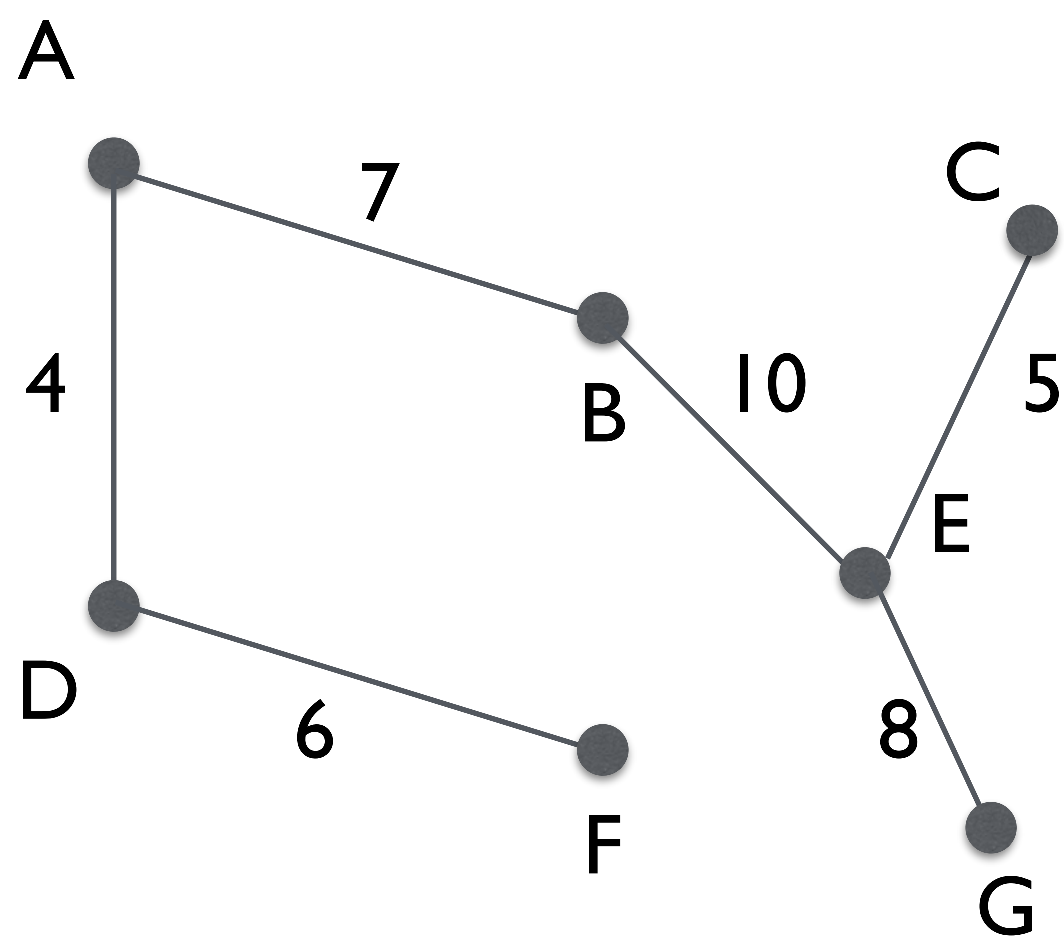
Idea: Add the cheapest remaining edge **that does not create a cycle**.

- Initialize $T = \emptyset$, $H \leftarrow E$
- While $|T| < n - 1$:
  - Remove cheapest edge e from H
  - If adding e to T does not create a cycle
    - $T \leftarrow T \cup \{e\}$
  - $H \leftarrow H - \{e\}$





Total weight: 40



Idea: Add the cheapest remaining edge **that does not create a cycle**.

- Initialize $T = \varnothing$, $H \leftarrow E$
- While $|T| < n - 1$:
  - Remove cheapest edge e from H
  - If adding e to T does not create a cycle
    - $T \leftarrow T \cup \{e\}$
  - $H \leftarrow H - \{e\}$

Kruskal's Analysis

- **Correctness**: Does it give us the correct MST?
- **Key Question:** Why is each edge (v, w) that we are adding safe?
  - Consider the step just before (v, w) is added
    - Let $S = \{x \in V \mid T$ contains a path from v to $x\}$
    - This is a valid cut in the graph (**Why**? Can $w \in S$?)
    - If there was a cheaper cut edge for cut $(S, V - S)$ which did not form a cycle, the algorithm would have already added it; this must be the min-cost cut edge for this cut
- **Runtime**.
  - How quickly can we find the minimum remaining edge?
  - How quickly can we determine if an edge creates a cycle?

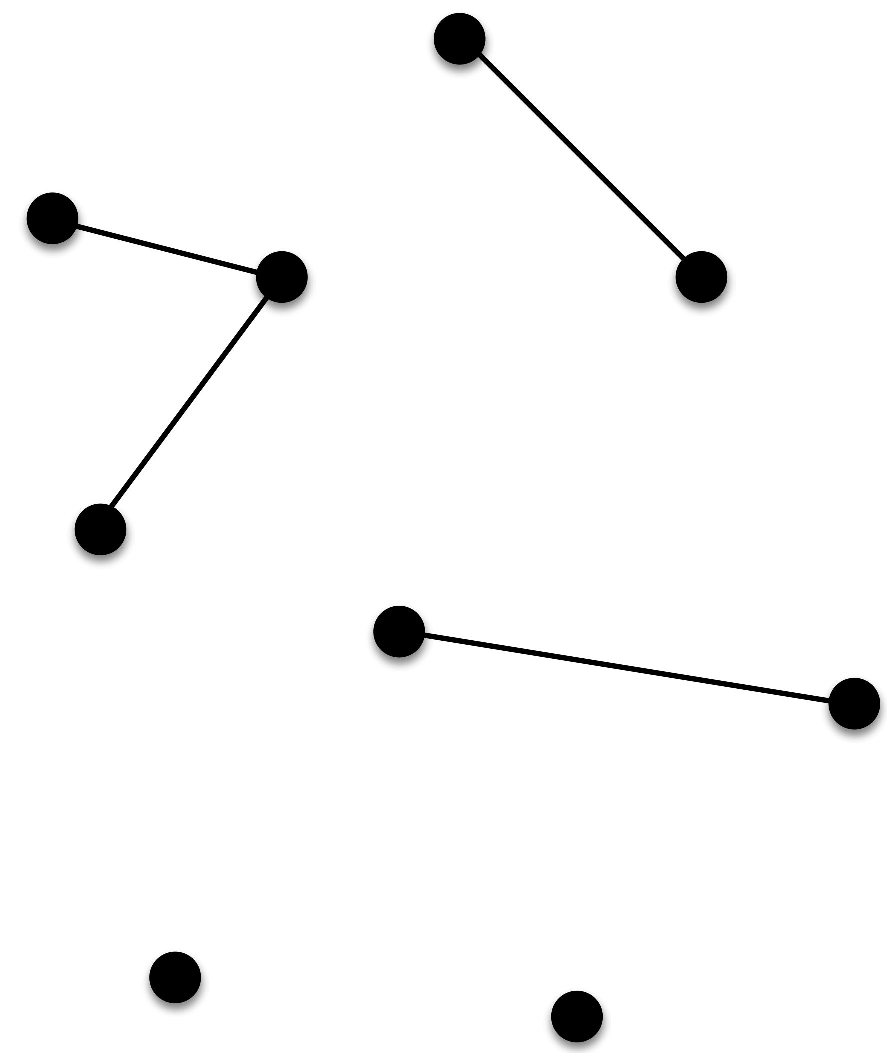
Kruskal's Implementation

What steps do we need to implement?

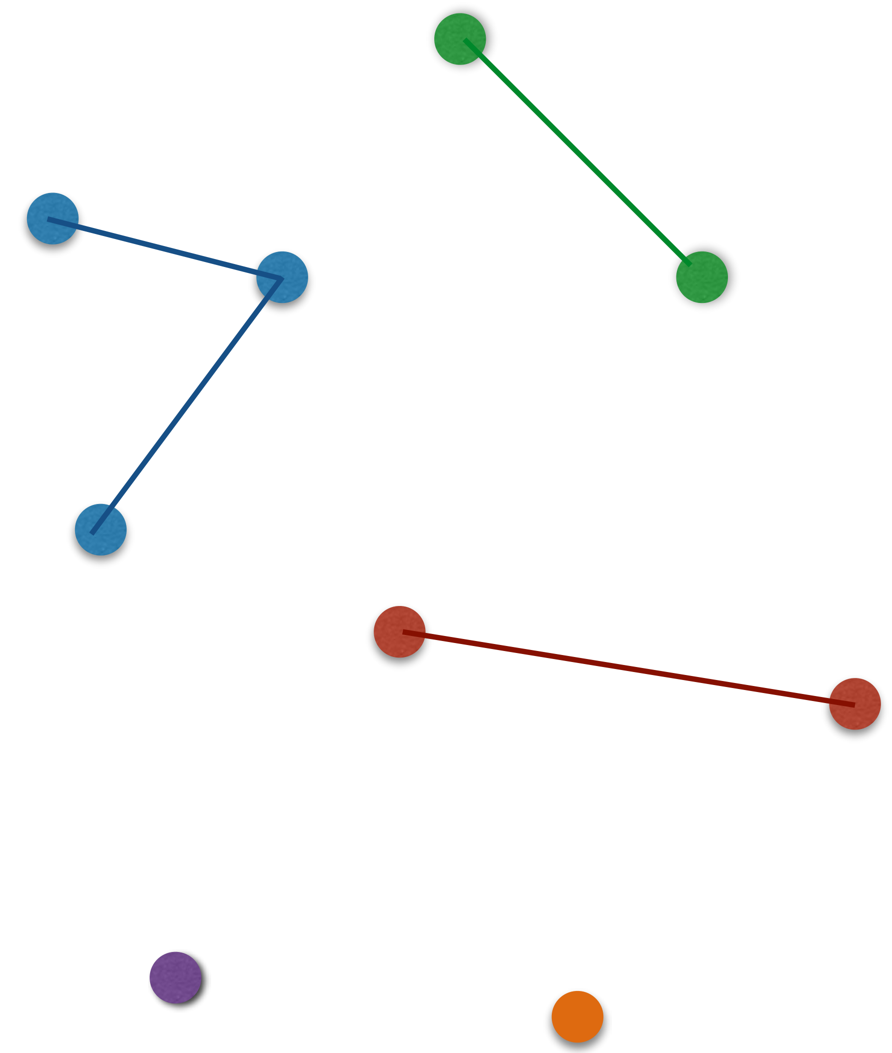
- Sort edges by weight (add to heap): $O(m \log m)$
    - If we do the rest efficiently, this is the dominant cost
- Determine whether $T \cup \{e\}$ contains a cycle
    - Ideas?
- Add an edge to T

Does this edge create a cycle?

- An edge creates a cycle if it connects a subtree to another vertex in the same subtree
- What if we could label the vertices in a tree? Then we could determine if an edge creates a cycle by comparing vertex labels



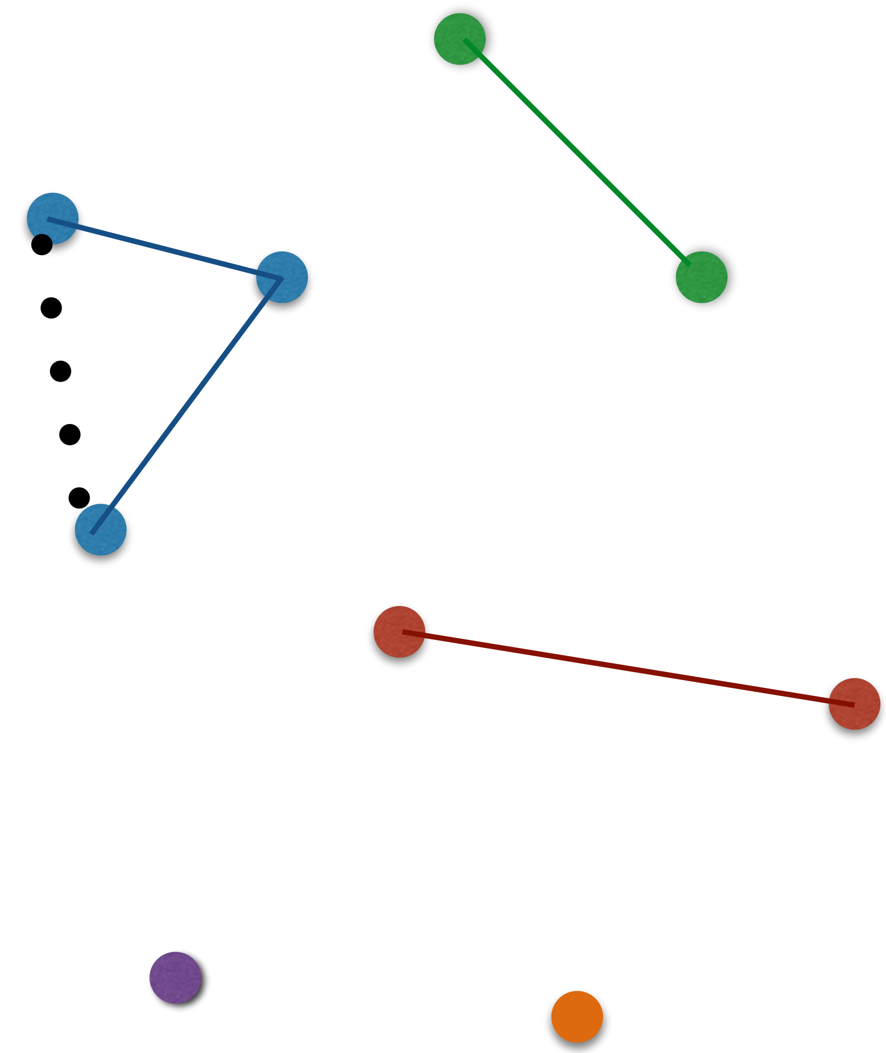
Does this edge create a cycle?



- An edge creates a cycle if it connects a subtree to another vertex in the same subtree
- What if we could label the vertices in a tree? Then we could determine if an edge creates a cycle by comparing vertex labels

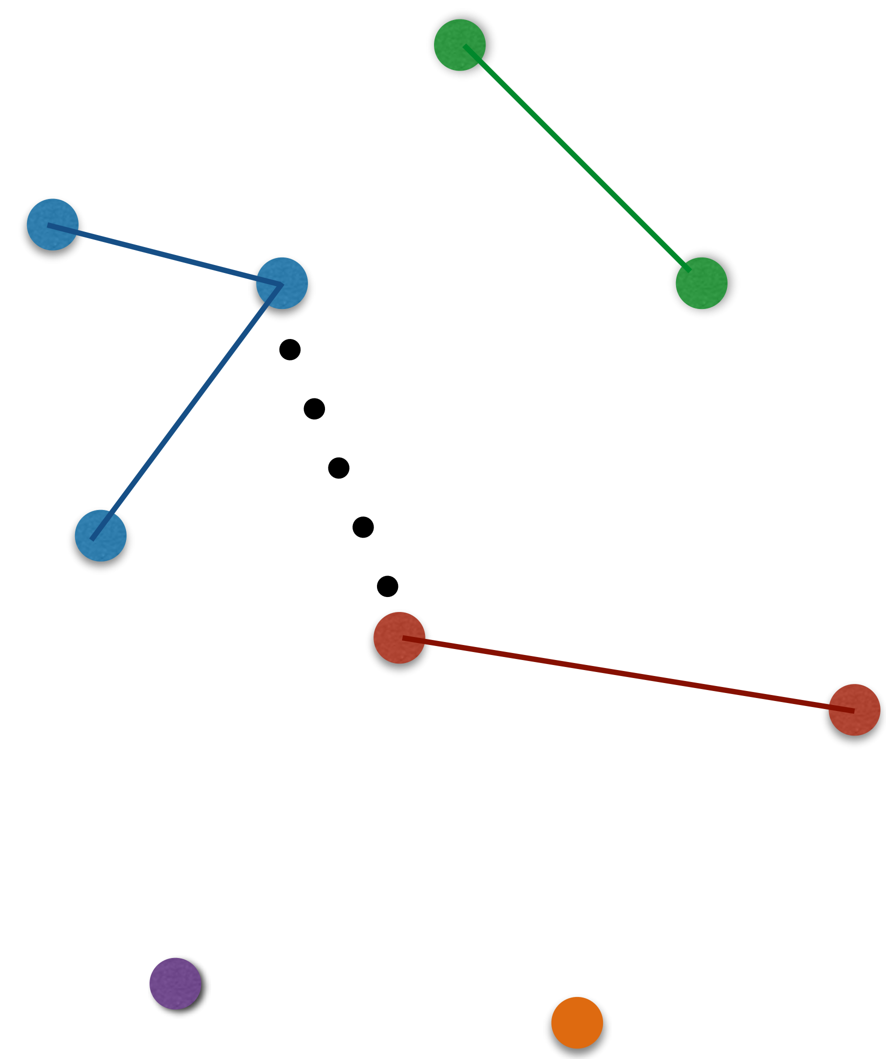
Does this edge create a cycle?

- An edge creates a cycle if it connects a subtree to another vertex in the same subtree
- What if we could label the vertices in a tree? Then we could determine if an edge creates a cycle by comparing vertex labels



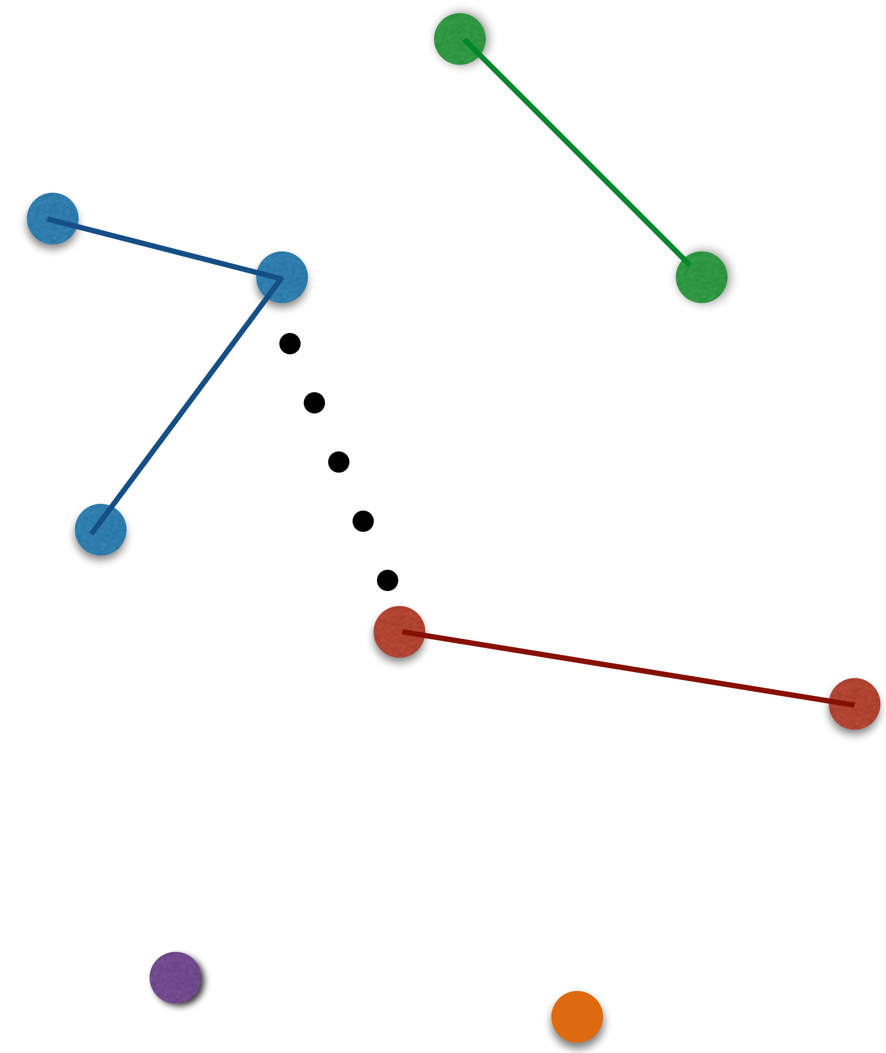
Does this edge create a cycle?

- An edge creates a cycle if it connects a subtree to another vertex in the same subtree
- What if we could label the vertices in a tree? Then we could determine if an edge creates a cycle by comparing vertex labels



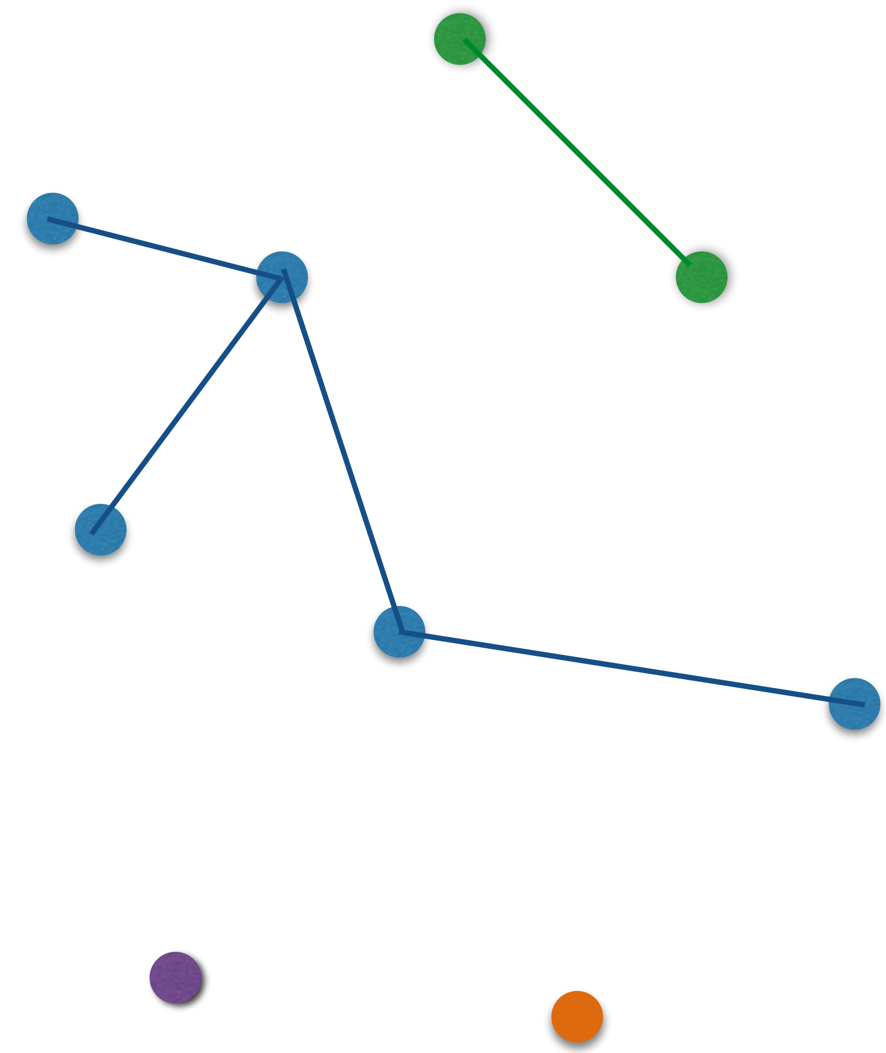
Does this edge create a cycle?

- An edge creates a cycle if it connects a subtree to another vertex in the same subtree
- What if we could label the vertices in a tree? Then we could determine if an edge creates a cycle by comparing vertex labels
- How can we update vertex labels when adding an edge?



Does this edge create a cycle?

- An edge creates a cycle if it connects a subtree to another vertex in the same subtree
- What if we could label the vertices in a tree? Then we could determine if an edge creates a cycle by comparing vertex labels
- How can we update vertex labels when adding an edge?



Ideally, what would we do?

- Start with each node as its own set
- Given a node, determine which set it's in (i.e., a label)
- Take two sets and combine them into a single set with a single label

Union-Find Data Structure

Manages a **dynamic partition** of a set S

- Provides the following methods:
    - `MakeUnionFind()`: Initializes each vertex/set with unique label
    - `Find(x):` Return label of set containing x
    - `Union(X, Y)`: Replace sets `X, Y` with $X \cup Y$ with single label

Kruskal's Algorithm can then use

- `Find` for cycle checking
- `Union` to update after adding an edge to T

Acknowledgments

- These slides are based on material from Shikha Singh.
- The pictures in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)