27 January 2025

Greedy Algorithms: Exchange Arguments

Plan

* MST
* Announcements
* Exchange Arguments
  - ↳ The Cycle Lemma
  - ↳ The Cut Lemma
* Implementing Kruskal.

Minimum Spanning Tree

Given a connected, undirected, weighted graph $G = (V, E, w)$

Find a minimum spanning tree $T = (V, E' \subseteq E)$

- minimum: minimize sum of edge weights.

$$w_T = \sum_{e \in T} w_e$$

- spanning: $\forall u, v \in V$, u and v connected in T
- tree: graph with no cycles

Two MST Algs.

On input $G = (V, E, W)$

Step 0. Sort edges by weight

$$w_{e_1} < w_{e_2} < \cdots < w_{e_m}$$

} Greedy priority.

Add Min

$T = (V, \{\})$

For $i = 1 \longrightarrow m$.

if $T \cup \{e_i\}$ does not form a cycle

Add e_i to T.

Return T.

Delete Max

$T = (V, E)$

For $i = m \longrightarrow 1$.

if $T \setminus \{e_i\}$ is still connected

Remove e_i from T

Return T.

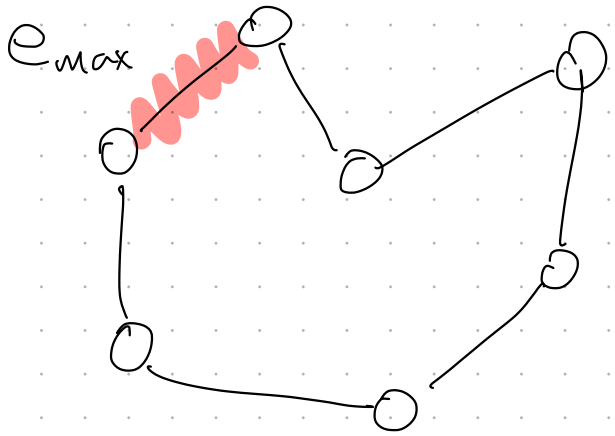
The Cycle Lemma.

$G = (V, E, W)$ w/ distinct edge weights.

Suppose C is a cycle within G and let e_{max} be the edge in C of maximum weight.

Then,

e_{max} is NOT in the MST of G.



Corollary. DeleteMax returns the MST of G.

Theorem. DeleteMax returns the MST.

Proof Steps

(1) Show that DeleteMax returns a spanning tree.
(Good practice exercise.)

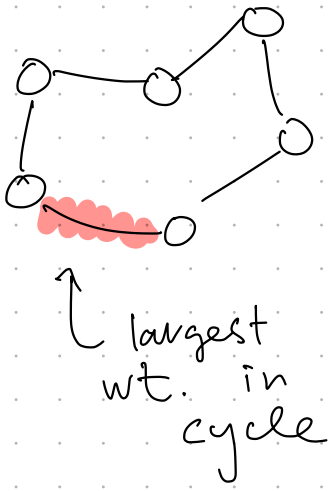
✓ (2) Assuming Cycle Lemma,
Show that DeleteMax returns MST
(Last time)

(3) Prove Cycle Lemma

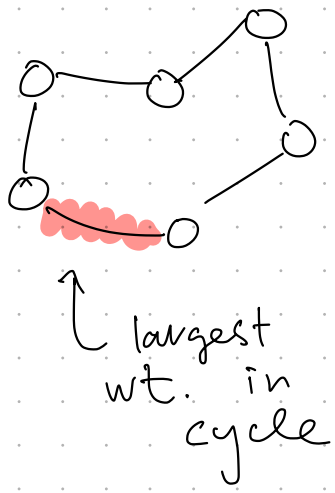
↳ this lecture.

Exchange Argument.

Goal. Show e_{max} cannot be in min spanning tree.



Exchange Argument.



Goal. Show e_{max} cannot be in min spanning tree.

Idea. Start w/ some spanning tree T containing e_{max}.

* "Exchange" e_{max} for some other edge such that

  ↳ Weight goes down

  ↳ New structure still a spanning tree.

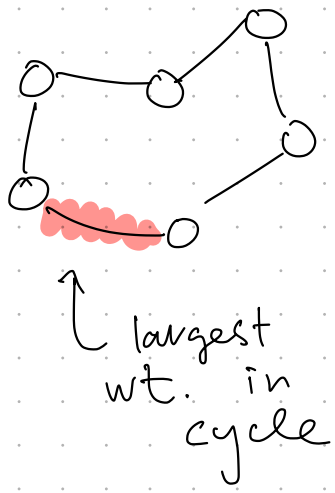
  } ⇒ T is NOT MST

Announcements,

* HW0 due Tues
  ↳ Optional, but highly Recommended!

* OH Schedule is live.

Exchange Argument.



Goal. Show e_{max} cannot be in min spanning tree.

Idea. Start w/ some spanning tree T containing e_{max}.

* "Exchange" e_{max} for some other edge such that

  ↳ Weight goes down

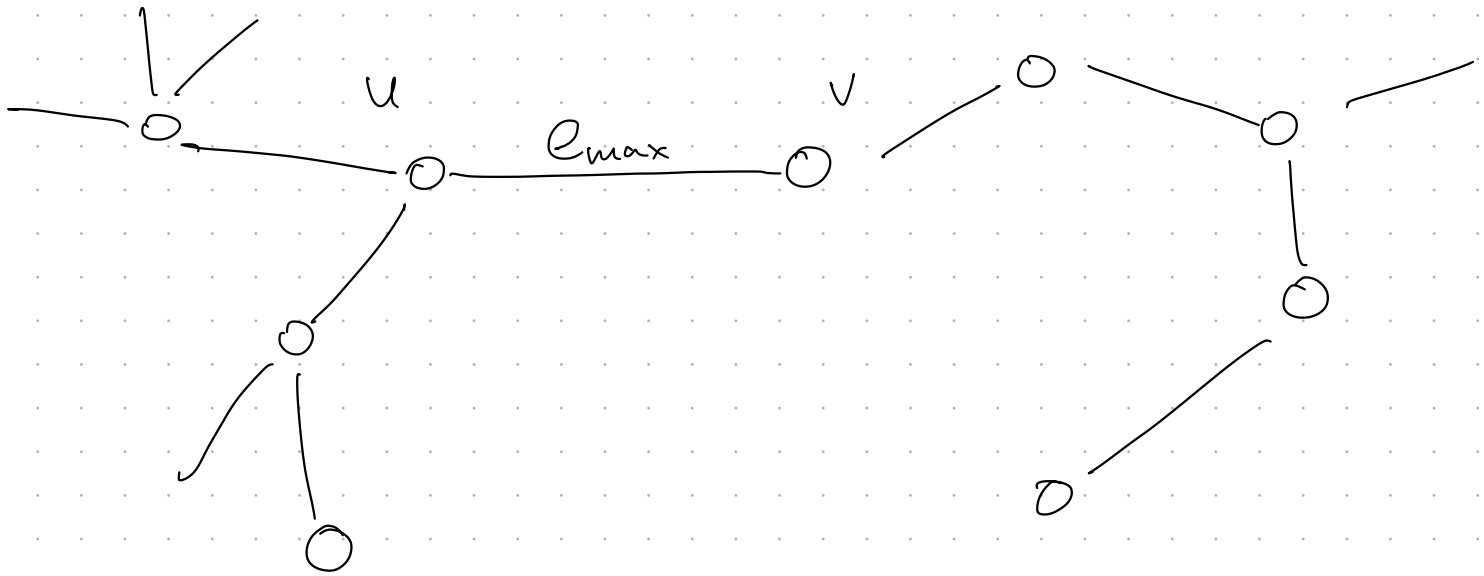
  ↳ New structure still a spanning tree.

  } ⇒ T is NOT MST

Pf. By exchange argument.

Suppose T is a spanning tree s.t. $e_{max} = (u, v) \in T$

We show that T is NOT the minimum spanning tree.

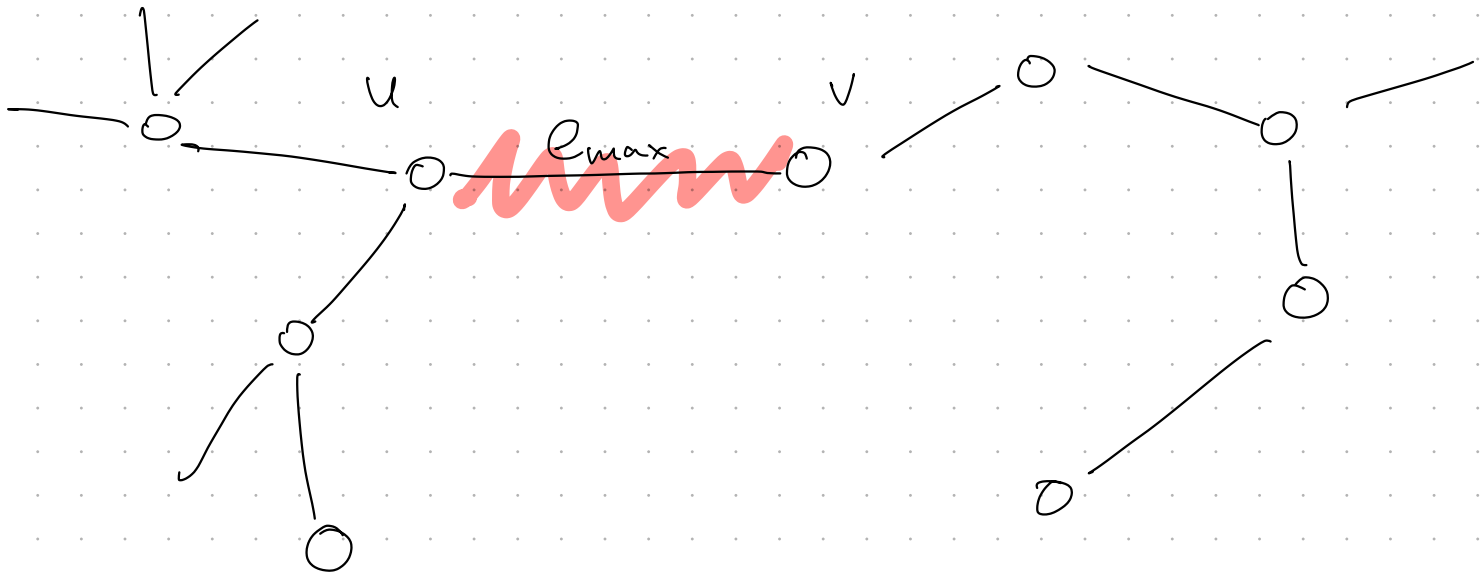


Pf. By exchange argument.

Suppose T is a spanning tree s.t. $e_{max} = (u, v) \in T$

We show that T is NOT the minimum spanning tree.

Consider removing e_{max} from T.

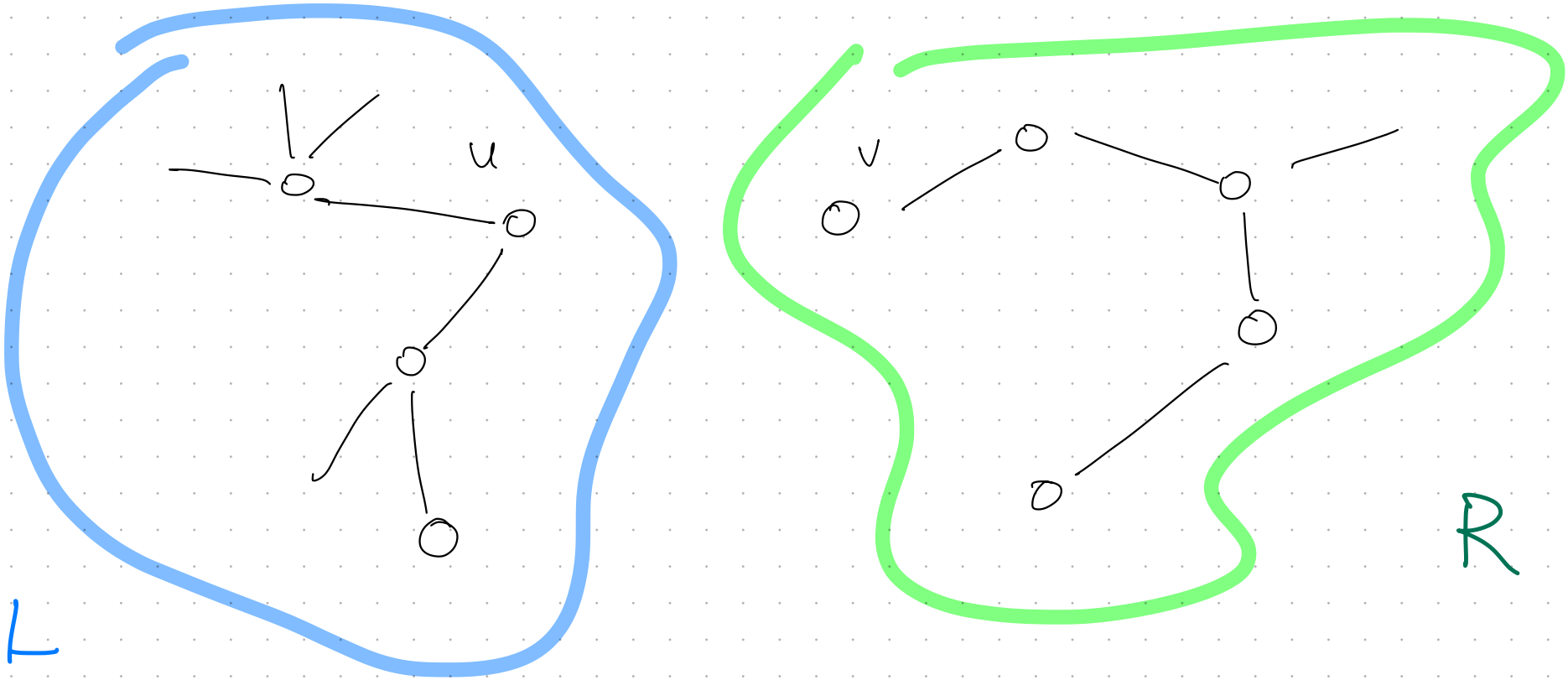


Pf. By exchange argument.

Suppose T is a spanning tree s.t. $e_{max} = (u, v) \in T$

We show that T is NOT the minimum spanning tree.

Disconnects T into two pieces.



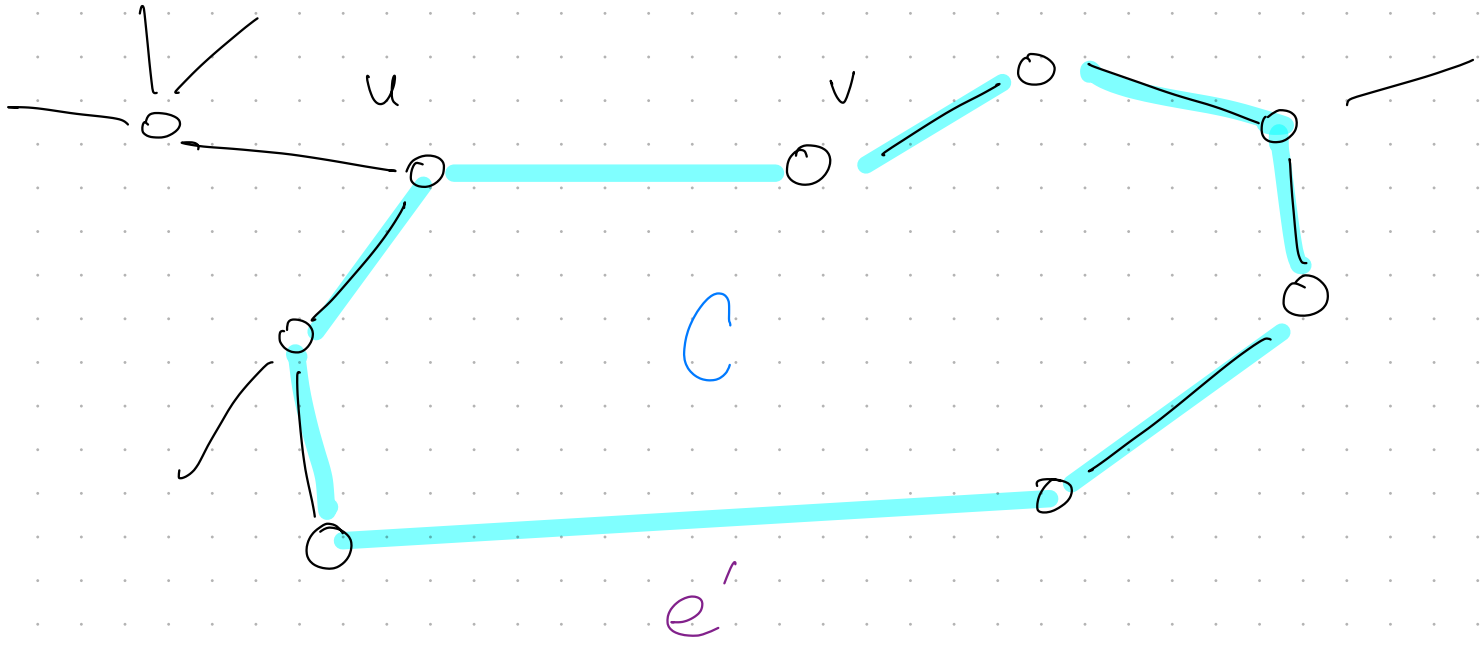
Pf. By exchange argument.

Suppose T is a spanning tree s.t. $e_{max} = (u, v) \in T$

We show that T is NOT the minimum spanning tree.

By assumption, e_{max} is in some cycle C within G.

$\Rightarrow$ there exists some edge $e' \in C$ (but not in T) from L to R



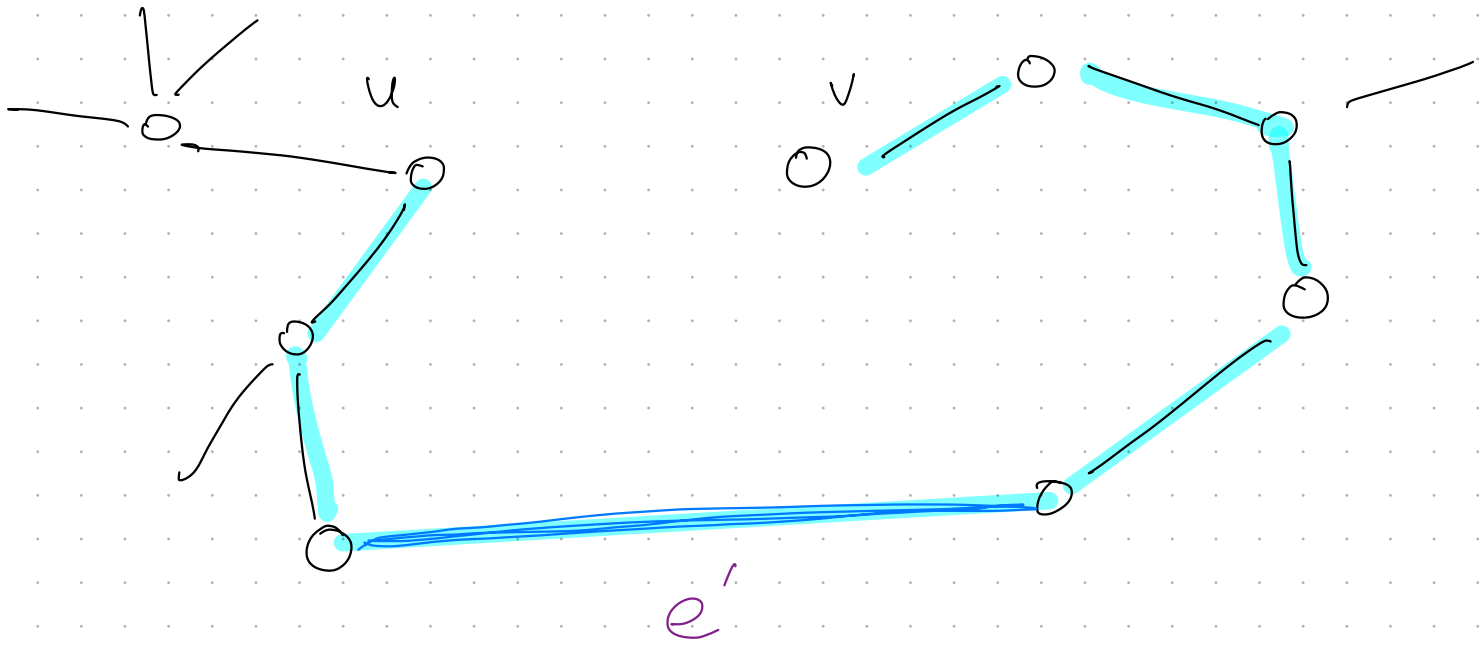
Pf. By exchange argument.

Suppose T is a spanning tree s.t. $e_{max} = (u, v) \in T$

We show that T is NOT the minimum spanning tree.

Consider exchanging e' for e_{max}.

$$T' = T \setminus \{e_{max}\} \cup \{e'\}$$



Claims.

① T' has smaller weight than T

② T' is a tree

③ T' is connected (i.e. a spanning tree)

Claims.

① T' has smaller weight than T

- e_{max} is the max wt. edge on C $\Rightarrow w_{T'} < w_T$
- exchanged for $e' \in C$.

② T' is a tree

③ T' is connected (i.e. a spanning tree)

Claims.

① T' has smaller weight than T

② T' is a tree

- if we add e' to T, we only introduce 1 cycle
- Removing e_{max} breaks this cycle.

③ T' is connected (i.e. a spanning tree)

Claims.

① T' has smaller weight than T

② T' is a tree

③ T' is connected (i.e. a spanning tree)

- Original T was spanning.
- Any path using (u,v) can use detour from u through L to start of e', then through R to v.

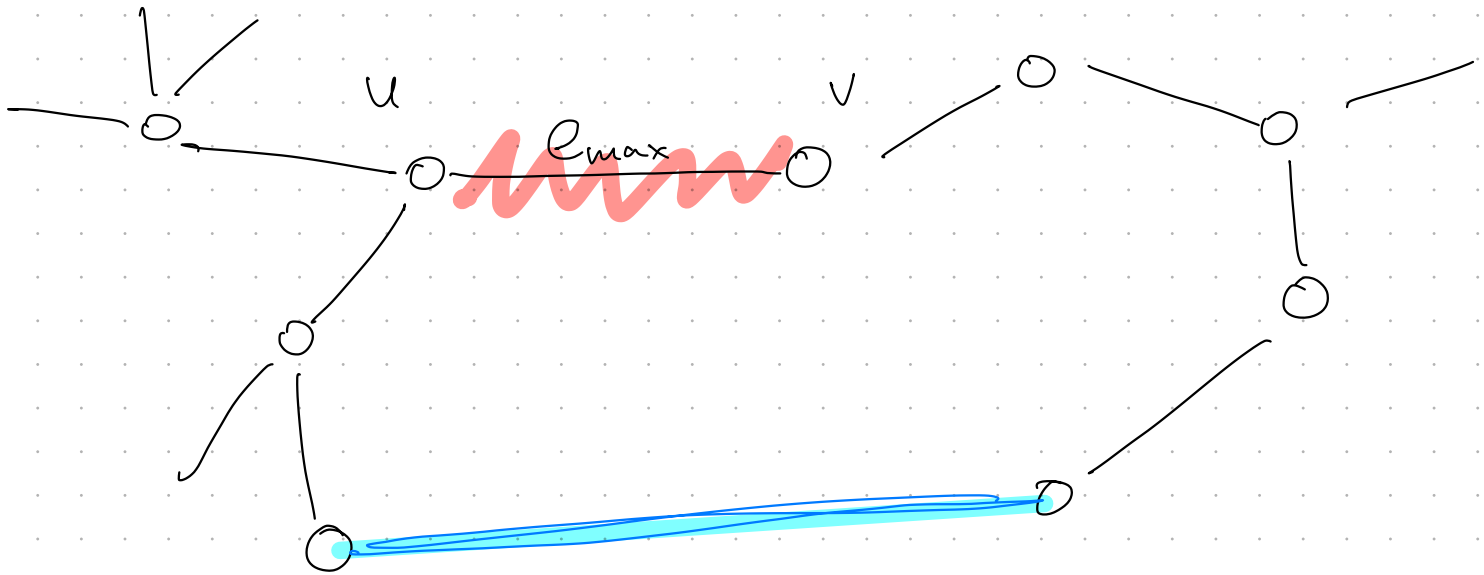
Pf. By exchange argument.

Suppose T is a spanning tree s.t. $e_{max} = (u, v) \in T$

We show that T is NOT the minimum spanning tree.

Thus, T' is a spanning tree w/ $w_{T'} < w_T$

So T is NOT the MST. ▨



Two MST Algs.

On input $G = (V, E, W)$

Step 0. Sort edges by weight

$$w_{e_1} < w_{e_2} < \cdots < w_{e_m}$$

Add Min

$T = (V, \{\})$

For $i = 1 \longrightarrow m$.

if $T \cup \{e_i\}$ does not form a cycle

Add e_i to T.

Return T.

Delete Max

$T = (V, E)$

For $i = m \longrightarrow 1$.

if $T \setminus \{e_i\}$ is still connected

Remove e_i from T

Return T.

What about Add Min ?

Add Min

$T = (V, \{\})$

For $i = 1 \longrightarrow m$.

if $T \cup \{e_i\}$ does not form a cycle

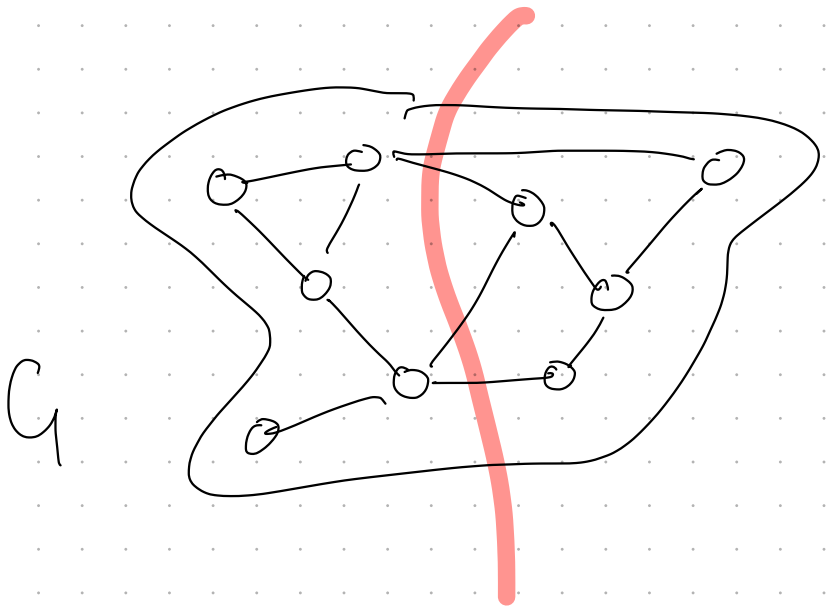
Add e_i to T.

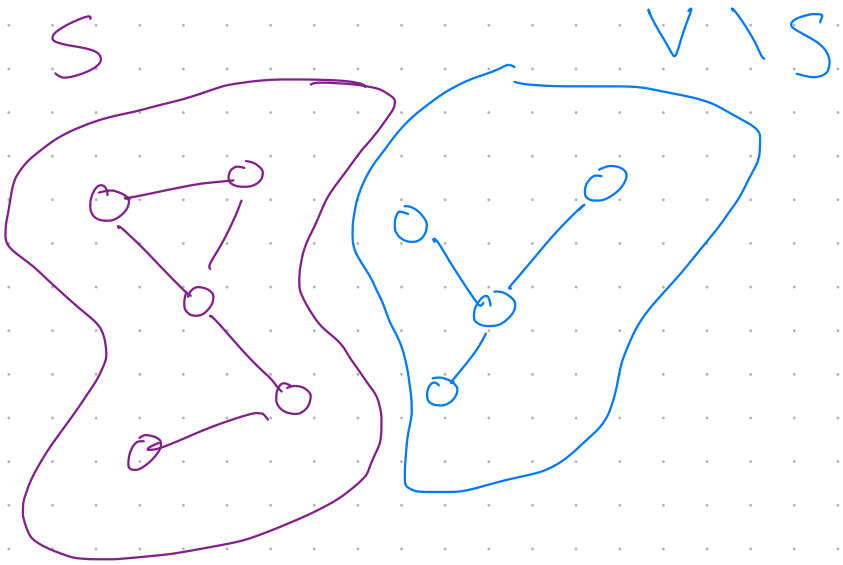
Return T.

Graph Cuts

Given a graph G on vertex set V.

a cut is a partition of the vertices into $S \subseteq V$ and $V \setminus S$.





Are there any edges
that MUST be in the MST?

(Think about cuts
in the graph)

The Cut Lemma.

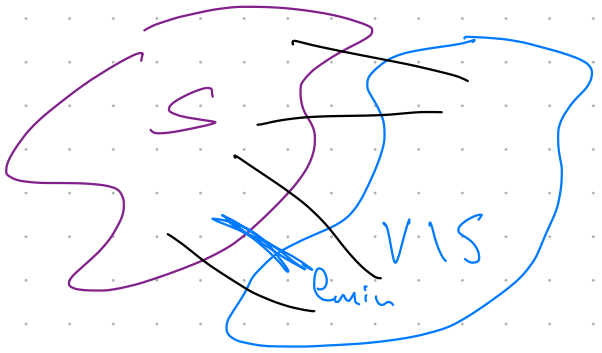
Given a connected, weighted graph $G=(V,E,w)$ w/ distinct edge weights,

* Consider any nontrivial cut $(S, V \setminus S)$.

* Let e_{min} be the minimum weight edge crossing $(S, V \setminus S)$

Then

e_{min} is in the MST



$e_{min} = (u,v)$ for $u \in S, v \in V \setminus S$

Idea for Proof of Cut Lemma. Exchange Argument.

1. Start w/ spanning tree T s.t. $e_{min} \notin T$.
2. Find an exchange s.t.
   - * Exchange includes e_{min} & some heavier e'
   - * Exchange forms a spanning tree
   - * Exchanged weight drops $\Rightarrow$ T is not MST

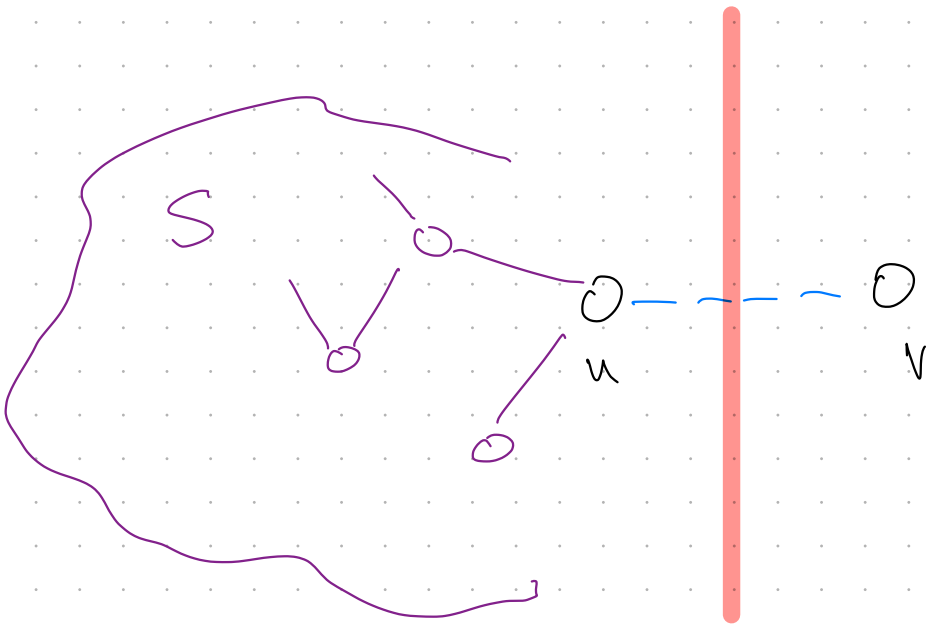
See Section 4.5 of KT.

Corollary. ~~Add Min~~ returns the MST.

aka Kruskal's Algorithm

Proof Idea.

1. Show Kruskal returns a spanning tree
2. Show that every edge (u,v) that Kruskal adds is the minimum weight edge e_{min} crossing some cut $S \subseteq V$.



Implementing Kruskal

Implementing Kruskal's Algorithm

$T = (V, \{\})$

For $i = 1 \longrightarrow m$.

  if $T \cup \{e_i\}$ does not form a cycle

    Add e_i to T.

Return T.

Implementing Kruskal's Algorithm

$T = (V, \{\})$

For $i = 1 \longrightarrow m$.

  if $T \cup \{e_i\}$ does not form a cycle

    Add e_i to T.

Return T.

How do we test efficiently?

Implementing Kruskal's Algorithm

$T = (V, \{\})$

For $i = 1 \longrightarrow m$.

  if $T \cup \{e_i\}$ does not form a cycle

    Add e_i to T.

Return T.

How do we test efficiently?

Idea. Maintain a list of connected components of T

Component (u) = Component (v) $\Longleftrightarrow$ Adding (u, v) forms a cycle!

Union - Find Kruskal

$T = (V, \{\})$

Initialize Components : $\{\{v_1\}, \{v_2\}, \dots, \{v_n\}\}$

For $i = 1 \longrightarrow m$.

- $(u, v) \leftarrow e_i$
- if Component $(u) \neq$ Component (v)
  - * Add e_i to T.
  - * Merge (Component (u), Component (v))

Return T.

Union - Find Kruskal

$T = (V, \{\})$

Initialize Components : $\{\{v_1\}, \{v_2\}, \cdots, \{v_n\}\}$

For $i = 1 \longrightarrow m$.

  $(u, v) \leftarrow e_i$

  if Find $(u) \neq$ Find (v)

    * Add e_i to T.

    * Union (Find (u), Find (v))

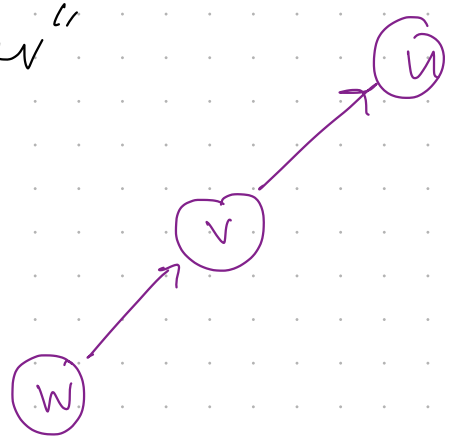
Return T.

Theorem. There is an implementation of the Union - Find data structure that guarantees Kruskal's Algorithm runs in $O(m \log m)$ time.

Union - Find Data Structure

* Every Component maintains a "leader"

  * Vertices maintain a pointer towards the component leader



$K = \{u, v, w\}$

* Find(w) ⟶ follow pointers until end
  Return the leader

* Union(u, v) ⟶ add a pointer from u to v (or v to u)

More on MST

* Union-Find is even better than $O(m \log m)$!

KT § 4.6

* Prim's Algorithm
  - ↳ Similar to Dijkstra's Algo. but w/ different priority
  - ↳ Also $O(m \log m)$

KT § 4.5