Lecture 17: Minimum Spanning Trees

Michael Dinitz

October 29, 2024
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

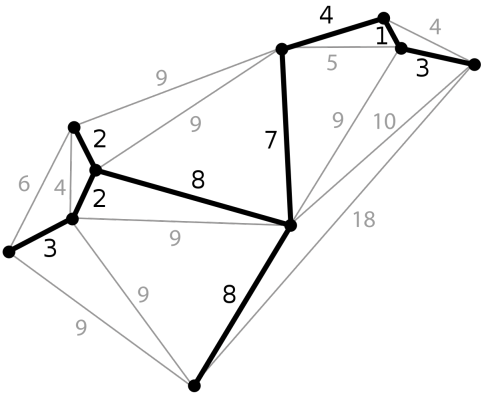
Minimum Spanning Tree problem (MST):

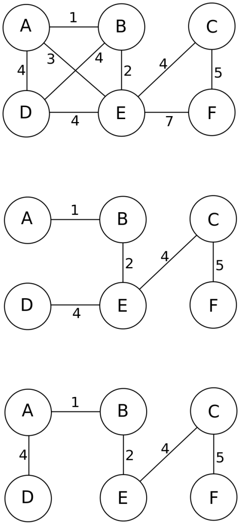
- Input:
  - Undirected graph $G = (V, E)$
  - Edge weights $w : E \to \mathbb{R}_{\geq 0}$
- Output: Spanning tree minimizing $w(T) = \sum_{e \in T} w(e)$.

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

Generic Greedy

Definition

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

```
Generic-MST {
    A = ∅
    while(A not a spanning tree) {
        find an edge e safe for A
        A = A ∪ {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.

Theorem

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

```
Generic-MST {
    A = ∅
    while(A not a spanning tree) {
        find an edge e safe for A
        A = A ∪ {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 = ∅
    while(A not a spanning tree) {
        find an edge e safe for A
        A = A ∪ {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 $\boldsymbol{A}$ is subset of *some* MST. If $\boldsymbol{A} \cup \{\boldsymbol{e}\}$ is also a subset of some MST, then $\boldsymbol{e}$ is *safe* for $\boldsymbol{A}$.

```
Generic-MST {
    A = ∅
    while(A not a spanning tree) {
        find an edge e safe for A
        A = A ∪ {e}
    }
    return A
}
```

Theorem

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

Proof.

Induction.

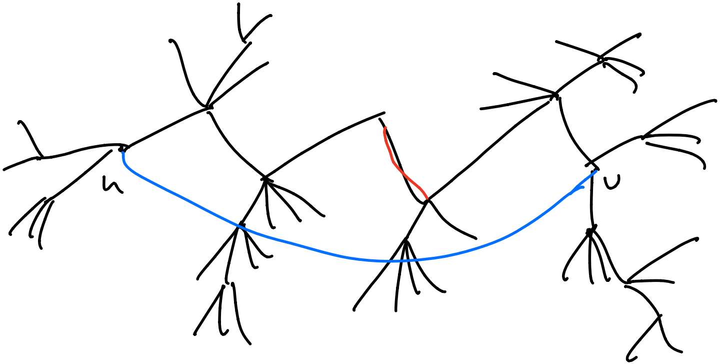
Claim: $\boldsymbol{A}$ always a subset of some MST.
Base case: ✓
Inductive step: ✓ □

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

Structural Properties

Lemma

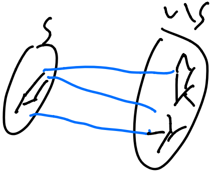
Let $\boldsymbol{T}$ be a spanning tree, let $\boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{V}$, and let $\boldsymbol{P}$ be the $\boldsymbol{u}-\boldsymbol{v}$ path in $\boldsymbol{T}$. If $\{\boldsymbol{u}, \boldsymbol{v}\} \notin \boldsymbol{T}$, then $\boldsymbol{T}' = (\boldsymbol{T} \cup \{\{\boldsymbol{u},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



Structural Properties

Definition

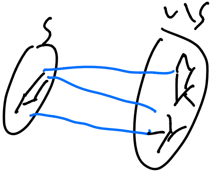
A *cut* $(S, V \setminus S)$ (or $(S, \bar{S})$ or just S) is a partition of V into two parts. Edge e *crosses* cut $(S, \bar{S})$ if e has one endpoint in S and one endpoint in $\bar{S}$.



Structural Properties

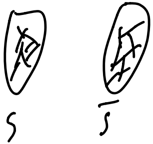
Definition

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 $(S, \bar{S})$



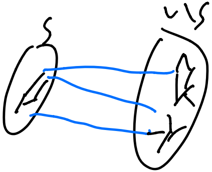
Structural Properties

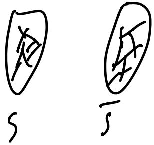
Definition

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

Main Structural Theorem

Theorem

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

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

Main Structural Theorem

Theorem

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

Need to show there is an MST containing $\boldsymbol{A} \cup \{\boldsymbol{e}\}$.
If $\boldsymbol{e} \in \boldsymbol{T}$: ✓

Main Structural Theorem

Theorem

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

Need to show there is an MST containing $\boldsymbol{A} \cup \{\boldsymbol{e}\}$.
If $\boldsymbol{e} \in \boldsymbol{T}$: ✓ Otherwise:

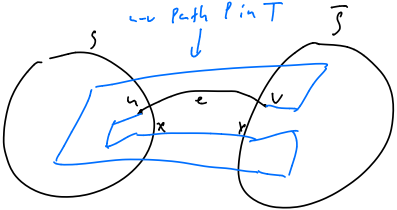
Main Structural Theorem

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 $T' = (T \cup \{e\}) \setminus \{\{x, y\}\}$
$\Longrightarrow$ T' a spanning tree by first lemma



Main Structural Theorem

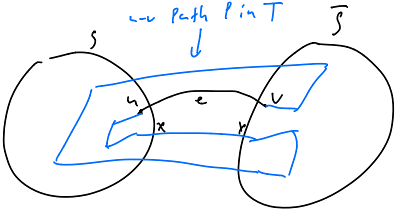
Theorem

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 $T' = (T \cup \{e\}) \setminus \{\{x, y\}\}$
$\implies$ T' a spanning tree by first lemma

$\{x, y\} \notin A$, since $(S, \bar{S})$ respects A
$\implies$ $A \cup \{e\} \subseteq T'$



Main Structural Theorem

Theorem

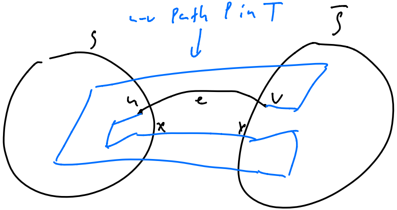
Let $A \subseteq E$ be a subset of some MST T, let $(S, \bar{S})$ be a cut respecting A, and let $e = \{u, v\}$ be a light edge for $(S, \bar{S})$. Then e is safe for A.

Need to show there is an MST containing $A \cup \{e\}$.
If $e \in T$: ✓ Otherwise:

Let $T' = (T \cup \{e\}) \setminus \{\{x, y\}\}$
$\implies$ T' a spanning tree by first lemma

$\{x, y\} \notin A$, since $(S, \bar{S})$ respects A
$\implies$ $A \cup \{e\} \subseteq T'$

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



Main Structural Theorem

Theorem

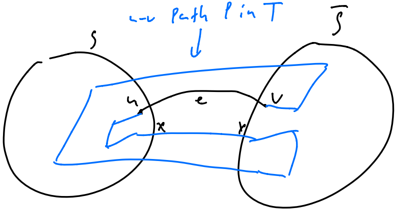
Let $A \subseteq E$ be a subset of some MST T, let $(S, \bar{S})$ be a cut respecting A, and let $e = \{u, v\}$ be a light edge for $(S, \bar{S})$. Then e is safe for A.

Need to show there is an MST containing $A \cup \{e\}$.
If $e \in T$: ✓ Otherwise:

Let $T' = (T \cup \{e\}) \setminus \{\{x, y\}\}$
$\implies$ T' a spanning tree by first lemma

$\{x, y\} \notin A$, since $(S, \bar{S})$ respects A
$\implies$ $A \cup \{e\} \subseteq T'$

$w(T') = w(T) + w(e) - w(x, y) \leq w(T)$
$\implies$ T' an MST containing $A \cup \{e\}$



Prim's Algorithm

Prim's Algorithm

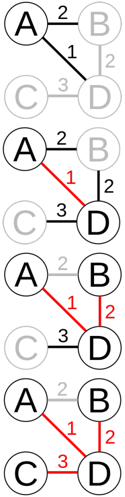
Idea: start at arbitrary node $\boldsymbol{u}$. Greedily grow MST out of $\boldsymbol{u}$.

$\boldsymbol{A} = \varnothing$
Let $\boldsymbol{u}$ be an arbitrary node, and let $\boldsymbol{S} = \{\boldsymbol{u}\}$
while($\boldsymbol{A}$ is not a spanning tree) {
    Find an edge $\{\boldsymbol{x}, \boldsymbol{y}\}$ with $\boldsymbol{x} \in \boldsymbol{S}$ and $\boldsymbol{y} \notin \boldsymbol{S}$ that is light for $(\boldsymbol{S}, \bar{\boldsymbol{S}})$
    $\boldsymbol{A} \leftarrow \boldsymbol{A} \cup \{\{\boldsymbol{x}, \boldsymbol{y}\}\}$
    $\boldsymbol{S} \leftarrow \boldsymbol{S} \cup \{\boldsymbol{y}\}$
}
return $\boldsymbol{A}$

Prim's Algorithm

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

$A = \varnothing$
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 A (induction).
- If edge e added then light for $(S, \bar{S})$
- Hence e safe for 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?

$\boldsymbol{A} = \varnothing$
Sort edges by weight (small to large)
For each edge $\boldsymbol{e}$ in this order $\{$
    if $\boldsymbol{A} \cup \{\boldsymbol{e}\}$ has no cycles, $\boldsymbol{A} = \boldsymbol{A} \cup \{\boldsymbol{e}\}$
$\}$
return $\boldsymbol{A}$

Correctness

Theorem

Kruskal's algorithm computes an MST.

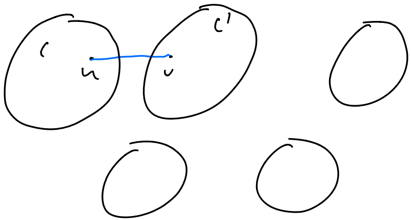
Want to show just Generic-MST: when $\{u, v\}$ added, it was safe for A.

Correctness

Theorem

Kruskal's algorithm computes an MST.

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

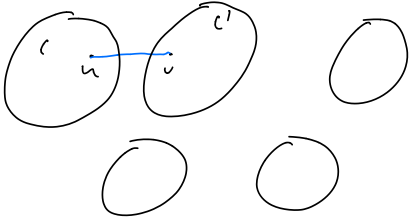


Correctness

Theorem

Kruskal's algorithm computes an MST.

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



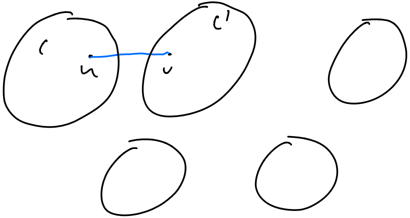
Consider cut $(C, \bar{C})$. Respects A, and $\{u, v\}$ light for it.

Correctness

Theorem

Kruskal's algorithm computes an MST.

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



Consider cut $(C, \bar{C})$. Respects A, and $\{u, v\}$ light for it.
Main structural theorem $\implies$ $\{u, v\}$ safe for A

Running Time

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

Running Time

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

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

Running Time

Sorting edges: $\boldsymbol{O(m \log m) = O(m \log n)}$
Easy analysis: $\boldsymbol{m}$ iterations, DFS/BFS in each iteration to check if endpoints already connected.

- $\boldsymbol{O(m(m+n)) = O(m^2 + mn)}$

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 speak 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 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 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 speak this up with data structures?

Union-Find! Connected components of 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 speak this up with data structures?

Union-Find! Connected components of 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 speak 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 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 speak 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 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 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.