Lecture 4

Dynamic trees (beginning of dynamic graphs)

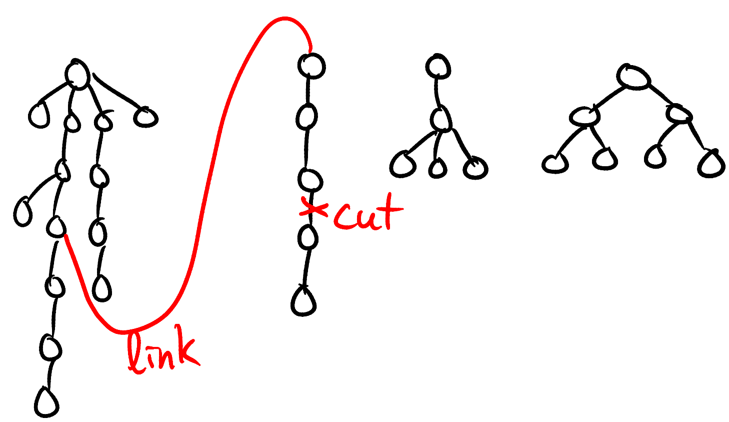
Problem: maintain a forest of rooted trees
topology — any # children, unordered

Operations:

- maketree: return new vertex in new singleton tree
- link(v, w): make v new child of w — add edge (v, w)
  v: root of tree not containing w
- cut(v): delete edge (v, parent(v)) (v not root)
- findroot(v): return root of tree containing v

Link-cut trees [Sleator & Tarjan — JCSS 1983; Tarjan — book 1984]

- even though represented trees in forest may be unbalanced, achieve $O(\lg n)$ time/op. (amortized)

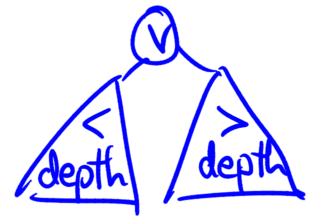


Link-cut trees are like Tango trees:

- represented tree split into paths:
  - preferred child of node $v = w$ if last access to v's subtree was in w's subtree, or none if v or no last access to v's subtree
  - preferred path = chain of preferred edges
  - preferred paths partition nodes of represented tree
- preferred path represented by auxiliary tree
  - splay tree keyed on depth
  - at root node, store path parent: path's top node's parent in represented tree (can't store path children — may be many)
  - auxiliary trees + path parent pointers = tree of auxiliary trees (potentially high degree)

access(v): make preferred path from root to v
& make v the root of auxiliary tree
$\Rightarrow$ v is the root of tree of aux. trees

- splay v (within its aux. tree) $\Rightarrow$ at root
- remove v's preferred child:
  - path parent(right(v)) $\leftarrow$ v
  - right(v) $\leftarrow$ none (+sym.)



- until v is in root's preferred path
  - w $\leftarrow$ path parent(v) (none $\Rightarrow$ finished)
  - splay w (within its aux. tree)
  - switch w's preferred child to v:
    - path parent(right(w)) $\leftarrow$ w
    - right(w) $\leftarrow$ v (+sym.)
    - path parent(v) $\leftarrow$ none
  - v $\leftarrow$ w

+sym.: + symmetric setting of parent ptr.

findroot(v):

- access(v) ⇒ root of aux. tree containing root
- find min. (depth) node in aux. tree:
    - v ← left(v) until left(v) = none
- splay v — to make faster for next time
- return v

cut(v):

- access(v)
- declare left(v) to root new tree of aux. trees:
    - left(v) ← none (+sym.)

note: right(v) = none too: no preferred child

link(v, w):

- access(v) ⇒ v alone in pref. path/aux. tree
- access(w) ⇒ w at root
- left(v) ← w ⇒ v becomes deepest node in w's preferred path
  (+sym.)

$O(\lg^2 n)$ bound: (amortized)

- link & cut cost $O(1)$ beyond access
- findroot costs access + time to find & splay min.
- access costs time to splay · # preferred child changes
- splay analysis works in this setting (splits/concats)

$\Rightarrow O(\lg n) \cdot (m + \underbrace{\text{total \# preferred child changes}}_{\text{claim: } O(m \lg n)})$

Heavy-light decomposition: in represented tree

- size(v) = # nodes in v's subtree
- edge (v, parent(v)) is heavy if $\text{size}(v) > \frac{1}{2}\,\text{size}(\text{parent}(v))$
  & light otherwise

$\Rightarrow$ at most 1 heavy child of each node

- heavy paths partitions nodes of tree
- light depth (v) = # light edges on root-to-v path $\leq \lg n$

$\rightarrow$ represented edge can be $\begin{pmatrix}\text{preferred}\\ \text{not}\end{pmatrix}$ & $\begin{pmatrix}\text{heavy}\\ \text{light}\end{pmatrix}$

$O(m \lg n)$ preferred child changes

- count # light preferred edge creations
- access(v) creates preferred edges along root-v path

$\Rightarrow \leq \lg n$ of them are light — so all about heavy

- count # heavy preferred edge destructions
  & add $\leq n+1$ for end configuration
- destruction of heavy preferred edge
  $\Rightarrow$ creation of light preferred edge
  except possibly preferred child of v

$\Rightarrow \leq \lg n + 1$

$\Rightarrow$ access(v) creates $O(\lg n)$ preferred edges

- link(v, w) "heavens" nodes on root-to-w path

$\Rightarrow$ edges hanging off this path may become light
(worry: create light pref./destroy heavy pref.)

- but access(w) made these edges unpreferred

- cut(v) lightens ancestors of v
- but $\leq \lg n$ of them can be(come) light
- also possibly destroy heavy pref. edge (v, parent(v))

$O(\lg n)$ bound: (amortized)

- $W(v)$ = # nodes in v's subtree in tree of aux. trees
  = # nodes in v's subtree within v's aux. tree
  + Σ #nodes in each "descendent" aux. tree of these nodes
- potential $\Phi = \sum_v \lg W(v)$ (splay potential)
- claim: amortized cost of access
  $= O(\lg n) + O(1) \cdot$ # preferred child changes
  $(= O(\lg n)$ amortized$)$

Proof:

- splay(v) costs $\le 3(\lg W(u) - \lg W(v)) + 1$
  where u = root of v's auxiliary tree
- splay(v) affects W's only within v's aux. tree
  $\Rightarrow$ standard splay analysis holds
- changing v's preferred child doesn't change W's: tree of aux. trees stays same
- $W(v) \le W(w)$ – w next element to splay

$\Rightarrow$ telescopes

$\Rightarrow$ access cost $\le 3\underbrace{(\lg W(r) - \lg W(v))}_{= O(\lg n)} + O\begin{pmatrix}\text{\# pref.} \\ \text{child} \\ \text{changes}\end{pmatrix}$

- cut(v) only decreases W's $\Rightarrow \Phi$ only decreases
- join(v, w) just increases W(w), by $\le n$

$\Rightarrow O(\lg n)$ increase in Φ $\square$