Minimum Spanning Tree

~~Lemma~~: (Recall problem and lemma from last lecture)

Prim's Algorithm

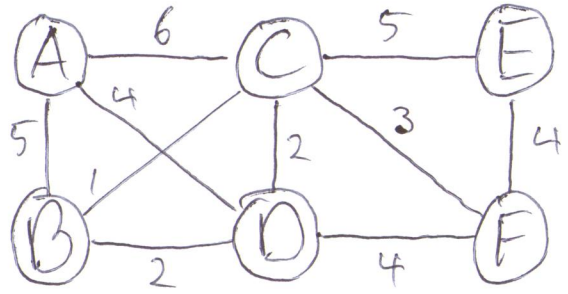
```
T ← {v} (arbitrary)
while |T| < n do
  . add the shortest
    edge (a,b) with
    a∈T and b∉T to T
return T
```

Kruskal's Algorithm

```
E ← sort all edges by length
T ← ∅
for i ← 1 to |E| do
    add E[i] to T if it
    does not create a cycle
return T
```

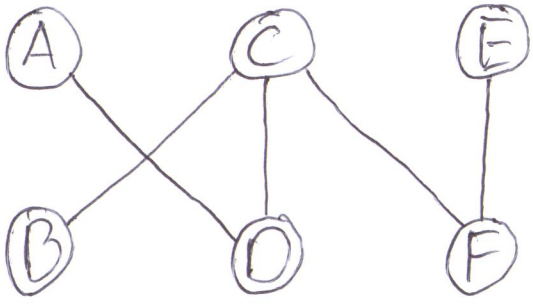
I'll analyze Kruskal's algorithm - Prim's is in A4!

Example:



Edges:
BC, CD, BD,
CF, DF, EF,
AD, AB, CE, AC

Result:



- Correct? There are two parts to this:

Claim: ~~A~~The graph returned by Kruskal is connected.

Proof: Suppose it is not. Then there are at least two connected components C_1 and C_2. ~~Each edge joining C_1 and C_2~~ But the shortest edge joining C_1 and C_2 would have been added to T, so this is impossible. □

Claim: Before and after each iteration of the loop, there is an MST containing all edges of T.

Proof: The claim holds initially, as any MST contains the empty set.

Suppose that the claim is true before the loop. Let ~~T~~ M be ~~the~~ an MST containing all edges of T, and let e be the next edge considered. If e creates a cycle in T, or $e \in M$ then we are done. If e is added to T, but $e \notin M$, then e creates a cycle in M that has an edge $f \notin T$. Then $T' = T - f + e$ is also a spanning tree, and it has the same weight as T, since the algorithm took e ~~@~~ before f, so $wt(e) \leq wt(f)$. □

- Terminates? Yes.

- Efficient?

Sorting the edges takes $O(|E| \log |E|) = O(|E| \log |V|)$ since ~~$|E| = O(|V|^2)$~~. $|E| \leq |V|^2$.

Creating T and adding edges to it can be done in $O(|V|)$ time.

If we can check for a cycle in $O(s)$ time, the for-loop takes $O(|E|s)$ time.

$\Rightarrow$ Total: $O(|E|(\log |V| + s))$

So if we can check for cycles in $O(\log |V|)$ time, this is $O(|E| \log |V|)$ total.

We use a <u>union-find</u> data structure. This ~~can ave~~ supports two operations:

- Union(x,y): merge the sets containing x and y
- Find(x): return ~~thea~~ unique identifier of the set containing x

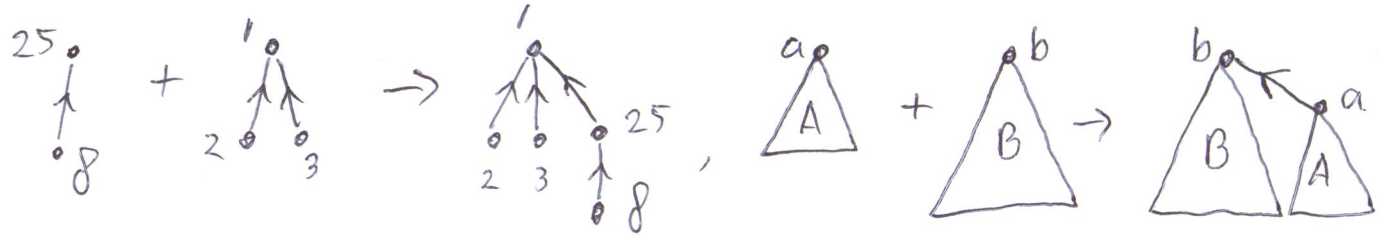
"does (u,v) create a cycle" $\Rightarrow$ "Find(u) $\stackrel{?}{=}$ Find(v)?"

It works as follows:

- start by making each vertex it's own set:

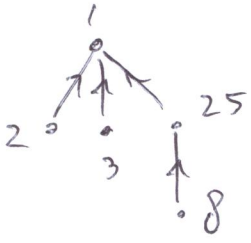
- When 2 sets merge, we create a rooted tree:

- When 2 trees merge, we make the ~~smaller~~ a tree with fewer levels child of the ~~larger~~ one with more:



If each root stores the number of levels, ~~merge~~ union(x, y) takes O(1) time.
merging

- We implement Find(x) by walking to the root of x's tree:



Find(8) = Find(25) = Find(1) = 1

Find(2) = Find(1) = 1

This takes O(#levels) time.

Claim: A tree with k levels has at least 2^{k-1} nodes.

Proof: This is true initially, since $2^{1-1} = 2^0 = 1$.

If the level doesn't increase with a merge, it stays true. If the level does increase, both trees had k levels, so the new tree has at least $2^{k-1} + 2^{k-1} = 2 \cdot 2^{k-1} = 2^k = 2^{(k+1)-1}$ nodes. □

This means that the maximum level with n vertices is $O(\log n)$.

Thus, both Union and Find take $O(\log n)$ time.

With a small improvement, called path compression, this can be brought down to $O(\alpha(n))$. Here $\alpha(n)$ is an extremely slowly growing function called the inverse Ackermann function. For any realistic input size, $\alpha(n) < 5$.