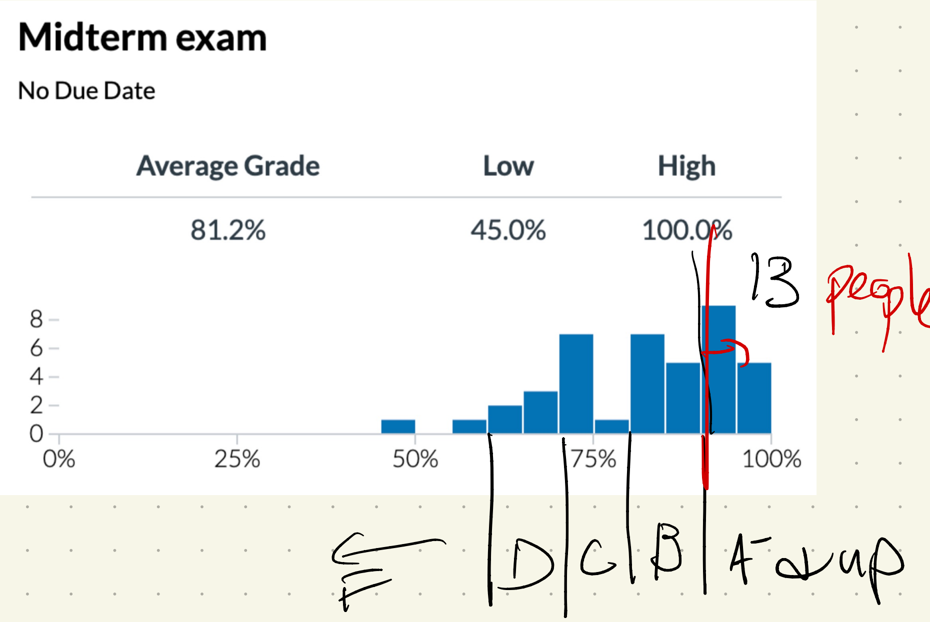
Algorithms - Spring '25

MSTs

Recap

- Midterm results:



Midterm exam

No Due Date

Average Grade	Low	High
81.2%	45.0%	100.0%

- Reminder:

  Only 20% of grade — so not over!

- HW this week - on paper

- Evaluations open

  ↳ close tonight

Minimum Spanning Trees

Goal: Given a weighted undirected Graph G, $w: E \to \mathbb{R}^+$ the weight function, find a spanning tree T of G that minimizes:

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

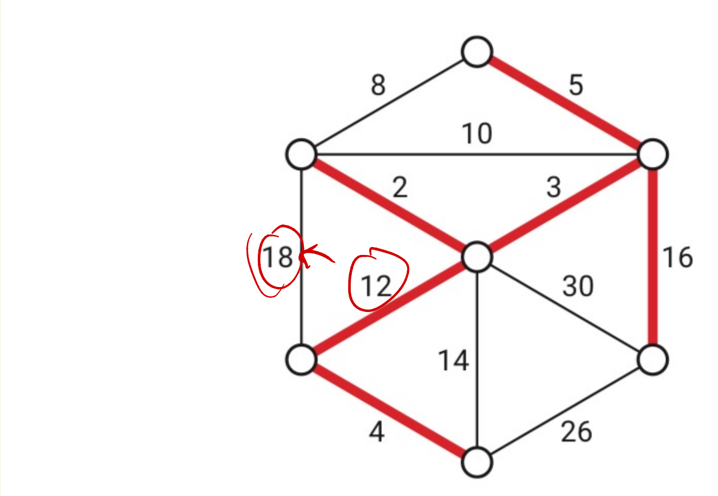


Figure 7.1. A weighted graph and its minimum spanning tree.

Assume MST is unique.

The magic truth of MSTs:

You can be SUPER greedy

Almost any natural idea will work!

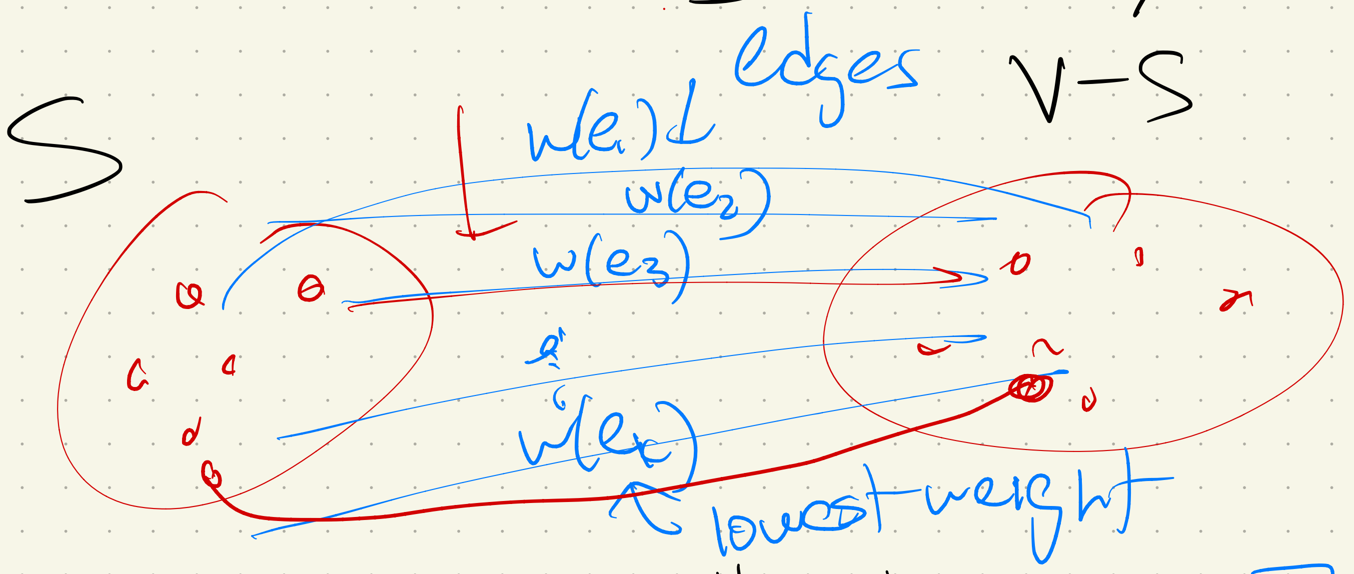
This is highly unusual, + there's a reason for it:

these are a (rare) example of something called a → matroid.

(Way beyond this class...)

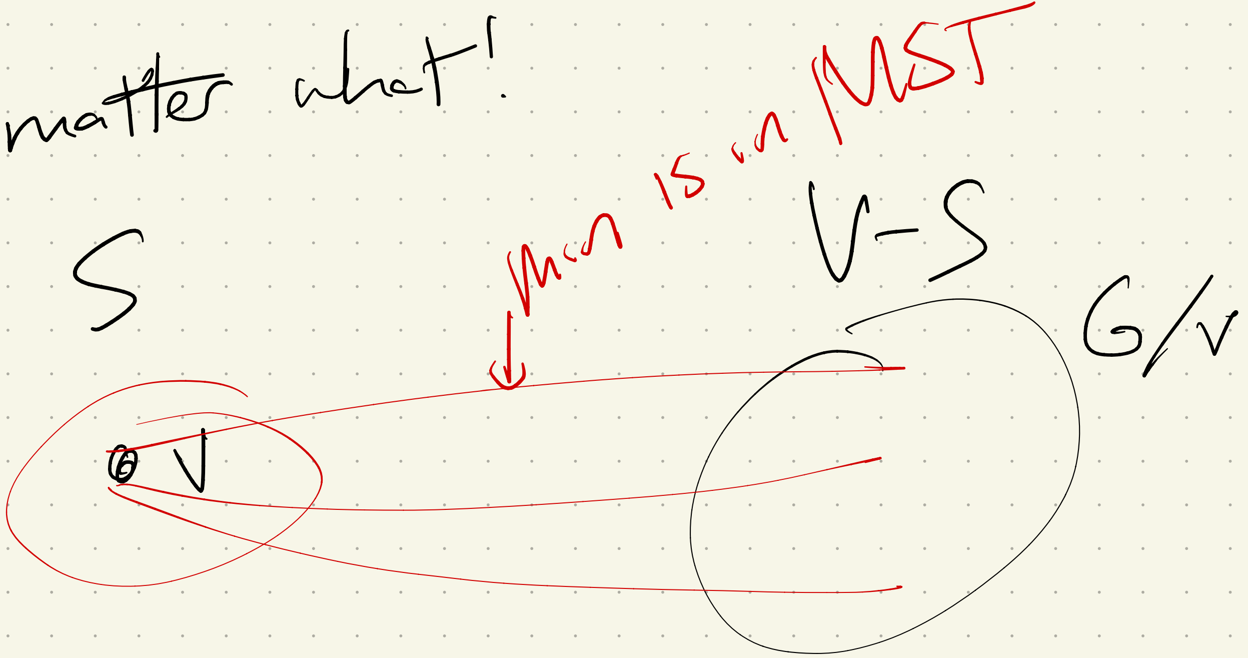
Key property:

Consider breaking G into two sets: S and V/S



The MST will always contain the lowest edge connecting the two sides.

No matter what!



Generic Algorithm:

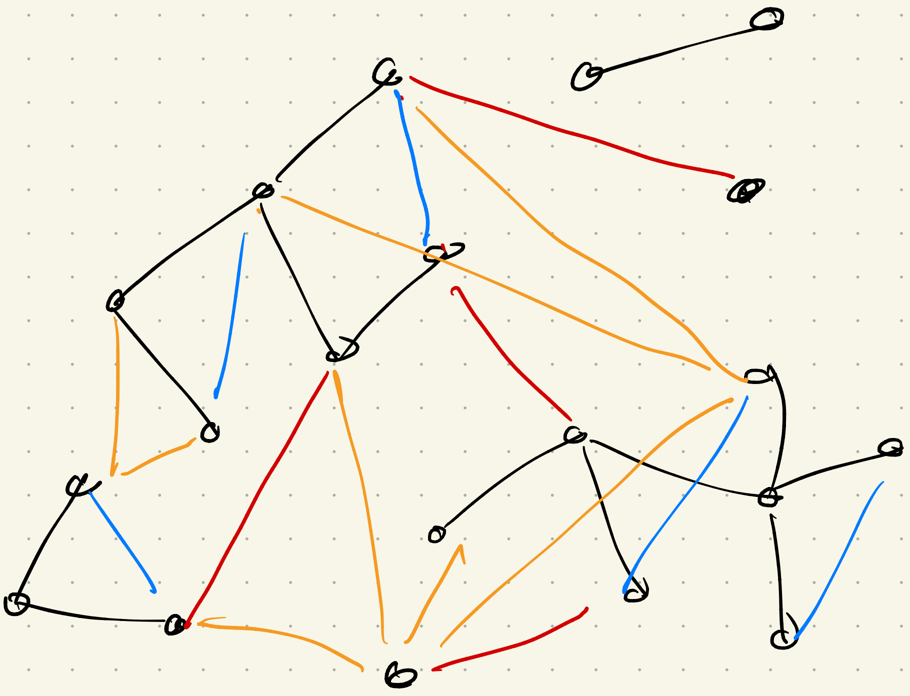
Build a forest: an acyclic subgraph.

Dfn: An edge is useless if it connects 2 endpts in same component of F

An edge is safe if it is minimum edge from some component of F to another.

also edges that are neither.

F:



So idea:

Add safe edges
until you get a tree

If everything isn't connected,
must have some safe
edge.

Why?

Use lemma: 2 comps (or more)
so make one of them = S.

Add it + recurse.

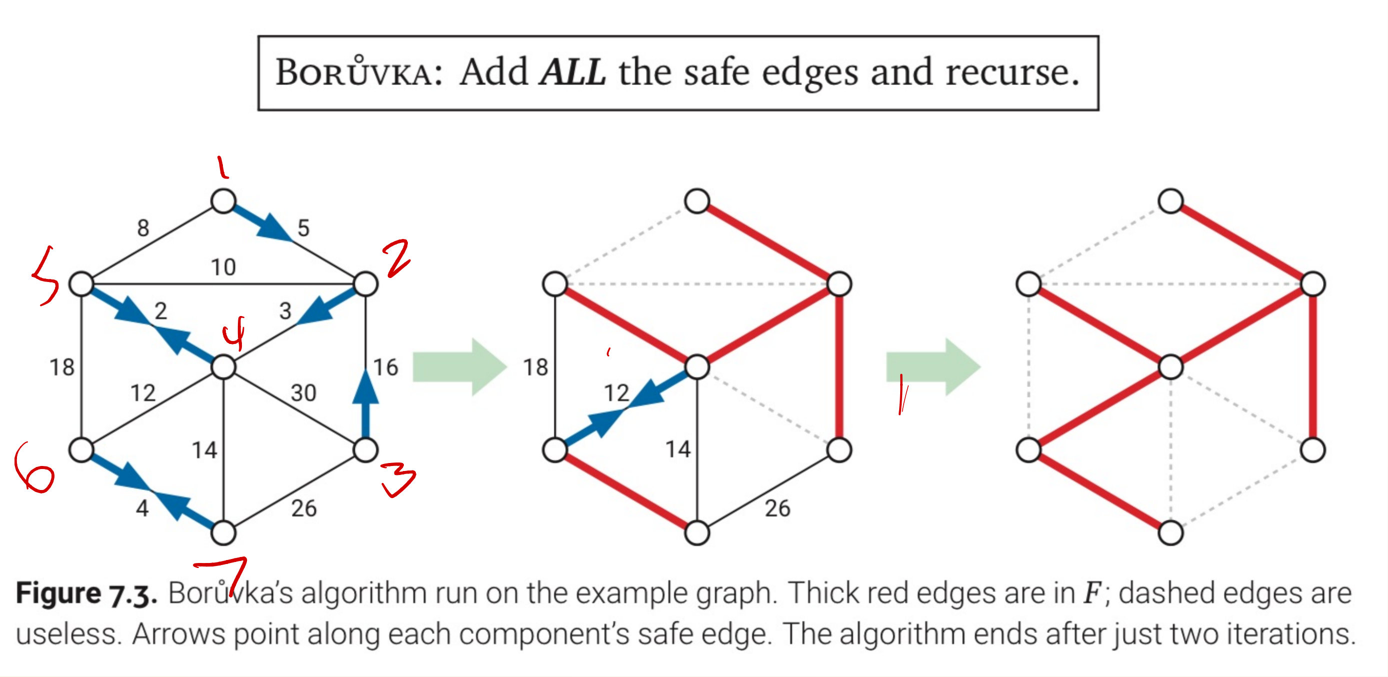
We'll see 3 ways:

① Find <u>all</u> safe edges.
Add them + recurse.

② Keep a single connected component.
At each iteration, add 1 safe edge.

③ Sort edges + loop through them.
If edge is safe, add it.

First one: (1926-ish)



BORŮVKA: Add ***ALL*** the safe edges and recurse.

Figure 7.3. Borůvka's algorithm run on the example graph. Thick red edges are in F; dashed edges are useless. Arrows point along each component's safe edge. The algorithm ends after just two iterations.

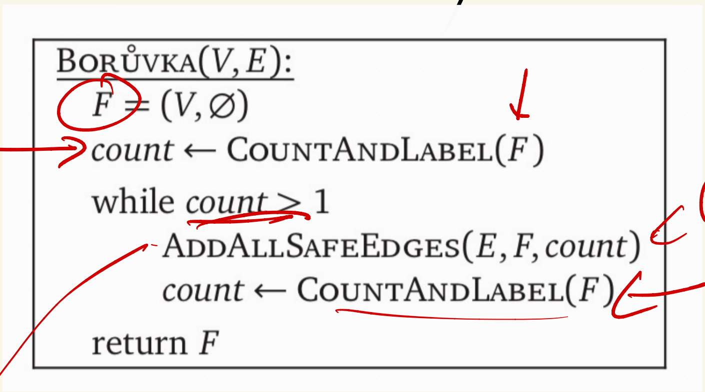
So we need to:

While more than 1 component:

- Track components
- Find all safe edges
- Add them

use WFS

More formally:



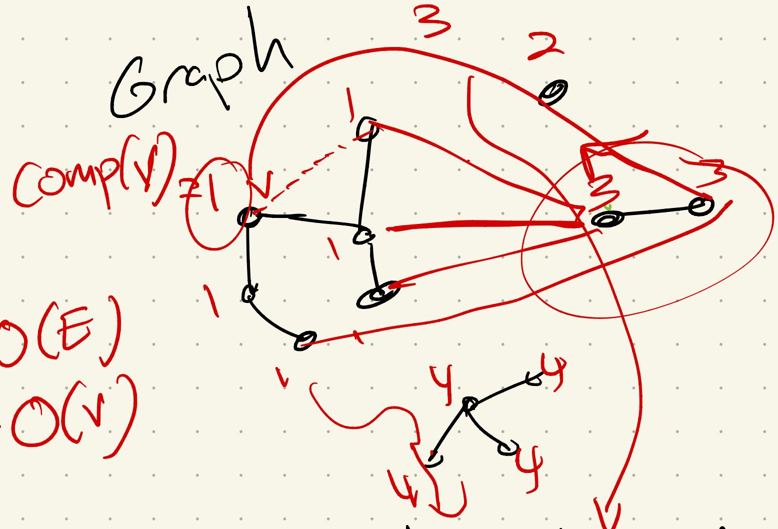
```
BORŮVKA(V,E):
    F = (V, ∅)
    count ← COUNTANDLABEL(F)
    while count > 1
        ADDALLSAFEEDGES(E, F, count)
        count ← COUNTANDLABEL(F)
    return F
```

O(V)

O(E)

O(V)

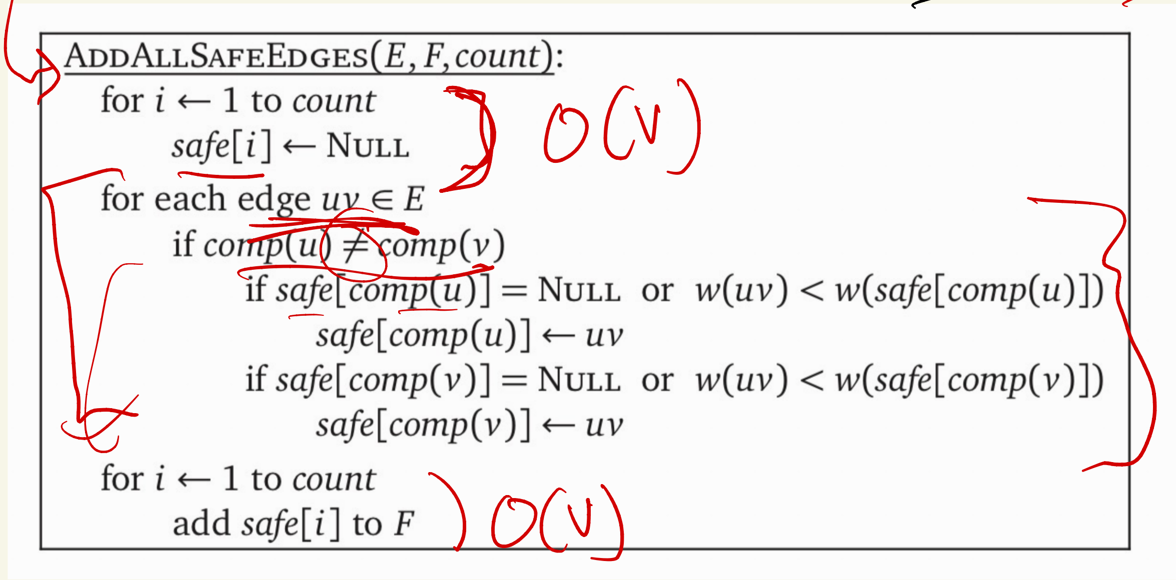
Graph



Safe:

∅	∅	∅	∅
1	2	3	4

count = 4



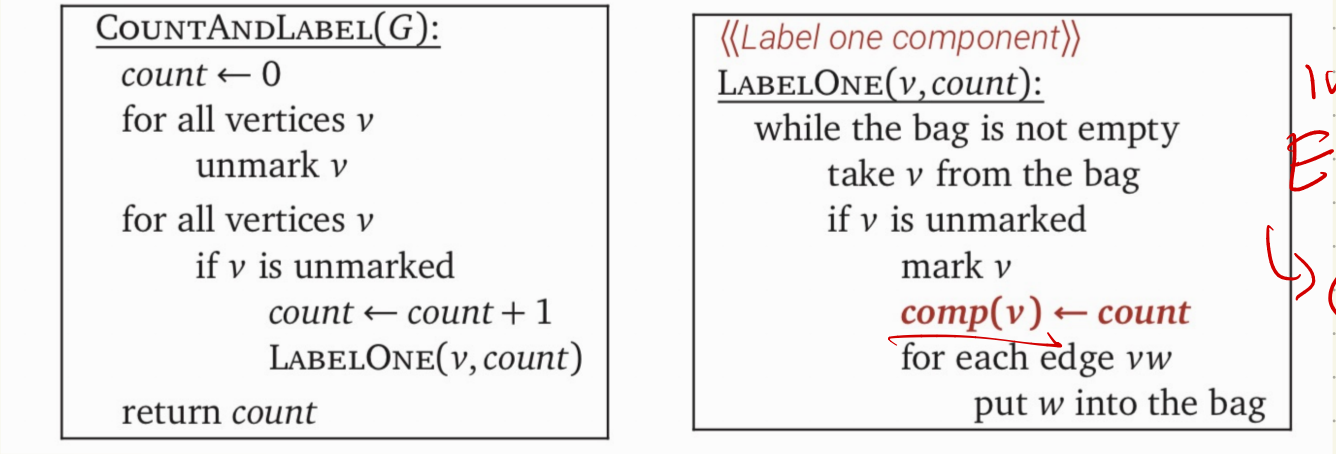
```
ADDALLSAFEEDGES(E, F, count):
    for i ← 1 to count
        safe[i] ← NULL
    for each edge uv ∈ E
        if comp(u) ≠ comp(v)
            if safe[comp(u)] = NULL or w(uv) < w(safe[comp(u)])
                safe[comp(u)] ← uv
            if safe[comp(v)] = NULL or w(uv) < w(safe[comp(v)])
                safe[comp(v)] ← uv
    for i ← 1 to count
        add safe[i] to F
```

O(V)

O(E)

O(V)

Uses WFS-variant from Ch 5: O(V+E)



```
COUNTANDLABEL(G):
    count ← 0
    for all vertices v
        unmark v
    for all vertices v
        if v is unmarked
            count ← count + 1
            LABELONE(v, count)
    return count
```

```
⟨⟨Label one component⟩⟩
LABELONE(v, count):
    while the bag is not empty
        take v from the bag
        if v is unmarked
            mark v
            comp(v) ← count
            for each edge vw
                put w into the bag
```

in forest

E ≤ V-1

O(V)

Correctness:

☆ - MST must have any safe edge (by lemma)

- We keep computing safe edges & adding
- Stop when #connected components = 1

⟹ Have the MST!

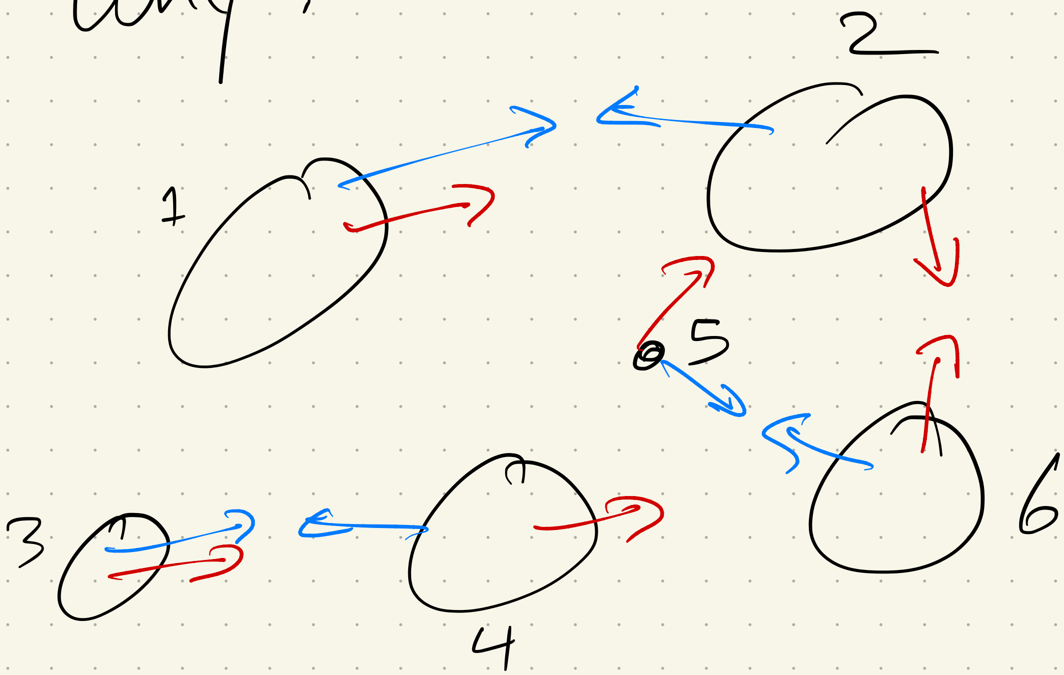
Run time:

A bit trickier!

Depends on how many safe edges we get.

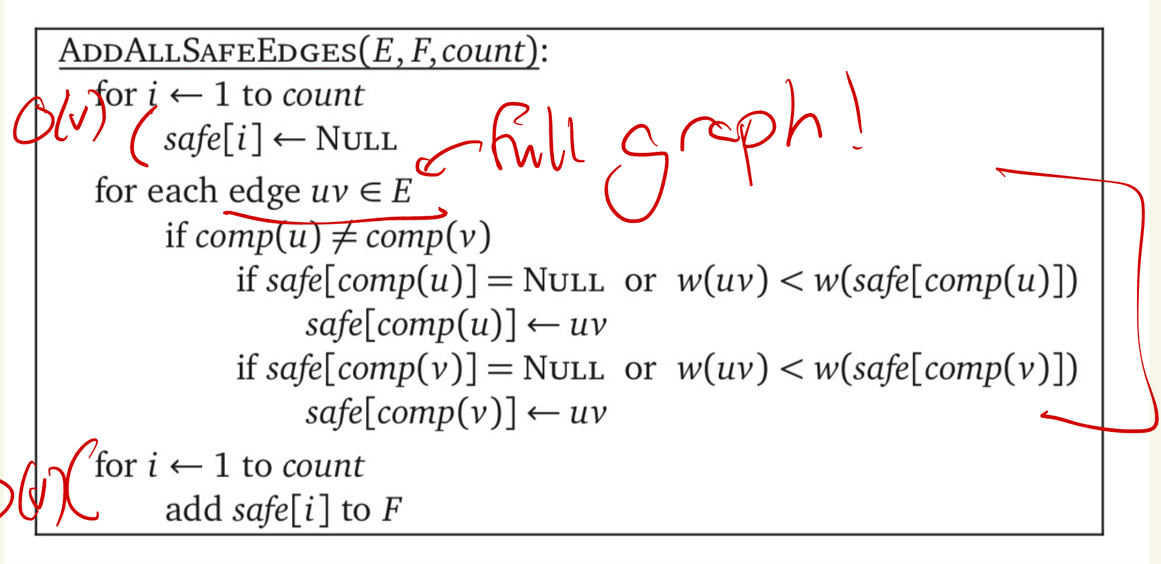
Claim: There are at least #components/2 safe edges each time.

Why?



each comp picks one edge.

So: runtime:



```
ADDALLSAFEEDGES($E, F, count$):
    for $i \leftarrow 1$ to $count$
        $safe[i] \leftarrow$ NULL
    for each edge $uv \in E$
        if $comp(u) \neq comp(v)$
            if $safe[comp(u)] =$ NULL  or  $w(uv) < w(safe[comp(u)])$
                $safe[comp(u)] \leftarrow uv$
            if $safe[comp(v)] =$ NULL  or  $w(uv) < w(safe[comp(v)])$
                $safe[comp(v)] \leftarrow uv$
    for $i \leftarrow 1$ to $count$
        add $safe[i]$ to $F$
```

$O(v)$

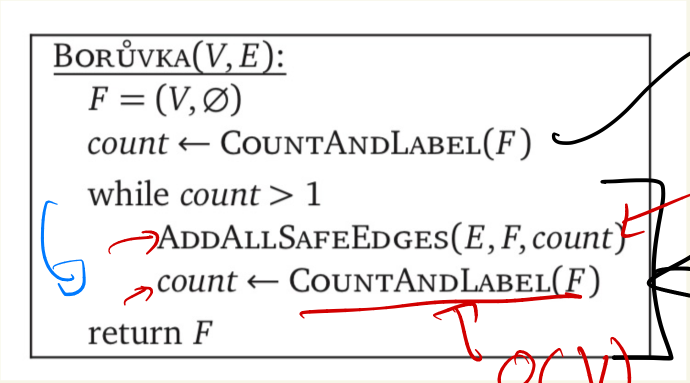
full graph!

$O(E)$

$O(v)$

↑ Looks at each vertex + edge in worst case:

$O(E+V) = O(V)$



```
BORŮVKA($V, E$):
    $F = (V, \varnothing)$
    $count \leftarrow$ COUNTANDLABEL($F$)
    while $count > 1$
        ADDALLSAFEEDGES($E, F, count$)
        $count \leftarrow$ COUNTANDLABEL($F$)
    return $F$
```

BFS/DFS on tree: $O(E)$

$O(V+E)$

$O(V)$

How many iterations?

$\log_2 V$ in worst case

Total: $O((V+E) \cdot \log_2 V)$

Prim's algorithm:

(really Jarník, we think)

Keep one spanning subtree.

While $|T| \neq V$:

    add next safe edge

JARNÍK: Repeatedly add T's safe edge to T.

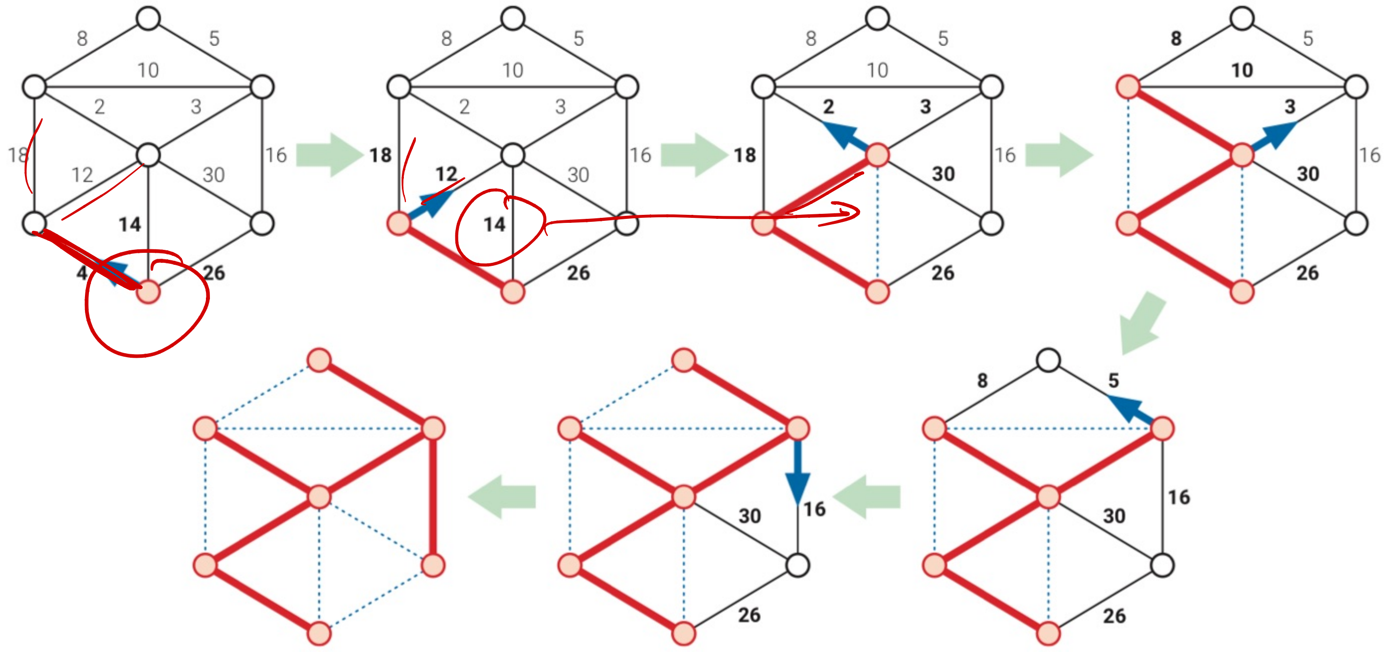


Figure 7.4. Jarník's algorithm run on the example graph, starting with the bottom vertex. At each stage, thick red edges are in T, an arrow points along T's safe edge; and dashed edges are useless.

Implementation:

From all edges going from $V(T)$ to $V(G)-V(T)$, add safe one.

↑ how??

Q: Which data structure?

priority queue

→ heap

runtime? $\log_2 n$

for insert or ExtractMin

Runtime:

loop V-1 times

each edge will be added (+ possibly removed) from heap

$$O(V + \underbrace{E \log E}_{E \log V}) \qquad E \le V^2$$

Aside:

Book goes over alternative —

uses a cooler data structure.

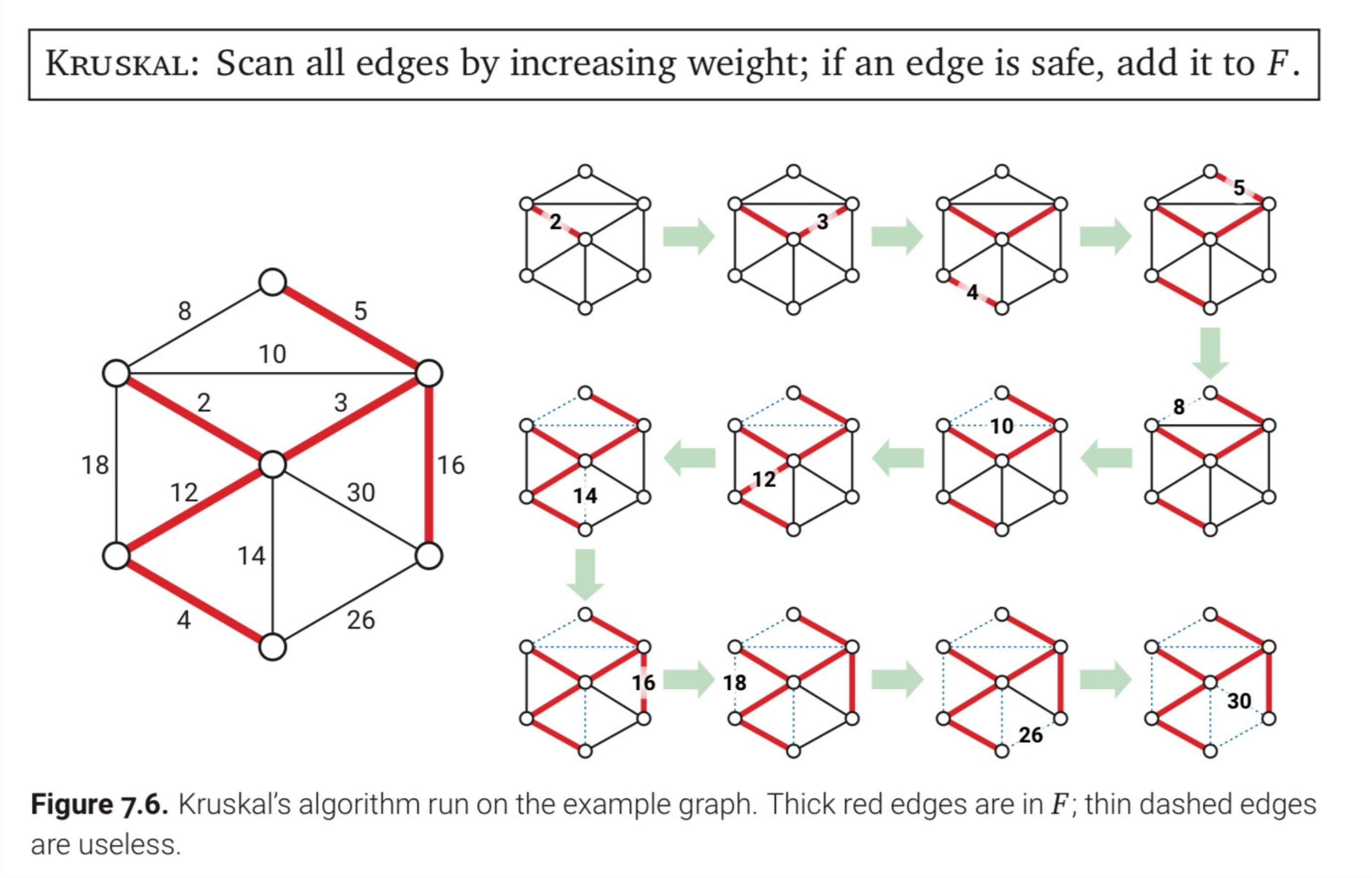
Comparison to Borůvka:

Faster (worst case), unless $E = O(V)$

But practically?

Borůvka usually wins

Kruskal's Algorithm :

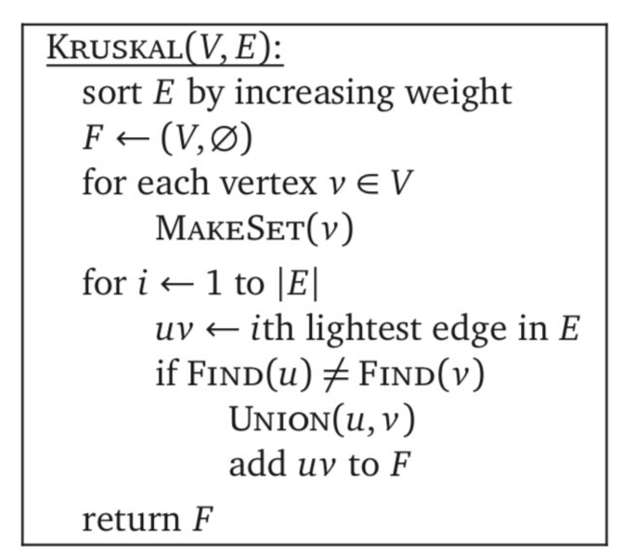


KRUSKAL: Scan all edges by increasing weight; if an edge is safe, add it to F.

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

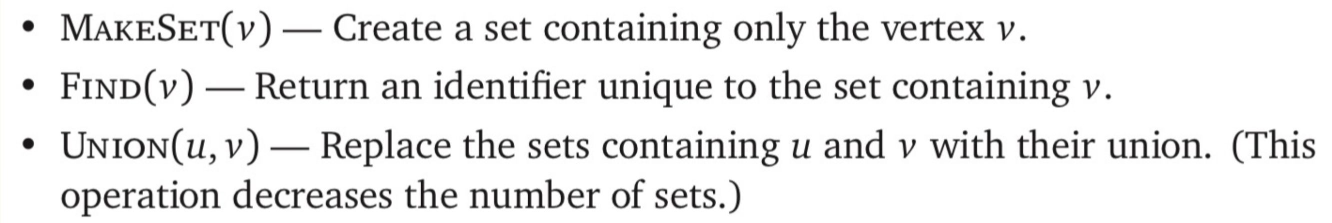
How to implement?

Algorithm :



```
KRUSKAL($V, E$):
    sort $E$ by increasing weight
    $F \leftarrow (V, \varnothing)$
    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 $F$
```

Data structure:
Union find



- 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.)

Comparison:

- Borurka: $O(E \log V)$

- "Prim": $O(E + V \log V)$

- Kruskal: $O(E \log V)$

- remember: worst case! (plus - hidden constants)
- also: $E \neq V$. Might have $E \approx V^2$, but also could be smaller.