Minimum Spanning Trees / Amortized Analysis

Today's lecture.

- The Minimum Spanning Tree (MST) problem.

Given an undirected 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)

Note: If a graph is connected then it has a spanning tree (a tree that contains all the vertices). Tree = no cycles, so has $n-1$ edges.

So can enumerate over all possible spanning trees:

for each compute the cost — 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 1951]

1. Sort the edges in non-decreasing order of weight.
   
   Say $w(e_1) \leq w(e_2) \leq \ldots \leq w(e_m)$.
   
   Set $E^0 = \emptyset$ (empty set).

2. For $i = 1$ to $m$

   if adding edge $e_i$ to $E^0$ does not create cycles, add it.

   $(E^* \leftarrow E^* \cup \{e_i\}$)

   Output $T = (V, E^*)$

   else $E^* \leftarrow E^* - e_i$.
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's prove useful lemma
say a set of edges is safe

if \( \exists \) a MST \( T' = (V, E') \) such that \( S \subseteq E' \).

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

\( \{ \text{Plan: } E_0 = \emptyset \text{ is safe, initially. Show that } E_i \text{ is safe } \forall i. \} \)

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 \( e = (a, v) \) is a least weight edge crossing

this, and no edge of \( S \) crosses this partition. Then \( S \cup \{v, a \} \) is safe.

Pf: \( S \) is safe. So \( \exists \) MST \( T' = (V, E') \) at \( E \subseteq S \). If \( E' \) also contains \( e \),

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 the tree \( T' \) from \( u \) to \( v \). Say \( \forall e \in A \neq v \setminus V \). 

\( \exists \) an edge \( e' \) on path \( P \) crosses from \( A \) side to \( V \setminus A \) side.

Call this edge \( e' \).

**Note:** \( w(e') > w(e) \) since \( e \) is lowest-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 \( MST \).

If \( E_0 \) 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 \( S \), \( A \), and rest in \( V \setminus A \).

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. \( \Rightarrow E_i \) is safe by induction.
Em is safe $\Rightarrow$ it can be extended into an MST.
But Em is set of edge of a spanning tree itself

$\Rightarrow$ it is an MST.

Aside: Can use Lemma 2 to show other algorithms also produce MSTs.

Eg. Prim's Algorithm
Boruvka's Algo.

Can discuss these some other time.

Good: What about runtime.

Step 1: Sorting edges takes $O(m \log m)$ steps.

Recall: algorithms like mergesort will do the job.
quicksort gets this time in expectation $\mathcal{O}(m \log m)$

Step 2: Want that at each step, given a set $E_i$ of edges, and a new edge $e_{in}$
check if $E_i \cup e_{in} \subset \mathcal{F}$ contains a cycle.

In other words, if $e_{in} = \{u, v\}$, do $u$ and $v$ lie in different connected components of $(V, E_i)$?
**Algorithm 1:** Just run Depth First Search on $(V, \mathcal{E}_i)$ from $u$ to see if $v$ is reachable. May take $O(i)$ time.

\[ \Rightarrow \text{total time over all steps } \sum_{i=1}^{n} O(i) = O(n^2). \]

Actually, always stop when $|\mathcal{E}_i|$ has $n-1$ edges, so can reduce to $O(mn)$.

Still bad.

**Idea:** Maintain data structures to speed up lookups.

Each component is a linked list, everyone maintains a pointer to "root" offset.

To check if $u$ & $v$ in same component:

**Check:** $u\cdot \text{root} = v\cdot \text{root} \ ?$

- If not, want to add $(u,v)$, so merge the components.
  - Need to merge the lists together.
  - Update root pointers.
Soy

\[ L_v = r_v \rightarrow \ldots \rightarrow u \rightarrow x \rightarrow \ldots \]

\[ L_u = r_u \rightarrow \ldots \rightarrow u \rightarrow x \rightarrow \ldots \]

These are the two lists

Merge lists:

\[ r_u \rightarrow u \rightarrow x \rightarrow \ldots \]

\[ r_v \rightarrow v \rightarrow y \rightarrow \ldots \]

And then reset pointers of the second list to point to \( r_u \) now.

\[ r_u \rightarrow \ldots \rightarrow u \rightarrow r_v \rightarrow v \rightarrow y \rightarrow x \rightarrow \ldots \]

newly grafted part.

Great. How much time?

\( O(\text{length of second list}) \).

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

\[ \Rightarrow O(n) \text{ each time we add a new edge} \Rightarrow O(n^2) \text{ overall}. \]

Final ingredient:

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

\[ \Rightarrow \text{say } |L_u| \geq |L_v| \]

then "charge" each element in \( L_v \) one $ each.
Use this to pay for the merge and resetting pointers.

So to bound the total cost, ask:

How much does each element pay over the entire algo?

Each element pays 1 each time it's list is merged and it is in shorter list.

So its list length doubles each time it pays.

Final list length ≤ n.

⇒ it pays ≤ log₂n times.

⇒ total payment = O(n log n) for merges.

+ O(m) to check for cycles.

Kruskal's Algorithm: O(m log m) + O(m + n log n)  
\[ \text{sorting} \] \[ \text{checking for cycles} \]

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

"Amortized" = "on average."

Used a bank account argument:

- each operation was paid for by someone.
- no one paid more than O(log n).  ⇒ total cost ≤ O(n log n)
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 only when they see improvement.

  Improves only $\leq \log_2 n$ times

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

  Will see more of it in course.