Minimum Spanning Tree

Given an undirected graph \( G = (V, E) \) with edge weights \( w : E \to \mathbb{R} \).
(can be negative)
Want a minimum spanning tree.

Tree: acyclic, (subgraph) graph $\Gamma$ connected

Spanning: contains all vertices of $G$

Minimum: $\min \left\{ w(T) : T \subseteq \Gamma \right\}$

(Assuming $G$ is connected. Other no spanning trees)
We'll assume \( w(e) \neq w(e') \)
if \( e \neq e' \).

Otherwise there could be multiple min spanning trees.
Let $T$ be the min spanning tree (we want this!).

We'll iteratively add one or more edges we know belong to $T$.

Say we're midway through:

$F \subseteq T$: the intermediate spanning forest of edges picked so far.

Acyclic; may have several components.
Initially, $F$ is just 1) isolated vertices. Stop when $F$ is connected (of $1V1-1$ edges).

Two kinds of edges with respect to $F$.

1) **Useless**: $F U \emptyset e 3$ has a cycle.

2) $T$ has no useless edges!
For any component of $F$, its safe edge is the lightest edge with exactly one endpoint in the component.

An edge is safe if it is safe for at least one endpoint's component.

Lemma (Prim '57): Min spanning tree $T$ contains every safe edge wrt $F \subseteq T$. (with respect to)
Proof: Will show. For any subset $S \subseteq V$ of vertices, the lightest edge with exactly one endpoint in $S$ belongs to $T$.

Let $T$ be a (the) min spanning tree.

Let $e$ be the lightest edge leaving $S$.

If $e \in T$, we're done.

Suppose otherwise...
Let $uv = e$.

There is a path $P$ from $u$ to $v$.

Some edge $e'$ of $P$ goes from $V \backslash S$ to $S$.

$T - e'$ has no $u$-$v$ path, so $T' = T - e' + e$ is a spanning tree.
e is safe for S, so
\[ w(e) < w(e') \]
\[ w(T') < w(T) \]

I must contain e after all!

Alg idea: Start with no edges in F.
Repeatedly add one or more safe edges until we have a spanning tree.
Kruskal '56:
Scan edges in increasing weight order. If edge is safe, add it to $F$.

Use disjoint set.
Make Set$(v)$: creates a set containing only $v$.
Find$(v)$: Returns the name of a vertex in $v$'s component.
Find(u) = Find(v) if and only if u and v are in the same component.

Union(u, v): Tells data structure we're combining u and v's components.

Kruskal(V, E):
- sort E by increasing weight
- \( F \leftarrow (V, \emptyset) \)
- for each vertex \( v \in V \)
  - MakeSet(\( v \))
- for \( i \leftarrow 1 \) to \( |E| \)
  - \( uv \leftarrow \) ith lightest edge in \( E \)
  - if FIND(\( u \)) \( \neq \) FIND(\( v \))
    - UNION(\( u, v \))
    - add \( uv \) to \( F \)
- return \( F \)

\[ O(E \log V) \leftarrow O(E \log V^2) \leftarrow O(E \log V) \]

\( O(E \log E) = O(E \log V^2) = O(E \log V) \)

sort

if graph is simple, then \( E \leq \binom{V}{2} = V^2 \)
$O(V)$ Make Set + Union
$O(F)$ Find

Simple answer: $O(\log V)$ time per operation, so $O(E \log V)$ total.

Disjoint sets with path compression structure:

$O(E \alpha (V)) = o(E \log V)$ time total

very very very small inverse Ackermann function

$\alpha (V) \leq 4$ for any $V \leq \# stars$
But we had to sort:

$O(E \log V)$ total
Jarník '29:
Prim '57:

If $F$ has one non-trivial component, always add a safe edge for that one component.
One implementation:

Keep a priority queue of edges leaving the component.
Add edges leaving some vertex $v$.
Repeat until queue is empty
Delete min $uv$ from queue
If either endpoint unmarked
Mark both endpoints
Add $uv$ to $F$
Add outgoing edges of newly marked vertex

Return $F$

Binary Heap gives $O(\log E) = O(\log V)$ time per operation

$O(\log V) \times \text{Total}$
We'll make this faster next week.
Borůvka '26:
Add all safe edges + repeat.

$O(E)$ time to compute components of $F$ and find their safe edges.

You'll have at most half the # components each iteration.

$\Rightarrow O(\log V)$ iterations

$\Rightarrow O(E \log V)$ time total
A variant runs in $O(V)$ time for certain nice types of graphs (planar).