CSE 417 Autumn 2025

Lecture 11: Minimum spanning trees

Glenn Sun

Logistics

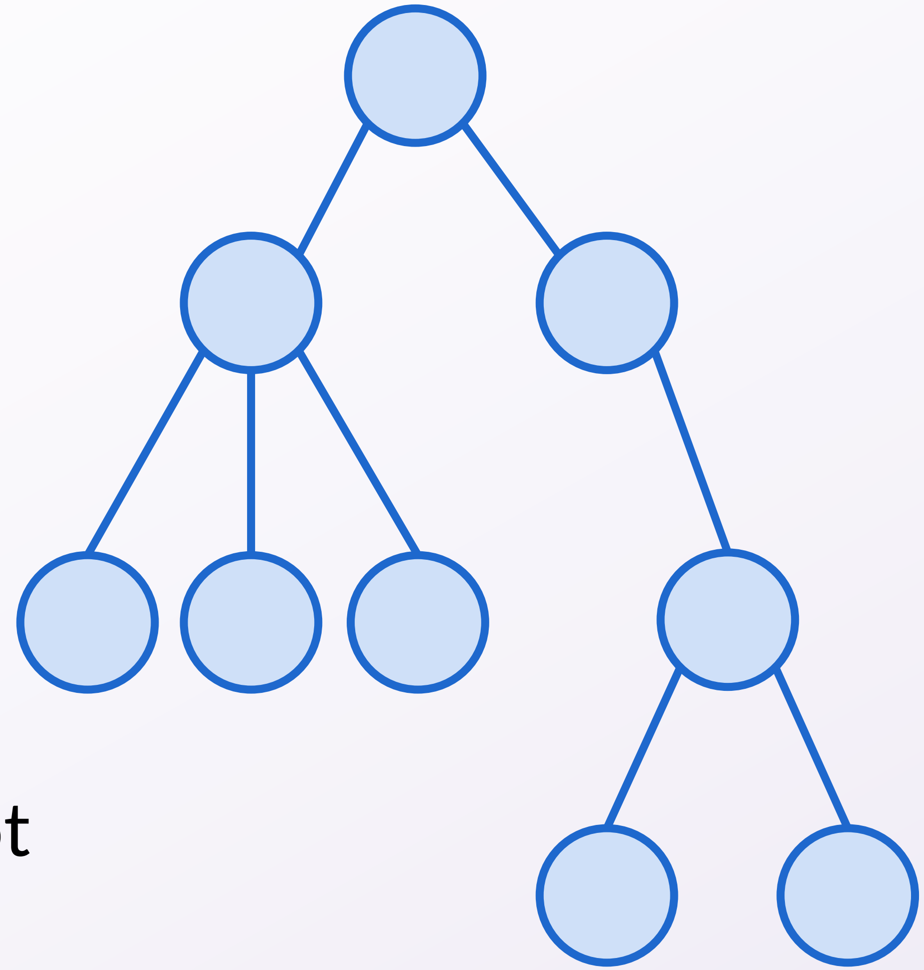
- HW 4 on graphs out after class:
  - Problem 7, 7X.1/2: Using MSTs (today's topic) for clustering
  - Problem 8: A graph modeling problem
- HW 1 solutions are out on Canvas!
- Practice quiz out on Canvas/website tonight.

More graph review

Trees

In a **rooted tree**:

- Each vertex has one ***parent*** above it (except the root, which has none)
- Each vertex can have zero or more ***children*** below it
- One way to reach each vertex from root



Examples: file tree, binary search tree, etc.

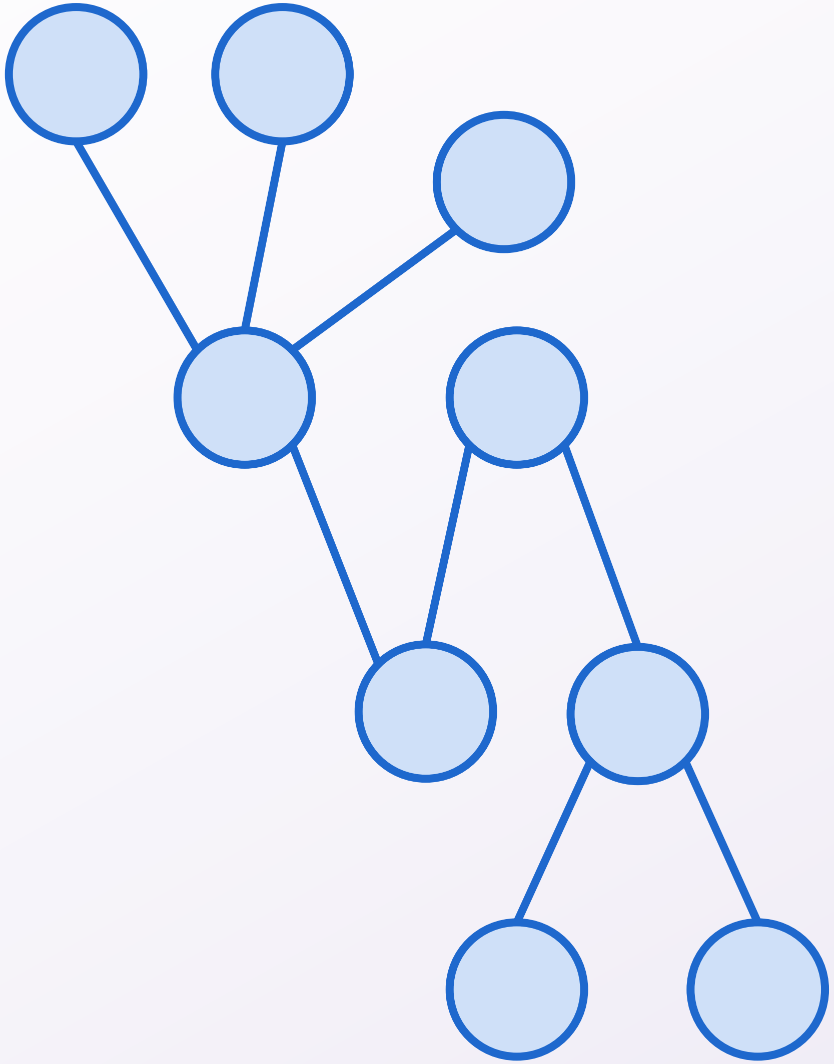
Trees

In an (unrooted) **tree**:

- No concept of parents, children, root, etc.
- **A connected graph with no cycles**

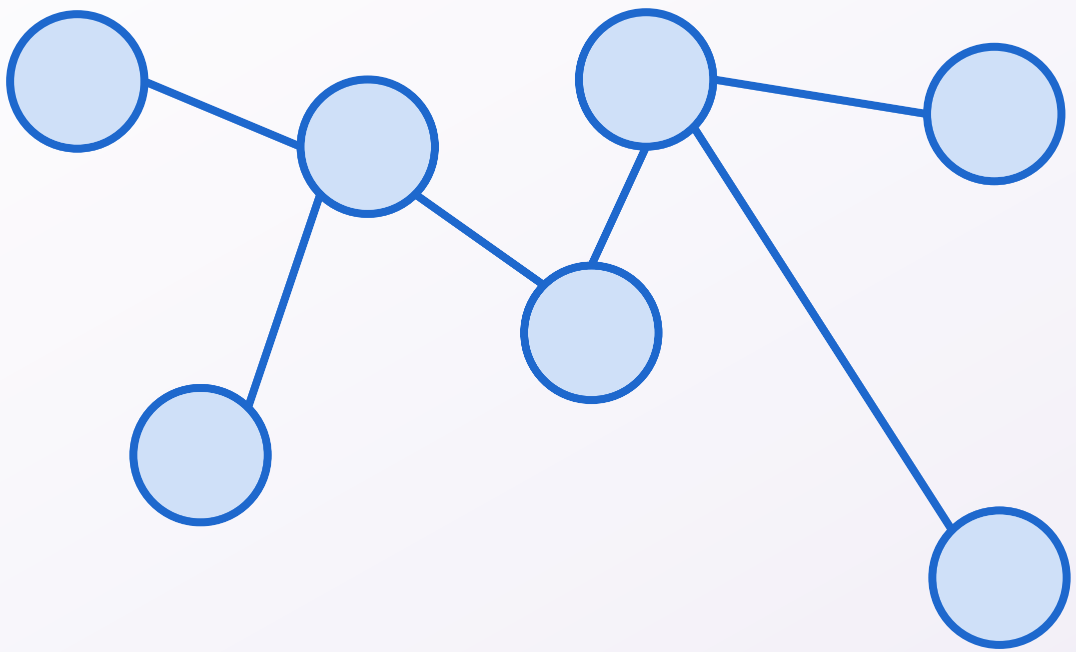
In other words, take a rooted tree and "forget" what the root is.

Resulting connectivity structure is a tree!



Number of edges

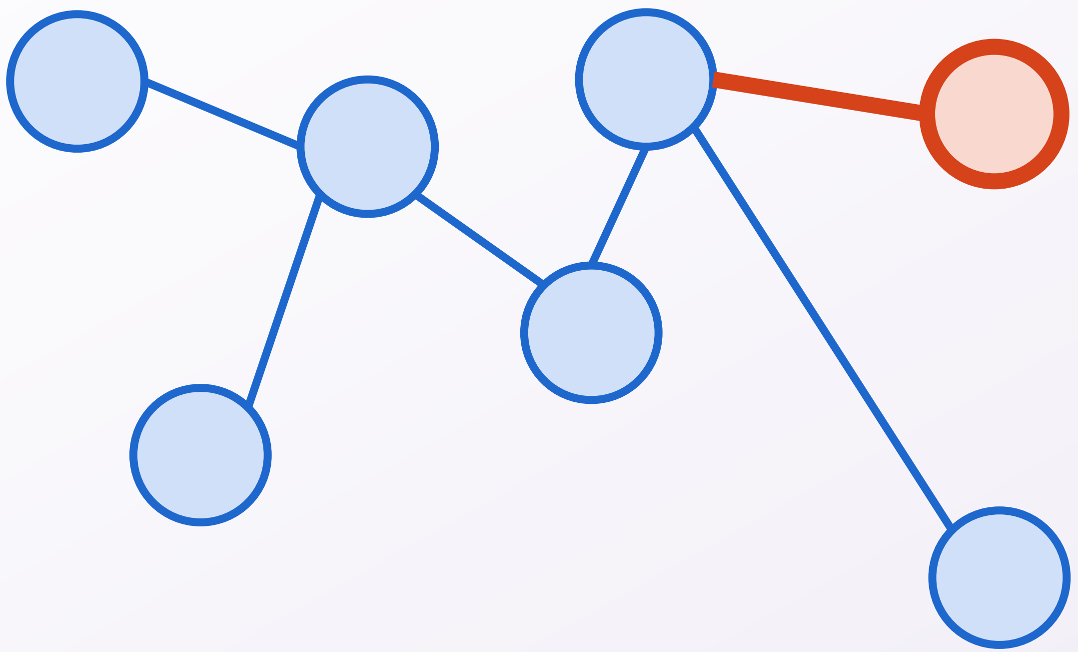
Claim. A tree with n vertices has $n - 1$ edges.



Number of edges

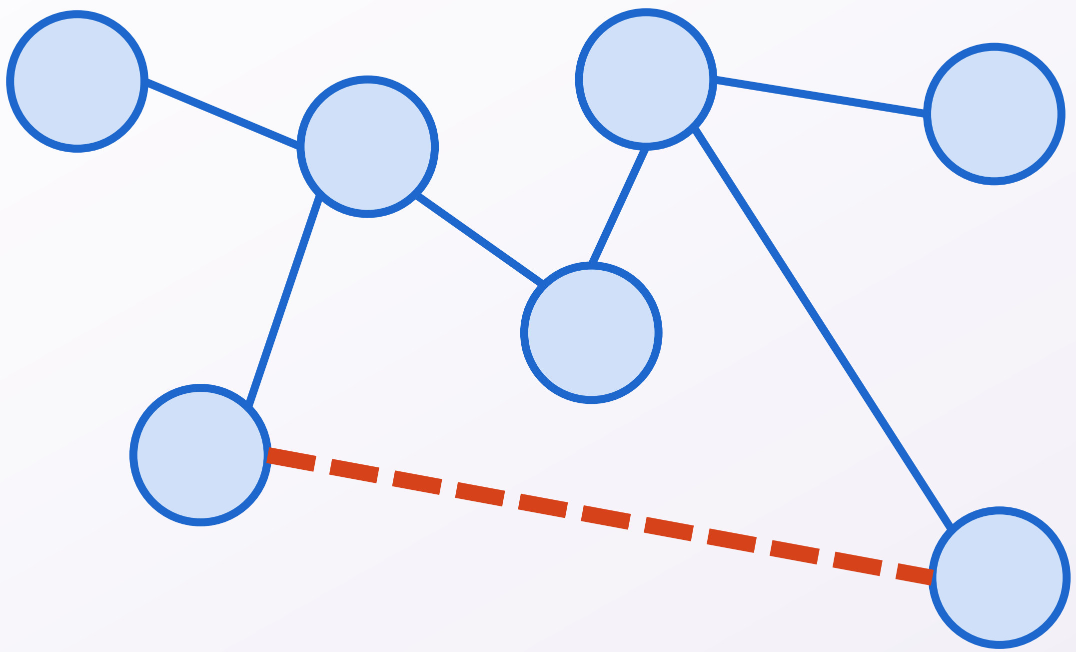
Claim. A tree with n vertices has $n - 1$ edges.

Proof. (Handwaving a bit) If you keep removing leaves, you remove 1 vertex and 1 edge at a time, until you just have one node.



Adding edges to trees

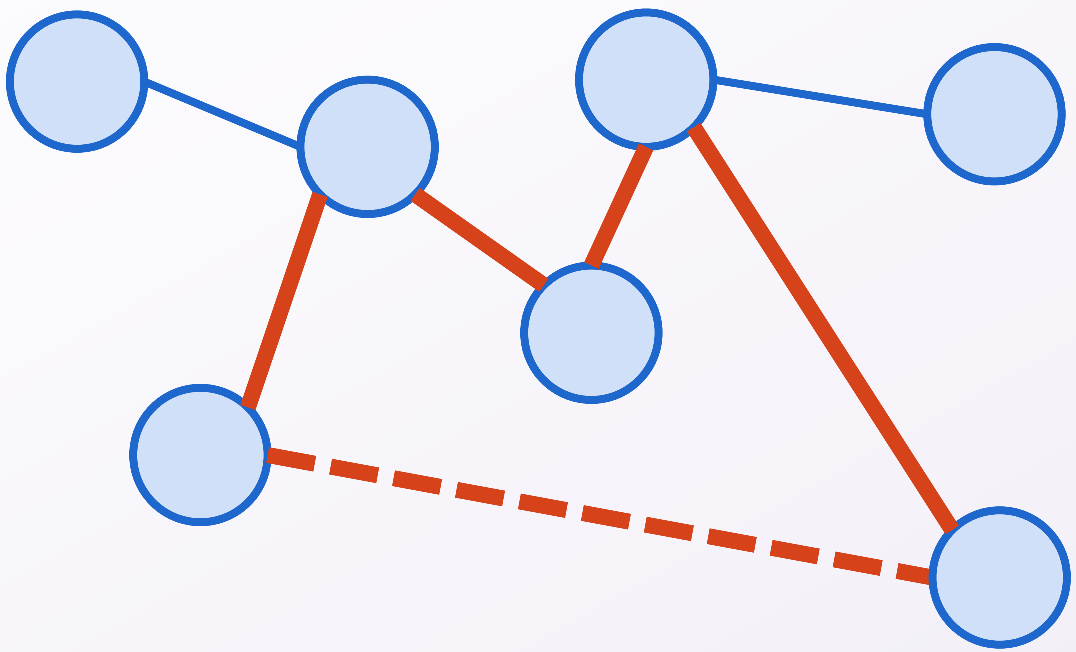
Claim. Adding any edge to a tree creates a cycle.



Adding edges to trees

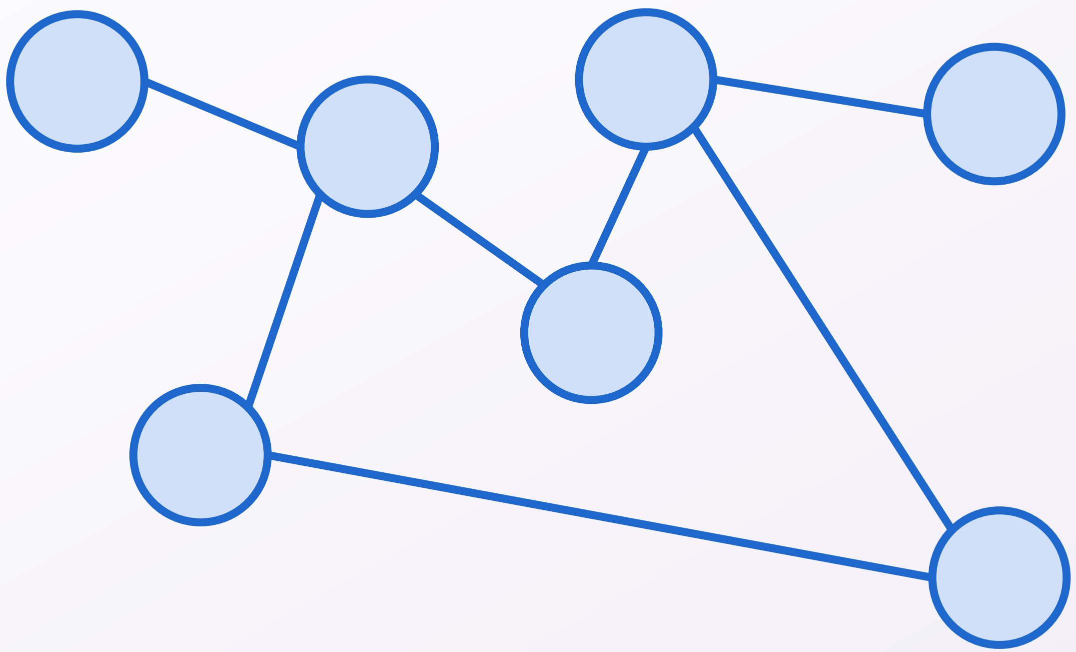
Claim. Adding any edge to a tree creates a cycle.

Proof. The graph is already connected, so this edge turns the original path into a cycle.



Minimal connectedness

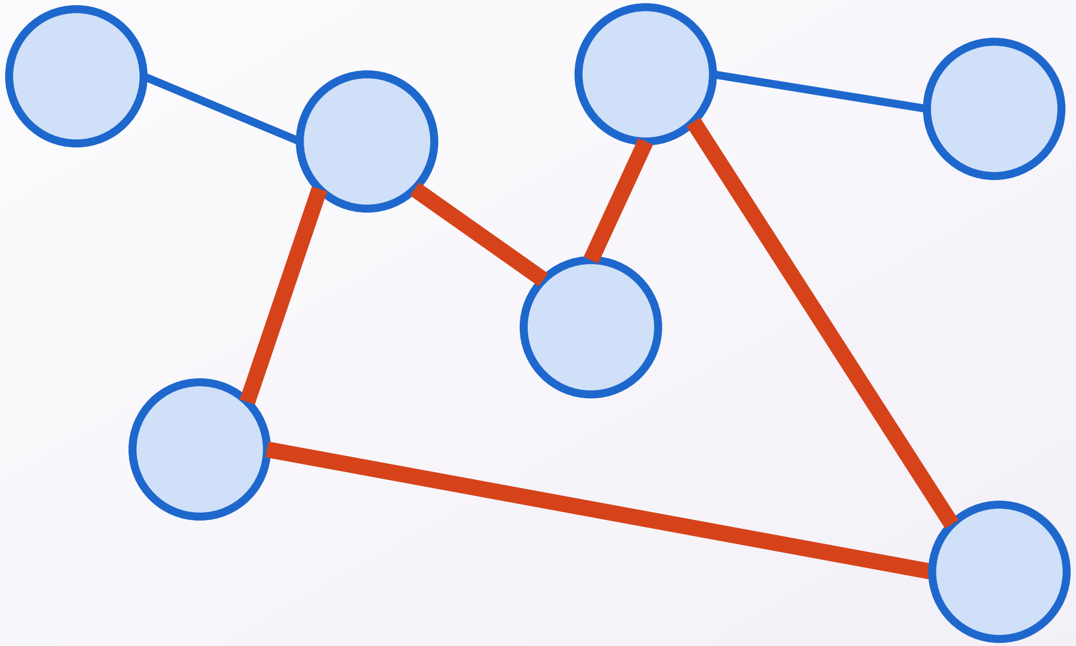
Claim. If a connected graph is not a tree, you can remove an edge and still be connected.



Minimal connectedness

Claim. If a connected graph is not a tree, you can remove an edge and still be connected.

Proof. Take any cycle and delete any edge in it.

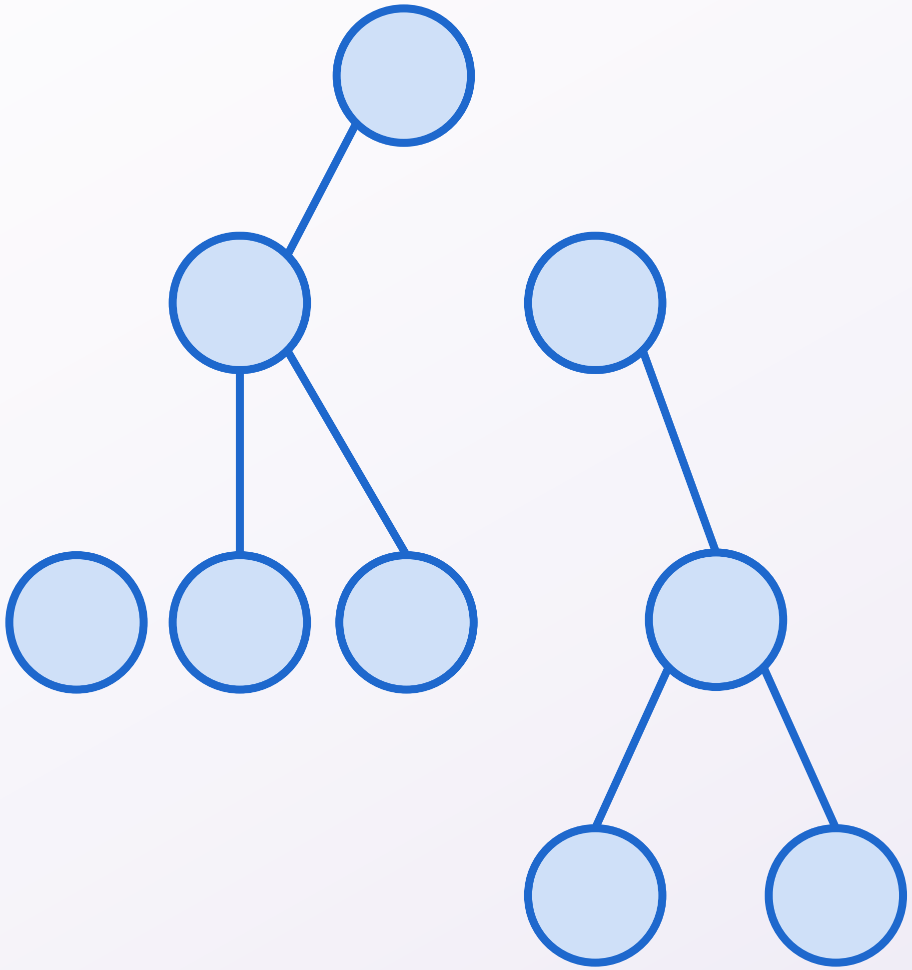


Forests

Multiple trees in a single graph form a **forest**.

(0 trees and 1 tree are also technically forests.)

Equivalently, any undirected graph with no cycles is a forest.

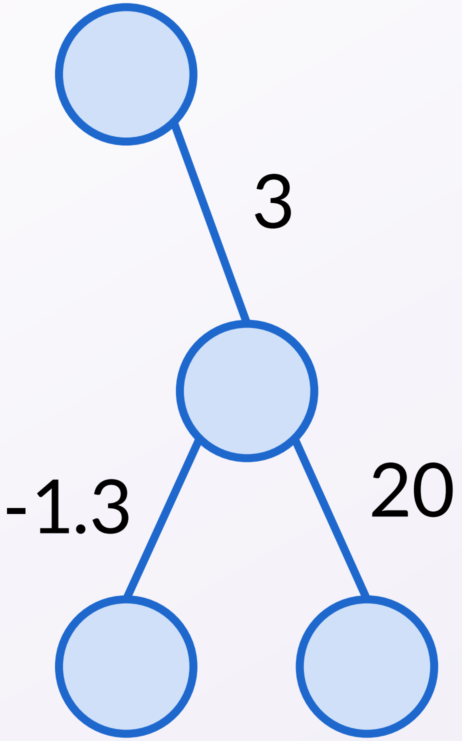


Weighted graphs

Sometimes, we put "weights" on edges.

They can represent:

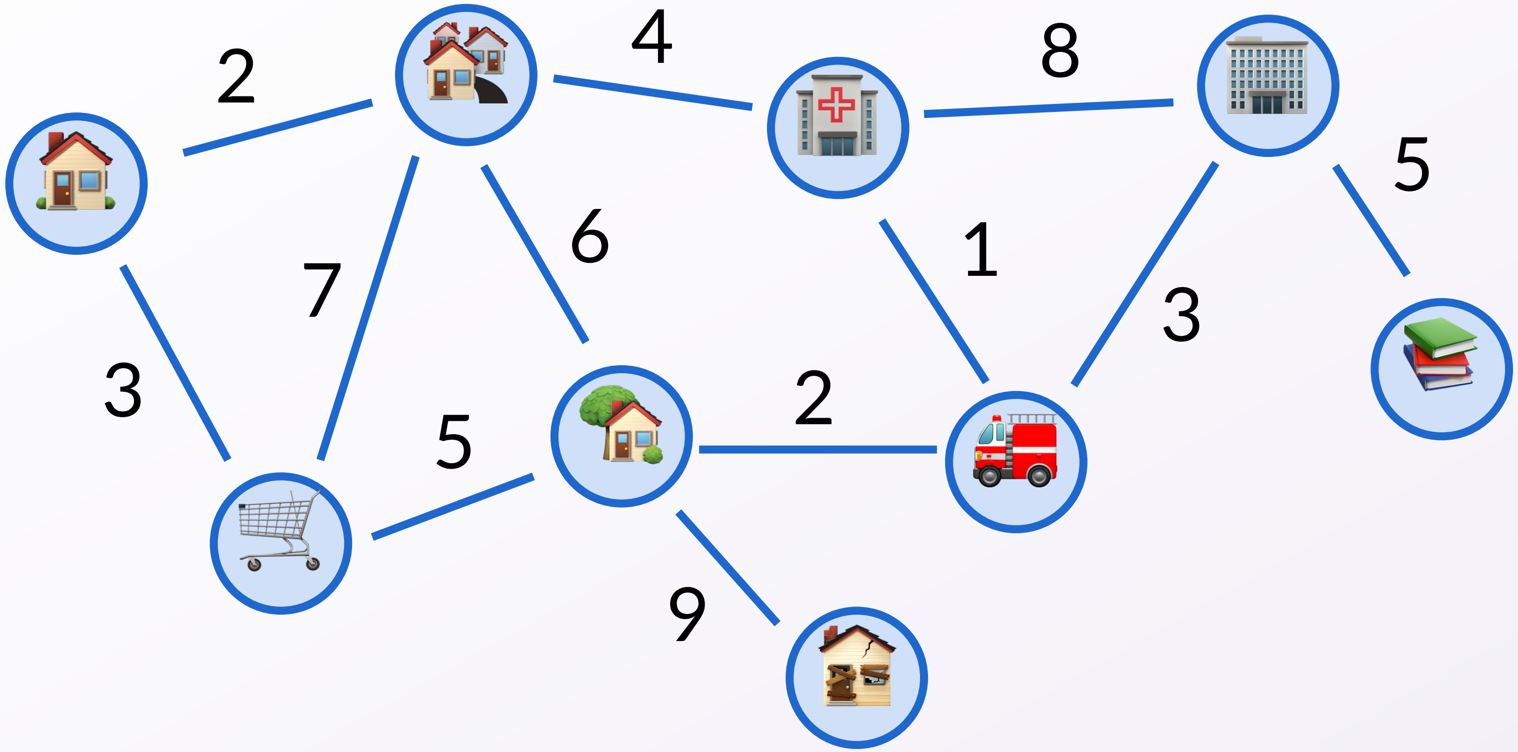
- Distance
- Cost
- Capacity
- Etc.



Minimum spanning trees

Emergency snow network

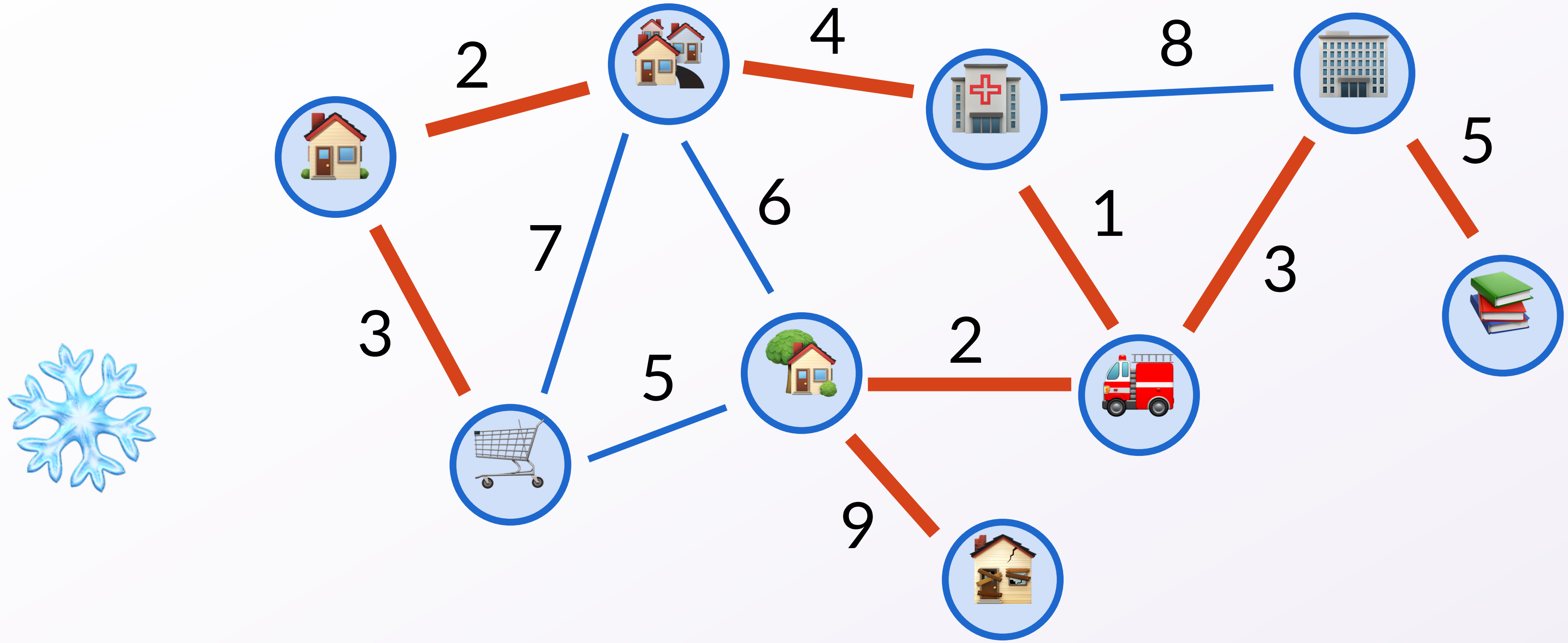
A city has a network of roads of various lengths. (diagram not to scale)



Emergency snow network

Goal: Find a minimum length network to plow that connects everyone during a snowstorm.

(Tree, by minimal connectedness!)

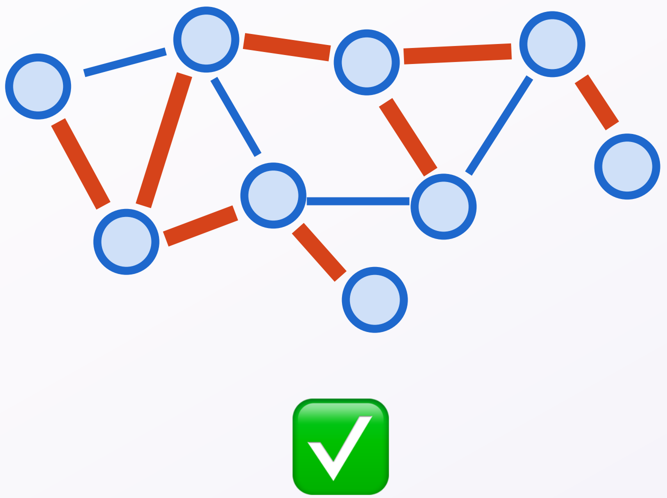


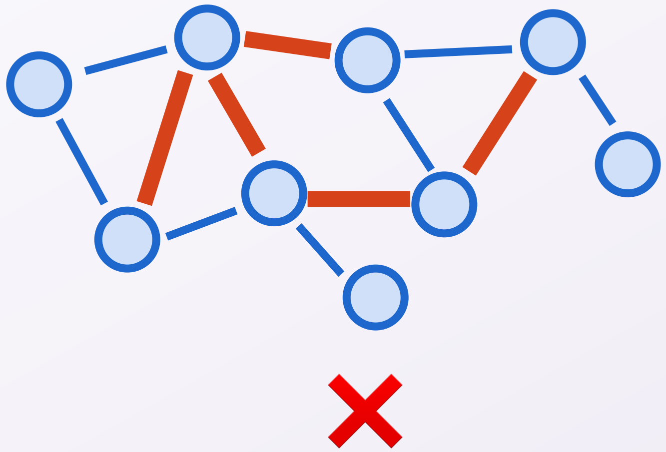
Minimum spanning tree

Input: A connected, undirected, weighted graph with vertices V and edges E

Goal: Find a spanning tree of minimum total weight

Spanning tree: subset of E that forms a tree on all of V





MST algorithms

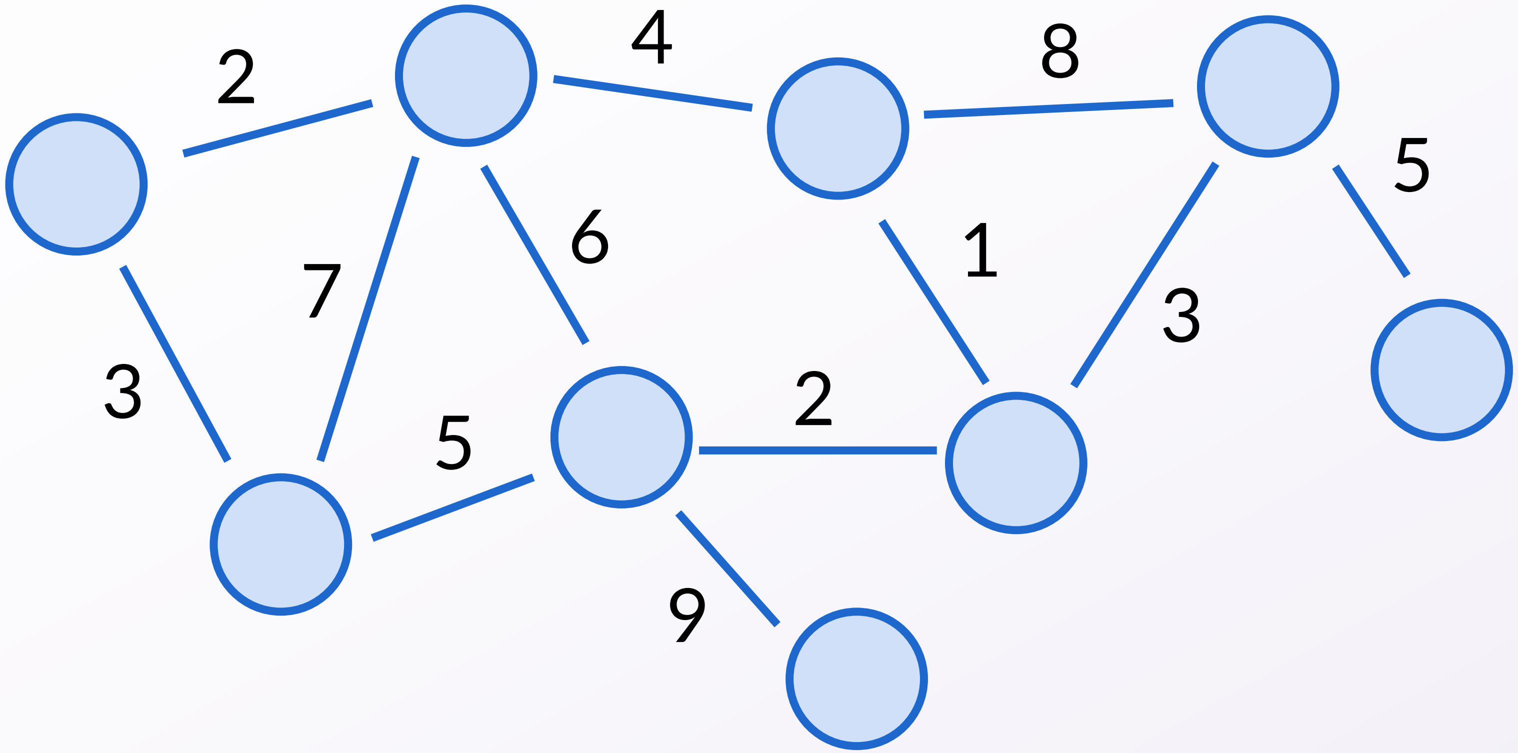
Kruskal's algorithm:

1. **repeat $n - 1$ times**
2. Pick the cheapest edge that doesn't create a cycle.

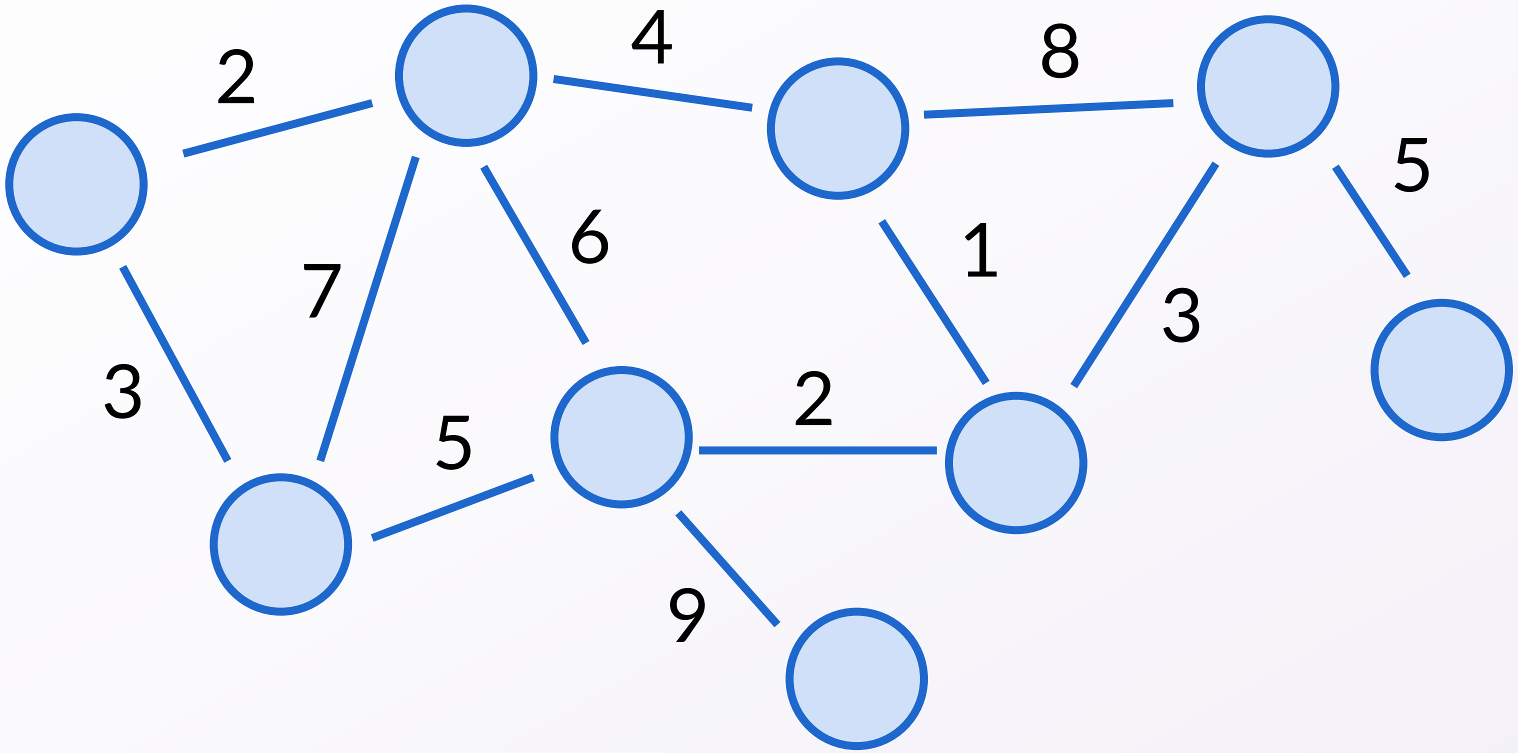
Prim's algorithm:

1. **repeat $n - 1$ times**
2. Pick the cheapest edge that extends the current tree to a new vertex.

Kruskal’s algorithm demonstration



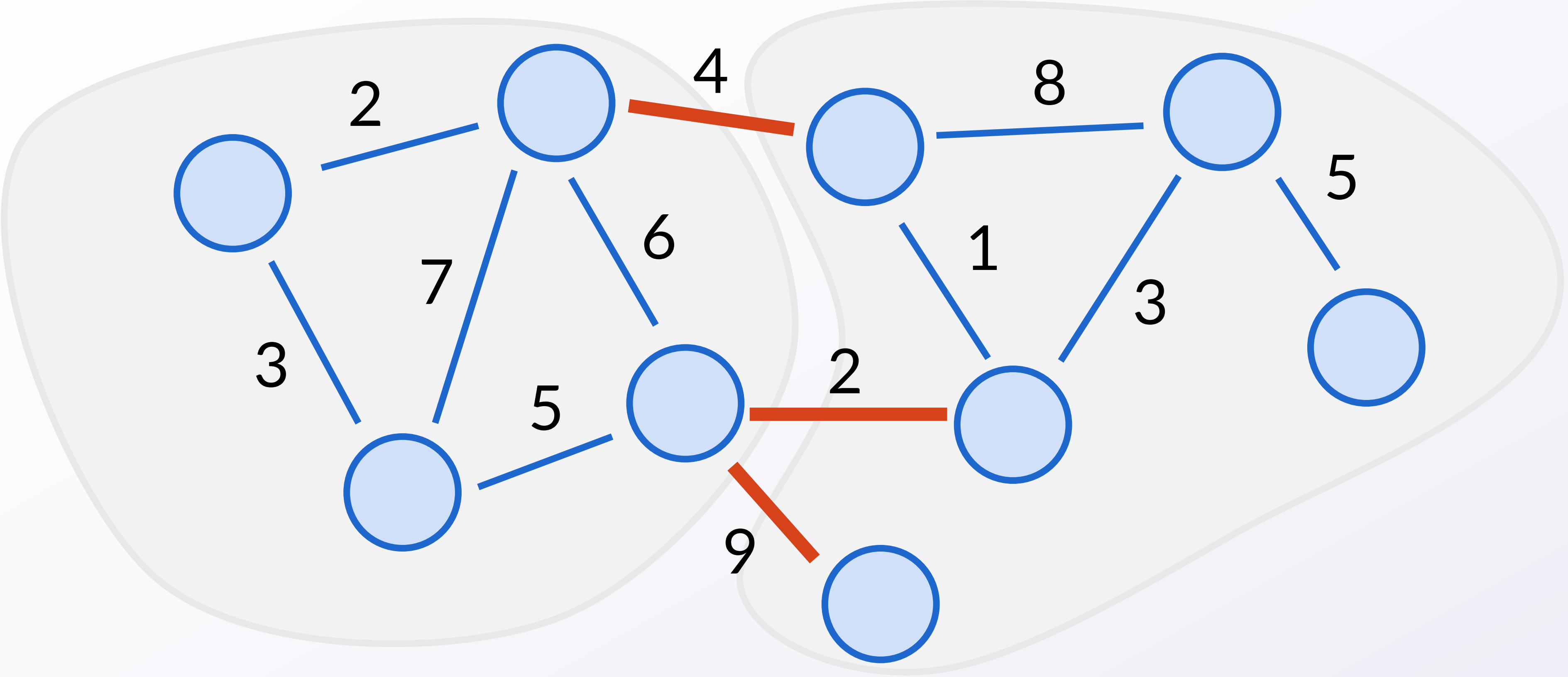
Prim's algorithm demonstration



Cut property of MSTs

A **cut** splits the vertices of a graph into two parts.

An edge **crosses the cut** if it has one endpoint in each part.

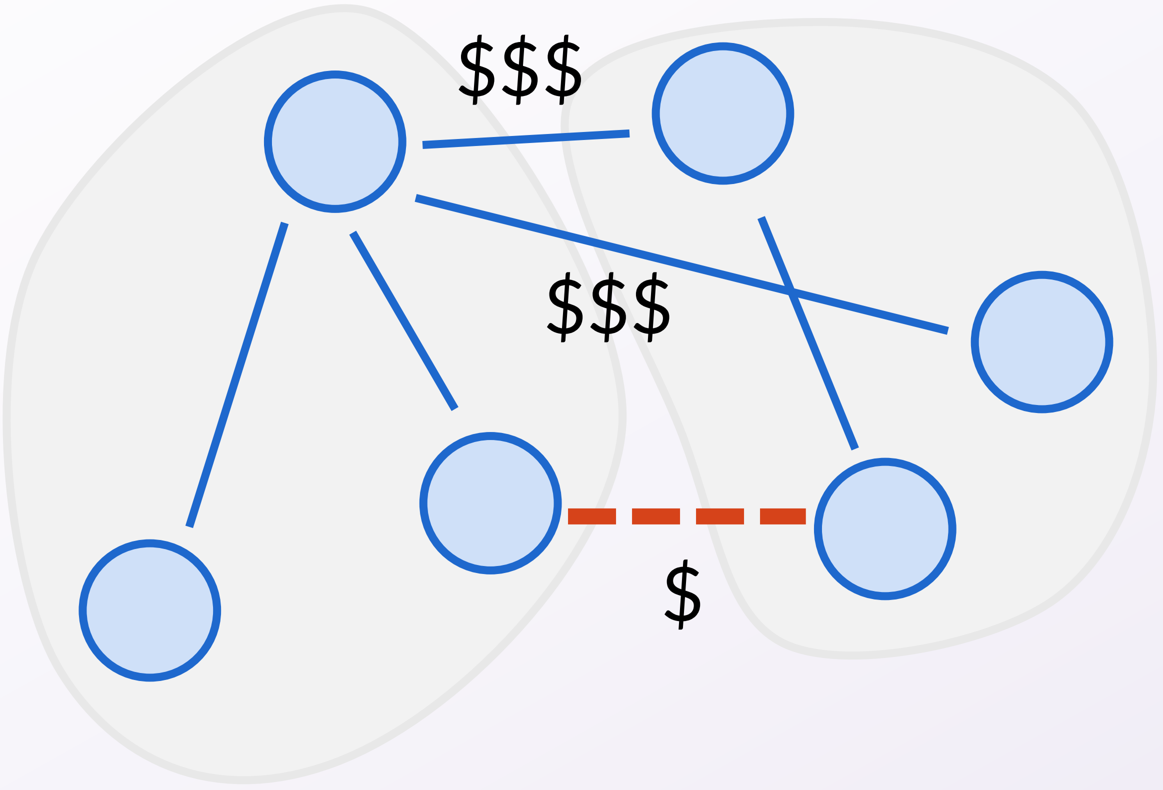


Cut property of MSTs

Theorem. If an edge is the minimum cost edge across some cut, then it must be in every MST.

Proof. By contradiction.

If an MST doesn't have the edge, we can make a smaller spanning tree by swapping it in!

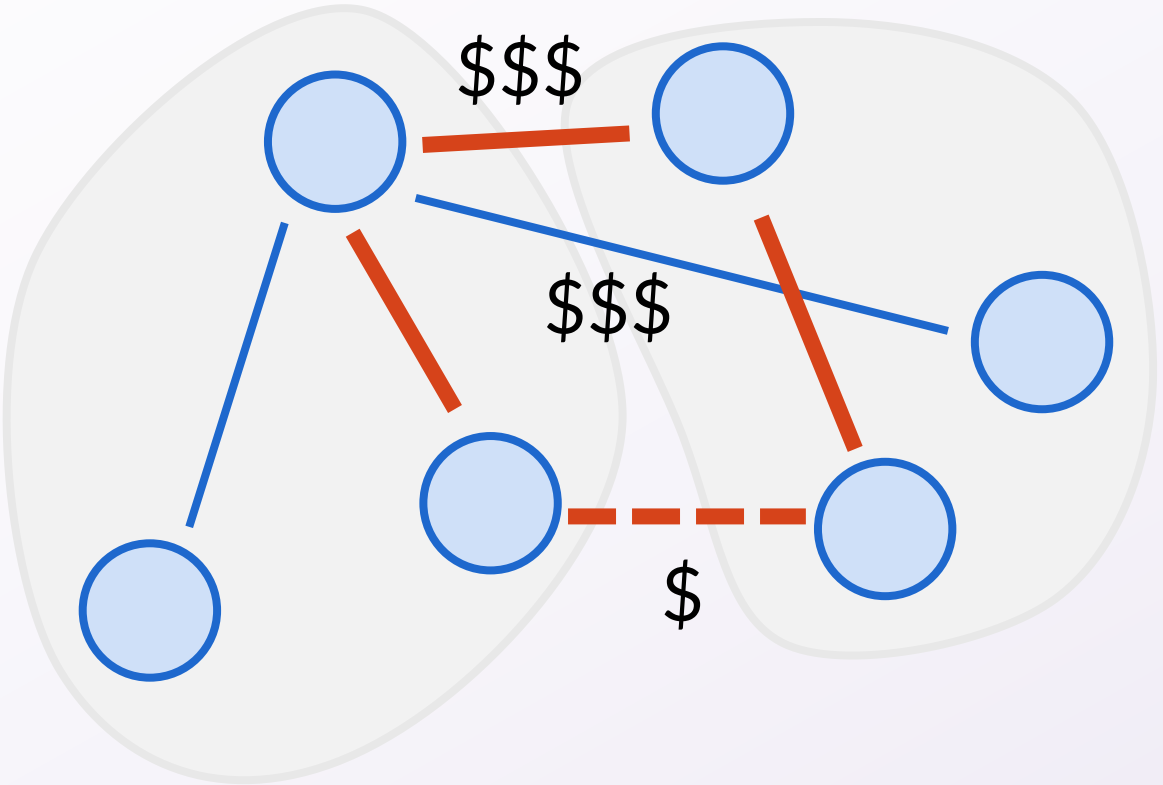


Cut property of MSTs

Theorem. If an edge is the minimum cost edge across some cut, then it must be in every MST.

Proof. By contradiction.

If an MST doesn't have the edge, we can make a smaller spanning tree by swapping it in!

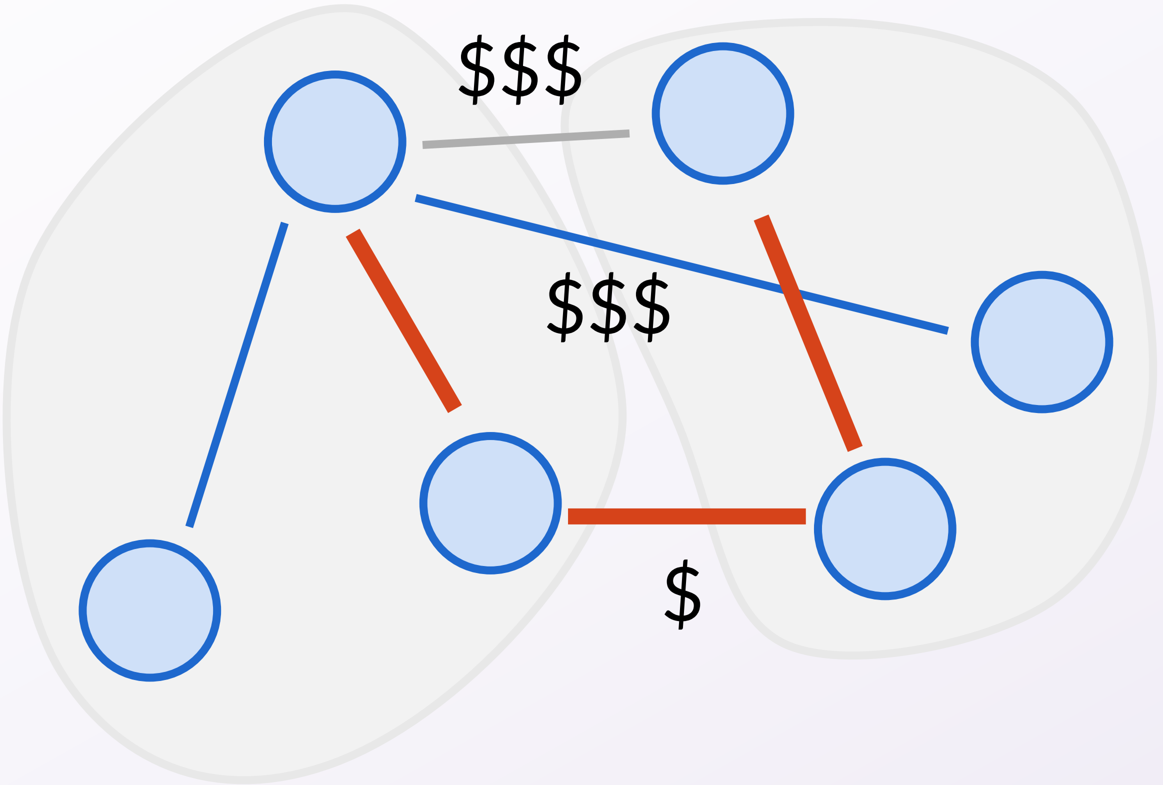


Cut property of MSTs

Theorem. If an edge is the minimum cost edge across some cut, then it must be in every MST.

Proof. By contradiction.

If an MST doesn't have the edge, we can make a smaller spanning tree by swapping it in!



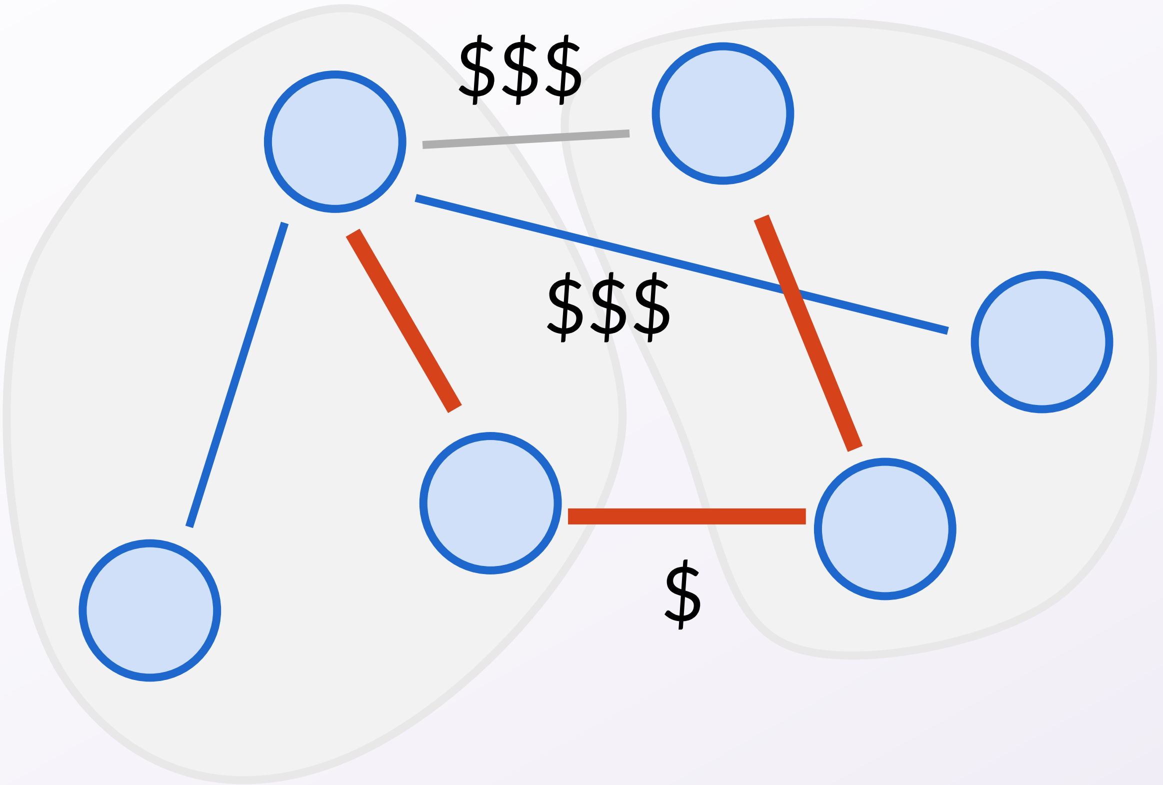
Cut property of MSTs

Proof. By contradiction.

If an MST doesn't have the edge (u, v), find the path from u to v.

The path must cross the cut, swap that edge with the cheap one.

Still a tree, because it's connected (go "long way around" with cheap edge) and still $n - 1$ edges.

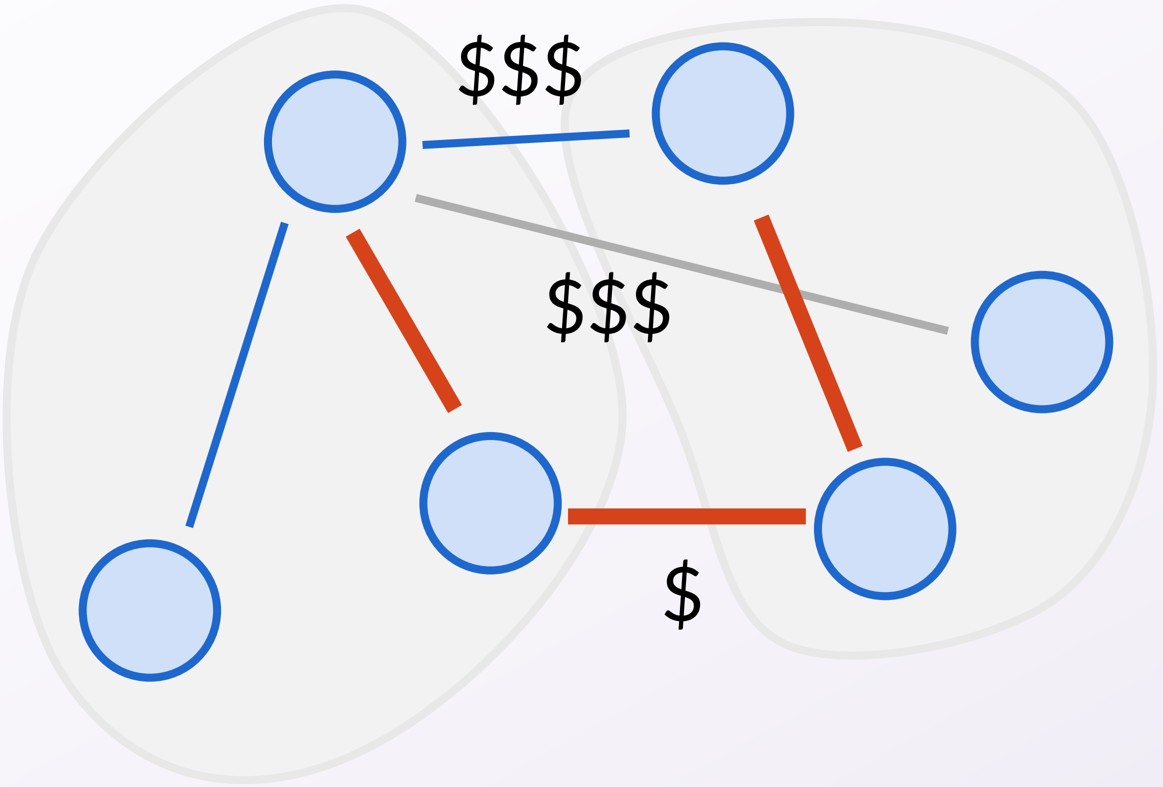


Cut property of MSTs

Warning:

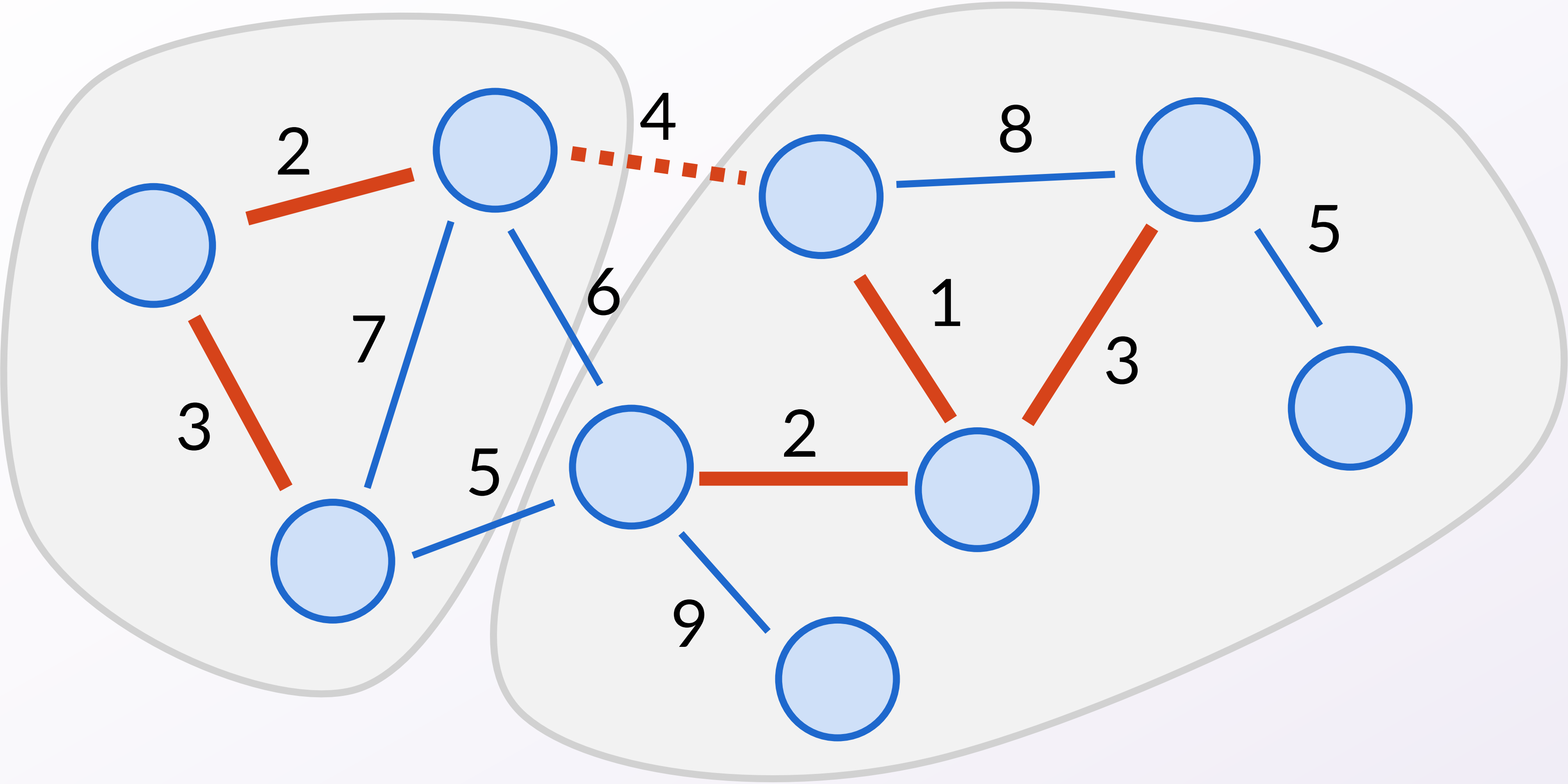
Must swap with an edge in a cycle!

Otherwise, cost will go down but you don't get a tree.



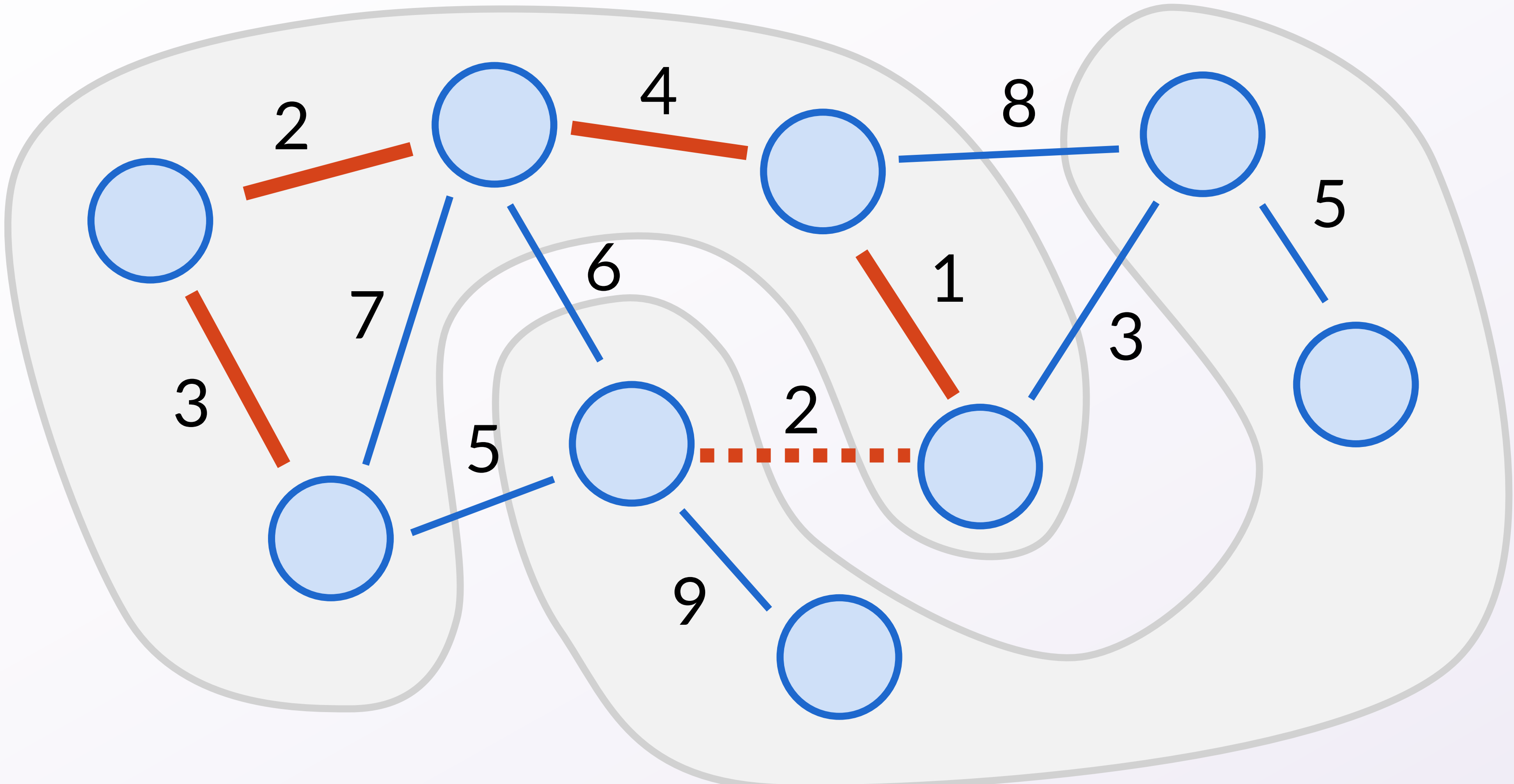
Correctness of Kruskal's algorithm

The edge we picked is the cheapest edge across any cut that puts connected components of endpoints on separate sides.



Correctness of Prim's algorithm

The edge we picked is the cheapest edge across the cut that puts our working tree all on one side.



Running time of Kruskal's algorithm

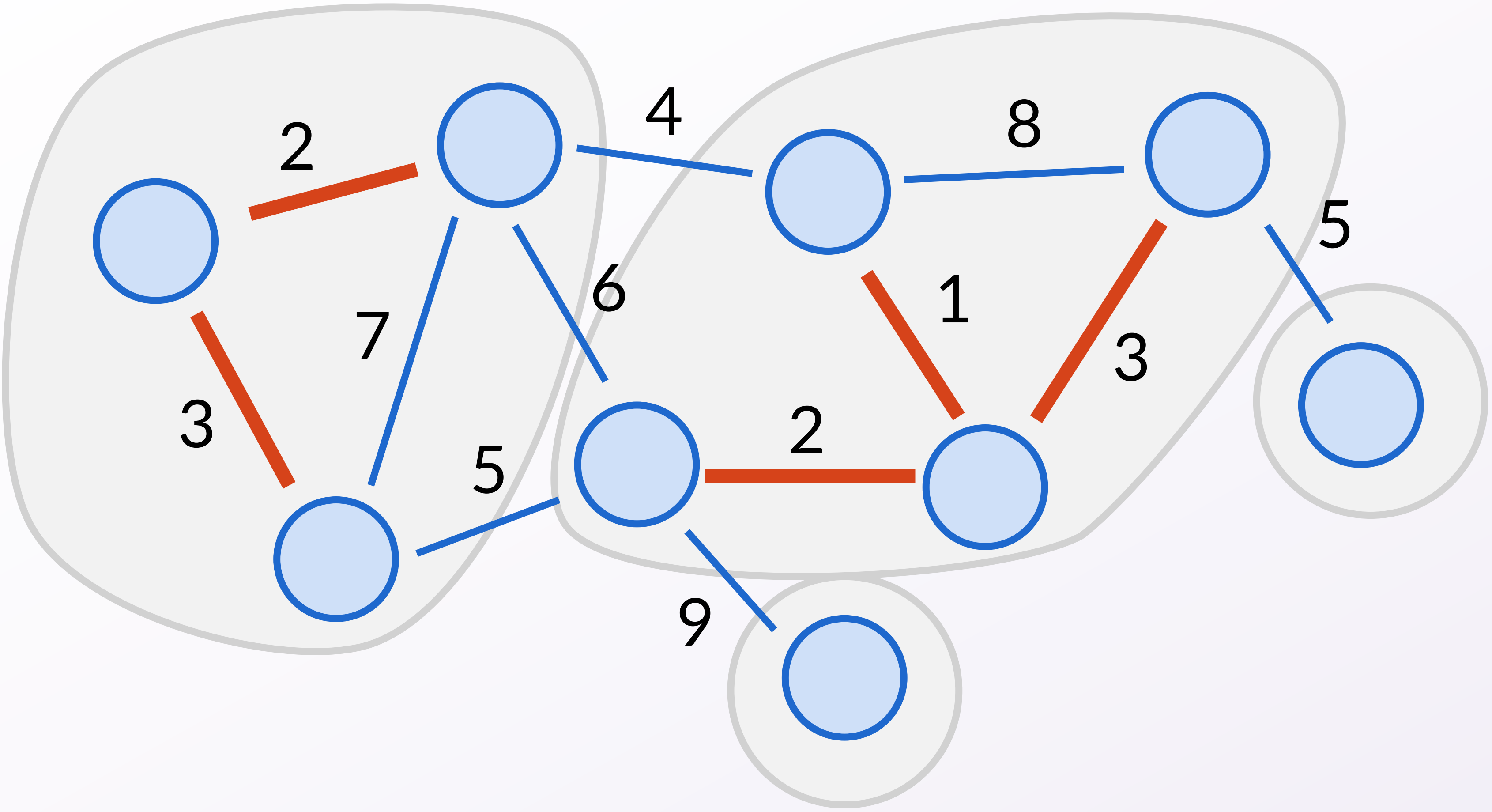
1. repeat $n - 1$ times
2. Pick the cheapest edge that doesn't create a cycle.

Sort edges beforehand:
additive $O(m \log m)$

Run DFS every iteration?
$O(n)$ per iteration

Idea: Just check that endpoints are in two different connected components!

Union-Find data structure



Union-Find data structure

Need the following operations:

- **Union** two connected components
- **Find** if two vertices belong to the same CC

Idea:

- For each vertex, store the name of its CC (e.g. the alphabetically smallest vertex in the CC)
- Also store the reverse lists (list of vertices in each CC)
- **Union**: Overwrite the names of the **smaller** CC and merge lists
- **Find**: Query if the CC names are the same

Union-Find data structure

Each individual union make take up to $O(n)$ time.

- Example: merging two CCs of size $n/2$

But: any k consecutive unions takes only $O(k \log k)$ time!

Proof.

- k consecutive unions can only affect $2k$ vertices
- Each vertex's component at least doubles in size every update
- Each vertex's component updates at most $\log 2k$ times!

Union-Find data structure

With more optimizations, k consecutive unions can take just $O(k\alpha(k)) \approx O(k)$ time! (Practically, $\alpha(k) \leq 4$ for all k.)

This is the "inverse Ackermann function".

Values of *A*(*m*, *n*)

m \ *n*	0	1	2	3	4
0	1	2	3	4	5
1	2	3	4	5	6
2	3	5	7	9	11
3	5	13	29	61	125
4	13 $= 2^{2^{2}} - 3$ $= 2 \uparrow\uparrow 3 - 3$	65533 $= 2^{2^{2^{2}}} - 3$ $= 2 \uparrow\uparrow 4 - 3$	$2^{65536} - 3$ $= 2^{2^{2^{2^{2}}}} - 3$ $= 2 \uparrow\uparrow 5 - 3$	$2^{2^{65536}} - 3$ $= 2^{2^{2^{2^{2^{2}}}}} - 3$ $= 2 \uparrow\uparrow 6 - 3$	$2^{2^{2^{65536}}} - 3$ $= 2^{2^{2^{2^{2^{2^{2}}}}}} - 3$ $= 2 \uparrow\uparrow 7 - 3$

Running time of Kruskal's algorithm

1. **repeat $n - 1$ times**
2. Pick the cheapest edge that doesn't create a cycle.

Sort edges beforehand:
additive $O(m \log m)$

~~Run DFS every iteration?~~
~~$O(n)$ per iteration~~

Use Union-Find.
$O(n \log n)$ in total!

Total: $O(m \log m) = O(m \log n)$

Running time of Prim's algorithm

1. repeat $n - 1$ times
2. Pick the cheapest edge that extends the current tree to a new vertex.

Key optimization: remember only cheapest edge for every discovered vertex

- List of possible vertices: up to $O(n)$
- Pick cheapest edge: $O(n)$ per iteration
- Traversing edges: additive $O(m)$

Total: $O(n^2)$ – better than Kruskal for graphs where $m \approx n^2$

Running time of Prim's algorithm

1. **repeat $n - 1$ times**
2. Pick the cheapest edge that extends the current tree to a new vertex.

Key optimization: remember only cheapest edge for every discovered vertex

Can also do with a **priority queue**: will discuss more Monday!

Final reminders

HW4 out, HW3 due tonight @ 11:59pm.

Practice quiz will posted tonight!

I have OH now-12:30pm:

- Meet at front of classroom, we'll walk over together
- CSE (Allen) 214 if you're coming later

Nathan has online OH 12–1pm:

- https://washington.zoom.us/my/nathanbrunelle