Minimum spanning tree.
Lecture in a minute.

Tree Definitions:
- \( n - 1 \) edges and connected.
- \( n - 1 \) edges and no cycles.
- All pairs of vertices connected by unique path.

Minimum Spanning Tree: \( G = (V, E) \), weights \( w : E \)
Kruskal: Sort edges.
Add edges in this order if no cycle.
Cut property:
Exists MST with minimum weight edge across cut.

Union-Find Data Structure.
Pointer implementation: \( \pi(u) \).
makeset(s) \(-\pi(u) = u.
find(x) \) – returns root of pointer structure.
union(x,y) \(-\pi(find(x)) = \pi(find(y)).

Union by rank: \( O(\log n) \) depth for pointer structure.
union(x,y) - point to larger rank root.
increase rank if tied.
> \( 2^k \) nodes in rank \( k \) root tree.
\( O(\log n) \) depth structure.
Def: A tree is a connected graph with no cycles.

Property: A tree has $n - 1$ edges.

Start with empty graph with $n$ components.

Adding any edge between components reduces number of components by one.

After $n - 1$ additions one component!
(If more additions, inside component $\implies$ cycle!)
Def: A tree is a connected graph with no cycles.

Property: A connected graph with \( n - 1 \) edges is a tree.

If not, there is \( n - 1 \) edge connected graph with a cycle.

Remove edge on cycle, still connected. And \( n - 2 \) edges. Must have at least \( n - 1 \) edges to be connected.

Doh! \( \rightarrow \) no cycle.
Another Equivalent Definition

Def: A tree is a connected graph with no cycles.

Property: A graph is a tree if and only if it has a unique path between every pair of nodes.

If two paths:

\[ u \quad v \]

Diverge

Come back together.

\[ \implies \text{cycle! Not Tree!} \]

If yes, connected and no cycle. Tree!
Minimum Spanning Tree.

Given a graph, $G = (V, E)$, edge weights $w_e$, find the cheapest possible connected subgraph.

Will it be a tree?

Yes? No?

Yes. If edge weights positive.

If negative edges, then restrict to tree.

Given a graph, $G = (V, E)$, edge weights $w_e$, find the lowest weight spanning tree.
To MST or not!

MST - cheapest spanning tree of graph.

Shortest path tree
- contains shortest paths from s to other nodes.

MST -
- do not care about shortest paths!
  just lowest weight tree.
Example and Algorithm

MST: total cost is $2 + 4 + 3 + 1 + 5 = 15$. 
Cut property.

Smallest edge across any cut is in some MST.

\[ S \quad V - S \]

Tree Connected \[ \implies \]
there exists \( e' \) across cut! Replace \( e' \) with \( e \).

Every pair remains connected.

If used \( e' \) can use path through \( e \).

and \( n - 1 \) edges.

So still a tree and is no more costly (\( w(e) \leq w(e') \)).
Sort edges.
\[ F = . \text{ For each edge: } e \]
\[ \text{If no cycle, } F = F + e. \]

How to check for cycle for edge \((u, v)\) in \(F\)?
Check for path between \(u\) and \(v\) in \(F\).

Total Running time?
\(O(n)\) time \(\rightarrow O(nm)\) for Kruskals.
Kruskal

Sort edges.

$F = \ldots$ For each edge: $e = (u, v)$

If no cycle in $F$, add edge.

Main issue: Check for cycle.

Maintain connected components.

At beginning each node by itself.

Adding edge, joins component.

Edge $(u, v)$ in cycle? $u$ and $v$ in same component.

Disjoint Sets Data Structure.

makeset($x$) - makes singleton set \{x\}.

find($x$) - finds set containing $x$.

union($x, y$) - merge sets containing $x$ and $y$.

“If no cycle” $\equiv$ “$\text{find}(u) \neq \text{find}(v)$”

“Add edge” $\equiv$ “union($u, v$)”
Disjoint Set Data Structure

Maintain pointers: $\pi(x)$ for each $x$.

**makeset**$(x)$  $\pi(x) = x$.

**union**(x,y)

$\pi$($\text{find}(x)$) = $\text{find}(y)$

**find**(x)

if $\pi(x) == x$

return x

else

find($\pi(x)$)

How long does find take?

(A) $O(n)$

(B) $O(1)$

(C) Depends.

Want depth to be small!
Maintain pointers: $\pi(x)$ for each $x$.

**makeset(x)** $\pi(x) = x$.

**find(x)**
  
  if $\pi(x) == x$
    return $x$
  else
    find($\pi(x)$)

Make a bit less deep: union-by-rank.

union(x,y)
Use roots of $x$ and $y$.
Which points to which?
“smaller” to “larger” ..sort of.
Union by rank.

Initially: \( \text{rank}(x) = 0 \).

\textbf{union}(x,y) \\
\hspace{1em} r_x = \text{find}(x) \\
\hspace{1em} r_y = \text{find}(y) \\
\hspace{1em} \textbf{if} \ \text{rank}(r_x) < \text{rank}(r_y): \\
\hspace{2em} \pi(r_x) = r_y \\
\hspace{1em} \textbf{else:} \\
\hspace{2em} \pi(r_y) = r_x \\
\hspace{1em} \textbf{if} \ \text{rank}(r_x) == \text{rank}(r_y): \\
\hspace{2em} \text{rank}(r_x) += 1
**Lemma:** Pop’s got a higher rank:

$$\text{rank}(x) < \text{rank}(\pi(x))$$

if $x \neq \pi(x)$.

Duh!

Code enforces it.

```python
union(x, y):
    if rank(r_x) < rank(r_y):
        \( \pi(r_x) = r_y \)
    else:
        \( \pi(r_y) = r_x \)
        if rank(r_x) == rank(r_y):
            rank(r_x) += 1
```

Initially?
Big rank is a big dog!

union(x,y):

if \text{rank}(r_x) < \text{rank}(r_y):
\pi(r_x) = r_y
else:
\pi(r_y) = r_x
if \text{rank}(r_x) == \text{rank}(r_y):
\text{rank}(r_x) += 1

Lemma: Any rank \( k \) root node has \( \geq 2^k \) nodes in its tree.

Induction:

Base Case?

(A) \( 2^0 \geq 1 \)
(B) \( 2^1 \geq 1 \)

A. Initially \( \text{rank}(x) = 0 \), 1 node in tree.

Induction step:

When rank(x) goes up to \( k \).

rank(x) was \( k - 1 \) so has \( \geq 2^{k-1} \) nodes. by ind. hyp.
gains nodes from rank \( k - 1 \) node with \( \geq 2^{k-1} \) nodes
\implies \geq 2^{k-1} + 2^{k-1} = 2^k \) nodes.
Check your understanding?

Exactly $2^k$ nodes in tree of rank $k$? Yes? No?

No.

\[
\text{if } \text{rank}(r_x) < \text{rank}(r_y):
\]
\[
\pi(r_x) = r_y
\]

Gains nodes without gaining rank!
Back to complexity.

Find(x) is

(A) $O(\log n)$ time.

(B) $O(1)$ time

(C) $O(n)$ time.

A. (and (C)).

Rank $k$ root node has $\geq 2^k$ nodes.

Only $n$ nodes in any set.

Every rank at most $\log n$, (otherwise, $> 2^{\log n} = n$ nodes.)

Parent has higher rank. Code enforces it.

Only $k$ steps in find.

$O(k) = O(\log n)$ time.

Yay!

Can we do better? Yes. We will see better.
Kruskal Implementation.

$|V|$ unions. $|E|$ finds.

$O(|E| \log n)$ time!

Versus $O(|E||V|)$. 
Lecture in a minute.

Tree Definitions:
- \( n - 1 \) edges and connected.
- \( n - 1 \) edges and no cycles.
- All pairs of vertices connected by unique path.

Minimum Spanning Tree: \( G = (V, E) \), weights \( w : E \)
Kruskal: Sort edges.
  Add edges in this order if no cycle.
Cut property:
  Exists MST with minimum weight edge across cut.

Union-Find Data Structure.
  Pointer implementation: \( \pi(u) \).
  makeset(s) – \( \pi(u) = u \).
  find(x) – returns root of pointer structure.
  union(x,y) – \( \pi(find(x)) = \pi(find(y)) \).

Union by rank: \( O(\log n) \) depth for pointer structure.
  union(x,y) - point to larger rank root.
  increase rank if tied.
\( > 2^k \) nodes in rank \( k \) root tree.
\( O(\log n) \) depth structure.
See you on Wednesday!