CS 2100

Graphs:
MSTs

Recap

- HW due today
- HW4: graded (in git)
  Please check! ("git pull")
- Working on grading 7+8 now
- Last lab - due next Friday
- HW - due next Saturday

→ Both on ZyBooks

- Review: last day of class
  (sample final handed out in class next week)
- Final: Wednesday at 2pm
  No conflicts or testing ctr

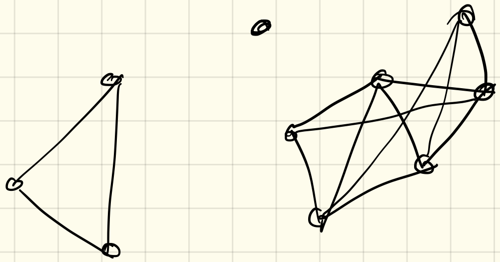
Today: Minimum Spanning Trees (MSTs)

Recall:

Dfn: A *tree* is a maximal acyclic graph, always with $n-1$ edges.

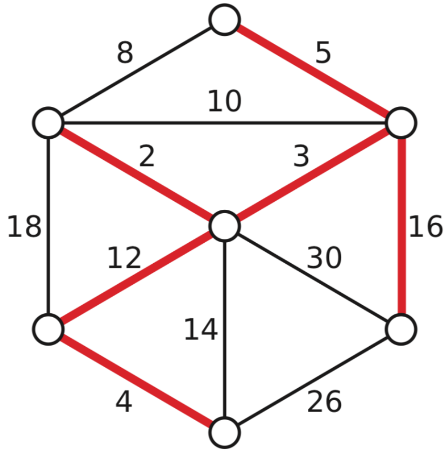
(DFS + BFS can both be used to get trees.)

Dfn: A *component* of a graph is a maximal connected subset of G.



Problem: Minimum Spanning Tree

Find a set of edges which connects all vertices & is as small as possible.



A weighted graph and its minimum spanning tree.

Applications: Obvious?

Note : Not a shortest path tree!

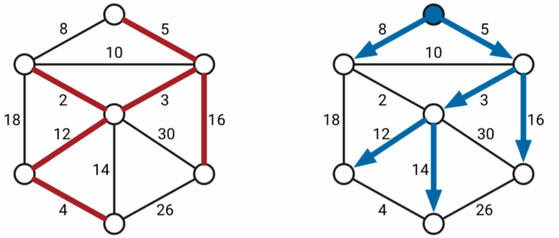


Figure 8.2. A minimum spanning tree and a shortest path tree of the same undirected graph.

These track fundamentally different structures:

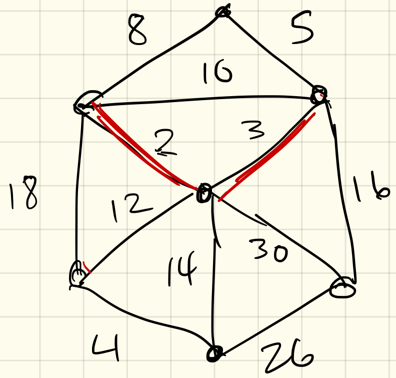
- Overall minimum weight vs distance to a particular goal.

So - different applications, depending on what you want to optimize!

High level idea for algorithm:

- We'll start by assuming edge weights are unique: so $w(e) \neq w(e')$ $\forall e, e' \in E$

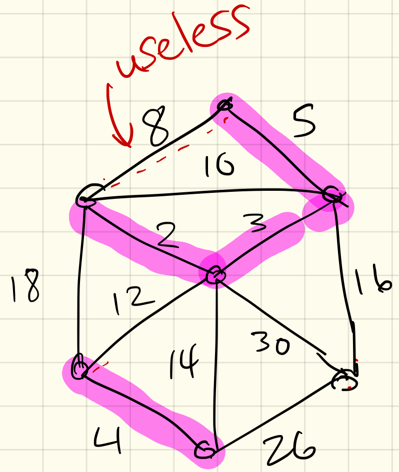
How to get started?



Pick smallest edge.

Intermediate stage

Now suppose we have a partial MST || a forest.

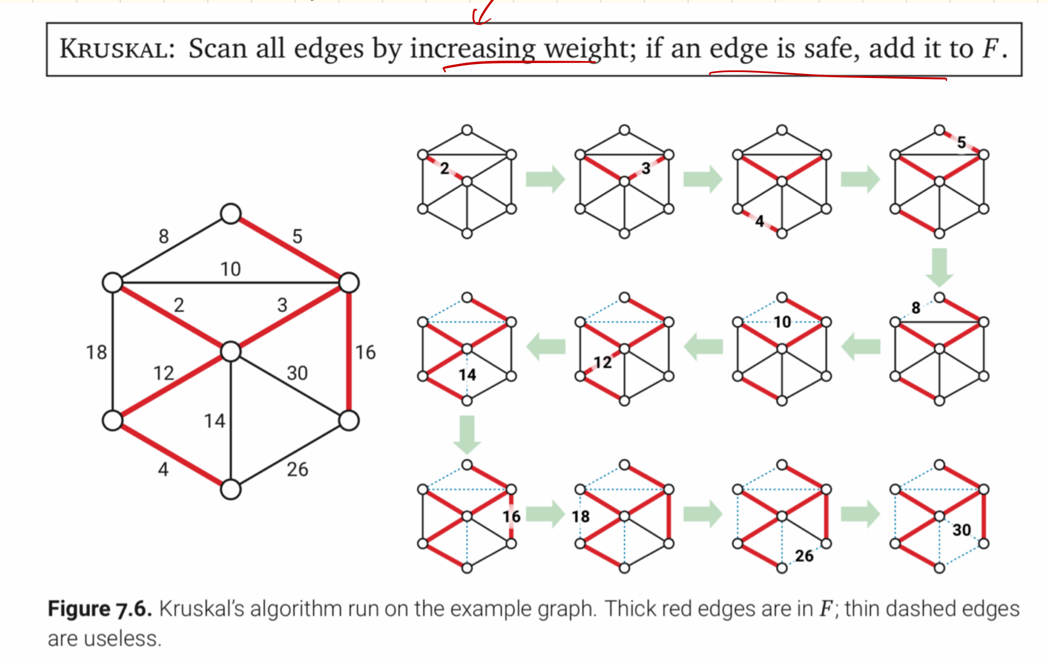


Classify edges:

- useless
- potential edges: connect 2 different components

Takeaway:

sort: $O(m \log m)$



KRUSKAL: Scan all edges by increasing weight; if an edge is safe, add it to F.

Figure 7.6. Kruskal's algorithm run on the example graph. Thick red edges are in F; thin dashed edges are useless.

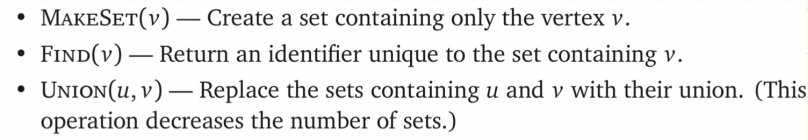
(Proof that it always works
↳ go take algorithms!)

Implementing:

Need to track <u>components</u> as we add edges.

(Zybooks called those vertexSets)

Really, need:



- MakeSet(v) — Create a set containing only the vertex v.
- Find(v) — Return an identifier unique to the set containing v.
- Union(u, v) — Replace the sets containing u and v with their union. (This operation decreases the number of sets.)

This is called <u>union-find</u> data structure.

↳ Ties to sets.
(more next week)

But - with just these 3 operations...

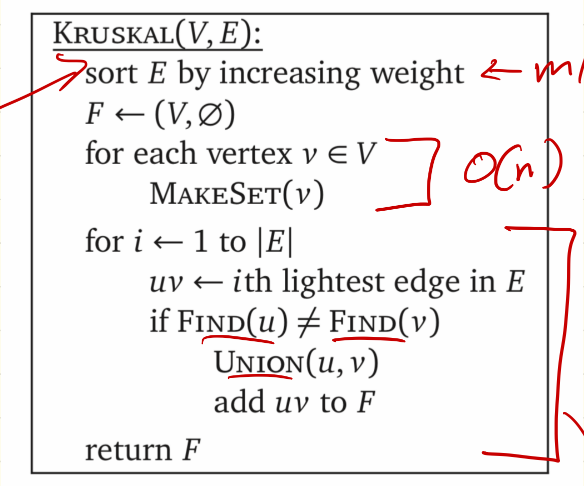
- Each vertex needs to "know" its component
- Initially, each vertex is its own → n labels
- When combining 2,
  - take smaller graph & relabel all of its vertices

    How? Model each component as a graph or tree, & do BFS/DFS

- Then, each time a component label changes, its set is ≥ twice as large.

  So: each label can change only $O(\log n)$ times.

Pseudocode:



```
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 $|E|$
        $uv \leftarrow i$th lightest edge in $E$
        if FIND($u$) $\neq$ FIND($v$)
            UNION($u, v$)
            add $uv$ to $F$
    return $F$
```

← m log m

O(n)

Runtime:

repeats $O(m) \overset{\shortparallel}{} O(|E|)$

$$O(m \log m + (n+m) \log n)$$

$$= O(m \log m)$$