MINIMUM SPANNING TREES

$G = (V, E)$ undirected, connected graph
$w : E \to \mathbb{R}$ weight function

} Throughout today's lecture

Ex 1) V, cities, E road network
2) V pins of circuit that should be wired together to be electronically equivalent
E pairwise cost for pins

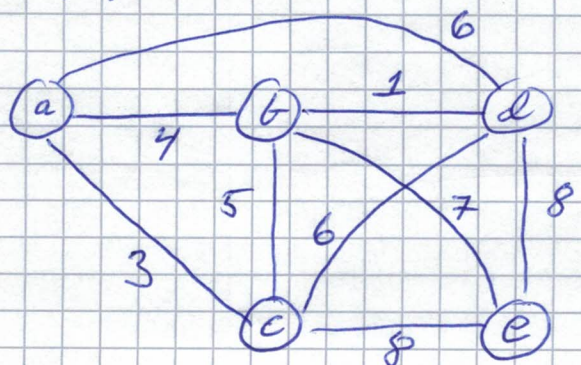
Want to find
1) smallest total length of roads to keep clear in case of snowstorm, say
2) cheapest wiring

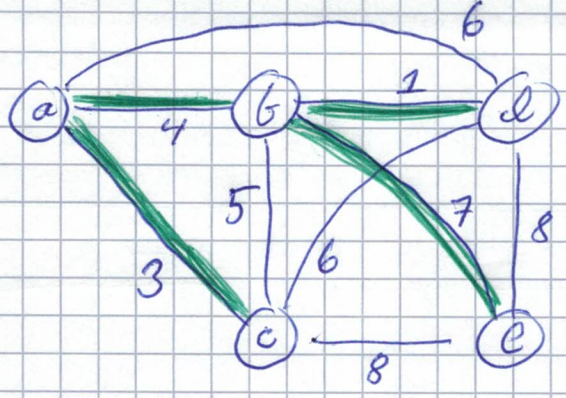
MINIMUM SPANNING TREE (MST)

Spanning tree T for G such that
$w(T) = \sum_{(u,v) \in E(T)} w(u,v)$ minimal

Actually minimum-weight spanning tree, but drop the word "weight"

Ex What is a minimum spanning tree for this graph?





Minimum spanning trees need NOT be unique.

Will build MSTs using GREEDY APPROACH:
Do what seems best locally at the moment

Greedy algorithms do not give optimal results in general, but here we can show that sequence of locally optimal choices give globally optimal result

General idea:
- Maintain subset of edges $A \subseteq E$ that is part of some MST
- At each step, add edge (u,v) that maintains this invariant, i.e., $A \cup \{(u,v)\}$ also part of some MST

Such edge (u,v) is a SAFE EDGE for A

GENERIC-MST (G, w)

```
A := ∅
while ( A not spanning tree)
    find edge (u,v) safe for A
    A := A ∪ {(u,v)}
return A
```

Suppose $|V| = n$, $|E| = m$

How many iterations in while loop?

$n-1$, because any tree on n vertices contains $n-1$ edges.

At all times, maintain $A \subseteq$ some MST

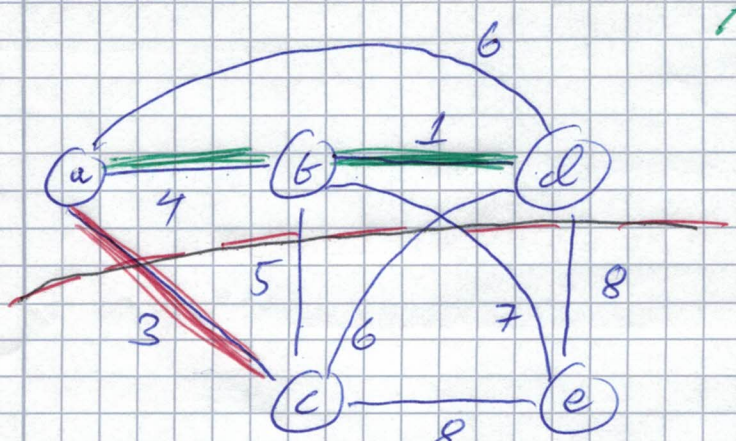
When $|A| = n-1$, the edges in A form MST.

Also note that always possible to add safe edge by definition, since always $A \subseteq$ some MST but A not yet tree

But how to find safe edges?

DEFINITIONS

- CUT $(S, V \setminus S)$: partition of vertices V
- Edge (u,v) CROSSES cut if $u \in S$, $v \in V \setminus S$
- Cut $(S, V \setminus S)$ RESPECTS $A \subseteq E$ if no edges in A cross the cut
- A LIGHT EDGE crossing a cut is a smallest weight edge crossing a cut (need not be unique)



$A = \{(a,b), (b,d)\}$

CUT $(\{a,b,d\}, \{c,e\})$ respects A

(a,c) is LIGHT EDGE for this cut

THEOREM 1

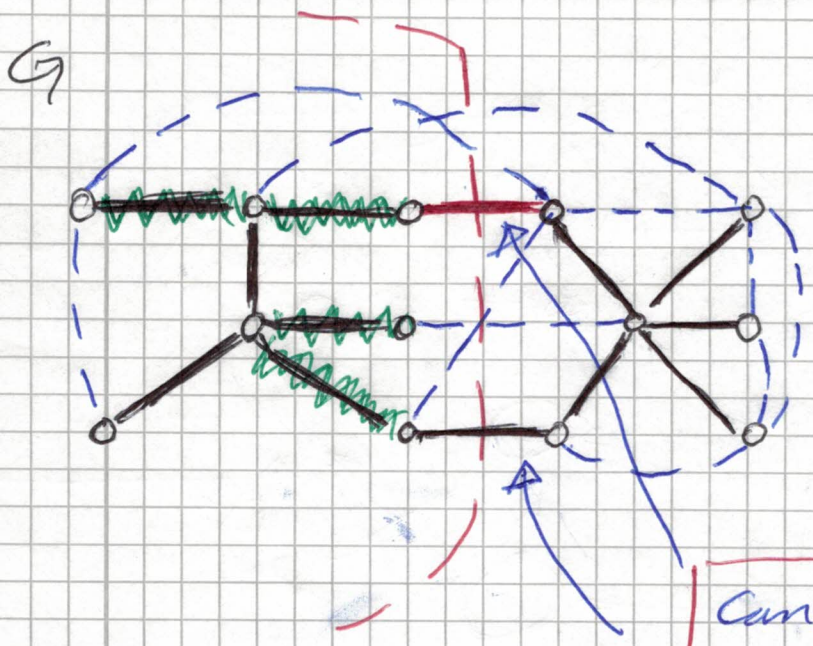
Let $G=(V,E)$ connected, undirected graph
$w: E \to \mathbb{R}$ weight function
Let $A \subseteq E$ be such that A is included in some MST
Let $(S, V \setminus S)$ cut that respects A
Let (u,v) light edge crossing $(S, V \setminus S)$
THEN (u,v) is safe for A

Proof Fix MST T including edges in A.
If (u,v) edge in T, then done, so suppose not.

Plan: Find edge (x,y) in T. Swap with (u,v). Argue still MST

u & v connected in T by (unique) path $P: u \rightsquigarrow v$
Add (u,v) to get cycle $C: v \rightsquigarrow u \to v$
Edge (u,v) crosses cut, so at least one more edge in C crosses cut
Fix any such edge (x,y)
Since (x,y) crosses cut, $(x,y) \notin A$ (respectfulness)
Remove edge (x,y) and add edge (u,v) to get T'. T' has $|V|-1$ edges and is connected (since T was connected)
[Prove this!]
Hence T' is tree.
Since (u,v) & (x,y) both cross cut and (u,v) light
$w(u,v) \leq w(x,y)$ Hence
$w(T') = w(T) - w(x,y) + w(u,v) \leq w(T)$
so T' also MST

∎



— Edges in T
Edges in A
Safe edge
Cut
other edges in G

Can swap these two edges since REDEDGE is LIGHT

A is always acyclic
Collection of trees — FOREST
A safe edge connects two subtrees in forest to form new subtree

COROLLARY 2

Let $A \subseteq E$ included in some MST
Let $C = (V_C, E_C)$ be connected component (i.e., subtree) in forese $G_A = (V, A)$
If (u,v) is light edge connecting C to some other component, then (u,v) safe for A.

Proof The cut $(V_C, V \setminus V_C)$ respects A
(u,v) is a light edge for this cut
Apply Theorem 1 ∎

PRIM'S ALGORITHM

Start in arbitrary root vertex r

Grow tree rooted in r.

In every step, add light edge to currently disconnected vertex

Will talk about implementation with HEAPS next week

Use **priority queue** for vertices v

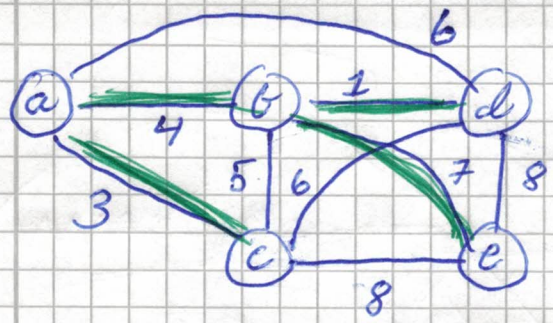
$v.key$ = min weight of edge connecting v to current tree (or ∞)

$v.\pi$ = parent of v in tree

Set A will be $A = \{(v, v.\pi) \mid v \in V \setminus (Q \cup \{r\})\}$

```
MST-PRIM(G, w, r)
    for v ∈ V(G)                          } n times
        v.key := ∞                        } O(1) time
        v.π   := NIL                      } O(n)
        v.marked := FALSE                 }
Q := BUILD-MIN-HEAP(V)        — O(n)
    Q.DECREASE-KEY(r, 0)      O(log n)
    while (NOT(Q.EMPTY))      n times
        u := Q.EXTRACT-MIN    — O(log n)
        u.marked := TRUE      — O(1)      } O(m)
        for v ∈ N(u)          — total of O(|E|) times
            if (NOT(v.marked) AND w(u,v) < v.key)
                v.π := u                  O(1)
                Q.DECREASE-KEY(v, w(u,v)) O(log n)
```

At termination MST edges are $\{(v, v.\pi) \mid v \in V \setminus \{r\}\}$



○ min queue element taken care of next

Key values in queue

a	b	c	d	e	
(0)	∞	∞	∞	∞	Initialize starting in a
0	4	(3)	6	∞	Dequeue a b.π := a ; key 4 c.π := a ; key 3 d.π := a ; key 6
0	(4)	3	6	8	Dequeue c EDGE (a,c) e.π := c ; key 8
0	4	3	(1)	7	Dequeue b EDGE (a,b) d.π := b ; key 1 e.π := b ; key 7
0	4	3	1	(7)	Dequeue d EDGE (b,d)
					Dequeue e EDGE (b,e)

Correctness

Follows from Theorem 1 + generic algorithm
We grow a tree from starting vertex
Always add light edge connecting this tree to new vertex

Time explx (recall $|V|=n$, $|E|=m$)

Initialization: $O(n)$

while loop: n times

$O(\log n)$ time except loop over adjacency list

Time in adjacency list for loop

total of $2m$ iterations

each iteration $O(\log n)$ time

Summing up: $O(n) + O(n \log n) + O(m \log n)$

$= \boxed{O(m \log n)}$

[$m \geq n-1 = \Omega(n)$ since graph is connected]

Faster priority queue would give faster algorithm. In coming algorithm courses, you will see how to use FIBONACCI HEAPS to get time $O(m + n \log n)$

— clearly better for dense graphs ($m \gg n$)

KRUSKAL'S ALGORITHM

Maintain sets of subtrees = FOREST

Add least-weight edge that connects any two distinct trees in forest (T_1 & T_2)

Correctness: If (u,v) is lightest edge connecting any two trees, say T_1 & T_2, then

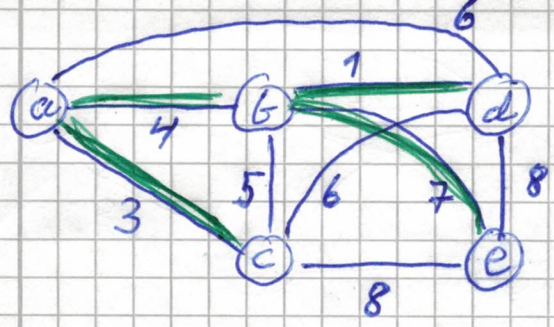
- $(V(T_1), V \setminus V(T_1))$ is a cut
- This cut respects all edges so far
- (u,v) is a light edge crossing this cut.

Formalizing this as an inductive proof, it follows that algorithm below must be correct.

MST-KRUSKAL (G, w)

```
A := ∅
for v ∈ V(G)                                   n times
    BUILD-SET({v})
L := all edges in E                            O(m)
SORT(L)   [in nondecreasing order by edge weight]   O(m log m)
for (u,v) in L in order                        m times
    if (NOT SAME-SET(u, v))
        A := A ∪ {(u,v)}                       O(1)
        JOIN-SETS(u,v)
return A
```

MST IX



Sort edges

	(b, d)	1
	(a, c)	3
	(a, b)	4
	(b, c)	5
	(a, d)	6
	(c, d)	6
	(b, e)	7
(c, e)	~~(a, d)~~	8
	(d, e)	8

Edge	Forest	
	{a} {b} {c} {d} {e}	
(b, d)	{a} {b, d} {c} {e}	Add (b, d)
(a, c)	{a, c} {b, d} {e}	Add (a, c)
(a, b)	{a, b, c, d} {e}	Add (a, b)
(b, c)	b & c in same set	——
(a, d)	a & d in same set	——
(c, d)	c & d in same set	——
(b, e)	{a, b, c, d, e}	Add (b, e)
~~(a, d)~~ (c, e)	c & e in same set	——
(d, e)	d & e in same set	——

To implement Kruskal efficiently, need data structures for (disjoint) sets

This is discussed in CLRS Chapter 21, which is not part of this course.

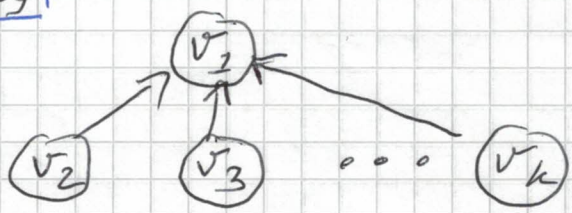
Basically, idea is as follows:

For each set, maintain tree with all edges pointing in direction of root.

Root is unique representative of set

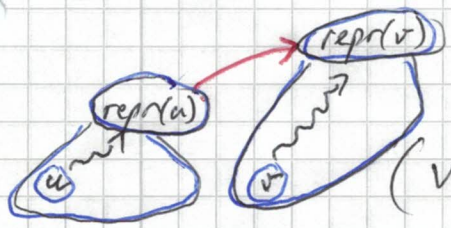
SAME-SET (u,v) checks if representative for u & v the same

BUILD-SET $(\{v_1, \ldots, v_k\})$ Build tree



(Representative of this set is v_1)

JOIN-SETS (u, v) : Find representative of u = root of u's tree. Add edge from rep(u) to repr(v)



(We can be slightly smarter, even.)

Running time of Kruskal dominated by sorting $O(m \log m)$ But $m \leq n^2$ and $\log(n^2) = 2 \log n$

So running time $O(m \log n)$