

# Lecture 18: Minimum Spanning Trees

Michael Dinitz

October 30, 2025

601.433/633 Introduction to Algorithms

# Introduction

## Definition

A **spanning tree** of an undirected graph  $G = (V, E)$  is a set of edges  $T \subseteq E$  such that  $(V, T)$  is connected and acyclic.

## Definition

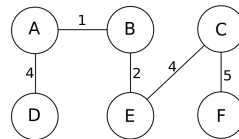
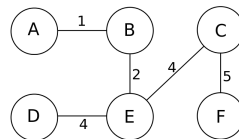
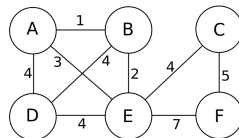
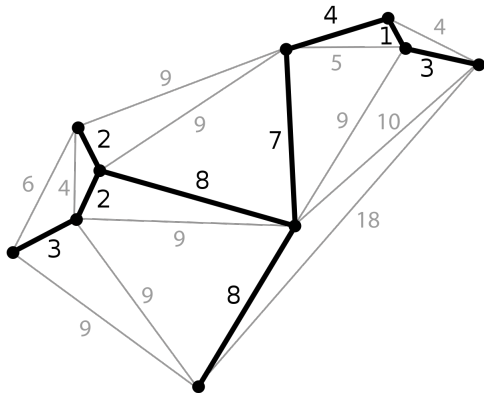
Minimum Spanning Tree problem (MST):

- ▶ Input:
  - ▶ Undirected graph  $G = (V, E)$
  - ▶ Edge weights  $w : E \rightarrow \mathbb{R}_{\geq 0}$
- ▶ Output: Spanning tree minimizing  $w(T) = \sum_{e \in T} w(e)$ .

Foundational problem in *network design*. Tons of applications.

Today: one “recipe”, two different algorithms from recipe. Main idea: greedy.

# Examples



# Generic Algorithm

# Generic Greedy

## Definition

Suppose that  $\mathbf{A}$  is subset of *some* MST. If  $\mathbf{A} \cup \{\mathbf{e}\}$  is also a subset of some MST, then  $\mathbf{e}$  is *safe* for  $\mathbf{A}$ .

# Generic Greedy

## Definition

Suppose that  $\mathbf{A}$  is subset of *some* MST. If  $\mathbf{A} \cup \{\mathbf{e}\}$  is also a subset of some MST, then  $\mathbf{e}$  is *safe* for  $\mathbf{A}$ .

```
Generic-MST {  
     $\mathbf{A} = \emptyset$   
    while( $\mathbf{A}$  not a spanning tree) {  
        find an edge  $\mathbf{e}$  safe for  $\mathbf{A}$   
         $\mathbf{A} = \mathbf{A} \cup \{\mathbf{e}\}$   
    }  
    return  $\mathbf{A}$   
}
```

# Generic Greedy

## Definition

Suppose that  $A$  is subset of some MST. If  $A \cup \{e\}$  is also a subset of some MST, then  $e$  is *safe* for  $A$ .

## Theorem

*Generic-MST is correct: it always returns an MST.*

```
Generic-MST {  
     $A = \emptyset$   
    while( $A$  not a spanning tree) {  
        find an edge  $e$  safe for  $A$   
         $A = A \cup \{e\}$   
    }  
    return  $A$   
}
```

# Generic Greedy

## Definition

Suppose that  $A$  is subset of some MST. If  $A \cup \{e\}$  is also a subset of some MST, then  $e$  is *safe* for  $A$ .

```
Generic-MST {  
     $A = \emptyset$   
    while( $A$  not a spanning tree) {  
        find an edge  $e$  safe for  $A$   
         $A = A \cup \{e\}$   
    }  
    return  $A$   
}
```

## Theorem

*Generic-MST is correct: it always returns an MST.*

## Proof.

Induction.

Claim:  $A$  always a subset of some MST.

Base case: ✓

Inductive step: ✓





# Generic Greedy

## Definition

Suppose that  $A$  is subset of some MST. If  $A \cup \{e\}$  is also a subset of some MST, then  $e$  is *safe* for  $A$ .

```
Generic-MST {  
     $A = \emptyset$   
    while( $A$  not a spanning tree) {  
        find an edge  $e$  safe for  $A$   
         $A = A \cup \{e\}$   
    }  
    return  $A$   
}
```

## Theorem

*Generic-MST is correct: it always returns an MST.*

## Proof.

Induction.

Claim:  $A$  always a subset of some MST.

Base case: ✓

Inductive step: ✓

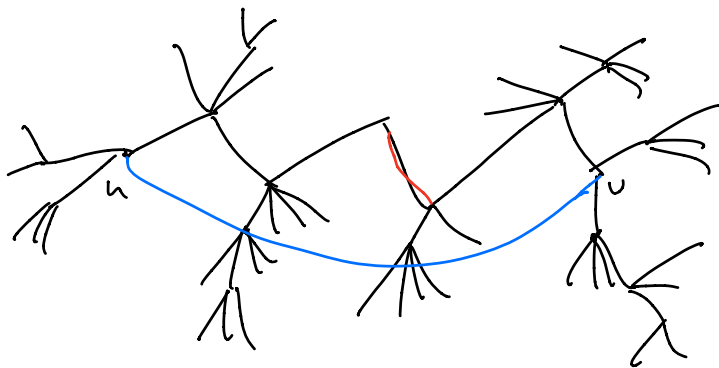


But how to find a safe edge? And which one to add?

# Structural Properties

## Lemma

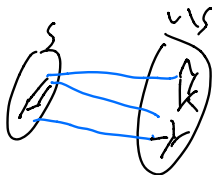
Let  $T$  be a spanning tree, let  $u, v \in V$ , and let  $P$  be the  $u - v$  path in  $T$ . If  $\{u, v\} \notin T$ , then  $T' = (T \cup \{\{u, v\}\}) \setminus \{e\}$  is a spanning tree for all  $e \in P$ .



# Structural Properties

## Definition

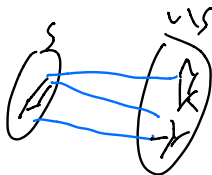
A **cut**  $(S, V \setminus S)$  (or  $(S, \bar{S})$  or just  $S$ ) is a partition of  $V$  into two parts. Edge  $e$  **crosses** cut  $(S, \bar{S})$  if  $e$  has one endpoint in  $S$  and one endpoint in  $\bar{S}$ .



# Structural Properties

## Definition

A **cut**  $(S, V \setminus S)$  (or  $(S, \bar{S})$  or just  $S$ ) is a partition of  $V$  into two parts. Edge  $e$  **crosses** cut  $(S, \bar{S})$  if  $e$  has one endpoint in  $S$  and one endpoint in  $\bar{S}$ .



## Definition

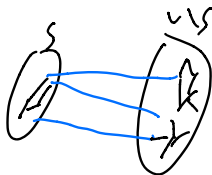
Cut  $(S, \bar{S})$  **respects**  $A \subseteq E$  if no edge in  $A$  crosses  $(S, \bar{S})$



# Structural Properties

## Definition

A **cut**  $(S, V \setminus S)$  (or  $(S, \bar{S})$  or just  $S$ ) is a partition of  $V$  into two parts. Edge  $e$  **crosses** cut  $(S, \bar{S})$  if  $e$  has one endpoint in  $S$  and one endpoint in  $\bar{S}$ .



## Definition

Cut  $(S, \bar{S})$  **respects**  $A \subseteq E$  if no edge in  $A$  crosses  $(S, \bar{S})$



## Definition

$e$  is a **light edge** for  $(S, \bar{S})$  if  $e$  crosses  $(S, \bar{S})$  and  $w(e) = \min_{e' \text{ crossing } (S, \bar{S})} w(e')$

# Main Structural Theorem

## Theorem

*Let  $\mathbf{A} \subseteq \mathbf{E}$  be a subset of some MST  $\mathbf{T}$ , let  $(\mathbf{S}, \bar{\mathbf{S}})$  be a cut respecting  $\mathbf{A}$ , and let  $\mathbf{e} = \{\mathbf{u}, \mathbf{v}\}$  be a light edge for  $(\mathbf{S}, \bar{\mathbf{S}})$ . Then  $\mathbf{e}$  is safe for  $\mathbf{A}$ .*

# Main Structural Theorem

## Theorem

*Let  $\mathbf{A} \subseteq \mathbf{E}$  be a subset of some MST  $\mathbf{T}$ , let  $(\mathbf{S}, \bar{\mathbf{S}})$  be a cut respecting  $\mathbf{A}$ , and let  $\mathbf{e} = \{\mathbf{u}, \mathbf{v}\}$  be a light edge for  $(\mathbf{S}, \bar{\mathbf{S}})$ . Then  $\mathbf{e}$  is safe for  $\mathbf{A}$ .*

Need to show there is an MST containing  $\mathbf{A} \cup \{\mathbf{e}\}$ .

# Main Structural Theorem

## Theorem

*Let  $\mathbf{A} \subseteq \mathbf{E}$  be a subset of some MST  $\mathbf{T}$ , let  $(\mathbf{S}, \bar{\mathbf{S}})$  be a cut respecting  $\mathbf{A}$ , and let  $\mathbf{e} = \{\mathbf{u}, \mathbf{v}\}$  be a light edge for  $(\mathbf{S}, \bar{\mathbf{S}})$ . Then  $\mathbf{e}$  is safe for  $\mathbf{A}$ .*

Need to show there is an MST containing  $\mathbf{A} \cup \{\mathbf{e}\}$ .

If  $\mathbf{e} \in \mathbf{T}$ : ✓



# Main Structural Theorem

## Theorem

*Let  $\mathbf{A} \subseteq \mathbf{E}$  be a subset of some MST  $\mathbf{T}$ , let  $(\mathbf{S}, \bar{\mathbf{S}})$  be a cut respecting  $\mathbf{A}$ , and let  $\mathbf{e} = \{\mathbf{u}, \mathbf{v}\}$  be a light edge for  $(\mathbf{S}, \bar{\mathbf{S}})$ . Then  $\mathbf{e}$  is safe for  $\mathbf{A}$ .*

Need to show there is an MST containing  $\mathbf{A} \cup \{\mathbf{e}\}$ .

If  $\mathbf{e} \in \mathbf{T}$ : ✓ Otherwise:

# Main Structural Theorem

## Theorem

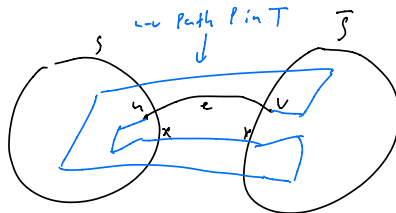
Let  $A \subseteq E$  be a subset of some MST  $T$ , let  $(S, \bar{S})$  be a cut respecting  $A$ , and let  $e = \{u, v\}$  be a light edge for  $(S, \bar{S})$ . Then  $e$  is safe for  $A$ .

Need to show there is an MST containing  $A \cup \{e\}$ .

If  $e \in T$ : ✓ Otherwise:

Let  $T' = (T \cup \{e\}) \setminus \{\{x, y\}\}$

$\Rightarrow T'$  a spanning tree by first lemma



# Main Structural Theorem

## Theorem

Let  $A \subseteq E$  be a subset of some MST  $T$ , let  $(S, \bar{S})$  be a cut respecting  $A$ , and let  $e = \{u, v\}$  be a light edge for  $(S, \bar{S})$ . Then  $e$  is safe for  $A$ .

Need to show there is an MST containing  $A \cup \{e\}$ .

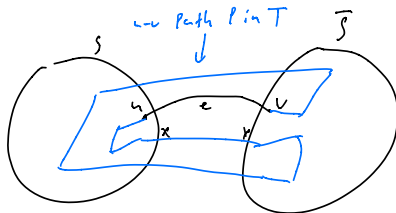
If  $e \in T$ : ✓ Otherwise:

Let  $T' = (T \cup \{e\}) \setminus \{\{x, y\}\}$

⇒  $T'$  a spanning tree by first lemma

$\{x, y\} \notin A$ , since  $(S, \bar{S})$  respects  $A$

⇒  $A \cup \{e\} \subseteq T'$



# Main Structural Theorem

## Theorem

Let  $A \subseteq E$  be a subset of some MST  $T$ , let  $(S, \bar{S})$  be a cut respecting  $A$ , and let  $e = \{u, v\}$  be a light edge for  $(S, \bar{S})$ . Then  $e$  is safe for  $A$ .

Need to show there is an MST containing  $A \cup \{e\}$ .

If  $e \in T$ :  $\checkmark$  Otherwise:

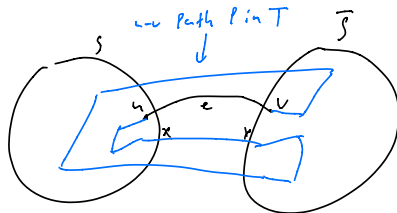
Let  $T' = (T \cup \{e\}) \setminus \{x, y\}$

$\Rightarrow T'$  a spanning tree by first lemma

$\{x, y\} \notin A$ , since  $(S, \bar{S})$  respects  $A$

$\Rightarrow A \cup \{e\} \subseteq T'$

$$w(T') = w(T) + w(e) - w(x, y) \leq w(T)$$



# Main Structural Theorem

## Theorem

Let  $A \subseteq E$  be a subset of some MST  $T$ , let  $(S, \bar{S})$  be a cut respecting  $A$ , and let  $e = \{u, v\}$  be a light edge for  $(S, \bar{S})$ . Then  $e$  is safe for  $A$ .

Need to show there is an MST containing  $A \cup \{e\}$ .

If  $e \in T$ : ✓ Otherwise:

Let  $T' = (T \cup \{e\}) \setminus \{x, y\}$

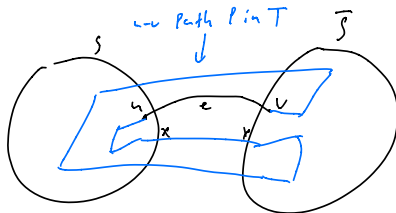
⇒  $T'$  a spanning tree by first lemma

$\{x, y\} \notin A$ , since  $(S, \bar{S})$  respects  $A$

⇒  $A \cup \{e\} \subseteq T'$

$w(T') = w(T) + w(e) - w(x, y) \leq w(T)$

⇒  $T'$  an MST containing  $A \cup \{e\}$



# Prim's Algorithm

# Prim's Algorithm

Idea: start at arbitrary node  $u$ . Greedily grow MST out of  $u$ .

$A = \emptyset$

Let  $u$  be an arbitrary node, and let  $S = \{u\}$

while( $A$  is not a spanning tree) {

    Find an edge  $\{x, y\}$  with  $x \in S$  and  $y \notin S$  that is light for  $(S, \bar{S})$

$A \leftarrow A \cup \{\{x, y\}\}$

$S \leftarrow S \cup \{y\}$

}

return  $A$

# Prim's Algorithm

Idea: start at arbitrary node  $u$ . Greedily grow MST out of  $u$ .

$A = \emptyset$

Let  $u$  be an arbitrary node, and let  $S = \{u\}$

while( $A$  is not a spanning tree) {

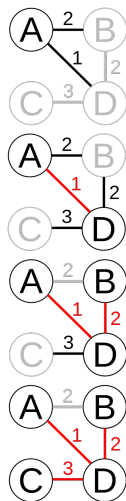
Find an edge  $\{x, y\}$  with  $x \in S$  and  $y \notin S$  that is light for  $(S, \bar{S})$

$A \leftarrow A \cup \{\{x, y\}\}$

$S \leftarrow S \cup \{y\}$

}

return  $A$





# Correctness

## Theorem

*Prim's algorithm returns an MST.*

# Correctness

## Theorem

*Prim's algorithm returns an MST.*

## Proof.

Just Generic-MST!

# Correctness

## Theorem

*Prim's algorithm returns an MST.*

## Proof.

Just Generic-MST!

- ▶  $(S, \bar{S})$  always respects  $\mathbf{A}$  (induction).
- ▶ If edge  $e$  added then light for  $(S, \bar{S})$
- ▶ Hence  $e$  safe for  $\mathbf{A}$  by main structural theorem.



# Running Time

Trivial analysis:

- ▶ Every spanning tree has  $n - 1$  edges  $\implies n - 1$  iterations
- ▶ In each iteration, look through all edges to find min-weight edge crossing  $(S, \bar{S}) \implies O(m)$  time
- ▶ Total  $O(mn)$

# Running Time

Trivial analysis:

- ▶ Every spanning tree has  $n - 1$  edges  $\implies n - 1$  iterations
- ▶ In each iteration, look through all edges to find min-weight edge crossing  $(S, \bar{S}) \implies O(m)$  time
- ▶ Total  $O(mn)$

Like Dijkstra's algorithm, do better by using a data structure: heap!

# Running Time

Trivial analysis:

- ▶ Every spanning tree has  $n - 1$  edges  $\implies n - 1$  iterations
- ▶ In each iteration, look through all edges to find min-weight edge crossing  $(S, \bar{S}) \implies O(m)$  time
- ▶ Total  $O(mn)$

Like Dijkstra's algorithm, do better by using a data structure: heap!

- ▶ Need to be able to get minimum-weight edge across  $(S, \bar{S})$

# Running Time

Trivial analysis:

- ▶ Every spanning tree has  $n - 1$  edges  $\implies n - 1$  iterations
- ▶ In each iteration, look through all edges to find min-weight edge crossing  $(S, \bar{S}) \implies O(m)$  time
- ▶ Total  $O(mn)$

Like Dijkstra's algorithm, do better by using a data structure: heap!

- ▶ Need to be able to get minimum-weight edge across  $(S, \bar{S})$

Heap of *vertices* in  $\bar{S}$ . Key of  $v$  is min-weight edge from  $v$  to  $S$ .

# Running Time

Trivial analysis:

- ▶ Every spanning tree has  $n - 1$  edges  $\implies n - 1$  iterations
- ▶ In each iteration, look through all edges to find min-weight edge crossing  $(S, \bar{S}) \implies O(m)$  time
- ▶ Total  $O(mn)$

Like Dijkstra's algorithm, do better by using a data structure: heap!

- ▶ Need to be able to get minimum-weight edge across  $(S, \bar{S})$

Heap of *vertices* in  $\bar{S}$ . Key of  $v$  is min-weight edge from  $v$  to  $S$ .

- ▶ When new vertex  $y$  added to  $S$ , need to update keys of nodes adjacent to  $y$



# Running Time

Trivial analysis:

- ▶ Every spanning tree has  $n - 1$  edges  $\implies n - 1$  iterations
- ▶ In each iteration, look through all edges to find min-weight edge crossing  $(S, \bar{S}) \implies O(m)$  time
- ▶ Total  $O(mn)$

Like Dijkstra's algorithm, do better by using a data structure: heap!

- ▶ Need to be able to get minimum-weight edge across  $(S, \bar{S})$

Heap of *vertices* in  $\bar{S}$ . Key of  $v$  is min-weight edge from  $v$  to  $S$ .

- ▶ When new vertex  $y$  added to  $S$ , need to update keys of nodes adjacent to  $y$ 
  - ▶ Happens at most  $m$  times total

# Running Time

Trivial analysis:

- ▶ Every spanning tree has  $n - 1$  edges  $\implies n - 1$  iterations
- ▶ In each iteration, look through all edges to find min-weight edge crossing  $(S, \bar{S}) \implies O(m)$  time
- ▶ Total  $O(mn)$

Like Dijkstra's algorithm, do better by using a data structure: heap!

- ▶ Need to be able to get minimum-weight edge across  $(S, \bar{S})$

Heap of *vertices* in  $\bar{S}$ . Key of  $v$  is min-weight edge from  $v$  to  $S$ .

- ▶ When new vertex  $y$  added to  $S$ , need to update keys of nodes adjacent to  $y$ 
  - ▶ Happens at most  $m$  times total
- ▶  $n$  Inserts,  $n$  Extract-Mins,  $m$  Decrease-Keys

# Running Time

Trivial analysis:

- ▶ Every spanning tree has  $n - 1$  edges  $\implies n - 1$  iterations
- ▶ In each iteration, look through all edges to find min-weight edge crossing  $(S, \bar{S}) \implies O(m)$  time
- ▶ Total  $O(mn)$

Like Dijkstra's algorithm, do better by using a data structure: heap!

- ▶ Need to be able to get minimum-weight edge across  $(S, \bar{S})$

Heap of vertices in  $\bar{S}$ . Key of  $v$  is min-weight edge from  $v$  to  $S$ .

- ▶ When new vertex  $y$  added to  $S$ , need to update keys of nodes adjacent to  $y$ 
  - ▶ Happens at most  $m$  times total
- ▶  $n$  Inserts,  $n$  Extract-Mins,  $m$  Decrease-Keys
- ▶ Like Dijkstra,  $O(m \log n)$  using binary heap.  $O(m + n \log n)$  with Fibonacci heap (only Extract-Min is logarithmic)

# Kruskal's Algorithm

# Algorithm

Intuition: can we be *even greedier* than Prim's Algorithm?

# Algorithm

Intuition: can we be *even greedier* than Prim's Algorithm?

$\mathbf{A} = \emptyset$

Sort edges by weight (small to large)

For each edge  $\mathbf{e}$  in this order {

    if  $\mathbf{A} \cup \{\mathbf{e}\}$  has no cycles,  $\mathbf{A} = \mathbf{A} \cup \{\mathbf{e}\}$

}

return  $\mathbf{A}$

# Correctness

## Theorem

*Kruskal's algorithm computes an MST.*

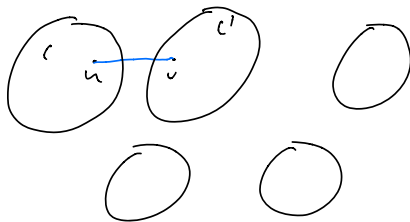
Want to show just Generic-MST: when  $\{u, v\}$  added, it was safe for **A**.

# Correctness

## Theorem

*Kruskal's algorithm computes an MST.*

Want to show just Generic-MST: when  $\{u, v\}$  added, it was safe for **A**.



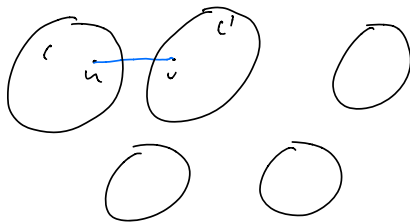


# Correctness

## Theorem

*Kruskal's algorithm computes an MST.*

Want to show just Generic-MST: when  $\{u, v\}$  added, it was safe for  $\mathbf{A}$ .



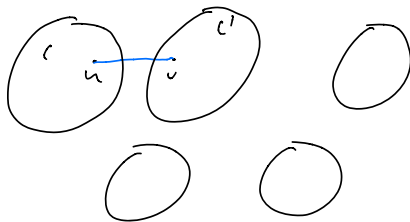
Consider cut  $(C, \bar{C})$ . Respects  $\mathbf{A}$ , and  $\{u, v\}$  light for it.

# Correctness

## Theorem

*Kruskal's algorithm computes an MST.*

Want to show just Generic-MST: when  $\{u, v\}$  added, it was safe for  $\mathbf{A}$ .



Consider cut  $(C, \bar{C})$ . Respects  $\mathbf{A}$ , and  $\{u, v\}$  light for it.

Main structural theorem  $\implies \{u, v\}$  safe for  $\mathbf{A}$

# Running Time

Sorting edges:  $O(m \log m) = O(m \log n)$

## Running Time

Sorting edges:  $O(m \log m) = O(m \log n)$

Easy analysis:  $m$  iterations, DFS/BFS in each iteration to check if endpoints already connected.

# Running Time

Sorting edges:  $O(m \log m) = O(m \log n)$

Easy analysis:  $m$  iterations, DFS/BFS in each iteration to check if endpoints already connected.

- ▶  $O(m(m + n)) = O(m^2 + mn)$

## Running Time

Sorting edges:  $O(m \log m) = O(m \log n)$

Easy analysis:  $m$  iterations, DFS/BFS in each iteration to check if endpoints already connected.

▶  $O(m(m + n)) = O(m^2 + mn)$

Can we speed this up with data structures?

# Running Time

Sorting edges:  $O(m \log m) = O(m \log n)$

Easy analysis:  $m$  iterations, DFS/BFS in each iteration to check if endpoints already connected.

►  $O(m(m + n)) = O(m^2 + mn)$

Can we speak this up with data structures?

Union-Find! Connected components of  $\mathbf{A}$  are disjoint sets.

# Running Time

Sorting edges:  $O(m \log m) = O(m \log n)$

Easy analysis:  $m$  iterations, DFS/BFS in each iteration to check if endpoints already connected.

- ▶  $O(m(m + n)) = O(m^2 + mn)$

Can we speak this up with data structures?

Union-Find! Connected components of  $\mathbf{A}$  are disjoint sets.

- ▶ Make-Sets:



# Running Time

Sorting edges:  $O(m \log m) = O(m \log n)$

Easy analysis:  $m$  iterations, DFS/BFS in each iteration to check if endpoints already connected.

- ▶  $O(m(m + n)) = O(m^2 + mn)$

Can we speed this up with data structures?

Union-Find! Connected components of  $\mathbf{A}$  are disjoint sets.

- ▶ Make-Sets:  $n$

# Running Time

Sorting edges:  $O(m \log m) = O(m \log n)$

Easy analysis:  $m$  iterations, DFS/BFS in each iteration to check if endpoints already connected.

- ▶  $O(m(m + n)) = O(m^2 + mn)$

Can we speed this up with data structures?

Union-Find! Connected components of  $\mathbf{A}$  are disjoint sets.

- ▶ Make-Sets:  $n$
- ▶ Finds:

# Running Time

Sorting edges:  $O(m \log m) = O(m \log n)$

Easy analysis:  $m$  iterations, DFS/BFS in each iteration to check if endpoints already connected.

- ▶  $O(m(m + n)) = O(m^2 + mn)$

Can we speed this up with data structures?

Union-Find! Connected components of  $A$  are disjoint sets.

- ▶ Make-Sets:  $n$
- ▶ Finds:  $2m$

# Running Time

Sorting edges:  $O(m \log m) = O(m \log n)$

Easy analysis:  $m$  iterations, DFS/BFS in each iteration to check if endpoints already connected.

- ▶  $O(m(m + n)) = O(m^2 + mn)$

Can we speak this up with data structures?

Union-Find! Connected components of  $\mathbf{A}$  are disjoint sets.

- ▶ Make-Sets:  $n$
- ▶ Finds:  $2m$
- ▶ Unions:

# Running Time

Sorting edges:  $O(m \log m) = O(m \log n)$

Easy analysis:  $m$  iterations, DFS/BFS in each iteration to check if endpoints already connected.

- ▶  $O(m(m + n)) = O(m^2 + mn)$

Can we speed this up with data structures?

Union-Find! Connected components of  $A$  are disjoint sets.

- ▶ Make-Sets:  $n$
- ▶ Finds:  $2m$
- ▶ Unions:  $n - 1$

# Running Time

Sorting edges:  $O(m \log m) = O(m \log n)$

Easy analysis:  $m$  iterations, DFS/BFS in each iteration to check if endpoints already connected.

▶  $O(m(m + n)) = O(m^2 + mn)$

Can we speak this up with data structures?

Union-Find! Connected components of  $\mathbf{A}$  are disjoint sets.

- ▶ Make-Sets:  $n$
- ▶ Finds:  $2m$
- ▶ Unions:  $n - 1$

$O(m \log^* n)$  using union-by-rank + path compression

$O(m + n \log n)$  using list data structure

# Running Time

Sorting edges:  $O(m \log m) = O(m \log n)$

Easy analysis:  $m$  iterations, DFS/BFS in each iteration to check if endpoints already connected.

- ▶  $O(m(m + n)) = O(m^2 + mn)$

Can we speak this up with data structures?

Union-Find! Connected components of  $\mathbf{A}$  are disjoint sets.

- ▶ Make-Sets:  $n$
- ▶ Finds:  $2m$
- ▶ Unions:  $n - 1$

$O(m \log^* n)$  using union-by-rank + path compression

$O(m + n \log n)$  using list data structure

Sorting dominates!  $O(m \log n)$  total.