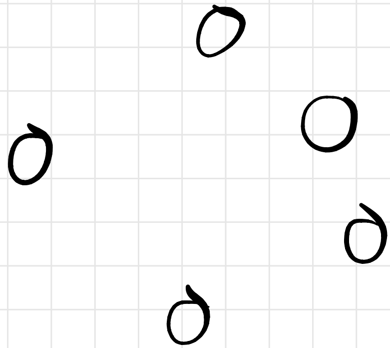
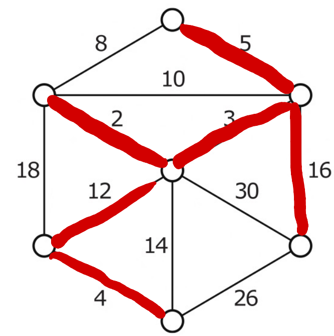




Minimum Spanning Tree

Given undirected graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}$.

(can be negative!)

Want a minimum spanning tree.

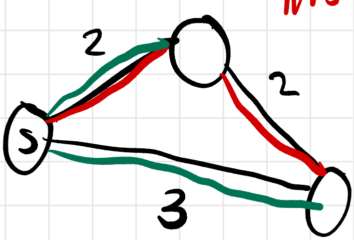
tree: acyclic (sub) graph + connected

spanning: contains all vertices of G

minimum: $\min w(T) := \sum_{e \in T} w(e)$

over all spanning trees T

MST



SSSP

(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 $|V|$ isolated vertices.

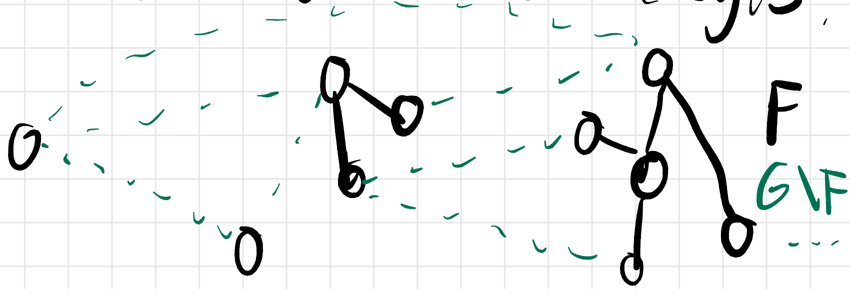
Stop when F is connected (at $|V|-1$ edges).

Two kinds of edges e with respect to F.

1) <u>Useless</u>: $F \cup \{e\}$ has a cycle.

T has no useless edges!

2)



For any component of F, its <u>safe</u> edge is the lightest edge with exactly one endpoint in the component.

e 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)

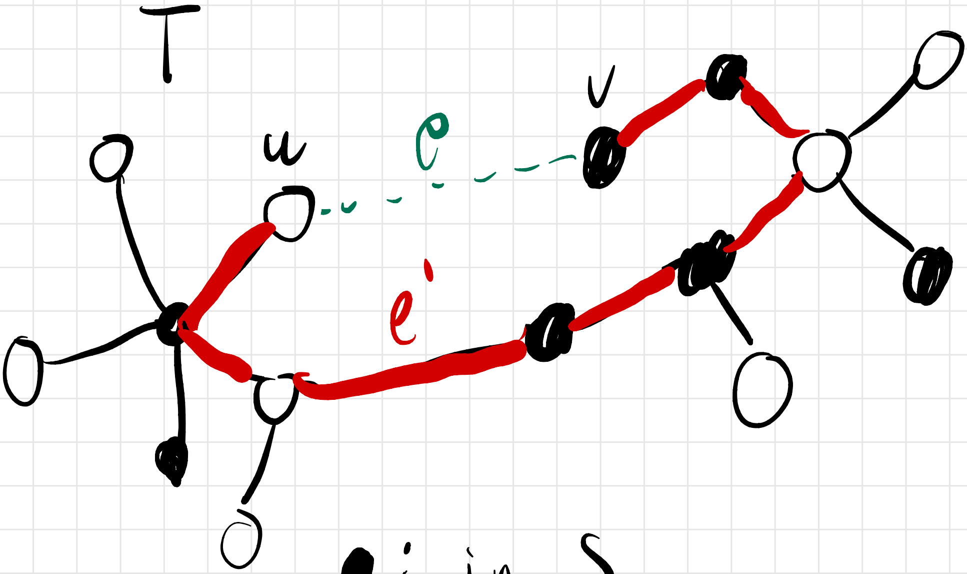
Proof: Will show: For any subset $S \subset V$ of vertices, the lightest edge with exactly one endpoint in S belongs to T.

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...



●: in S

○: in $V \setminus S$

Let $uv = e$.

There is a path P in T from u to v.

Same edge e' of P goes from $V \setminus S$ to S.

$T - e'$ has no u-v path so $T' = T - e' + e$ is a spanning tree e is safe for S, so

$$w(e) < w(e')$$

$$w(T') < w(T)$$

$\perp$

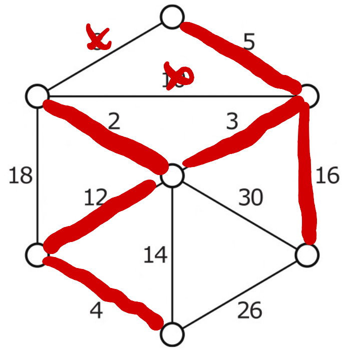
T must contain e after all!

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.

MakeSet(v): creates a set containing only v.

Find(v): Returns the name of a vertex in v's component.

Find(u) = Find(v) iff u & v are in same component.

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

KRUSKAL(V, E):
```
    sort E by increasing weight
    F ← (V, ∅)
    for each vertex v ∈ V
        MAKESET(v)
    for i ← 1 to |E|
        uv ← ith lightest edge in E
        if FIND(u) ≠ FIND(v)
            UNION(u, v)
            add uv to F
    return F
```

← $O(E \log V)$

← $E\alpha(V)$

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

sort

assuming graph is simple $\Rightarrow$ $E \leq \binom{V}{2} < V^2$

$O(V)$ MakeSet + Union

$O(E)$ 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 small

inverse Ackermann function

$\alpha(V) \leq 4$ for any $V = \#$ stars

But we had to sort :(

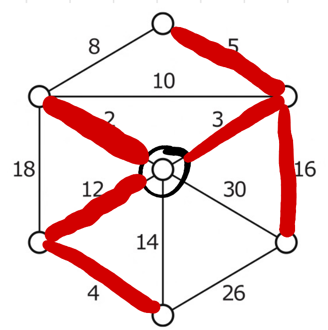
$O(E \log V)$ total

Jarník '29:

Prim '57:

F has one non-trivial component.

Alway add safe edge for that one component.



One implementation:

Keep a priority queue of edges leaving the component.

Add edges leaving same vertex s.

Repeat until queue is empty

DeleteMin 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

$O(E \log V)$ total

We'll make this faster next week.

Borůvka '26:

Add <u>all</u> safe edges & repeat.

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

You'll have <u>at most</u> 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).