6.851

ANDRÉ SCHULZ NOTES

Lecture 16

DYNAMIC TREES (LINK CUT TREES)

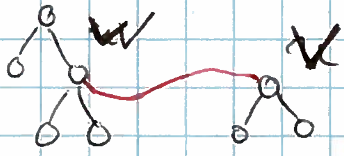
[SLEATOR, TARJAN '83]

- data structure to represent a forest of rooted trees
- OPN:

MAKE-TREE: Returns a new vertex as singelton tree

Root, in 2 different trees

LINK (v,w): Links the trees of v & w by making v a child of w



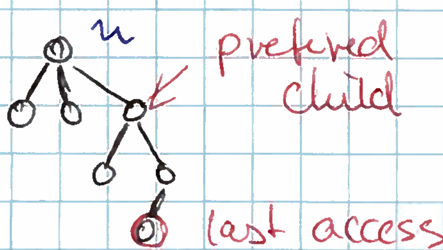
CUT (v): Remove the edge (v, parent

FIND ROOT (v): Return the root of v's tree

- A vertex was <u>accessed</u> if it was ~~pa~~ an argument of some OPN
- Every tree of the forest is ~~app~~ represented by a tree in the data structure

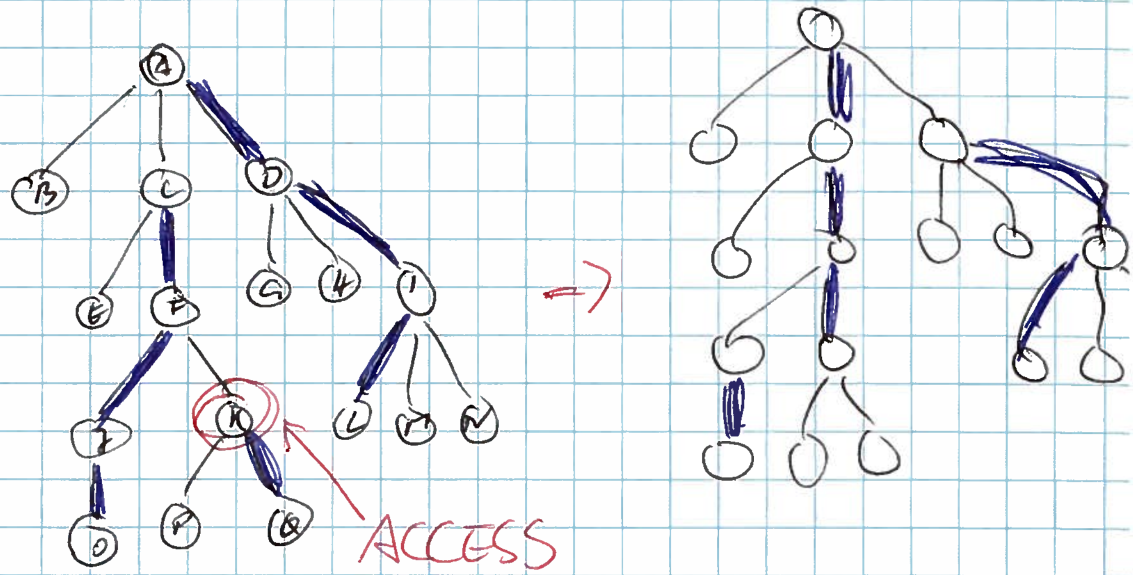
  nodes in this are prefered paths ~~as~~ of the original tree stored as splay-trees (key v = depth v)
- a <u>prefered child</u> of a node u in the original tree, is the child that contains the last accessed node in its subtree

  within the subtree below u



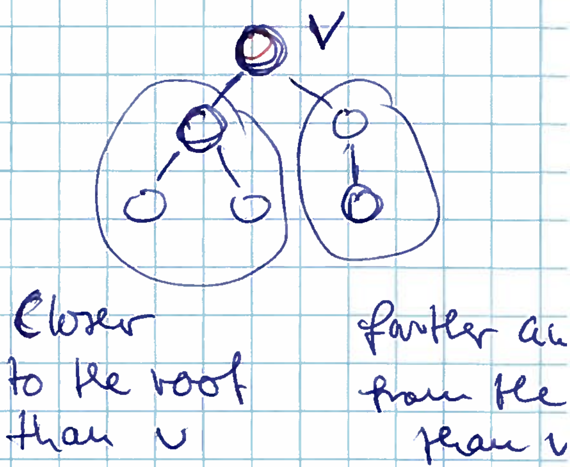
- a <u>prefered edge</u>: edge between u and its prefered child
- a prefered path: maximal connected component of prefered edges

Example:



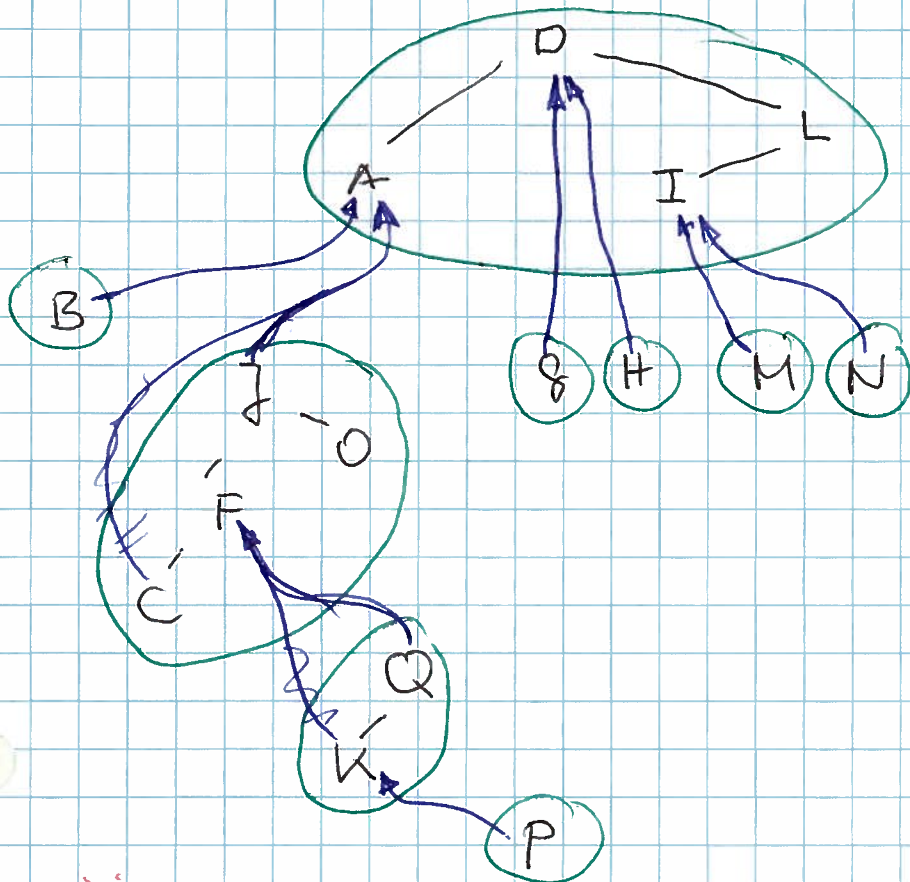
→ Every preferred path = node in the ds-tree
= splay tree keyed by depth
= AUXILIARY TREE

→



→ For every auxiliary tree we store a parent-path pointer in the root of the auxiliary tree

FOR the example (before ACCESSING)



→ SIMILAR TO TANGO TREES (BUT NOT BINARY and paths are not balanced BS)

- CUT / LINK / FIND-ROOT USE A COMMON SUB ROUTINE : ACCESS (n)

ASSURES that the ds tree is correct

How does it work?:
- Make n the root in its auxiliary tree
- update the prefered edges along n → root ds-tree

ACCESS(u):

(1) Splay u in its auxiliary tree
{update path_parent for aux. tree}

(2) Remove u's prefered child:

(2.1) ~~...~~
path_parent (right(u)) → u

(2.2) right(u) = NULL
parent (right(u)) = NULL } MAKE

(3) until we reached the root of the ds-tree:

(3.1) w ← parent (path (u))

(3.2) splay (w)

(3.3) path-parent (right (w)) ← w

(3.4) Make { right (w) ← u
parent (u) ← w

(3.5) path_parent (u) ← NULL

(3.6) u ← w

(4) SPLAY (u)

FIND-ROOT (v)

(1) ACCESS (v)

(2) Retrieve the smallest entry of the auxiliary tree of the ds-tree-root (aux-tree contains roo root has min depth)

$v \leftarrow left(v)$, until $left(v) = NULL$

(3) Return (v)

(4) Splay (v)

CUT (v)

(1) ACCESS (v)

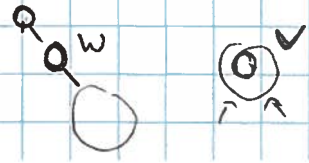
(2) $left(v) \leftarrow NULL$
$parent(left(v)) \leftarrow NULL$ } Make #

LINK (v, w)

(1) ACCESS (v) ← v alone in its aux. tree

(2) ACCESS (w) ← w root of its aux. tree

(3) Make { $left(w) \leftarrow v$
$parent(v) \leftarrow w$



ANALYSIS

- LINK, CUT costs: ACCESS + $O(1)$
- FIND-ROOT: ACCESS + find min + splay
- ACCESS: # pref child changes · splay costs
(splay analysis works in this setting)
- for m Op

$O(\log n) \cdot (m + \text{total \# of pref child changes})$

C

We will show: $C = O(m \log n)$

Heavy-light decomposition

- $size(v)$ = # nodes in v's subtree
- edge (u,v) is heavy, if $size(v) \geq \frac{1}{2} size(u)$ (u parent)
otherwise light
- ≤ 1 heavy edge / node
- $\leq \log(n)$ light edges on a path to root

- # of splays = # of prefered child changes
  = # of prefered edge constructions
  constructio

- # of light prefered child ~~creations~~
  $\leq O(\log n)$ (because we have $\leq \log n$ light edges on a root -> leaf pa

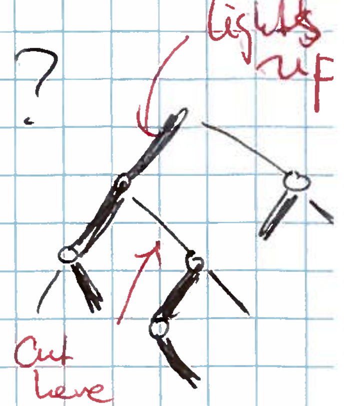
- # of heavy edge construction
  $\leq \underbrace{\text{\# of heavy edge destruction}}_{\text{every other created edge implies an destruction in prior}} + \underbrace{n+1}_{\text{diff be start \& en}}$

- destruction of a heavy edge as pc
  => construction of light edge as p
  ↑ only $O(\log n)$ / Access

=> total costs $O(\log n)$ / Access

bottom line: # of light prefered edge creations / O
$G \cdot O($
~~gives the bo~~ => $C \in O(\log n)$

Do link/cut interfere?

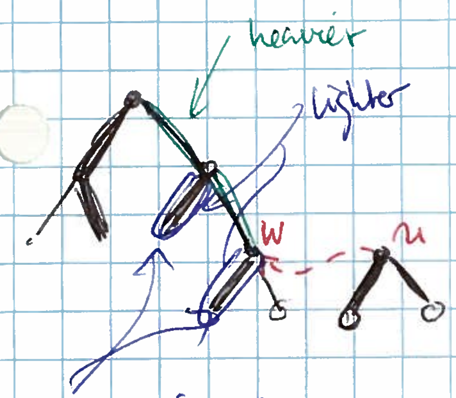


Cut : lightens ancestors of v

→ creation of log n light pref. edges (allowed per Opn)

Link (u, w) • heavens nodes on root-path, lightens nondes hanging o

• ~~these edges~~ edges gets lighter ⇒ because they hang of root-there are not prefered (→ACCESS (w) was executed



• no creation of p.c. light ea

⇒ total : $O(\log^2 n)$ / opn amorti

IMPROVEMENT

Review Splay trees:

ACCESS-Theorem

in our setting

w_i = weights for splay tree nodes $\equiv 1$

$S(x)$: $\sum_i \{ w_i \mid i$ is in subtree rooted at $x \}$

$\Phi = \sum_x \log S(x)$ potential

THEOREM

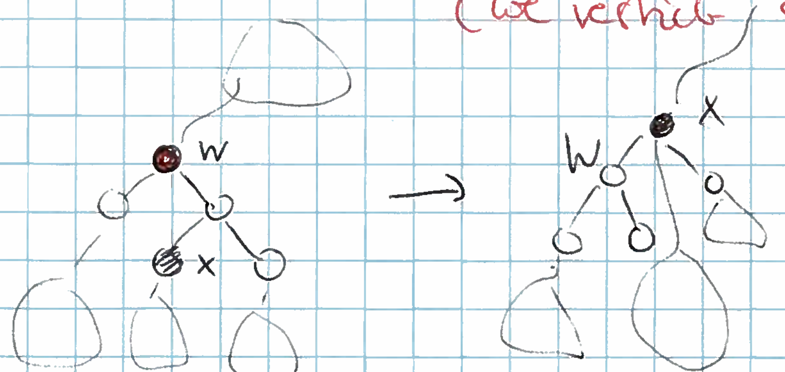
the amortized cost of accessing x

$\leq 3(\log s^{new}(x) - \log s^{old}(x))$

Condition: $\max\phi - \min\phi$ is small, which is true for general splays

(We use $w_i \equiv 1$)

- Observation 1: A splay only changes the $S(x)$ in ~~the~~ the aux. tree of x → Splay tree analysis holds (we restrict $S(x)$ to the auxiliary tree)



- **Observation 2**

$$S^{new}(x) = S^{old}(w)$$

$\Rightarrow$ costs $\leq 3 \left(\log S(w) - \log S(x) \right)$ (amortized for accessing x)

- **Observation 3**

  - Assume that we splay $x_1, x_2 \ldots x_2$ during an access

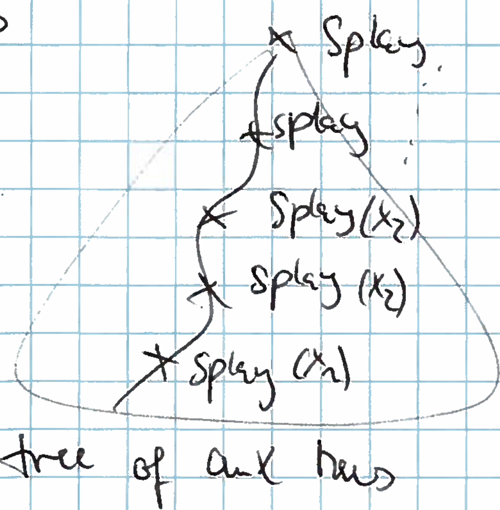
  $\Rightarrow$ $S(x_1) \leq S(x_2) \leq S(x_3) \leq \ldots \leq$

- **Observation 4**

  - Changing the perefered childs does not change the tree of aux trees

  $\rightarrow$ does not change $S(x)$ values

ACCESS



access cost

$\sum_i$ costs splay $(x_i$

$\Rightarrow$ accesscost $\leq \sum_i 3 \log(r_i) - \log(x_i) + 1$

r_i: root of aux tree of x

$s(x_i) \leq s(r_i) \leq s(x_{i+1})$!

telescope

accesscost $\leq 3 \log(s(\text{root}))$

$+ \underbrace{O(\#\text{prefered child changes})}_{O(\log n)}$

?

~~Claim~~ $s(\text{root}) \in O(\log n)$

- In order to make the amortization work, we have to show that $\Phi - \min \Phi + \max \Phi$ is small
- this holds for the access opn because we only changed inside
  1. Auxilary tree $\rightarrow$ usual splay analysis

CUT(V) : decreases $S(x)$

$\hookrightarrow \Phi$ decreases

JOIN(v,w) : increase only $S(w)$

by $\le n$

$\Rightarrow \Phi$ increases by $\le \log n$

Start configuration : Singeltons

$\Rightarrow \Phi = \sum_i \log(1)$

$= 0$ ← never drops below

End configuration : $\Phi \le m \log n$

$\uparrow$

of ~~all~~ joins

$\Rightarrow$ we can charge $\Phi_{max} - \Phi_{min}$

to the JOINS

$\rightarrow$ each join works in $O(\log n)$

amortized