Randomized Spanning Trees & MST verification

(Lecture #2)

Last time saw algorithms that all use the cut rule + fancy data structures to get improvements on runtimes $\rightarrow$ down to $O(m \log n)$ and better.

This time: (randomized) $O(m+n)$ time. Two ingredients:
(a) Sampling and using the cycle rule to delete a lot of edges. (Sparsify)
(b) MST verification

Light & Heavy:
- given a tree $T$ (or a forest $F$) and edge $e$ in $T$, $e$ is called $T$-light if $MSF(F \cup \{e\})$ contains $e$.
- An edge is $T$-heavy otherwise.

Fact: if $T$ is a MST then $\forall e \in T, e \in T$-light $\iff \exists e \in E(G)$.

Magic Blackbox: given $G = (V,E)$ and a tree $T$, there exists an algorithm Verify $(G,T)$ that returns a list of all $T$-light edges of $E$. And runs in time $O(m+n)$.

N.B.: if the list of $T$-light edges is just $E_T$ then $T$ is a MST of $G$.

So this, in linear time verifies if $T$ is a MST of $G$.

Namely: would need to check if $\forall e \in E \setminus E_T$, it is the heaviest edge on its fundamental cycle.
Karger’s Algorithm \( (V, E) \) \( |V| = n, |E| = m \), \( m > n \).

1. Let \( H \) be a random subset of edges of size \( \epsilon \) of \( E \) with probability \( \frac{1}{2} \).
2. \( E_1 \leftarrow \) random subset of \( E \) (with sampling probability \( \frac{1}{2} \)).
   + (some fixed spanning tree edges) drop.
3. \( T_1 \leftarrow \) Karger \((V', E_1)\)
4. discard all \( T_1 \)-heavy edges from \( E' \). (call this \( E_2 \), all light edges the remainder of \( E' \).
5. return Karger \((V', E_2)\)

**Fact 1:** \( E[\# \text{edges in } E_1] \leq 2n(\frac{1}{2})^{n-1} \)

**Fact 2:** \( E[\# \text{edges in } E_2] \leq 2n' \)

Says: all rest of work in steps 1, 2, 4 \( \leq cn \).

**Claim:** Let \( T(m, n) \) be expected runtime on every possible graph on \( n \) nodes.

\[
T(m, n) \leq E[T(1E_1, V')] + E[T(1E_2, V')] + m
\]

\[
\leq E[21E_1] + n' \epsilon + E[21E_2] + n' + m
\]

\[
\leq 2\left[m + n'\epsilon\right] + n' + 2\left[2n'\right] + n' + m
\]

\[
= 2m + 8n' \epsilon \leq 2m + n.
\]
Proofs of Facts:

Fact 1: easy. just note that each $\in E_i$, up to $\frac{1}{2}$. [must substitute
for $\frac{1}{2}$]

Fact 2: two different proofs depending on how $E_i$ is chosen.

Pf 1: [Karger]. $E_i$ is sampled by picking each edge in $E_i$ up to $\frac{1}{2}$ independently.

Recall: want to bound $\#T_i$. light edges, $T_i \subset$ MST on $E_i$.

$\Rightarrow$ Build $T_i$ using Kronkeal this way. Sort edges of $E_i$ and look at edges with

when looking at $e \in E_i$:

(i) if $e$ is a cycle with connected forest, ignore it. $\Rightarrow$ even if $e_i$ is a cycle,

(ii) if not, flip a coin for $e$. $\Rightarrow$ if tails, it will not hit $E_i$, it will be $T_i$.

$\Rightarrow$ if heads, $e_i$ is in $E_i$ and $e \in T_i$. $\Rightarrow$ also $T_i$ is light, but in $T_i$.

$\Rightarrow$ everything we are in case (ii) we make $T_i$ light edge.

But $\frac{1}{2}$ we add an edge to $T_i \Rightarrow E_i$ times before $T_i$ has $n-1$

edges $\Rightarrow E_i$ times before see $(n-1)$ heads $\Rightarrow 2(n-1)$.

QED.

Pf 2: [Chen]. $E_i$ is a random subset of $\frac{1}{2}n^2$ edges of $E_i$. plus a

Claim: $\forall e \in E_i$ $e \in T_i$ light with $\frac{1}{2}n$. [straight of op].

Pf: $T_i \subset$ MST of $E_i$.

if $e \in T_i$ light then $e \notin$ MST($T_i$, $U \cup E_i$).

Claim: Pick a random edge $e$, $\Pr[e \in \text{MST}(E_i)\text{-light}] \leq \frac{1}{2^n}$

$\Rightarrow \Pr[e \text{ is in } \text{MST}(E_i \cup e)]$ $\neq \Pr[e \text{ in } \text{MST}(E_i)\text{-light}]$ $\leq \frac{1}{2^n}$

but $\Pr[e \text{ in } \text{MST}(E_i \cup e)] \neq \Pr[e \text{ in } \text{MST}(E_i)$ randomly]

$\Rightarrow$ $\Pr[e \text{ in } \text{MST}(E_i \cup e)] \leq \frac{1}{n^2}$. 


Next: How to prove the Magic Black Box? MST verification.

(*) Given a tree T and pairs \((u_1, v_1), (u_2, v_2), \ldots, (u_m, v_m)\), return the minimum weight \(w_i\) of the heaviest edge on the path \(T[u_i, v_i]\), in time \(O(mn)\).

If we do this, can solve the MST verification problem.

---

**History:**

- 47: Dixon, Ranum, Tayan on RAM machine.
- 48: Backhaus-Kopplm, Rogers, Moniela, MFC.
- 47: Krevsky also on RAM machine.
- 07: Hagop C. Simony.

**Simplifications:** Given any tree, can assume that \(v_i\) is an ancestor of \(u_i\).

**Idea:** Find, for each \((u_i, v_i)\), the LCA \(LCA(u_i, v_i)\).

Create pairs \((u_i, LCA(u_i, v_i))\) and \((v_i, LCA(u_i, v_i))\) for each original \((u_i, v_i)\).

Now, given a soln for new instance, just return \(w_i = \min \{LCA(u_i, v_i)\}\).

**How long to find LCA’s of all pairs?**

**Thm [Harel & Tayan]:** Can preprocess a tree \(T\) in \(O(n)\) time so that

- can answer LCA queries in \(O(1)\) time.

Actually simpler: given \(T\) and all the pairs up front can answer in time \(O(mn)\), \(\approx\) easier problem.

May come back at end to give a soln \(O(m \log (mn) + n)\) time algorithm using min-find.
OK: do Tree $T$, pair $u_i, v_i$ with $v_i$ ancestor of $u_i$. Return $w_i = \max$ wt edge on $T[u_i, v_i]$.  

**Simplification #2:** $T$ is a fully branchy tree.

A fully-branchy tree $T$ is a rooted leveled tree with:

(a) all leaves on some level $d$.
(b) each internal node has at least 2 children.

Claim: take $T$, run Bronska on it, this gives a nice laminar structure which defines a tree $T'$

- height of $T' \leq \log_2 n$.

Fact:

$$\max \text{ wt edge on } T'[u_i, v_i] = \max \text{ wt edge on } T[u_i, v_i].$$  

[HW #1]

OK: so do simplification #2, and then #1 gives a fully branchy tree $T$ and pair $(u_i, v_i)$ of $v_i$ an ancestor of $u_i$, and $u_i$ is a leaf (by simplification #2).

How to solve max queries now!
For each edge $e = (u, v)$, look at all queries starting in $Tu$ and ending above $v$.

Say they go to $v_1, v_2, \ldots, v_k$ such that $d(v_1) < d(v_2) < \ldots < d(v_k)$.

And we have a "query string"

$Q_u = (d_1, d_2, \ldots, d_k)$.

For which we have an answer string

$A_u = (a_1, a_2, \ldots, a_k)$ where $a_i = \max$ wt. edge between $u$ and $v_i$ at depth $di$.

Crucial Fact: $a_1 \geq a_2 \geq \ldots \geq a_k$.

Now, want to extend this to $Q_x$ and get $A_x$.

Note: $Q_x \subseteq Q_u \cup \text{depth}(u)$.

So we can know that $A_u$ consists of

$\max(a_1, w(x, u)), \max(a_2, w(x, u)), \ldots, \max(a_k, w(x, u))$.

Can use binary search to find the right place, after which it in all $w(x, u)$.

$\#\text{comps} = \left\lceil \log_2(1 + |Q_u|) \right\rceil = \log_2|Q_u| + 1$.

$\Rightarrow \text{total \#comps} \leq \sum_u \log_2(1 + |Q_u|) + n$.

Claim: $\leq O(n \log \frac{m + n}{n}) = O(m)$. 

Kernels: just count # of comparisons for now.
Pf: let $n$ guys at level $i$ (issues at level 0).

$$\sum_{u \in \text{level}_i} \log (1 + |Qu|) \leq n_i \text{ Average} (\log_2 (1 + |Qu|))$$

$$\leq n_i \log_2 (1 + \frac{\sum_{u \in \text{level}_i} |Qu|}{n_i})$$

Simplifying,

$$= n_i \log_2 (1 + \frac{m_i}{n_i}) \leq n_i \log_2 \left( \frac{n + m_i}{n_i} \right)$$

$$\Rightarrow \sum_{u \in \text{all}} \log (1 + |Qu|) \leq n \log \left( \frac{n + m_i}{n} \right) + \sum n_i \log \left( \frac{m_i}{n_i} \right)$$

$$n_i \leq \frac{n}{2^i} \Rightarrow \sum_{i} n_i \log \left( \frac{m_i}{n_i} \right) \leq n \log \left( \frac{m}{n} \right) + n \log \left( \frac{m}{n} \right)$$

and $x \log \left( \frac{m}{n} \right)$ is increasing for $x \leq \frac{n}{2}.

$$\leq m + n \cdot \frac{n}{2^i} \cdot i$$

$$= O(n).$$

$$\Rightarrow \text{total: } O(n + n \log_2 \left( \frac{n + m_i}{n} \right)).$$

Remark: It is not surprising that I.a proof that $O(n)$ can be proven $\Rightarrow$ is MST.

[Just give sorted list of edges. etc.]

But that this proof can be found using $O(n)$ arrays. And in O(n) time.

Rest of analysis: How to implement all this in $O(n)$ time and not just

Using $O(n)$ comparisons, need to do table lookups and stuff.

High level idea: store the Qu as a log_2 n bit word.

Store answer also as a set of nodes: so $a_k$ is nearest the heaviest

edge on $(u, v_k)$ is one from $a_k$ to $p(a_k)$

now carefully do the same operations but instead of having to do

binary search explicitly, store the solutions in a 0 to 1 bitstrings and use them.
Wrap up: MST verification in O(mn) time.

Saw details except (a) LCA and (b) lemma from HW1.

→ RandomizedAlg in O(mn) time.

Can we make this deterministic? [Can we get an O(mn) algorithm where true or false is output?]

---

Postscript: LCAs in O(max(m,n)) time? At least when all queries are given up front: "offline"

List L of queries.

LCA(x)
  · makeSet(x)
  · for all children y of x.
    · LCA(y)
    · union(x, y).
    · if "head of find(x) ← x.
      · find(y)
  · x marked

if z st (z,e) ∈ List
  if z marked then LCA(x, z) = head of find(z).

Intuition/Proof:

If z marked then z has been explored and we're in another child of the LCA(= r)

⇒ find(z).head = r.

[Trayan's '79 paper]