

CSE 332: Data Structures & Parallelism

Lecture 21: MST



Hans Easton
Summer 2022

Announcements

- Final Review Session: MOR 220 Wed 10/17 from 3:00-4:00pm
 - There might be snacks, so come through! 😎

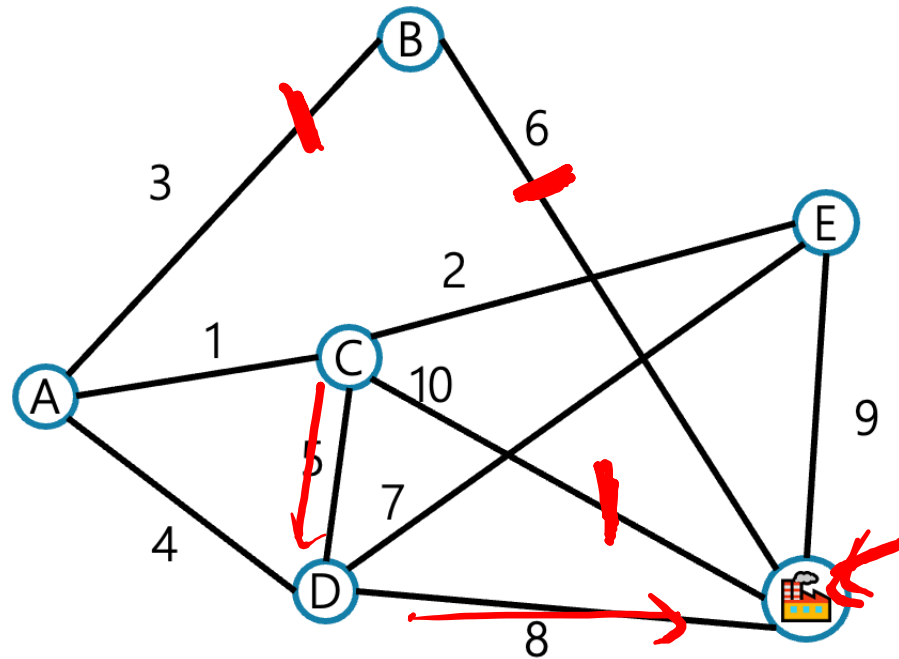
Outline for Today

- Intro to MST
- Prim's algorithm
- Kruskal's algorithm



Time travel - 1920s

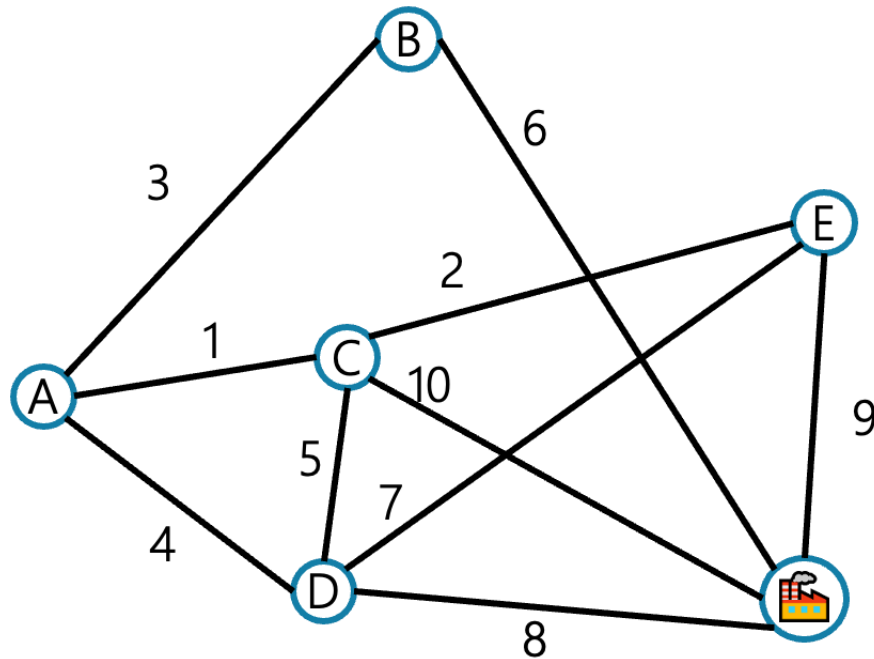
It's the 1920's. Your friend works at an electric company. They want to know where to build electrical wires to connect all cities to the powerplant.



They know how much it would cost to lay electric wires between any pair of locations, and they want the cheapest way to make sure there's electricity from the plant to every city.

Time travel - 1950s

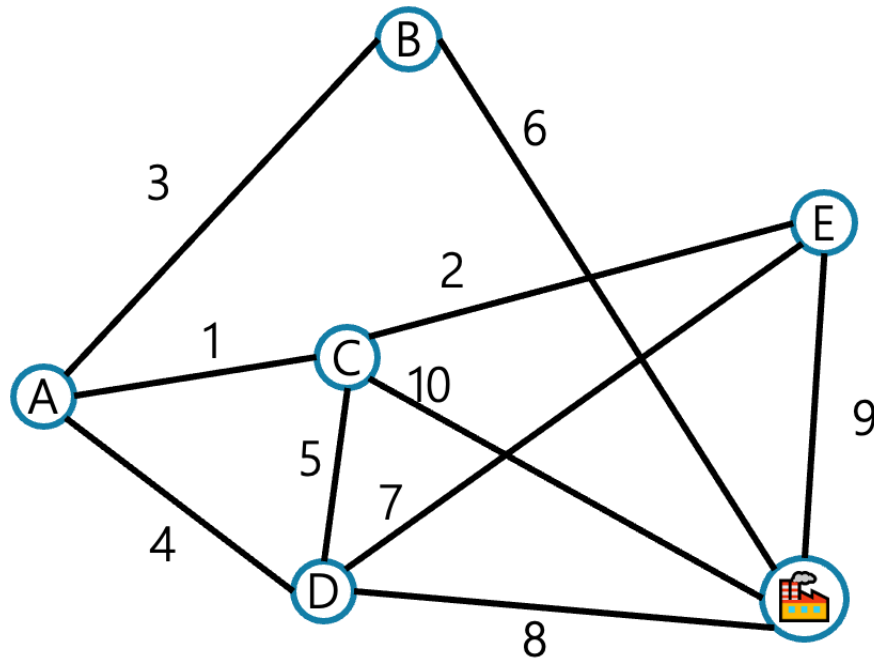
It's the 1950's. Your friend works at a phone company. They want to know where to build phone wires to connect phones to each other.



They know how much it would cost to lay phone wires between any pair of locations, and they want the cheapest way to make sure everyone can call everyone else.

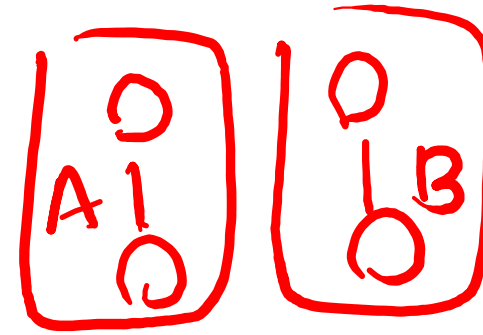
Time travel - 2020s

It's 2022! Your friend works at an internet company. They want to know where to build internet cables to connect all cities to the Internet.



They know much it would cost to lay internet cables between any pair of locations, and they want the cheapest way to make sure everyone can reach the server.

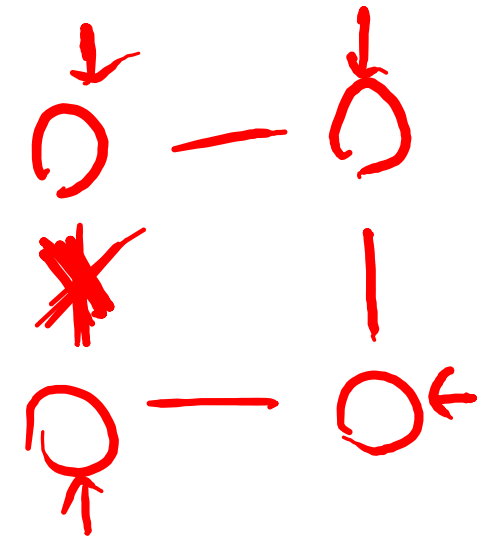
What are we looking for?



A set of edges such that...

- ⇒ 1. Every vertex touches **at least one edge** *edges span the graph*
- ⇒ 2. Graph on these edges is **connected**
- 3. Sum of edge weights is **minimized**

Claim: The set of edges we pick **never has a cycle**



Aside: Tree!

1. Don't need a root ✓
2. Varying numbers of ~~children~~ neighbors
3. Connected and no cycles



Tree (when talking about undirected graphs)

An undirected, connected acyclic graph.

MST Problem

A set of edges such that...

1. Edges **span** the graph ✓
2. Graph on these edges is **connected** ✓
3. Sum of edge weights is **minimized**
4. Contains **no cycle** ✓



We need a
minimum
spanning
tree

Minimum Spanning Tree Problem

Given: an undirected, weighted graph G

Find: A minimum-weight set of edges such that you can get from any vertex of G to any other on only those edges.

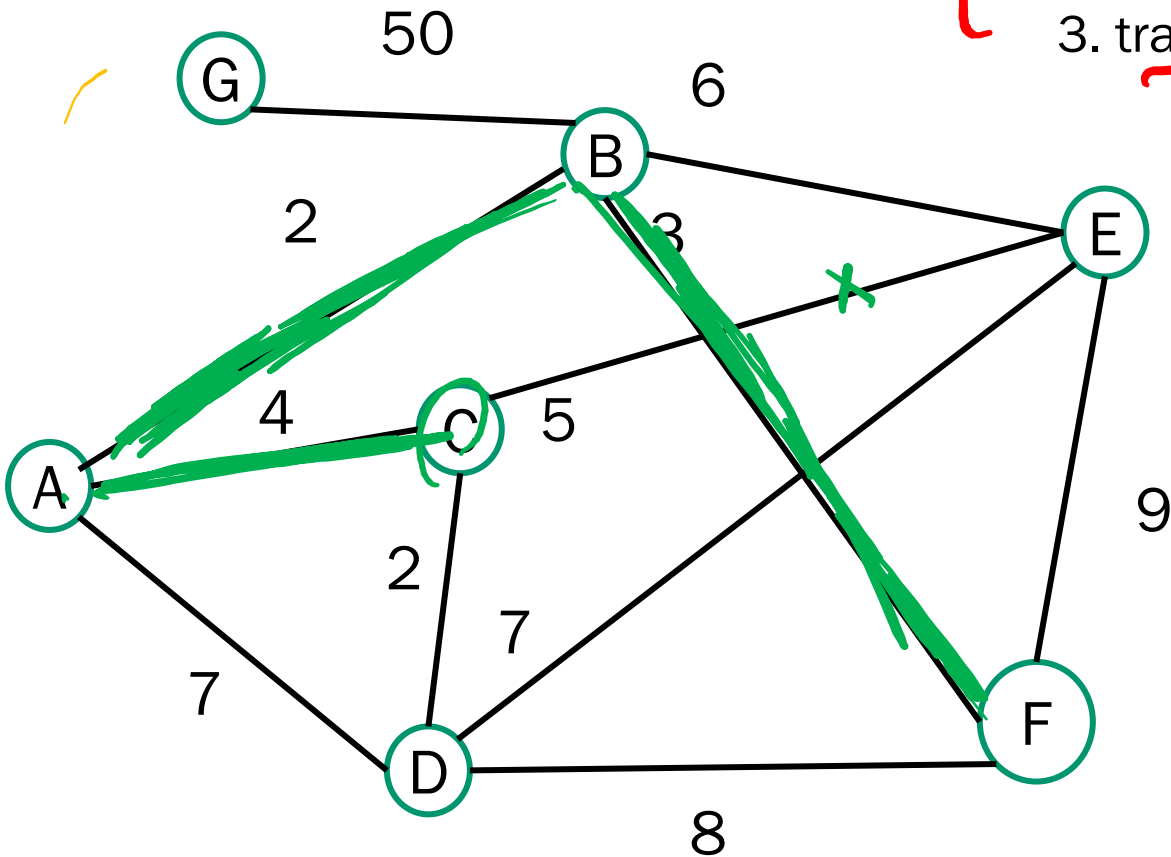
Prim's Algorithm

- Choose arbitrary starting point
- Add a new edge to the result each step
- How to choose the new edge?
 - Will let you reach more vertices
 - Is as light as possible

Try it out

Prim's travelling strategy as a broke college student!

- 1. research neighboring cities (update table) ✓
2. decide on the cheapest city to travel to (←)
3. travel to that city (mark graph)! ←

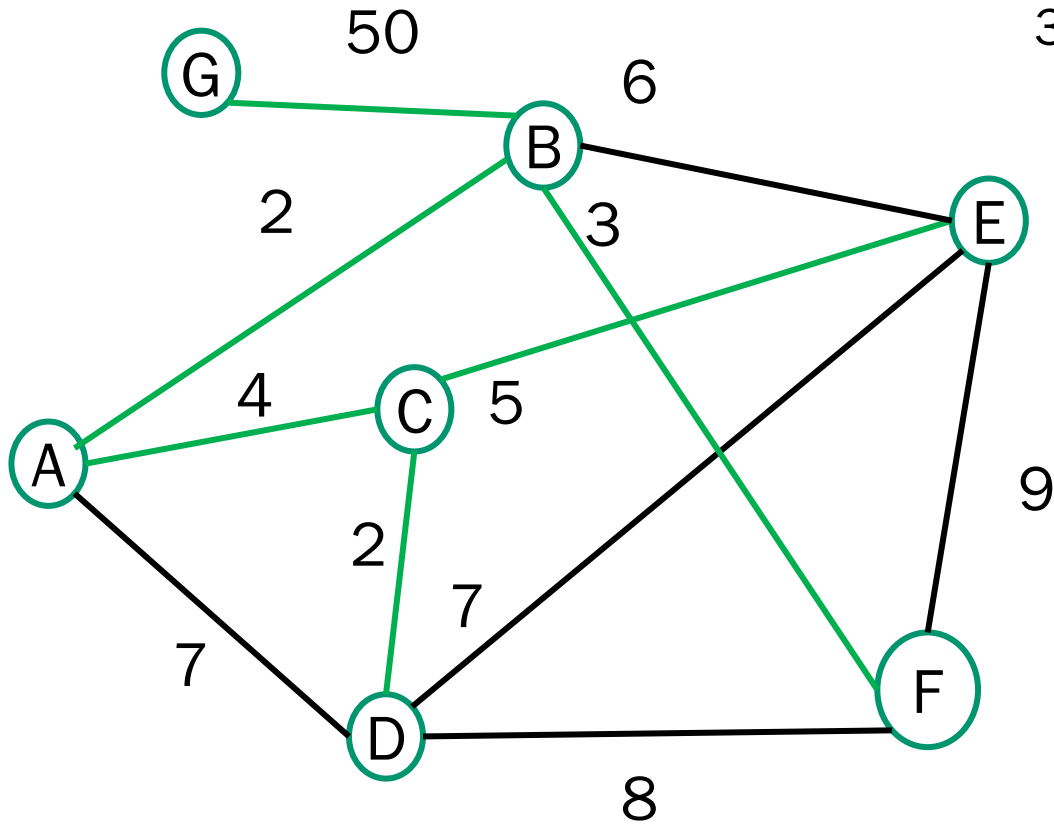


Vertex	Dist.	Best Edge	Processed
A	0	—	✓
B	2	(A, B)	✓
C	4	(A, C)	✓
D	2	(A, D) (C, D)	
E	3	(B, E) (C, E)	
F	3	(B, F)	✓
G	5	(B, G)	

Try it out

Prim's travelling strategy as a broke college student!

1. research neighboring cities (update table)
2. decide on the cheapest city to travel to
3. travel to that city (mark graph)!



Vertex	Dist.	Best Edge	Processed
A	--	--	Yes
B	2	(A,B)	Yes
C	4	(A,C)	Yes
D	7 -2	(A,D) (C,D)	Yes
E	6 -5	(B,E) (C,E)	Yes
F	3	(B,F)	Yes
G	50	(B,G)	Yes

Pseudocode

```
initialize distances to  $\infty$ 
mark source as distance 0
mark all vertices unprocessed
foreach(edge (source, v) )
    v.dist = w(source, v)
while(there are unprocessed vertices){
    let u be the closest unprocessed vertex
    add u.bestEdge to spanning tree
    foreach(edge (u, v) leaving u){
         $\Rightarrow$  if (w(u, v) < v.dist) {
            v.dist = w(u, v)
            v.bestEdge = (u, v)
        }
    }
    mark u as processed
}
```

$$\underline{\underline{E \log V + V \log V}}$$

Dijkstra's

$$\underline{u.dist + w(u, v)} < v.dist$$

Does Prim's Algorithm Always Work? *Check ya!*

- Prim's Algorithm is a greedy algorithm

- Always select the best option available at the moment
local optimum → global optimum

- Once it decides to include an edge in the MST, it never reconsiders its decision

- Life lesson? Be greedy!

global optimum

A different Approach

- Prim's algorithm - vertex by vertex

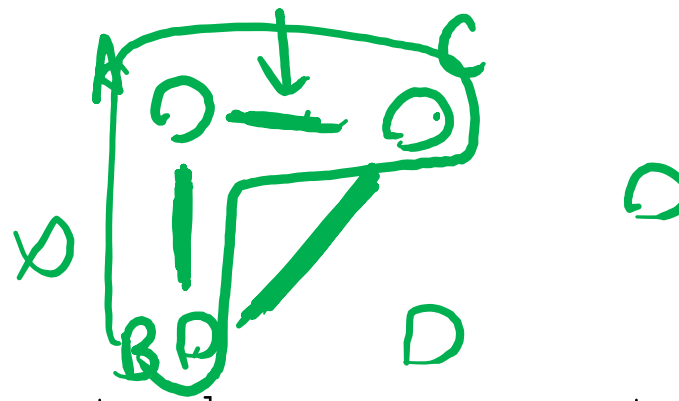
- What if you think edge by edge instead?

→ sort edges ↑

- Start from the lightest edge
 - add it if it connects new things to each other
 - don't add it if it would create a cycle

- This is Kruskal's Algorithm

Kruskal's Algorithm



KruskalMST(Graph G)

⇒ initialize each vertex to be a connected component

⇒ sort the edges by weight

⇒ foreach (edge (u, v) in sorted order) {
 if (u and v are in different components) {

 add (u, v) to the MST

 Update u and v to be in the same

component

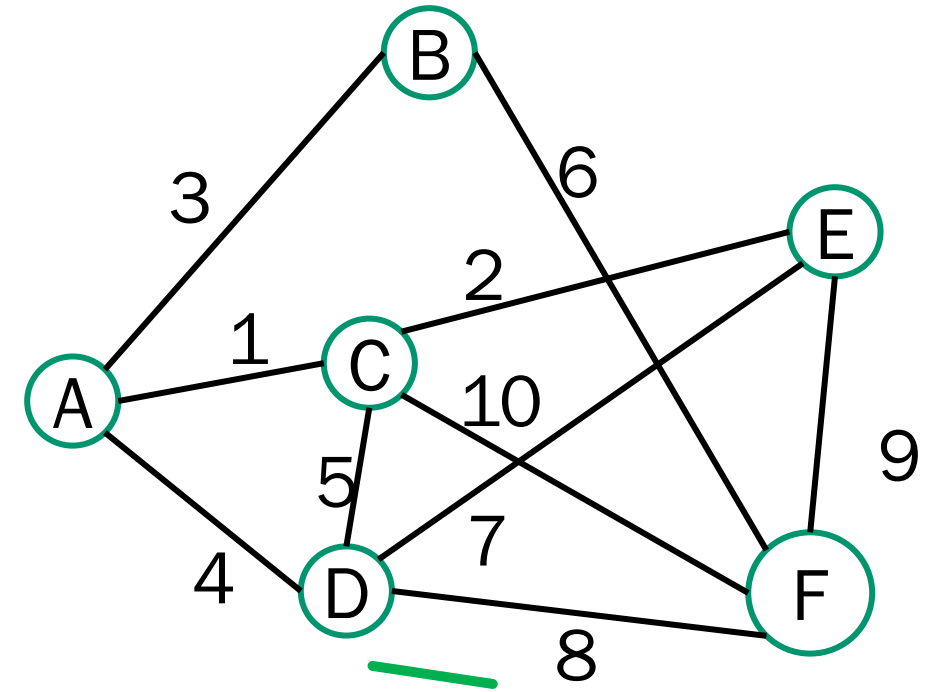
}

}

Try It Out

Two simple steps:

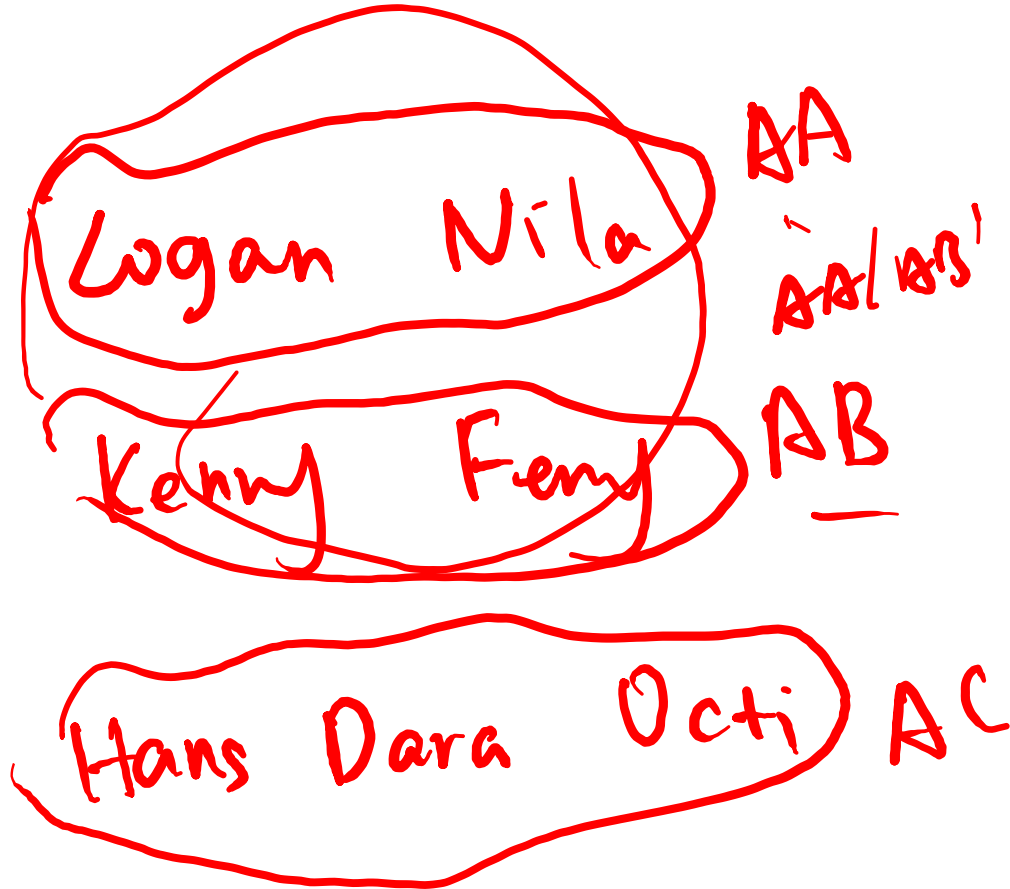
1. If edge connects different clouds then add edge, combine clouds
2. Otherwise, ignore



Edge	Include?	Reason
(A,C)	Y	
(C,E)	Y	
(A,B)	Y	
(A,D)	Y	
(C,D)	N	

Edge	Include?	Reason
(B,F)	Y	
(D,E)		
(D,F)		
(E,F)		
(C,F)		

Union-Find (Disjoint sets)



Union-Find ADT

state

Set of Sets

- Disjoint: No element appears in multiple sets
- No required order
- Each set has representative (*name*)

behavior

makeSet(x) – creates a new set where the only member (and the representative) is x.

findSet(x) – looks up the set containing element x, returns name of that set

union(x, y) – combines sets named x and y. Picks new name for combined set.

Union-Find Running Time

What's important for us?

Amortized running times! We care about the total time across the entire set of unions and finds, not the running time of just one.

Operation	Amortized	Non-amortized
MakeSet()	$\Theta(1)$	$\Theta(1)$
Union()	$O(\log^* n)$	$O(\log n)$
Find()	$O(\log^* n)$	$O(\log n)$

$\log^* n$

≤ 1

$\log^* n$: the number of times you need to apply $\log()$ to get a number at most 1.

E.g. $\log^*(16) = 3$

$$\log(16) = 4 \quad \log(4) = 2 \quad \log(2) = 1.$$

$\log^* n$ grows ridiculously slowly.

\uparrow

$$\log^*(10^{80}) = 5.$$

For all practical purposes, $\log^* n$ operations are constant time

Using Union-Find

Have each disjoint set represent a connected component

When you add an edge, you union those connected components.

How do you know if two vertices belong to the same component?

$$\underline{\text{find}(u)} == \text{find}(\underline{v})$$

Kruskal's with Union Find

$$\cancel{E} + \cancel{E} \log E + \cancel{E} \log V$$

$$E \geq V$$

KruskalMST (Graph G)

$$V \cdot 1 = V$$

$$\log E \geq \log V$$
$$\geq \log^2 V$$

initialize each vertex to be a connected component

sort the edges by weight

$$E \log E$$
$$E \log E$$

```
E ← foreach (edge (u, v) in sorted order) {  
    if (find(u) != find(v)) {  
        add (u, v) to the MST  
        union (find(u), find(v))  
    }  
}
```

$$\log^2(V) \cdot \cancel{E}$$

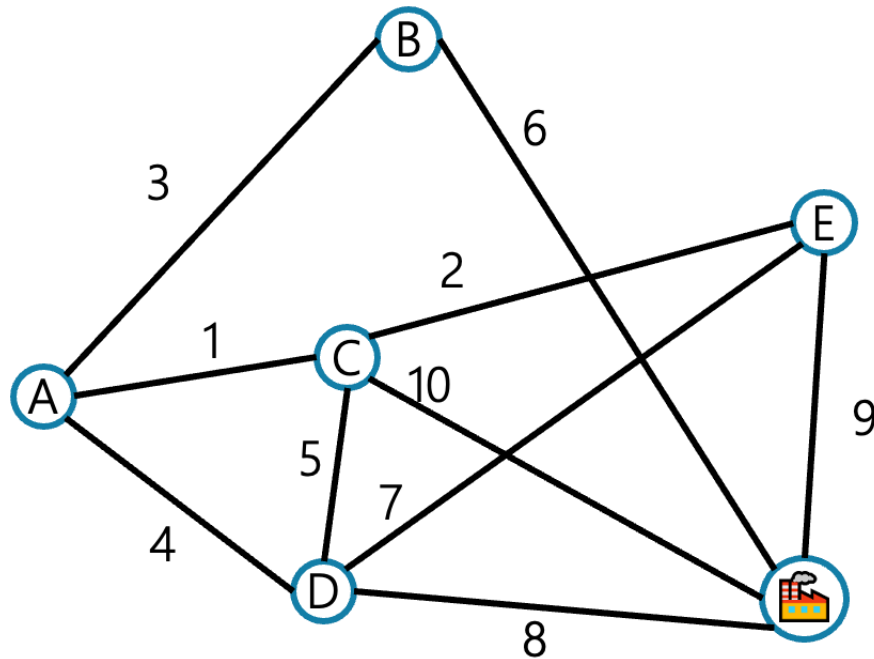
Full circle

Let's time travel again!

Time travel (again) - 1920s

Boruvka's algorithm

It's the 1920's. Your friend works at an electric company. They want to know where to build electrical wires to connect all cities to the powerplant.



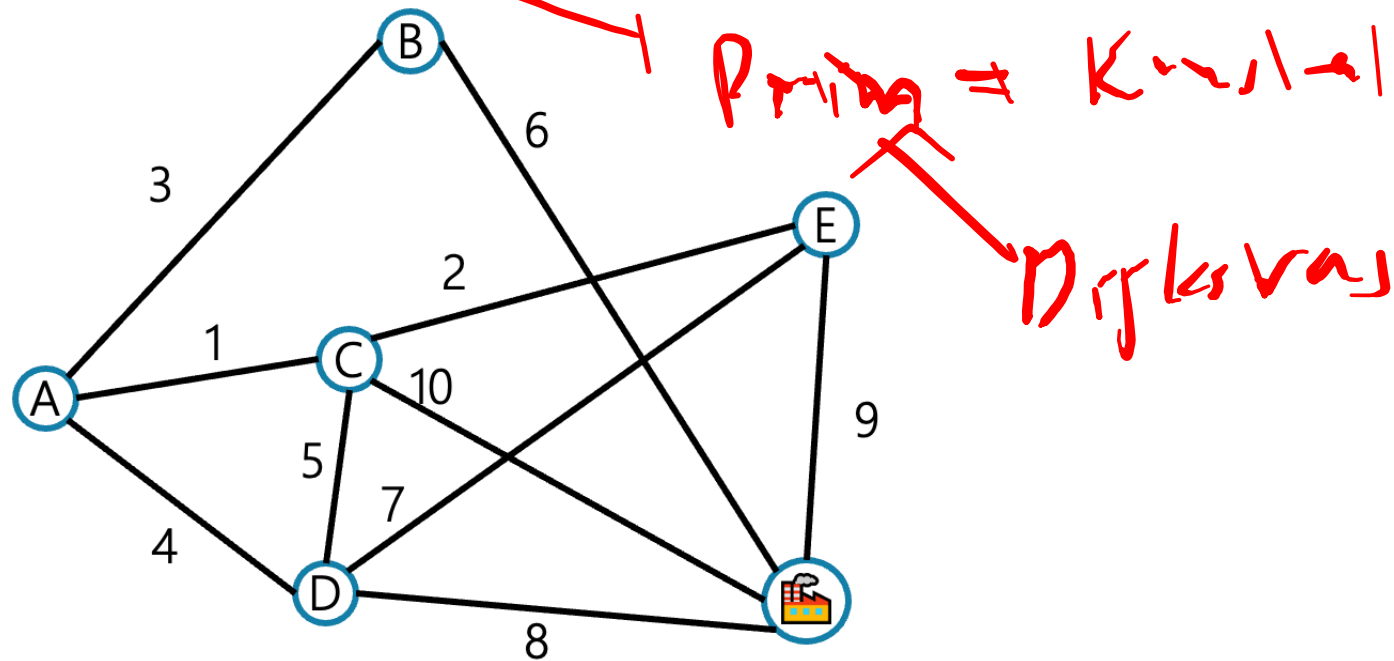
primed MST

They know how much it would cost to lay electric wires between any pair of locations, and they want the cheapest way to make sure there's electricity from the plant to every city.

Time travel (again) - 1950s

Bell Labs (M-100)

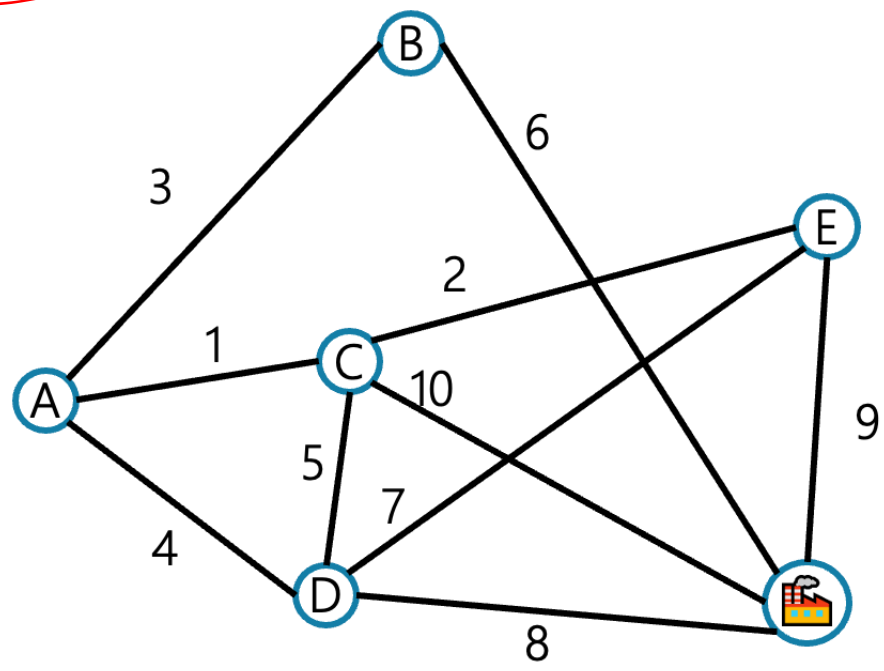
It's the 1950's. Your friend works at a ~~phone company~~. They want to know where to build phone wires to connect phones to each other.



They know how much it would cost to lay phone wires between any pair of locations, and they want **the cheapest way** to make sure **everyone can call everyone else**.

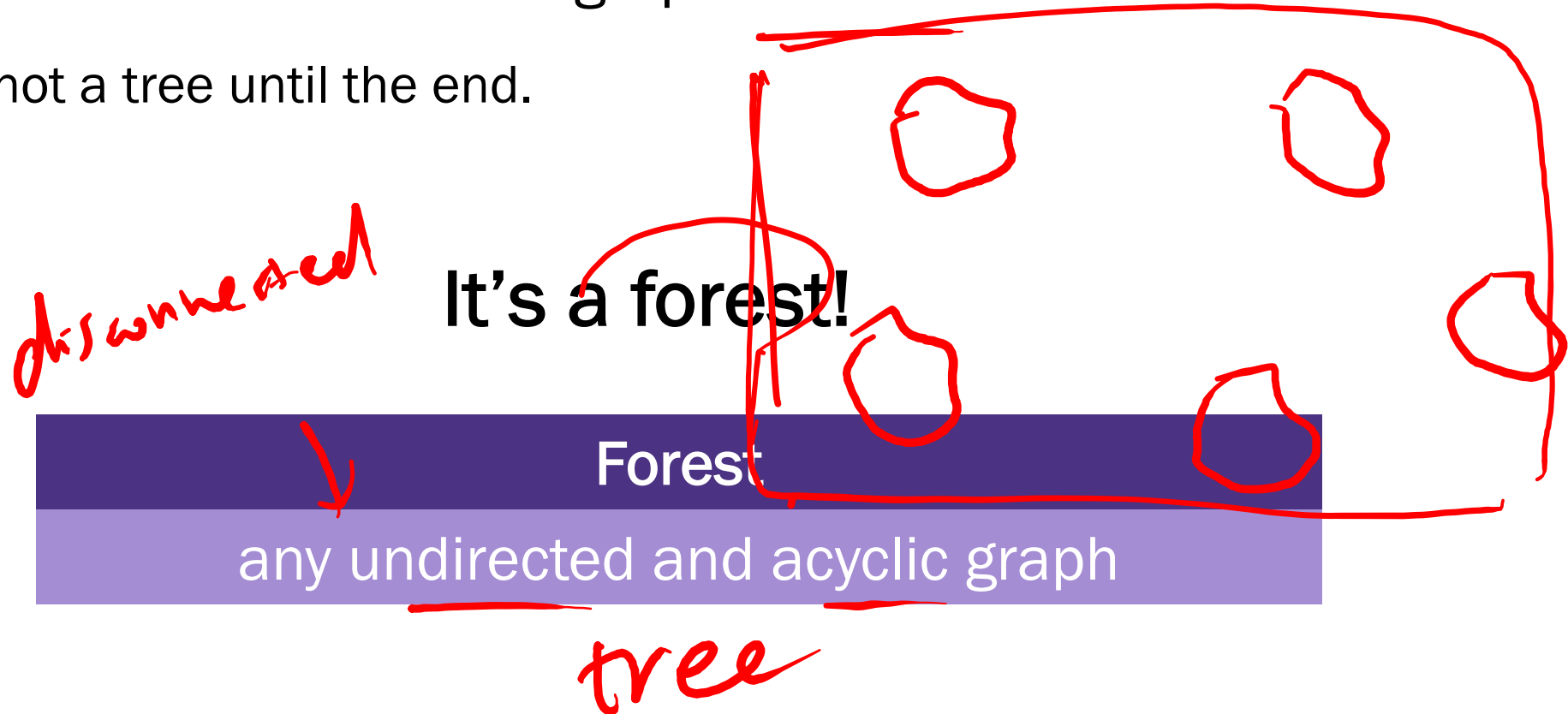
Time travel (again) - 2020s

It's 2022! You work at an internet company. They want to know where to build internet cables to connect all cities to the Internet. **Maybe you will come up with the next MST algorithm!**



A Graph of Trees

- Recall a tree is an undirected, connected, and acyclic graph.
- How would we describe the graph Kruskal's builds?
 - It's not a tree until the end.





EVERY TREE IS A FOREST.