Minimum Spanning Tree (MST)

Given connected undirected graph $G = (V, E)$.

Weights $w: E \rightarrow \mathbb{R}$ could be negative!

Goal: Find the minimum spanning tree, a spanning tree $T$ minimizing $w(T) = \sum_{e \in T} w(e)$.
Assumption: $w(e) \neq w(e')$ when $e \neq e'$.

$\Rightarrow$ guarantees MST is unique

Otherwise, could be multiple MST's

Ex. $w(e) = 1 \ \forall e \Rightarrow$ every spanning tree has weight $|V| - 1$
The One Algorithm:

T: the MST we want to find

want to select edges bit by bit

for T

part way through algorithm

F \subseteq T: the edges we chose so far

- acyclic (a forest)

- call it the intermediate spanning forest

- initially the set of (V) one-vertex trees
will add edges to make
\[ F' \supset F \text{ s.t. } F' \subseteq T. \]
- then recursively find
\[ \text{MST } T \supseteq F'. \]
- stop if \( F \) is connected
Given $F$, there are two special subsets of edges:

- **useless edges**: outside $F$ but both endpoints in components of $F$. If $F$ has no useless edges, $F + e$ has a cycle.
- Each component of $F$ has a safe edge: the lightest one leaving the component.
If $F \neq T$, there is at least one safe edge.
If $F = T$, all $e \in T$ are useless.

Claim: MST $T$ contains every safe edge. In fact, for all SCV tree $T$ has lightest edge with one endpoint in $S$.

If MST $T$ contains $e$.

O.w.

$e \in S$
$0 \in S$
exists a path in T between e's endpoints
path has an edge $e'$ that goes from in S to not in S
T-$e'$ has two components
Both components have one end point of e.
\[ \Rightarrow T-e'+e \text{ is a spanning tree.} \]
But $w(e) < w(e')$
so $w(T-e'+e) < w(T)$
so $e \notin T \text{ after all!}$
So... add one or more safe edges to $F$, and recurse!

...but which ones?
Kruskal [56]:

Scan edges in increasing weight order; add each safe edge you see.

Claim: When we scan e, all e’ s.t. \( w(e') \leq w(e) \) are in \( F \) or useless.

If e not useless, it is lightest for both endpoints’ components \( \Rightarrow \) safe.
Disjoint Sets: Maintains disjoint subsets over a collection of objects.

MakeSet($v$): creates set \{ $v$ \}

Find($v$): returns an "ID" for $v$'s set. $\text{Find}(u) = \text{Find}(v)$ if $u \cup v$ in same set

Union($u$,$v$): replaces sets for $u \cup v$ with union of the sets
**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 $F$

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

Total time \[ O(E \log V) + O(E \log V) = O(E \log V) \]
dominated by sorting

even with faster disjoint sets
Prim-Jarník:
Jarník [129], Prim [57].

$F$ always has one non-trivial component $T'$. Others are isolated vertices.

Jarník: Repeatedly add safe edge of $T'$ to $T'$.

($T'$ starts as any one vertex)
To implement:

keep a priority queue of edges incident to $T$.

each time you add an edge to $T'$, add new incident edges to $T'$.

when you extract at least one endpoint in $T$, check if edge leaves $T'$ & ignore it if not.

$O(\log E) = O(\log V)$ time per heap operation, so

$O(E \log V)$
But really, use Borůvka ('26).
See Erickson 7.3.