Minimum Spanning Trees / Amortized Analysis

$G=(V,E)$
$|V|=n$
$|E|=m$.

Today's lecture.

- the Minimum Spanning Tree (MST) problem.

[Given an <u>undirected</u> graph $G=(V,E)$, with each edge having a weight w_e, find a spanning tree of minimum weight.]

(assume it for rest of lecture)

<u>Note</u>: if graph is connected then $\exists$ a spanning tree (a tree that contains all the vertices). Tree = no cycles, so has $n-1$ edges. = forest that is connected

So can enumerate over all possible spanning trees,

for each compute the ~~cost~~ wt. — given T, $w(T) = \sum_{e \in T} w_e$.

and output one with least weight.

Exponential time!

Using the definition directly is not good here.

A better algorithm [Kruskal '95]

① Sort the edges in non-decreasing order of weight.

Say $w(e_1) \le w(e_2) \le \dots\dots \le w(e_m)$.

Set $E_0^* \leftarrow \phi$ (emptyset)

② For $i=1$ to m

if adding edge e_i to E_{i-1}^* does not create cycles, add it ($E_i^* \leftarrow E_{i-1}^* \cup \{e_i\}$)

else $E_i^* \leftarrow E_{i-1}^*$.

Output ~~E_m^*~~. $T=(V, E_m^*)$

Next Steps: ① Correctness

② Runtime.

Optional: get better algorithm, then go to step ①. :)

Correctness:

Observation 1: $T_M = (V, E_M)$ is a spanning tree (remember: G is connected)

(Sketch: else we would have added edge connecting disjoint components).

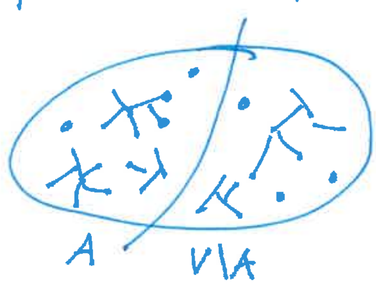
To prove T_M is a min-weight spanning tree (MST), ~~let us prove useful lemma~~ say a set of edges S is safe

~~Lemma 2: Suppose S is safe~~ if $\exists$ a MST $T' = (V, E')$ such that $S \subseteq E'$.

(so we can add more edges to S to get some MST.)

{ Plan: $E_0 = \phi$ is safe, trivially. Show that E_i is safe $\forall i$.
Let's prove a general useful lemma. }

Lemma 2: Suppose $S \subseteq E$ is safe. Also suppose we take a partition $(A, V \setminus A)$ of the vertex set and $\underline{\underline{e}}$ is a least weight edge crossing this, and no edge of S crosses this partition. Then $S \cup \{e\}$ is safe.



Pf: S is safe. So $\exists$ MST $T' = (V, E')$ st $E' \supseteq S$. If E' also contains e, done.

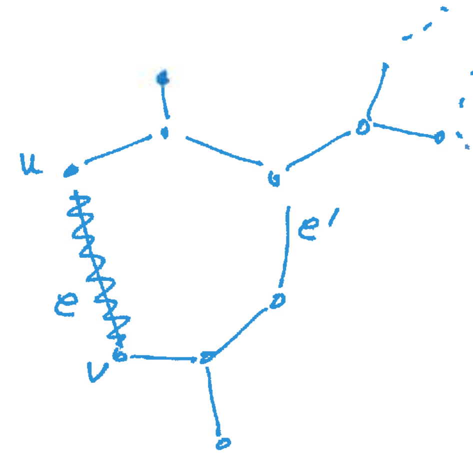
So $e \notin E'$. Hence adding e to spanning tree T' will create a cycle.

Say $e = \{u, v\}$.

Now consider the path P in tree T' from u to v. Say $u \in A$, $v \in V \setminus A$.

$\exists$ an edge ~~that~~ on Path P crosses from A side to $V \setminus A$ side.

Call this edge e'.



Note: $w(e') \geq w(e)$ since e is least-weight edge crossing partition $(A, V \setminus A)$

e' is not in S since S does not cross partition, and e' does.

$T'' = (V, E' - \{e'\} + e)$ is also connected, since we swapped e' for e, and both were on a cycle.

$\Rightarrow T''$ is another spanning tree.

and has weight at most as much as T'.

And it contains $S \cup \{e\}$.

So $S \cup \{e\}$ is safe too. ☺

Good: Now using Lemma 2, we can show Kruskal produces MST.

- E_0 is empty set, hence trivially safe.
- if E_{i-1} is safe, the edge e_i does not form a cycle, so it connects two components of E_{i-1}. Call any one of them A, and rest is $V \setminus A$.

Now e_i is cheapest edge to cross $(A, V \setminus A)$. Using Lemma 1, with $S = E_{i-1}$, get that $E_i = E_{i-1} \cup e_i$ is also safe. $\Rightarrow E_m$ is safe by induction

E_m is safe $\Rightarrow$ it can be extended into an MST.
But E_m is set of edges of a spanning tree itself

$\Rightarrow$ it <u>is</u> an MST.

<u>Aside</u>: Can use Lemma 2 to show other algorithms also produce MSTs.

<u>Eg</u>. $\begin{cases} \text{Prim's Algorithm} \\ \text{Boruvka's Algo.} \end{cases}$

Can discuss these some other time

<u>Good</u>: what about run time.

<u>Step 1</u>. sorting edges takes $O(m \log m)$ steps.

[Recall: algorithms like mergesort will do the job.
quicksort ~~does~~ gets this time <u>in expectation</u>]

<u>Step 2</u>: Want that at each step, given a set

E_i of edges, and a new edge e_{in},

check if $E_i \cup \{e_{in}\}$ contains a cycle.

In other words, if $e_{in} = \{u, v\}$, do u & v lie in

different connected components of (V, E_i) ?

Algo 1: Just run Depth First Search on (V, E_i) from u to see if v is reachable. May take $O(i)$ time.

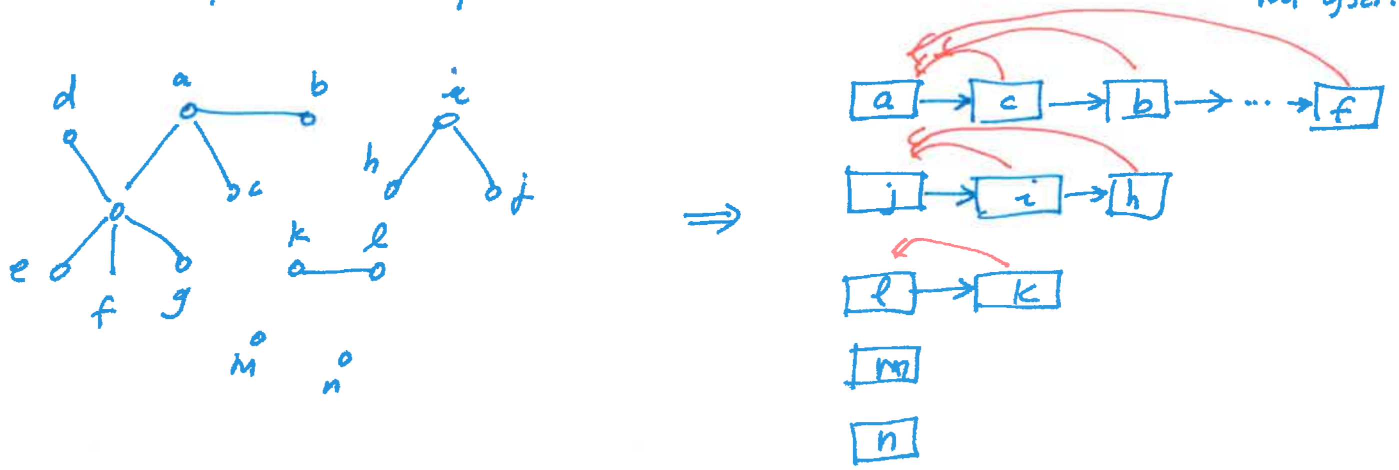
$\Rightarrow$ total time over all steps $\sum_{i=1}^{m} O(i) = O(m^2)$. ☹

Actually, always stop when $|E_i|$ has $n-1$ edges, so can reduce to $O(mn)$.

Still bad.

Idea: Maintain data structures to speed up lookups.

~~Each compo~~ Each component is a linked list, everyone maintains a pointer to "root" of set.



- to check if u & v in same component

  Check: u.root = v.root ?

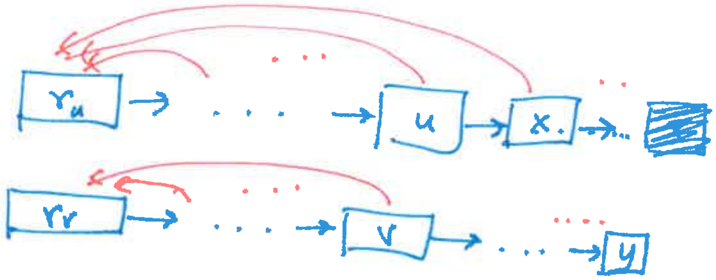
- if not, want to add (u,v). So merge the components.

  - need to merge the lists together.
  - update root pointers.

Say

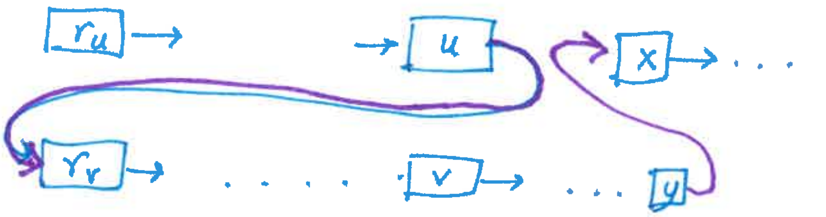
$L_u =$

L_v:



these are the two lists

Merge lists:



And then reset root pointers of the second list to point to r_u now.

newly grafted part.

Great. How much time?

O(length of second list).

Could be $O(n)$ in worst case.

$\Rightarrow O(n)$ each time we add a new edge $\Rightarrow O(n^2)$ overall.

Final Ingredient:

When merging two lists, merge the smaller into the larger.

say $|L_u| \geq |L_v|$.

then "charge" each element in L_v one \$ each.

Use this to pay for the merge and ~~the~~ resetting pointers.

<u>So to bound the total cost, ask:</u>

How much does each element pay over ~~all~~ the entire algo?

Each element pays 1 each time its list is merged
<u>and</u> it is in shorter list.

So its list length doubles each time it pays.

Final list length $\leq n$.

$\Rightarrow$ it pays $\leq \log_2 n$ times.

$\Rightarrow$ total payment $= O(n \log n)$ for merges.
$+\ O(m)$ to check for cycles.

<u>Kruskal's Algo</u> : $\underbrace{O(m \lg m)}_{\text{sorting}} + \underbrace{O(m + n \lg n)}_{\text{checking for cycles.}}$

<u>Amortized analysis</u>: some steps cost a lot,
but on average the cost is small.

"Amortized" = "on average."

Used a ~~token~~ bank account argument

- each ~~op~~ operation was paid for by someone.
- no one paid more than $O(\log n)$. $\Rightarrow$ total cost $\leq O(\underbrace{n}_{\#\text{people}} \underbrace{\log n}_{\text{max payment per person.}})$

Today's takeaways

- Sometimes greedy algorithms are good

  Saw it for MSTs.

  In fact this ~~extends~~ also works for a much broader class of min-weight subset problems. (See "matroids" if you are interested).

- Useful to keep data structure to avoid redoing work.

  - maintain connected components using linked lists.

  fast way to check if u & v share components.

- Choosing whom to merge into whom reduced time usage from $O(n^2)$ to $O(n \log n)$.

- Amortized analysis – clever way of accounting for the total cost.

  (Person pays ~~the~~ only when they see improvement.

  Improves only $\le \log_2 n$ times

  $\Rightarrow$ person pays only $\log_2 n$ times. QED)

  Will see more of it in course.