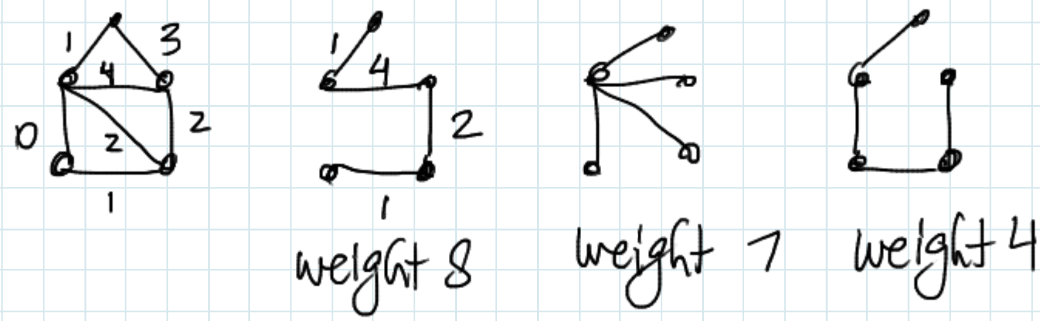
Minimum Spanning Tree

Problem: Given a graph $G=(V,E)$ with weights $w: E \to \mathbb{R}^{\geq 0}$ on the edges find a subset of the edges that connects all the vertices and has minimum weight.

e.g.



The edge subset will be a tree, ← why?
called the minimum spanning tree

Greedy algorithms will find min. spanning trees
You've seen some of this in MATH 239.

In fact, there are several possible correct greedy approaches, with different implementation challenges.

e.g. - add cheapest edge first, never build a cycle
Kruskal's alg.
- grow connected graph from one vertex
Prim's alg.
- throw away expensive edges, never disconnect

Kruskal's Algorithm

Order edges by weight $e_1 \cdots e_m$

$w(e_i) \le w(e_{i+1})$

$T \leftarrow \emptyset$

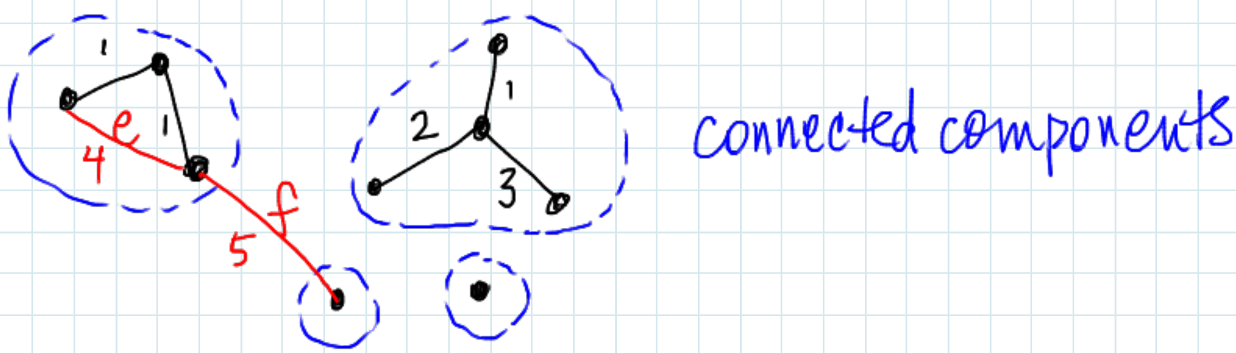
for $i = 1 .. m$

  if e_i does not make a cycle with T then

    $T \leftarrow T \cup \{e_i\}$

end

General situation



e makes a cycle with T iff e joins vertices in same connected component.

e.g. edge e makes a cycle $\Rightarrow$ throw it out

  edge f does not $\Rightarrow$ add f to T

Correctness – an exchange proof

Let T have edges $t_1 \cdots t_{n-1}$

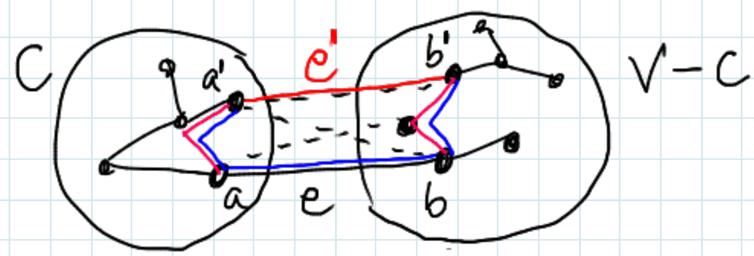
Prove by induction on i that there is a MST matching T on the first i edges

- basis case: $i = 0$

- Assume by induction that there is a MST M matching T on the first k edges.

$$\begin{array}{cccccccc} T & t_1 & \cdots & t_{i-1} & t_i & \cdots & & t_{n-1} \\ & \| & & \| & \not\| & & & \\ M & m_1 & \cdots & m_{i-1} & m_i & & \cdots & m_{n-1} \end{array}$$

Let $t_i = e = (a,b)$ and let C be the connected component of T containing a.



Look at $\boxed{\text{path}}$ in M from a to b

It must cross from C to $V-C$ (maybe multiple times)

Let e' be an edge of the path that goes from C to $V-C$.

Then $w(e) \le w(e')$ otherwise Kruskal would add e'

Claim $M' = (M - \{e'\}) \cup \{e\}$ is a MST

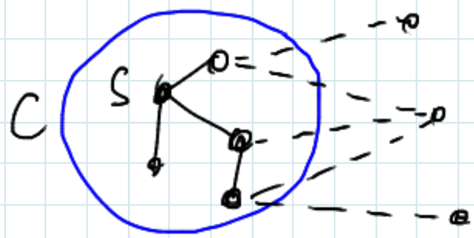
Then we're done, since M' matches T on i edges.
(note $e' \notin T$ so $t_1 \cdots t_{i-1}$ still in M')

Pf. ① M' is a spanning tree because it connects all vertices (replace e' by blue path from a' to a in M, edge e, from b to b' in M)
and has same no. of edges

② $w(M') = w(M) - w(f) + w(e) \le w(M)$
So M' is a min. spanning tree.

Prim's Algorithm.

Grow one connected component in a greedy fashion (i.e. by adding min. weight edge leaving the component.



Choose min. weight edge leaving C

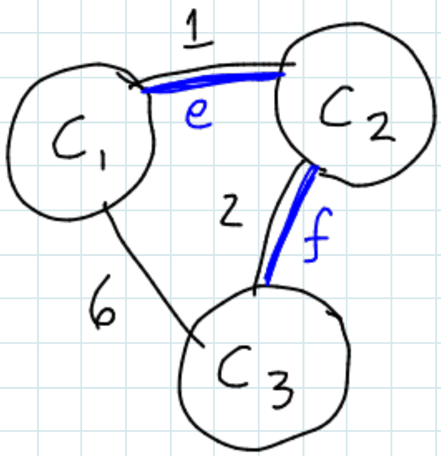
C = set of vertices reached by T so far

initialize $C \leftarrow \{s\}$, $T \leftarrow \emptyset$
while $C \neq V$
  find min. weight edge $e = (u, v)$ from
    $u \in C$ to $v \in V - C$
  $T \leftarrow T \cup \{e\}$
  $C \leftarrow C \cup \{v\}$
end

Correctness The exact same exchange argument works. And in fact, we could prove one lemma that gives correctness of both algs. (see text).

Other greedy MST algs:
- Kruskal backwards. $i = m$ down to 1
  throw away e_i if result is still connected

- Baruvka's Alg.



in each round
Add min. weight edge leaving each component.
Good parallel algorithm!

e.g. add e – min weight leaving C_1
f – min " " C_2
and " " " C_3

Implementing and analyzing MST algorithms.

Graph $G = (V, E)$ $|V| = n$ $|E| = m$

Kruskal

$O(m \log m)$ to sort edges $= O(m \log n)$
because $m \leq n^2$ so $\log m$ is $O(\log n)$

Then we need to maintain connected components as we add edges. Also test if edge (a,b) has a, b in same component (don't add edge), or different components (do add edge)

Union-Find Problem

Maintain a collection of disjoint sets

Operations

- Find(x) — which set contains element x?
- Union — unite two sets

In our case the elements are vertices and the sets are connected components of T, the tree so far

This Abstract Data Structure has a very simple implementation that gives $O(m \log n)$ for Kruskal.

There is a fancier implementation — CS 466
[alg. is pretty simple, analysis is hard and true run time involves inverse Ackerman's fn, very slow growing]

Simple implementation of Union Find.
Keep array $S[1..n]$, $S[i]$ = component of element i
and keep linked list of elements in each set

e.g. C_1: 1, 3, 5, 6
C_2: 2, 4
C_3: 7

Find is $O(1)$
Union – must rename one of the two sets join 2 linked lists $O(1)$ and
so $O(n)$ in worst case

BUT renaming smaller set does better!
e.g. to unite C_1 and C_2 do $C_1 \Leftarrow C_1 \cup C_2$
must fix $S(2) \Leftarrow 1$, $S(4) \Leftarrow 1$.

If an element's set number changes, then its set (more than) doubles
This happens $\leq \log n$ times
Therefore total renaming work is $O(n \log n)$

Total run time

$$\underbrace{O(m \log n)}_{\text{sort}} + \underbrace{O(m)}_{\text{Finds}} + \underbrace{O(n \log n)}_{\text{Unions}}$$

so $O(m \log n)$