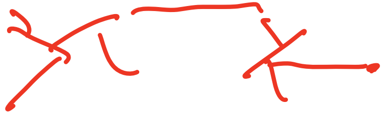
Lecture 16: Minimum Spanning Trees

Michael Dinitz

October 21, 2021
601.433/633 Introduction to Algorithms

Introduction



Definition

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

Definition

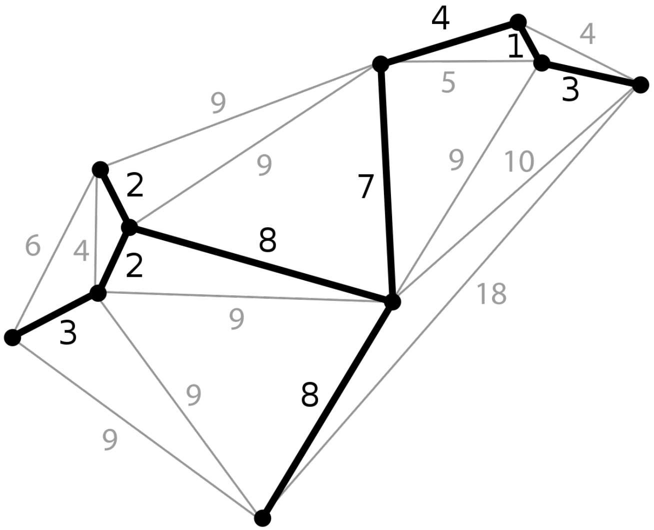
Minimum Spanning Tree problem (MST):

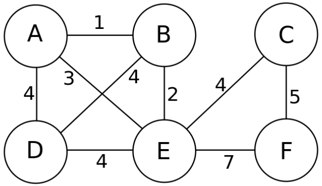
- Input:
  - Undirected graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$
  - Edge weights $\mathbf{w} : \mathbf{E} \to \mathbb{R}_{\geq 0}$
- Output: Spanning tree minimizing $\mathbf{w}(\mathbf{T}) = \sum_{\mathbf{e} \in \mathbf{T}} \mathbf{w}(\mathbf{e})$.

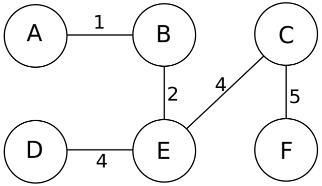
Foundational problem in *network design*. Tons of applications.

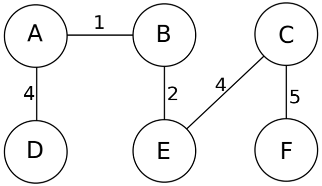
Today: one "recipe", two different algorithms from recipe. Main idea: greedy.

Examples


4
1
4
5
3
9
9
9
10
2
7
8
6
4
2
18
9
3
9
8
9


A
B
C
D
E
F
1
3
4
4
2
4
5
4
7


A
B
C
D
E
F
1
2
4
5
4


A
B
C
D
E
F
1
4
2
4
5

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

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

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 {
    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.

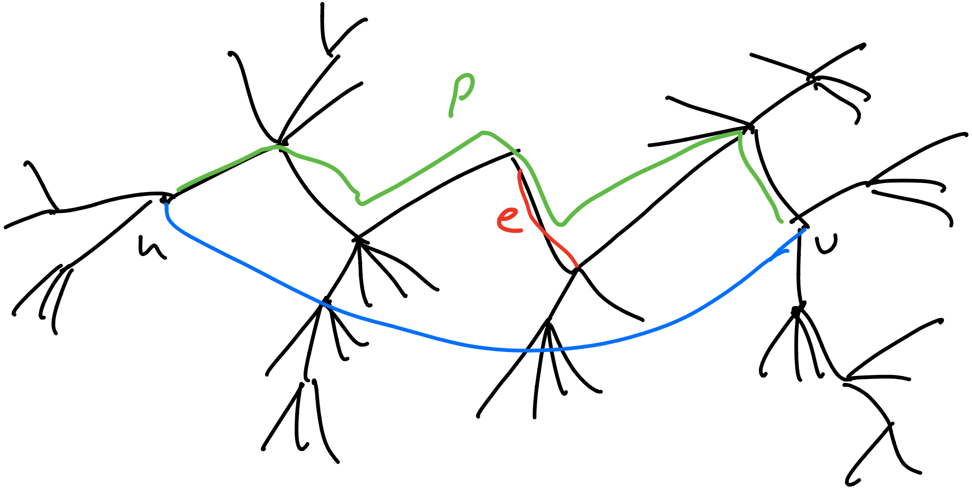
Claim: $\mathbf{A}$ always a subset of some MST.
Base case: ✓
Inductive step: ✓ □

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

Structural Properties

Lemma

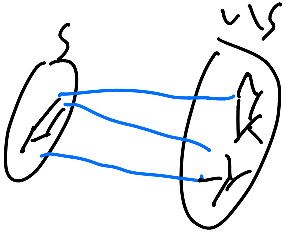
Let $\mathbf{T}$ *be a spanning tree, let* $\mathbf{u}, \mathbf{v} \in \mathbf{V}$, *and let* $\mathbf{P}$ *be the* $\mathbf{u} - \mathbf{v}$ *path in* $\mathbf{T}$. *If* $\{\mathbf{u}, \mathbf{v}\} \notin \mathbf{T}$, *then* $\mathbf{T}' = (\mathbf{T} \cup \{\{\mathbf{u}, \mathbf{v}\}\}) \setminus \{\mathbf{e}\}$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



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 $\bar{S}$.



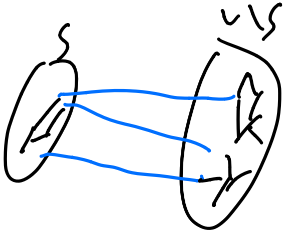
Structural Properties

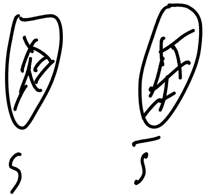
Definition

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

Definition

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





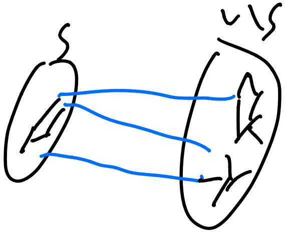
Structural Properties

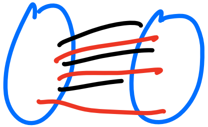
Definition

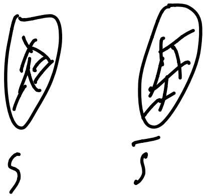
A *cut* $(\mathbf{S}, \mathbf{V} \setminus \mathbf{S})$ (or $(\mathbf{S}, \bar{\mathbf{S}})$ or just $\mathbf{S}$) is a partition of $\mathbf{V}$ into two parts. Edge $\mathbf{e}$ *crosses* cut $(\mathbf{S}, \bar{\mathbf{S}})$ if $\mathbf{e}$ has one endpoint in $\mathbf{S}$ and one endpoint in $\bar{\mathbf{S}}$.

Definition

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







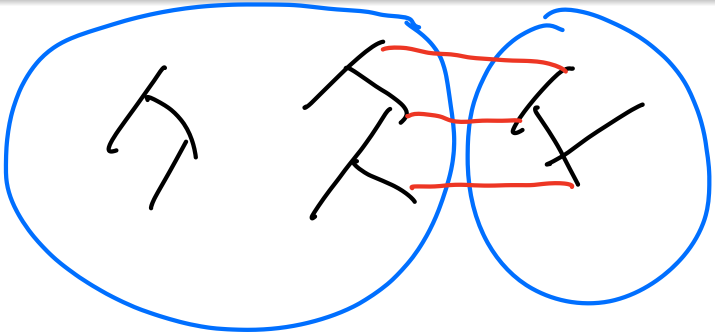
Definition

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

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* $\mathbf{A}$.



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

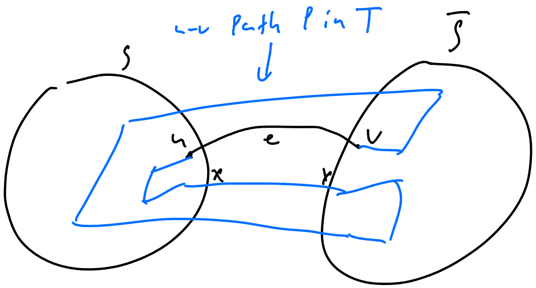
Let $\mathbf{A} \subseteq \mathbf{E}$ be a subset of some MST $\mathbf{T}$, let $(\mathbf{S}, \bar{\mathbf{S}})$ be a cut respecting $\mathbf{A}$, and let $\mathbf{e} = \{\mathbf{u}, \mathbf{v}\}$ be a light edge for $(\mathbf{S}, \bar{\mathbf{S}})$. Then $\mathbf{e}$ is safe for $\mathbf{A}$.

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

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

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

$\implies$ $\mathbf{T}'$ a spanning tree by first lemma



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 $\mathbf{A} \cup \{\mathbf{e}\}$.

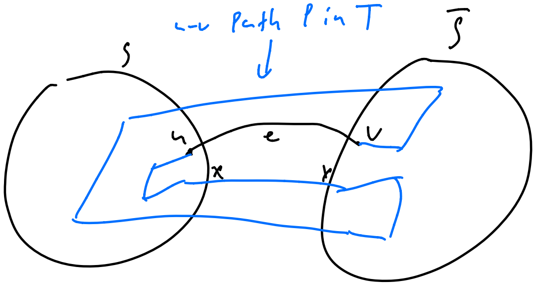
If $\mathbf{e} \in \mathbf{T}$: ✓ Otherwise:

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

$\Longrightarrow$ $\mathbf{T}'$ a spanning tree by first lemma

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

$\Longrightarrow$ $\mathbf{A} \cup \{\mathbf{e}\} \subseteq \mathbf{T}'$



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:

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

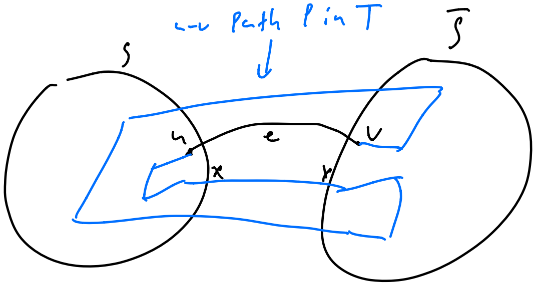
$\implies$ $\mathbf{T}'$ a spanning tree by first lemma

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

$\implies$ $\mathbf{A} \cup \{\mathbf{e}\} \subseteq \mathbf{T}'$

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

=



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 $\mathbf{A}$.

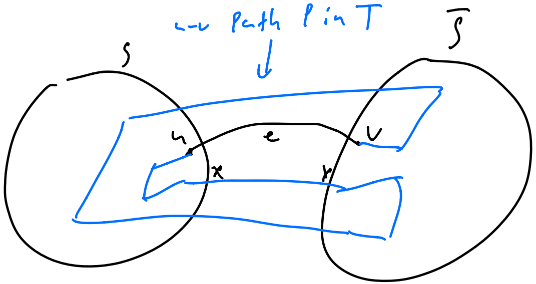
Need to show there is an MST containing $\mathbf{A} \cup \{\mathbf{e}\}$.
If $\mathbf{e} \in \mathbf{T}$: ✓ Otherwise:

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

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

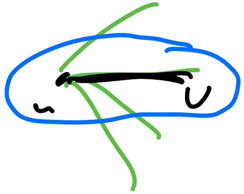
$$\mathbf{w}(\mathbf{T}') = \mathbf{w}(\mathbf{T}) + \mathbf{w}(\mathbf{e}) - \mathbf{w}(\mathbf{x}, \mathbf{y}) \leq \mathbf{w}(\mathbf{T})$$

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



Prim's Algorithm

Prim's Algorithm



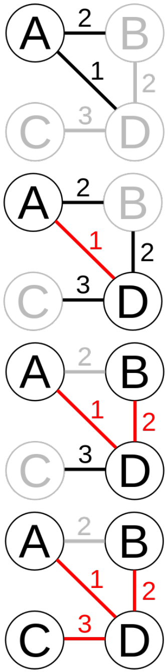
Idea: start at arbitrary node $\mathbf{u}$. Greedily grow MST out of $\mathbf{u}$.

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

Prim's Algorithm

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

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



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

Running Time

Trivial analysis:

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

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

Running Time

Trivial analysis:

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

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

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

Running Time

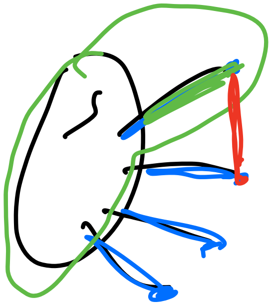
Trivial analysis:

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

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

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

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



Running Time

Trivial analysis:

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

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

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

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

- When new vertex $\mathbf{y}$ added to $\mathbf{S}$, need to update keys of nodes adjacent to $\mathbf{y}$

Running Time

Trivial analysis:

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

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

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

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

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

Running Time

Trivial analysis:

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

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

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

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

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

Running Time

Trivial analysis:

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

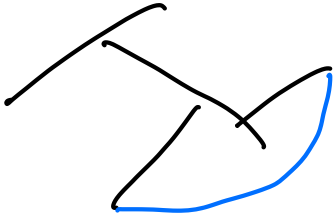
- When new vertex y added to S, need to update keys of nodes adjacent to y
  - Happens at most m times total
- n Inserts, n Extract-Mins, m Decrease-Keys
- Like Dijkstra, $O(m \log n)$ using binary heap. $O(m + n \log n)$ with Fibonacci heap (only Extract-Min is logarithmic)

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?

$\mathbf{A} = \varnothing$
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.

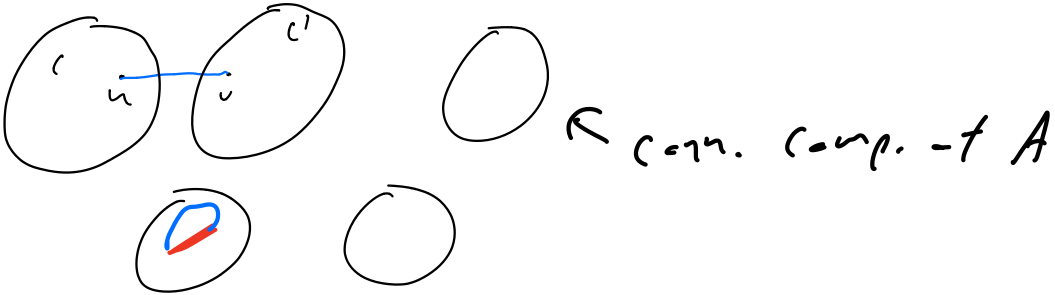
Want to show just Generic-MST: when $\{\mathbf{u}, \mathbf{v}\}$ added, it was safe for $\mathbf{A}$.

Correctness

Theorem

Kruskal's algorithm computes an MST.

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

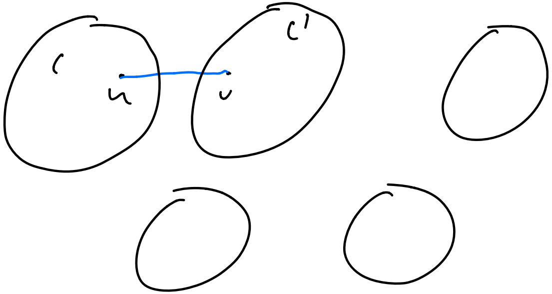


Correctness

Theorem

Kruskal's algorithm computes an MST.

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



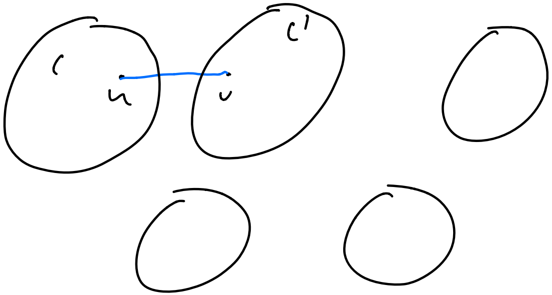
Consider cut $(\mathbf{C}, \bar{\mathbf{C}})$. Respects $\mathbf{A}$, and $\{\mathbf{u}, \mathbf{v}\}$ light for it.

Correctness

Theorem

Kruskal's algorithm computes an MST.

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



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

Running Time

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

Running Time

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

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

Running Time

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

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

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

Running Time

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

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

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

Can we spe~~ak~~ed 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: $\mathbf{O(m \log m) = O(m \log n)}$

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

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

Can we speak this up with data structures?

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

- Make-Sets: $\mathbf{n}$
- Finds:

Running Time

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

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

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

Can we speak this up with data structures?

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

- Make-Sets: $\mathbf{n}$
- Finds: $\mathbf{2m}$

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

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

Can we speak this up with data structures?

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

- Make-Sets: $\mathbf{n}$
- Finds: $\mathbf{2m}$
- Unions: $\mathbf{n-1}$

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: $\mathbf{O(m \log m) = O(m \log n)}$

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

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

Can we speak this up with data structures?

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

- Make-Sets: $\mathbf{n}$
- Finds: $\mathbf{2m}$
- Unions: $\mathbf{n - 1}$

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

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

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