6.854 Optional Lecture: LINK-CUT TREES

Zeyuan A. Zhu.
zeyuan@mit.edu.

☐ Roadmap.

§1. Goal [Op.]

§2. Link-cut Tree Structure

§3. Implementations

§4. Amortized Analysis

§5. Applications on Blocking-flows

[Notes will be provided]

☐ Goal.

- represent a forest of rooted trees
- each node has an arbitrary # of unordered children
- FIND-ROOT(v) : return the root of the tree that contains v.
- CUT(v) : delete the edge between v and par(v).
- LINK(v, w): make v a new child of w
  (user guarantee: v is the root of some tree)
- Will consider UPDATE/FIND-MIN to deal with cap./val. later.

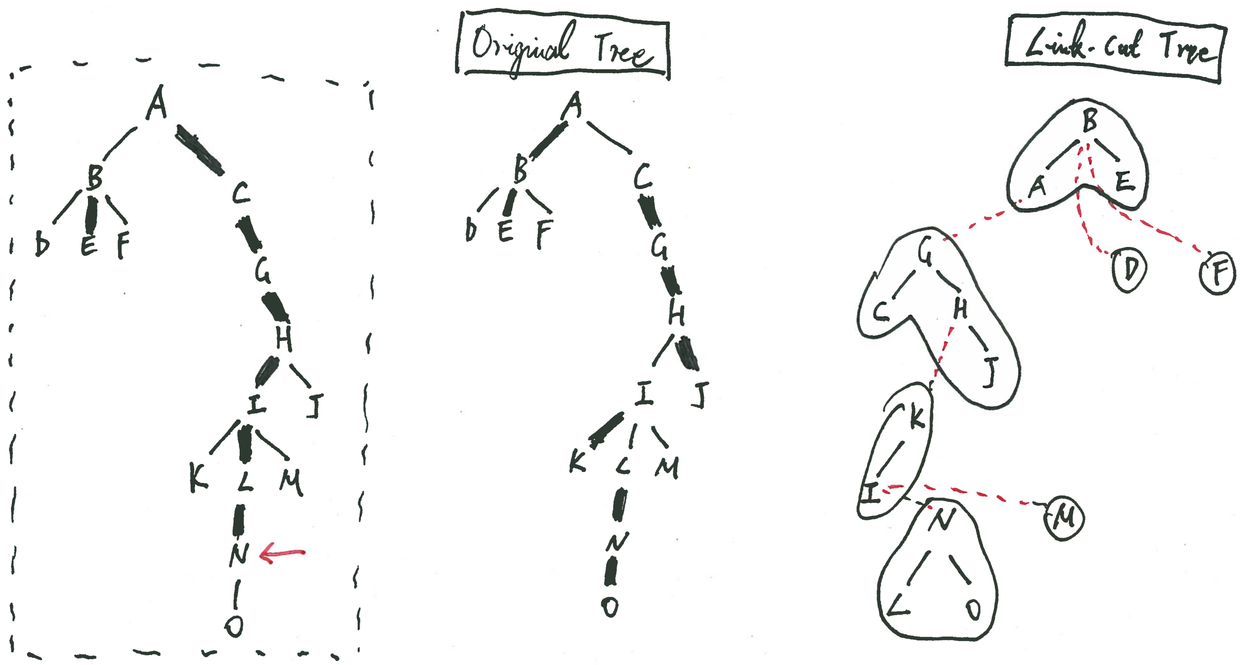
☐ Warm-Up.

- Maintain the tree ⇒ $O(1)$ cut/link but $O(n)$ FINDROOT.
- No CUT op. ⇒ $O(\alpha(n))$ LINK/FINDROOT using disjoint-union
- [Sleator Tarjan '82] "A data structure for dyn. trees"
  $O(\log n)$ amortized, using Link-cut Trees.

☐ LINK-CUT TREES.

- Intuition. { if there is a single chain — use a splay tree, ordered by depth
  otherwise, do path decomposition, each path by a splay tree.

- A node has been <u>accessed</u> if was passed to some op. above.
- For a node v, its <u>preferred child</u> is the one that contains the last accessed node in T(v).
- A <u>preferred edge</u> is (v, w) where w is v's preferred child.
- A <u>preferred path</u> is a path of preferred edges. [Draw Original Tree]



- Represent (each of) the original tree as a <u>tree of auxiliary trees</u>, one for each preferred path
- Each auxiliary tree is a (binary) splay tree keyed by depth [left = closer to the root]

[Draw Link-cut Tree]

- Each root of the auxiliary tree has a path-parent pointer [except for the root]

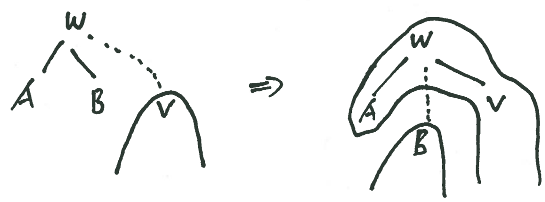
[can't store parent-to-child pointers]

□ Operations / Implementations

- Access(v): no actual change to the original tree
  but change the preferred-path decomposition

[Go to the example, Access(N)]

- Splay(v). [bring v to the top of his auxiliary tree, right = descendants on original tree]
- path-parent(right(v)) ← v, right(v) ← null [setting par(right(v)) ← null]
- v' ← v.
- WHILE v ≠ root of the link-cut tree Do. (*)
  - w ← path-parent(v)
  - Splay(w)
  - path-parent(right(w)) ← w, right(w) ← v
  - path-parent(v) ← null
  - v ← w.

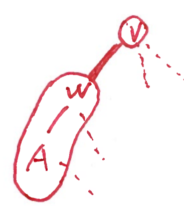


- Splay(v')

- Find-Root(v): – Access(v)
  – w ← smallest elem. in the auxiliary tree of v.
  – Splay(w), return w.

- Cut(v): – Access(v)
  – left(v) ← null [no need to set up the path-parent]

- Link(v, w): – Access(v), Access(w)
  – left(v) ← w.



□ Amortized Analysis – $O(\log^2 n)$

- Suffices to show Access(v) is in $O(\log^2 n)$ amortized.
- Each Splay $O(\log n)$, suffices to show loop (*) has $O(\log n)$ iter. (amortized)

Am I cheating? n A ops. $n\log n$, 1, 1, . . . 1 ⇒ $\log n$ amortized
n B ops. $n\log n$, 1, 1, - - - 1 ⇒ – – –
Together $n^2\log^2 n / n = n\log^2 n$?

$\underbrace{SSS\cdots S}_{\text{Access}}$ $\underbrace{SS}_{\text{Access}}$ . . . $\underbrace{SS\cdots S}_{\text{Access}}$ t Access
⇒ $t\log n$ Splay
⇒ $t\log^2 n$ time

- Heavy-Light Decomposition
  - An edge $(v, par(v))$ is heavy if $size(v) > \frac{1}{2} size(par(v))$.
  - light-depth(v) := # of light edges root $\leftrightarrow v$. $O(\log n)$
  - [An edge in the original graph can be light/heavy. preferred/unpreferred]

- # of iter. in (*) = O(# edge that become preferred)
- At most $O(\log n)$ of light edges will become prefered for each Access.

This part will be made clear on the 6th page

$$
\begin{aligned}
\#(\text{heavy edges become prefered}) &\le \#(\text{heavy edges become unprefered}) + (n-1) \\
&\le \#(\text{light edges become prefered}) + (n-1) \quad (+O(t)) \\
&\le \cancel{O(t \log n)}\ O(t \cdot \log n). \quad \text{for } t \text{ Access}
\end{aligned}
$$

$\Rightarrow O(\log n)$ iter. amortized.

☐ Amortized Analysis – $O(\log n)$

- $\Phi(v) = \log s(v)$ $\quad$ $s(v)$: # nodes in the ~~auxiliary tree~~ link-cut tree [tree of aux. trees]

- Does not affect ops. in splay trees [imagine that we have weights now]

  $\text{cost}(\text{splay}(v)) \le 3(\log s(u) - \log s(v))$

  where u is the root of the auxiliary tree that contains v.

- Access(v) takes $O(\log n)$ amortized, due to telescoping
- Cut(v): potential ↓ ; Link(v,w): potential ↑ } of $\le O(\log n)$.

☐ Applications

- Let each edge be associated with a capacity

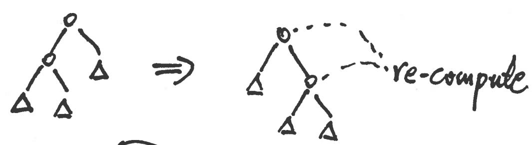
  store it on the node, $c(v) := c(v,w)$ where w is v's parent in the original tree.

  $c(root) := \infty$

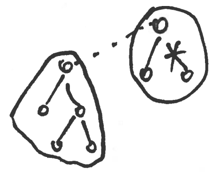
- Update(v, x): add x to all edges on root(v) — v.

- Find-Min(v): return min cap. node ~~in v's path to root(v)~~ on root(v) — v

- Maintain the following extra fields
  - $\Delta c(v) = c(v) - c(w)$ if $w = \mathrm{par}(v)$ on the aux. (splay) tree.
    $= c(v)$ if $v = \mathrm{root}$
  - ~~min(v)~~ = value of the min cap. node in subtree(v).
    $\Delta \min(v) = \min(v) - c(v) \leq 0$
  - argmin(v)
- All in $O(1)$ to maintain: SPLAY :

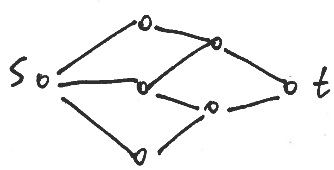


LINK :



- UPDATE(v, x): ACCESS(v), $\Delta c(v) \leftarrow \Delta c(v) + x$
  FIND-MIN(v, x): ACCESS(v), return argmin(v).
  CAP(v): ACCESS(v), return $\Delta c(v)$.

☐ Blocking-Flow



STEP 0 : each vertex forms an ind. tree

STEP 1 : $v \leftarrow$ FIND-ROOT(s)

STEP 2 : If $v \neq t$ { (Advance) select an edge (v, w), LINK(v, w, cap)
(Retreat ~~Return~~) if no such edge, for each (w, v), if in, CUT(w).

GOTO 1.

STEP 3 : If $v = t$ (Augment), { $v \leftarrow$ FIND-MIN(s)
UPDATE(s, −CAP(v))
CUT(v) }
GOTO 1.

- $O(m \log n)$ alg. for blocking-flow, $\Rightarrow$ $O(mn \log n)$ for max-flow.

☐ Fixing a bug on page 4.

- I claimed earlier that #(heavy edges become preferred) $\le$ #(heavy edges become unpreferred) + $(n-1)$, since "a heavy edge must become unpreferred before it can be preferred again".
- The above claim is INCORRECT! The same mistake has also appeared in the lecture notes of 6.854 in 2007, and 6.851 in 2007.
- The correction is as follows:

  #(heavy edges become preferred) $\le$ #(heavy-preferred edges get destroyed) + $(n-1)$

  $=$ #(heavy-preferred $\Rightarrow$ unpreferred) + #(heavy-preferred $\Rightarrow$ light) + $(n-1)$.

  $=:$ ① + ② + $(n-1)$.

- ① can only happen in ACCESS. But, if a heavy edge gets unpreferred, there must be a light one that gets preferred (except one of them for each ACCESS). But, #(light edges get preferred) $= O(\lg n)$ for each ACCESS. In sum, ① $= O(t \lg n) + t = O(t \lg n)$.
- ② can only happen during LINK/CUT.
  - It won't happen in LINK(v,w) because all edges on the way $w \leftrightarrow$ root are preferred after ACCESS(w), and they will only get heavier.
  - It happens in CUT(v). All edges on $v \leftrightarrow$ root will be preferred after ACCESS(v), but at most $O(\lg n)$ of them ~~are~~ will become light. $\Rightarrow$ ② $= O(t \lg n)$
- $\Rightarrow$ #(heavy edges become preferred) $= O(t \lg n)$.