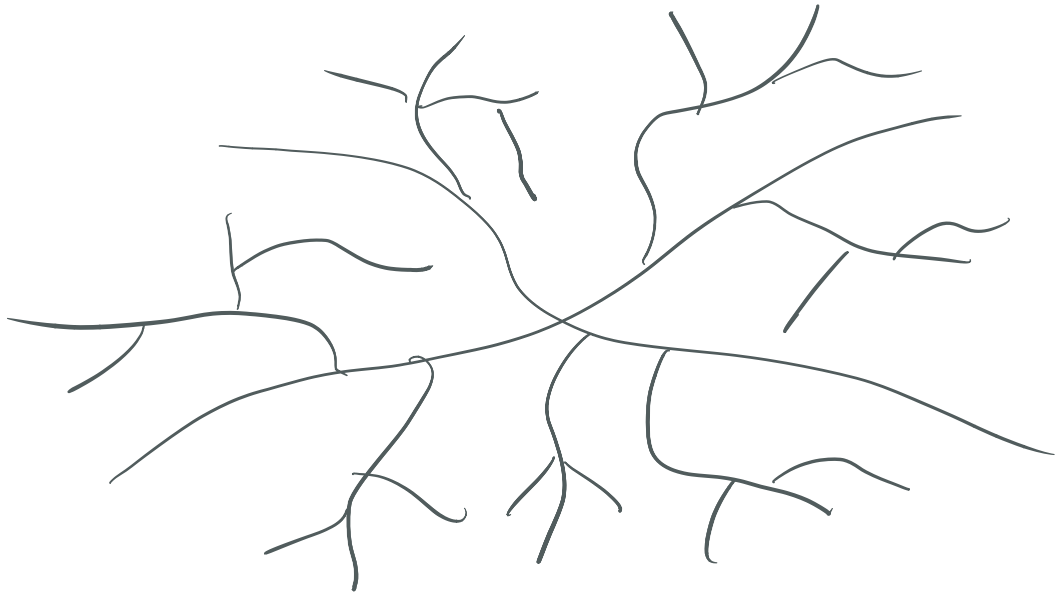
Minimum Spanning Trees

(by Kent Quanrud)



Input: Undirected graph $G=(V,E)$, edgeweights $w: E \to \mathbb{R}$

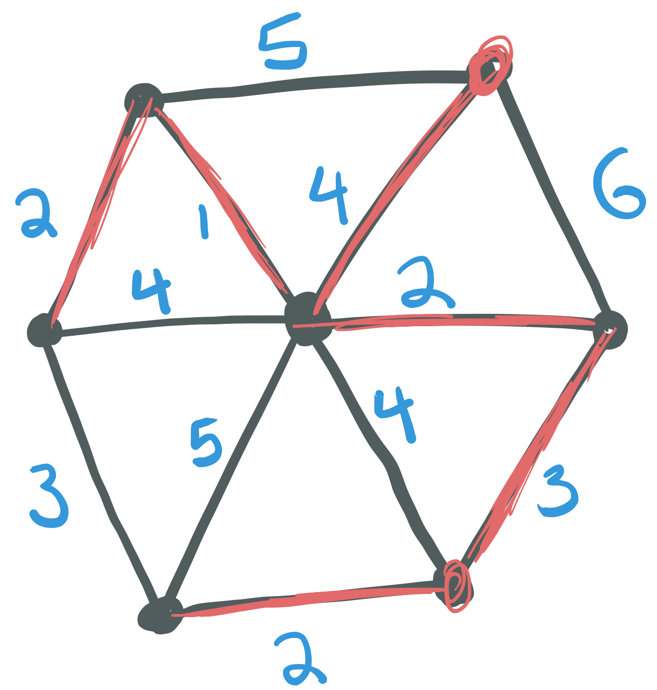
$\left[\begin{matrix} m = |E| \\ n = |V| \end{matrix}\right]$

A spanning Tree is a tree in G containing all of V (e.g., n-1 edges)

Goal: Compute the minimum weight spanning tree (abbr. MST) in G

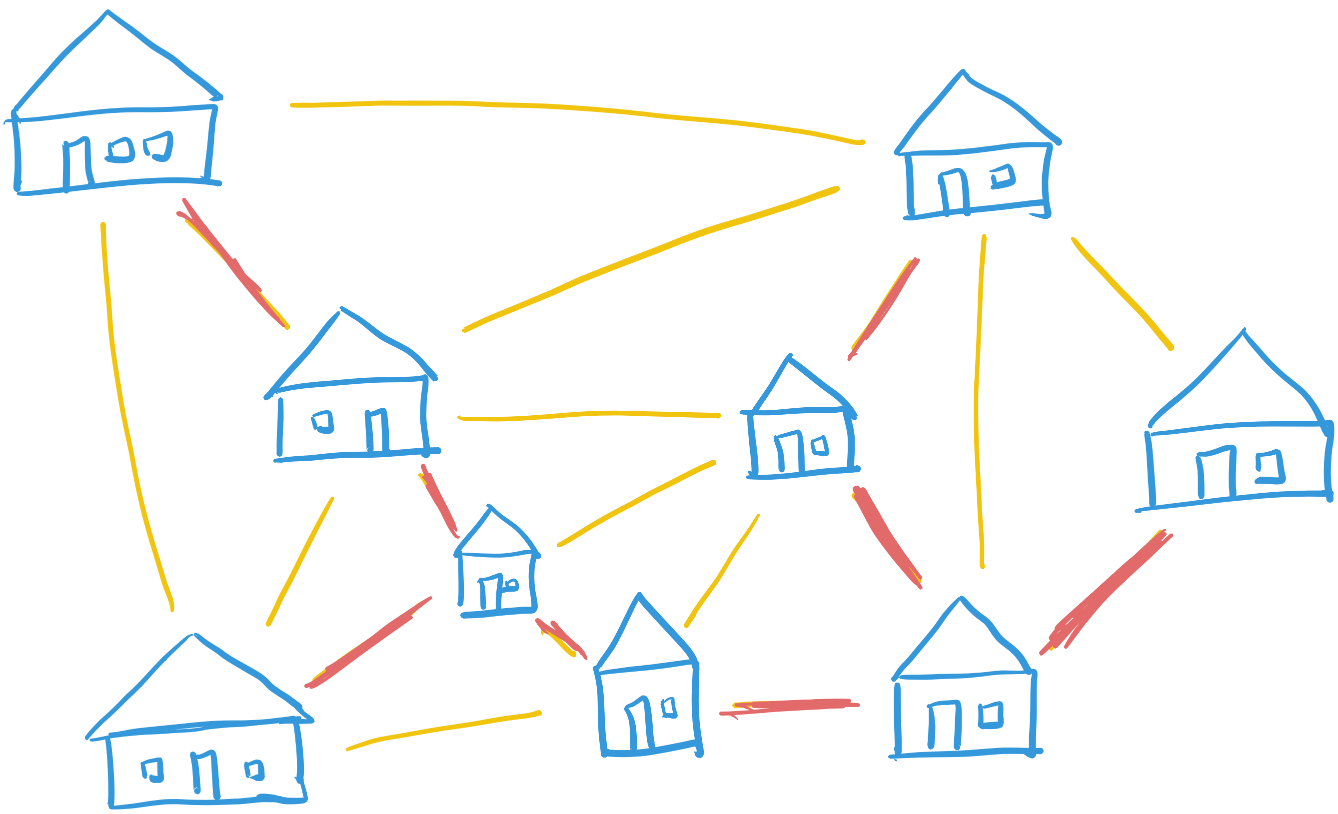
[weight of tree = sum of edge weights

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



Applications

- Network design
- Approximations for harder problems like traveling salesman
- deep connections across theory, comb. OPT



GOAL: Connect town w/ minimum amout of electrical wire

Preliminary obs:

- min-ST w/r/t w = max-ST w/r/t $-w$
- we can assume (WLOG) that all edge weights are distinct by breaking ties consistently.

e.g. number edges $e_1, e_2, \dots, e_m$

e_i "weighs less than" e_j if

$w(e_i) < w(e_j)$

or $w(e_i) = w(e_j)$ and $i < j$.

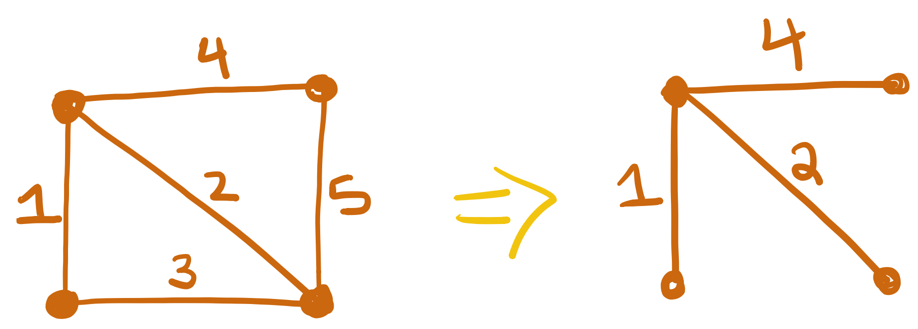
Outline:

(1) Describe 4 different algorithms

(2) Prove all of their correctneess at the same time

(3) Discuss data structures and pin down the running time

Running example:



Prim's algorithm

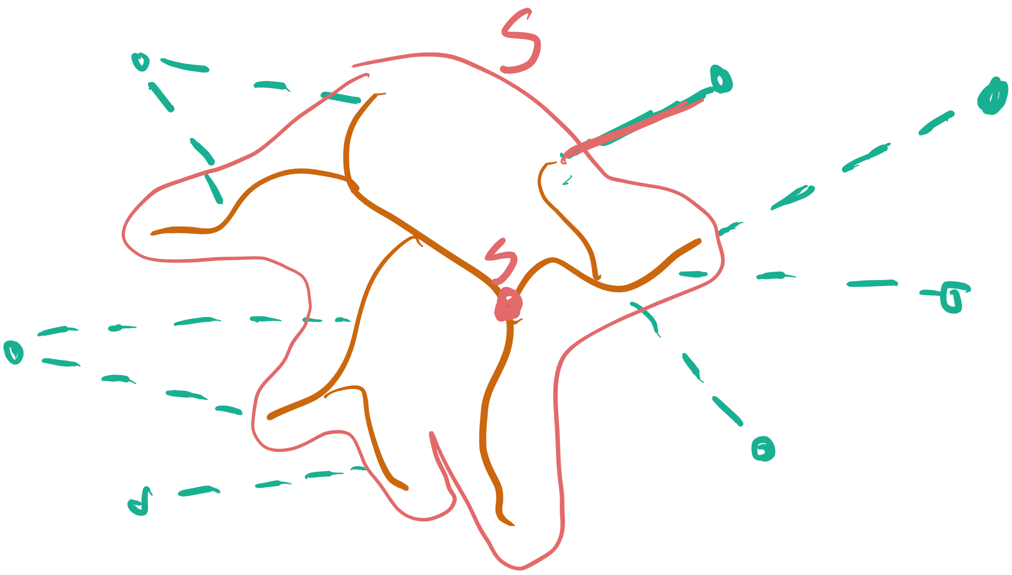
repeatedly adds the minimum weight edge w/ one endpoint in T

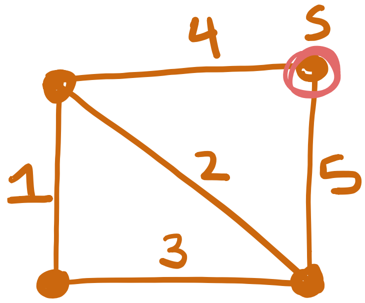
$\text{PRIM}(G=(V,E),\ w: E \to \mathbb{R})$

1. $T \leftarrow \emptyset$, $S \leftarrow \{s\}$ for some vertex $s \in V$
2. while $S \neq V$
   a. $e \leftarrow$ min weight edge crossing S
   b. $T \leftarrow T + e$, $S \leftarrow S \cup \{e\}$ $\quad e = \{u, v\}$
3. return T

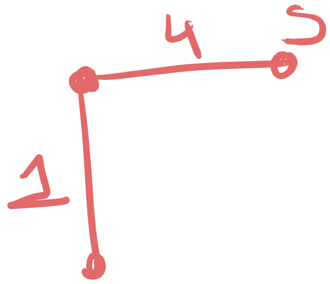
$e = \{u, v\}$
$u \in S$, $v \notin S$, $S \leftarrow S + v$

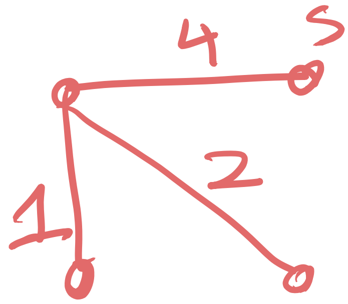
// Key invariant: T is a tree connecting S




4
s
1
2
5
3


4
s


4
s
1


4
s
2
1

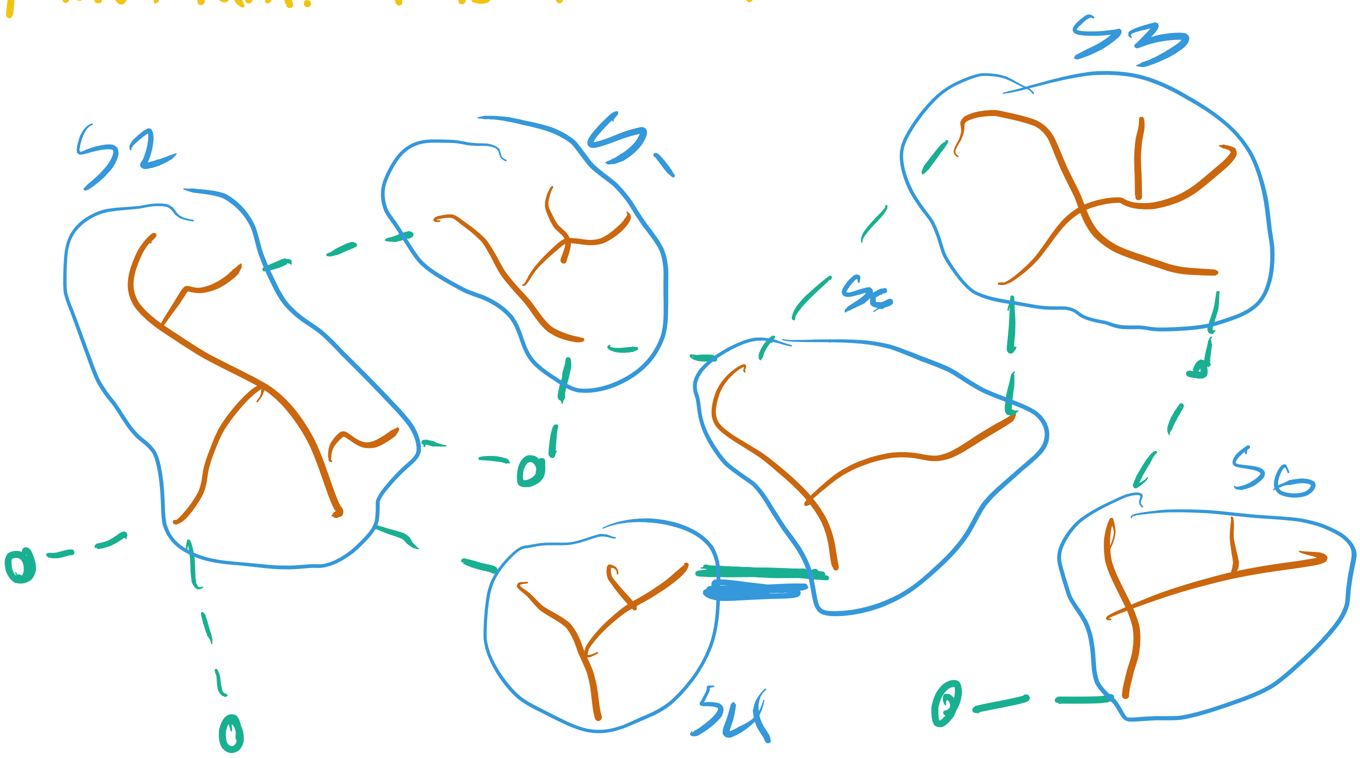
Kruskal's algorithm

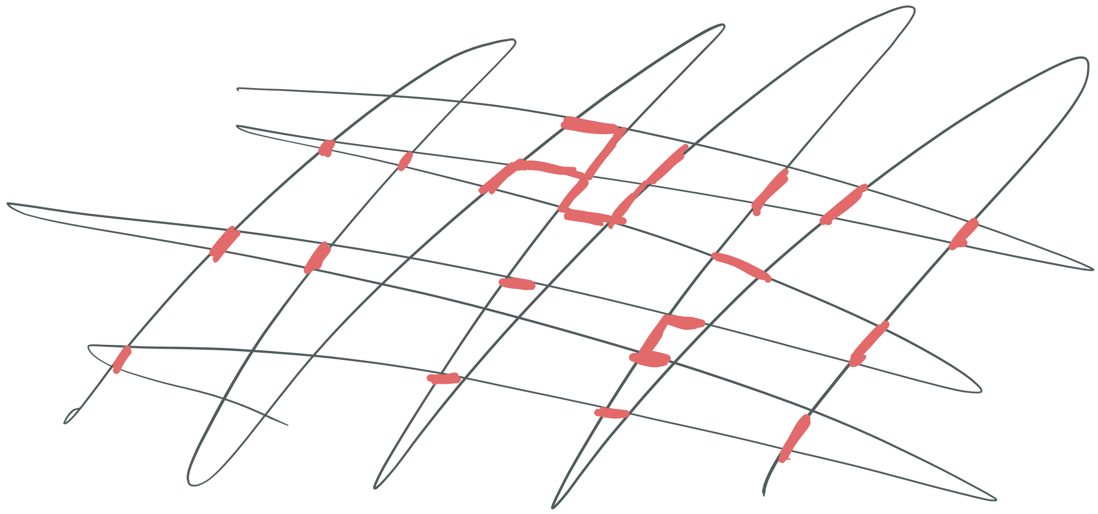
repeatedly adds the minimum weight edge that does not create a cycle

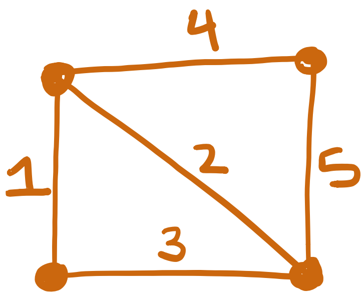
KRUSKAL$(G=(V,E), w)$

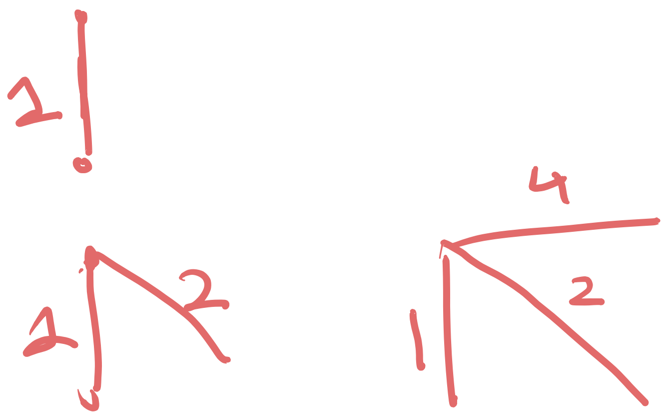
1. $T \leftarrow \emptyset$
2. while T does not span all of V
   a. $e \leftarrow$ min weight edge in $E \setminus T$
      st $T+e$ is acyclic
   b. $T \leftarrow T+e$
3. return T

// Key invariant: T is a forest






4
1
2
5
3


1
1
2
4
1
2

Borůvka:

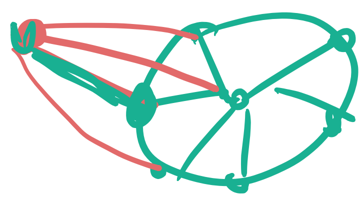
grow all connected components w/ min-weight crossing edge in parallel

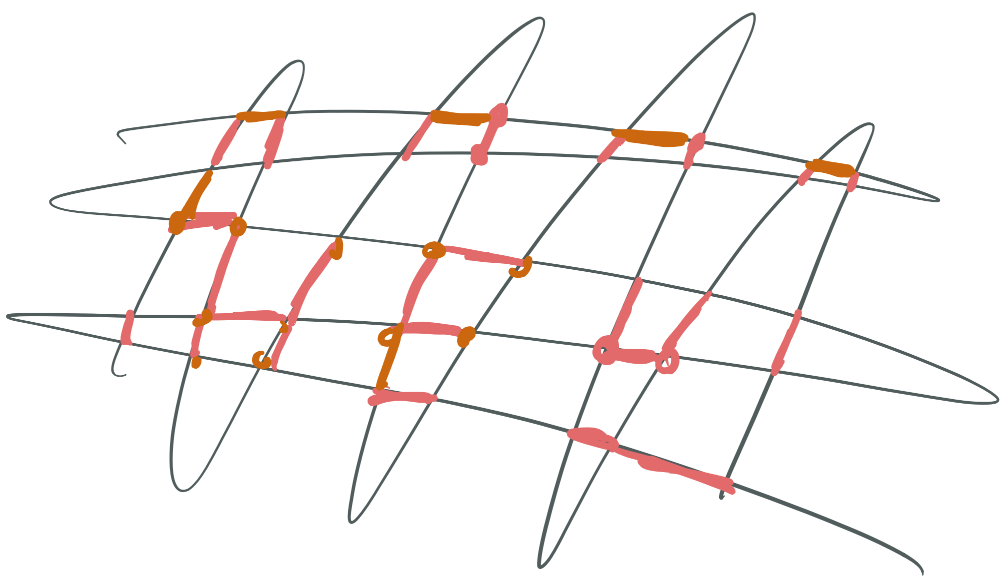
Borůvka:

1. $T \leftarrow \emptyset$
2. while T is not spanning
   - A. $U \leftarrow \emptyset$
   - B. for each component $S \subset V$ w/r/t T
     - i. $e \leftarrow$ min weight edge w/ 1 endpoint in S

safe edge

     - ii. $U \leftarrow U + e$
   - C. $T \leftarrow T \cup U$
3. return T





[simulation]

reverse-delete

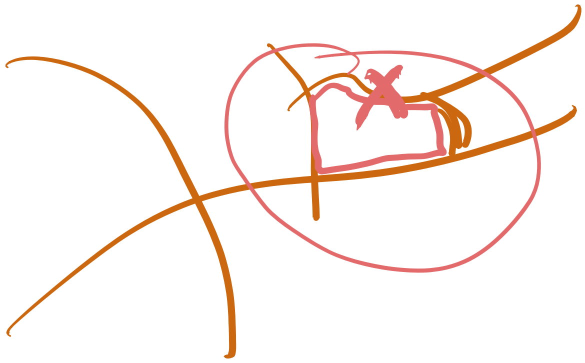
repeatedly removes max-weight edge that does not disconnect graph

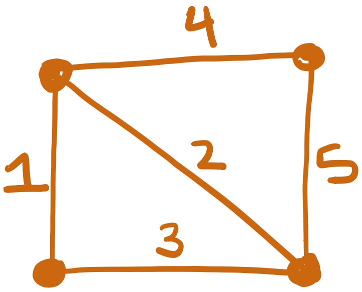
REVERSE-GREEDY($G=(V,E)$, w):

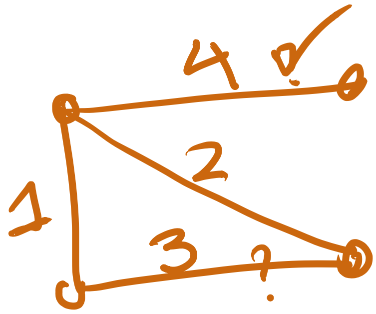
1. $T \leftarrow E$
2. while $E \neq \emptyset$
   A. $e \leftarrow$ max weight edge in E
   B. $E \leftarrow E - e$
   C. if $T - e$ is connected
      i. $T \leftarrow T - e$
3. return T

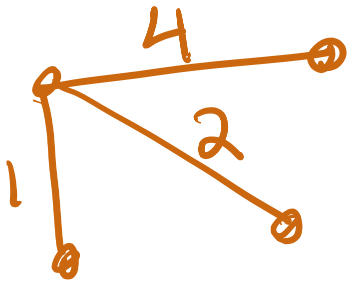
// Key invariant: T is a connected subgraph spanning V

T

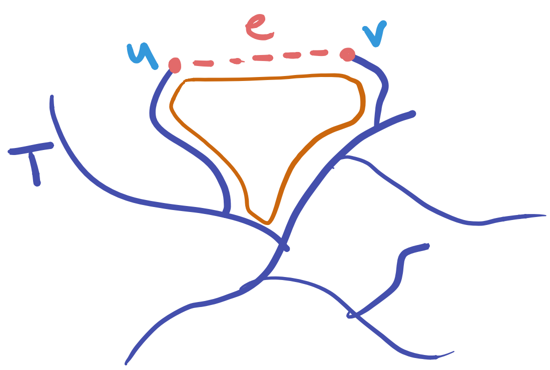








On to proofs!



Lemma let T be a spanning Tree, $e \in E \setminus T$

Then $T+e$ contains a unique cycle, which contains e.

Proof let $e = \{u, v\}$. since T is a spanning, there is a unique path P from u to v in T. $C = P + e$ is our cycle

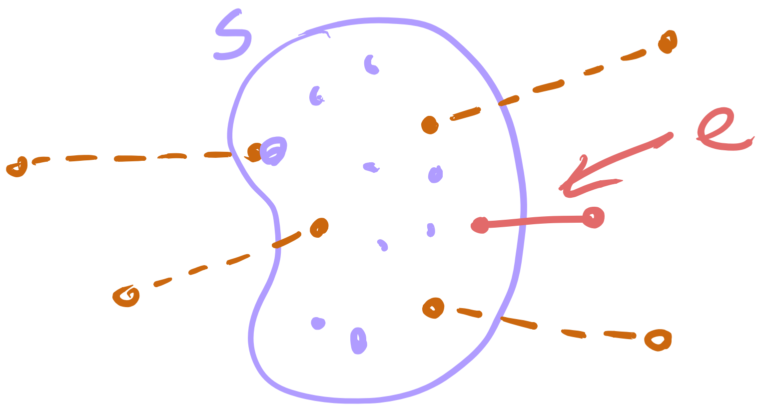
Suppose there is another cycle

$D \subseteq T + e$.

- $e \in D$
- $D - e$ is a path in T

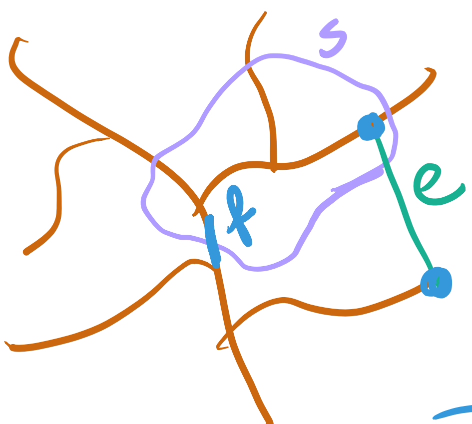
P is the unique path, $D = P + e$

Safe edges An edge e is safe if there is a set of vertices S such that e is the min-weight edge w/ one endpoint in S



Lemma Any safe edge e is in every MST

Proof Let e be a safe edge w/r/t (say) S.
Let T be an MST s.t. $e \notin T$



$w(e) < w(f)$

$T' = T - f + e$

$\overline{w}(T') = \overline{w}(T) - w(f) + w(e)$
$< \overline{w}(T)$

let C be the unique ~~cy~~ cycle in $T+e$

$C - e$ is a path starting in S

ending $V \setminus S$.

Lemma (Suppose edge weights are distinct)

Then Prim/Kruskal/Boruvka compute spanning trees where every edge is safe

Proof

(add min weight cross S)

(inspection)

Theorem (suppose edge weights are distinct)

There are exactly $n-1$ safe edges and they form the unique MST

Proof

① first Lemma (safe $\subseteq$ MST)

$\Rightarrow \le n-1$ safe edges

② (safe $\supseteq$ Kruskal, Prim's, ...)

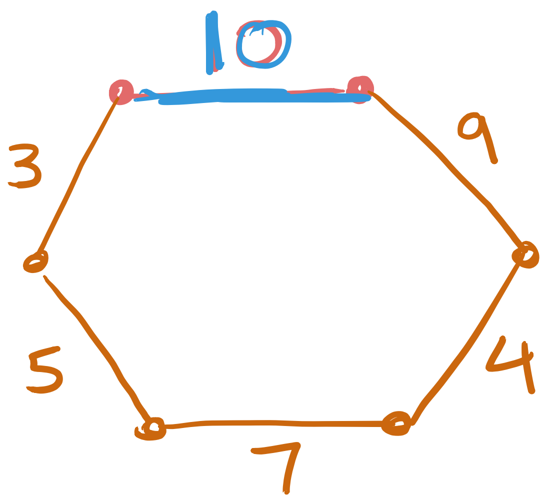
$\Rightarrow \ge n-1$ safe edges

Corollary Prim's, Kruskal's, Borůvka's algos all return MST's.

Proof $\square$ ✓

Unsafe edges

an edge e is unsafe if there is a cycle C where e is the uniquely maximum weight edge

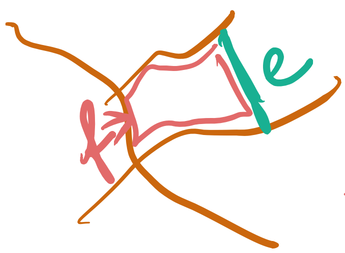


Lemma (Suppose distinct edge weights).

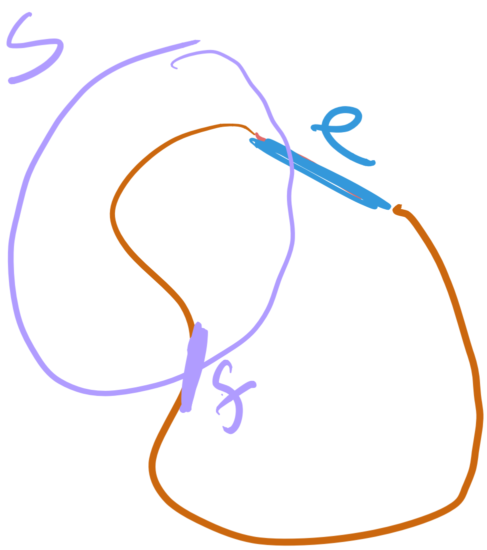
All edges are either safe or unsafe

Proof suppose e is not safe. let T be the MST. $e \notin T$. let C be the cycle in $T+e$.

$w(e) < w(f)$ for some $f \in C - e$, then $T - f + e$ has smaller weight



suppose e is safe, and C is a cycle containing e.



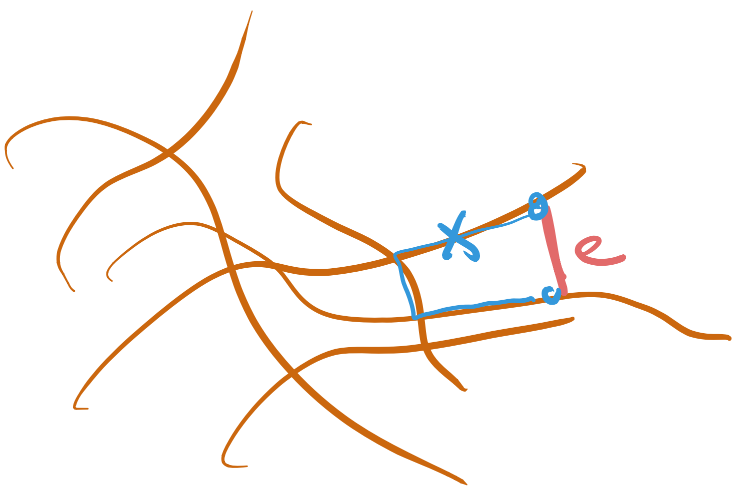
$w(e) \leq w(f)$

Lemma Let T be a connected subgraph, and $e \in T$ the max weight edge*. Then e is an unsafe edge.

* s.t. $T-e$ is still connected.

Proof Since $T-e$ is connected,

T contains a cycle C containing e.



then e is max weight on cycle $\Rightarrow$ unsafe.

Corollary

Reverse-Delete returns the MST

Implementation

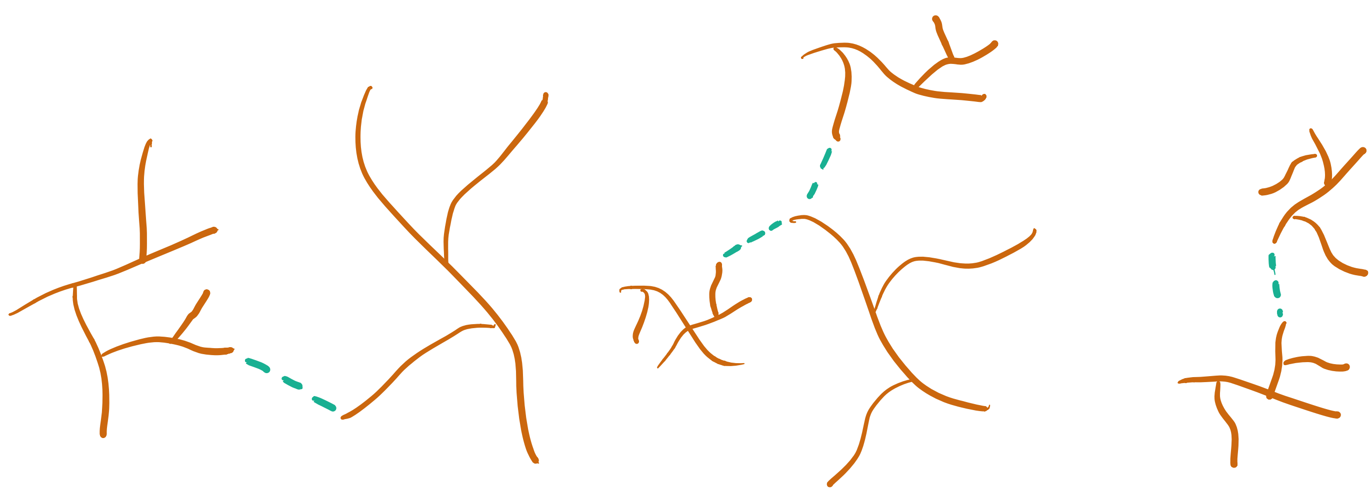
Borůvka	$O(m \log n)$
Kruskal	$O(m \log n)$
Prim	$O(m + n \log n)$

Borůvka:

grow all connected components w/ min-weight crossing edge in parallel

Borůvka:

1. $T \leftarrow \emptyset$
2. while T is not spanning
   A. $U \leftarrow \emptyset$
   B. for each component $S \subset V$ w/r/t T
      i. $e \leftarrow$ min weight edge w/ 1 endpoint in S
      ii. $U \leftarrow U + e$
   C. $T \leftarrow T \cup U$
3. return T



Borůvka running time

// adds edges crossing each component in parallel

- Each round halves # connected comp.

  $\Rightarrow O(\log n)$ rounds

- Each round we look at each edge, pick out one edge per component

  $\Rightarrow O(m)$ per round

$\Rightarrow O(m \log n)$ total

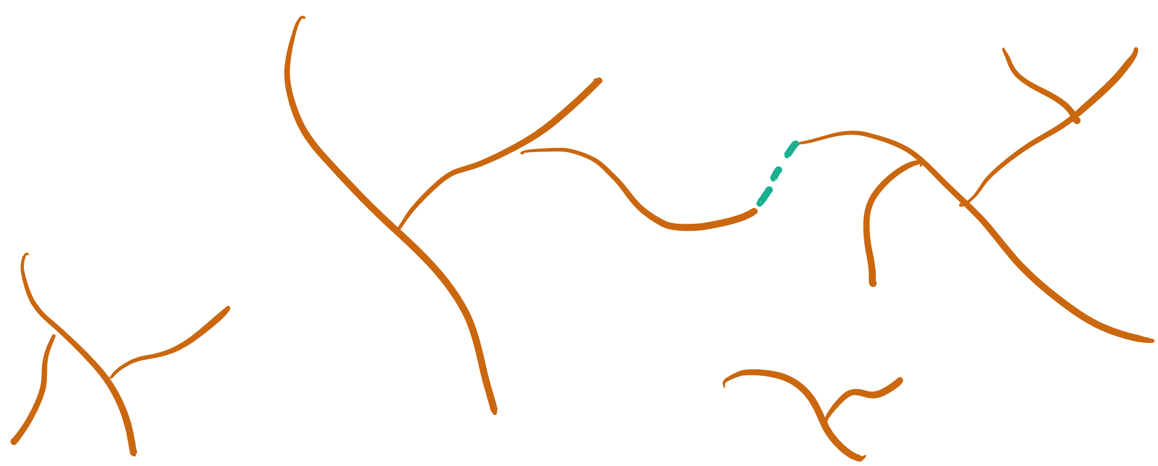
Kruskal's algorithm

repeatedly adds the minimum weight edge that does not create a cycle

KRUSKAL$(G=(V,E), w)$

1. $T \leftarrow \emptyset$
2. while T does not span all of V
    a. $e \leftarrow$ min weight edge in $E \setminus T$
       st $T+e$ is acyclic
    b. $T \leftarrow T+e$
3. return T

// Key invariant: T is a forest



Kruskal (refactored)

1. $T \leftarrow \phi$
2. for each $e = \{u,v\}$ in increasing order of $w(e)$
   A. if u,v are in diff. components of T
      i. $T \leftarrow T + e$
3. return T

We need to

(a) maintain connected components of T

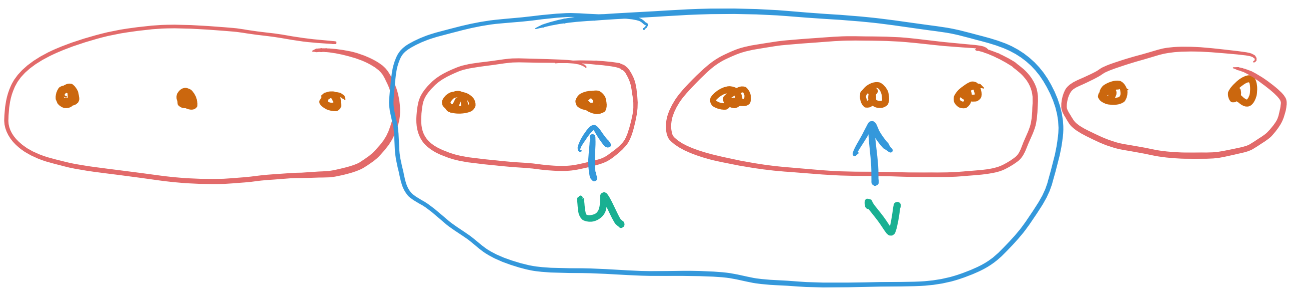
(b) quickly decide if 2 vertices are in same component

Union-Find data structure

maintains collection of disjoint sets s.t.

Union(u,v): combine the set containing u and the set containing v

Together(u,v): returns True iff u and v are in the same set



$O(\alpha(n))$

union-find can be implemented very fast (almost $O(1)$ amortized per op.).

Bottleneck of Kruskal is sorting

$\Rightarrow O(m \log n)$

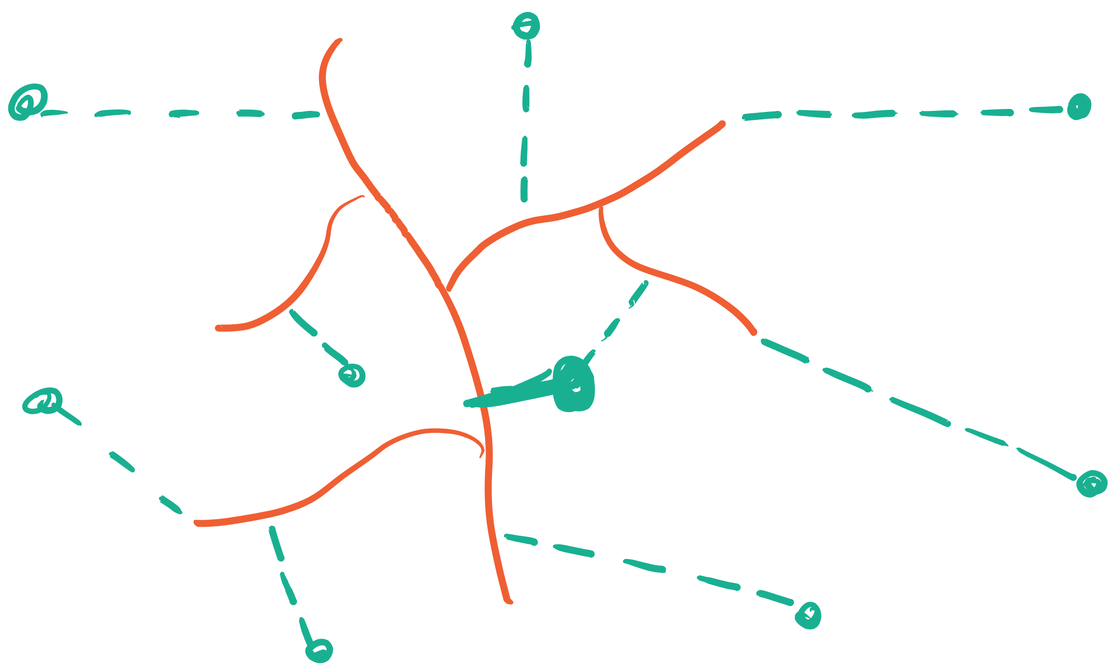
Prim's algorithm

repeatedly adds the minimum weight edge w/ one endpoint in T

PRIM($G=(V,E)$, $w: E \to \mathbb{R}$)

1. $T \leftarrow \emptyset$, $S \leftarrow \{s\}$ for some vertex $s \in V$
2. while $S \neq V$
   a. $e \leftarrow$ min weight edge crossing S
   b. $T \leftarrow T+e$, $S \leftarrow S \cup \{e\}$
3. return T

// Key invariant: T is a tree connecting S



Need: quickly identify nearest vertex outside the tree to the tree

Priority queue data structure

- insert(k,p): insert key k w/ priority p
- decrease(k,p'): decrease the priority of a key k (already in the queue) to a smaller priority p'
- extract-min: remove and return the key w/ the minimum priority

For Prim's algo:

keys = vertices not in the tree

priority = min weight of any edge from vertex to tree

		Fibonacci Heap
$O(n)$	insertions	$O(1)$
$O(n)$	extract-min	$O(\log n)$
$O(m)$	decrease-key	$O(1)$ amortized

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