Trees.

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

**Property:** A tree has \( n - 1 \) edges.

Start with an empty graph with \( n \) components.

![Diagram of a tree with labeled vertices](image)

Adding any edge between components reduces the number of components by one. After \( n - 1 \) additions, one component!

(If more additions, inside component \( \implies \) cycle!)

---

**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.

---

**Equivalent Definition.**

**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:

Diverge

Come back together.

\( \implies \) cycle! Not Tree!

If yes, connected and no cycle. 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:**

Given a graph, \( G = (V,E) \), 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:**

\( \pi(u) \).

- \( \text{makeSet}(s) \rightarrow \pi(u) = u \).
- \( \text{find}(x) \rightarrow \text{return root of pointer structure} \).
- \( \text{union}(x,y) \rightarrow \pi(\text{find}(x)) = \pi(\text{find}(y)) \).

Union by rank:

- \( O(\log n) \) depth for pointer structure.
- \( O(\log n) \) depth structure.

![Diagram of a tree with labeled vertices](image)

- Increase rank if tied.

- > 2\( k \) nodes in rank \( k \) root tree.

*Union by rank: \( O(\log n) \) depth for pointer structure.*

- \( \pi(u) \).
- \( \text{makeSet}(s) \rightarrow \pi(u) = u \).
- \( \text{find}(x) \rightarrow \text{return root of pointer structure} \).
- \( \text{union}(x,y) \rightarrow \pi(\text{find}(x)) = \pi(\text{find}(y)) \).

Union by rank:

- \( O(\log n) \) depth for pointer structure.
- \( O(\log n) \) depth structure.

- Increase rank if tied.

- > 2\( k \) nodes in rank \( k \) root tree.

*Union by rank: \( O(\log n) \) depth for pointer structure.*

- \( \pi(u) \).
- \( \text{makeSet}(s) \rightarrow \pi(u) = u \).
- \( \text{find}(x) \rightarrow \text{return root of pointer structure} \).
- \( \text{union}(x,y) \rightarrow \pi(\text{find}(x)) = \pi(\text{find}(y)) \).

Union by rank:

- \( O(\log n) \) depth for pointer structure.
- \( O(\log n) \) depth structure.

- Increase rank if tied.

- > 2\( k \) nodes in rank \( k \) root tree.

*Union by rank: \( O(\log n) \) depth for pointer structure.*

- \( \pi(u) \).
- \( \text{makeSet}(s) \rightarrow \pi(u) = u \).
- \( \text{find}(x) \rightarrow \text{return root of pointer structure} \).
- \( \text{union}(x,y) \rightarrow \pi(\text{find}(x)) = \pi(\text{find}(y)) \).

Union by rank:

- \( O(\log n) \) depth for pointer structure.
- \( O(\log n) \) depth structure.

- Increase rank if tied.

- > 2\( k \) nodes in rank \( k \) root tree.

*Union by rank: \( O(\log n) \) depth for pointer structure.*
**To MST or not!**

- Shortest Path Tree from s!
  - MST? Yes!
  - Shortest path from s to v in tree? No!
  - 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**

```
A
B
C
D
E
F
5
3
5
2
6
1
5
3
```

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

**Cut property.**

- Smallest edge across any cut is in some MST.

```
S
V −S
e
e′
Tree Connected =⇒
there exists e′ across cut! Replace e′ with e.
If used e′ can use path through e.
and n −1 edges.
So still a tree and is no more costly (w(e) ≤w(e′).)
```

**Disjoint Set Data Structure**

- Maintain pointers: \( \pi(x) \) for each x.
  - makeset(x) \( \pi(x) = x \).
  - find(x) \( \pi(\text{find}(x)) = \text{find}(y) \).
  - union(x,y) \( \pi(\text{find}(x)) ) = \text{find}(y) \).

```
A
B
C
D
E
F
```

**Kruskal**

- Sort edges.
  - \( F = \) For each edge: \( e \)
    - 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.

```
A
B
C
D
E
F
```

- Add edge
  - \( n \) −1 edges.
  - So still a tree and is no more costly (\( w(e) \leq w(e') \)).

**Disjoint Sets Data Structure**

Maintain pointers: \( \pi(x) \) for each \( x \).
- makeset(x) \( \pi(x) = x \).
- find(x) \( \pi(\text{find}(x)) = \text{find}(y) \).
- union(x,y) \( \pi(\text{find}(x)) = \text{find}(y) \).

```
A
B
C
D
E
F
```

- How long does find take?
  - (A) \( O(n) \)
  - (B) \( O(1) \)
  - (C) Depends.

Want depth to be small!
Disjoint Set Data Structure

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

\[
\text{makeSet}(x) \quad \pi(x) = x.
\]

\[
\text{find}(x)
\]

\[
\quad \text{if} \quad \pi(x) = x
\quad \text{return} \quad x
\]

\[
\quad \text{else}
\quad \text{find}(\pi(x))
\]

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

\[
\text{union}(x, y)
\]

Use roots of \( x \) and \( y \).

Which points to which?

"smaller" to "larger"...sort of.

Big rank is a big dog!

\[
\text{union}(x, y)
\]

\[
\quad \text{if rank}(x) < \text{rank}(y):
\quad \pi(x) = y
\]

\[
\quad \text{else}:
\quad \pi(y) = x
\]

\[
\quad \text{if rank}(x) == \text{rank}(y):
\quad \text{rank}(x) + = 1
\]

Why rank?

\[
\text{Lemma: Pop's got a higher rank:}
\]

\[
\quad \text{rank}(x) < \text{rank}(\pi(x))
\]

\[
\quad \text{if} \quad x \neq \pi(x).
\]

Duh!

Code enforces it.

\[
\text{union}(x, y):
\]

\[
\quad : \quad \text{if rank}(x) < \text{rank}(y):
\quad \pi(x) = y
\]

\[
\quad \text{else}:
\quad \pi(y) = x
\]

\[
\quad \text{if rank}(x) == \text{rank}(y):
\quad \text{rank}(x) + = 1
\]

Check your understanding?

Exactly \( 2^k \) nodes in tree of rank \( k \)? Yes? No?

No.

\[
\text{Gains nodes without gaining rank!}
\]

Back to complexity.

\[
\text{Find}(x)
\]

\[
\quad \text{is}
\]

\[
\quad (A) \quad \text{O(log} n\text{)} \text{ time.}
\]

\[
\quad (B) \quad \text{O(1)} \text{ time}
\]

\[
\quad (C) \quad \text{O(n)} \text{ time.}
\]

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.

\[
\text{O}(k) = \text{O(log} n\text{)} \text{ time.}
\]

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: π(u).
makeset(s) – π(u) = u.
find(x) – returns root of pointer structure.
union(x,y) – π(find(x)) = π(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!