SoS Talk: Dynamic Connectivity on Forests with Link-Cut Trees

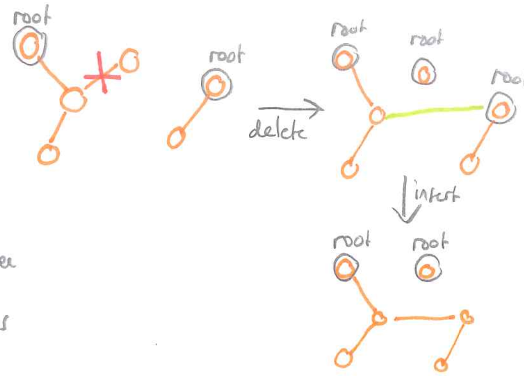
Daniel Graf
8.12.2016

Setting: Object of study: Forest of rooted trees

Wanted: **Queries:** · Connectivity (u,v)?
are u & v in the same tree?

Updates: · InsertEdge(u,v) // where v is a root
· DeleteEdge(u,v) // makes v the root of its tree

Goal: small (amortized) runtime for all (and more) operations



Warmups: ∘ Incremental: union-find in $O(\alpha(n))$ [Tarjan'75]

∘ Decremental: – use component labels for $O(1)$ queries
– upon delete, only relabel the smaller remaining tree
find it with parallel BFS → $O(\log n)$ updates [Even-Shiloach'75]
(with bit tricks in $O(1)$ [Alstrup et al.'97].)

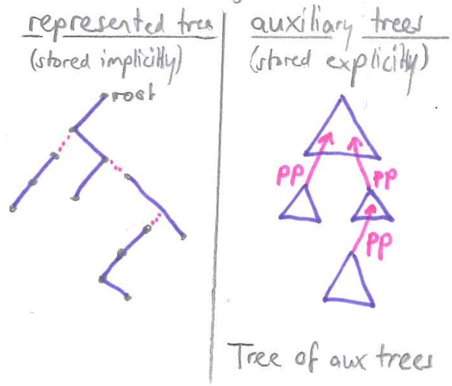
Link-Cut Trees (all in $O(\log n)$ amortized) [Sleator, Tarjan 1983]

Idea: ∘ take unbalanced tree
∘ decompose it into disjoint paths
∘ store each path in a balanced tree

Preferred Path Decomposition

preferred child of v = $\begin{cases} w, & \text{if last access to } v\text{'s subtree ended in } w\text{'s subtree} \\ \text{none}, & \text{if last access ended at } v \end{cases}$

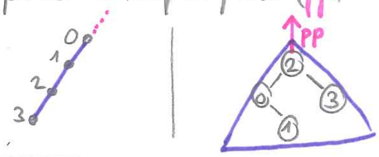
after accessing v:
- root-v-path preferred
- some older path pieces too



Auxiliary Tree

∘ represent each preferred path by a splay tree, keyed by depth

∘ root of this aux. tree stores a pointer to the path parent (pp)



"Basic" Operation (building block for all others)

access(v) (make root-v-path preferred)

⓪

① · splay v (within its aux. tree)

② · cut the preferred path below v
v.right.pathparent = v
v.right.parent = none

go up the tree of aux trees

· until v.pathparent = none

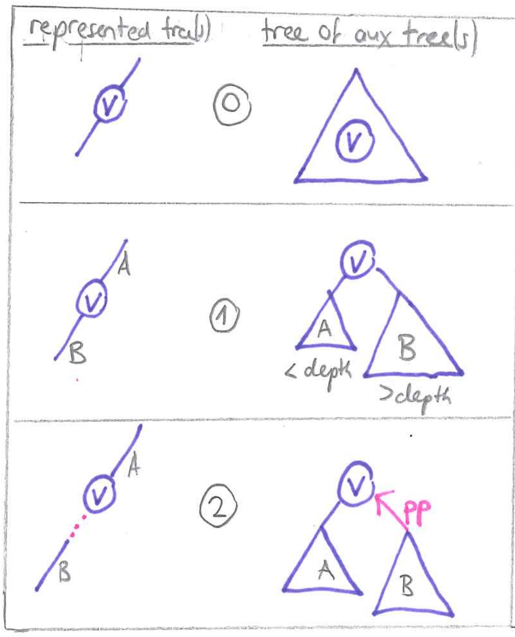
w = v.pathparent

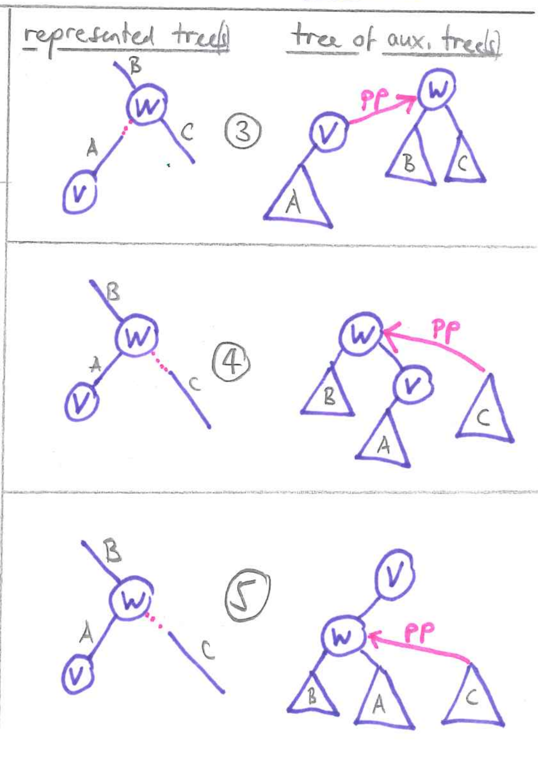
③ splay w

④ switch w's preferred child

⑤ splay v (rotate)

exit condition: v is the root of the tree of aux. trees

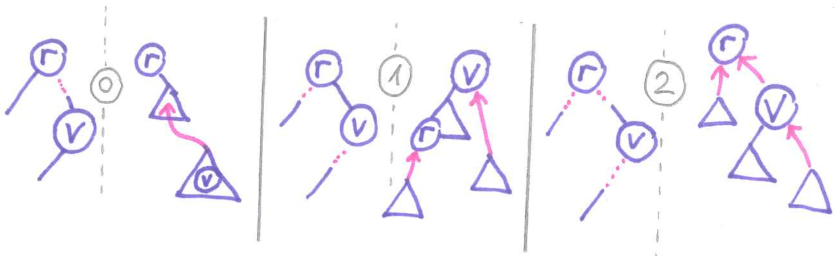




„Standard" Operations

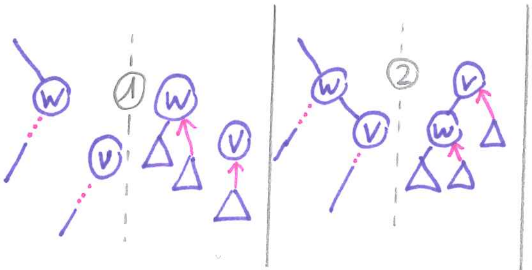
findroot(v)

- ⓪ ① • access(v) // root-v-path preferred
- • walk left to find r // minimum of aux. tree
- ② • access(r) // so that it is fast next time



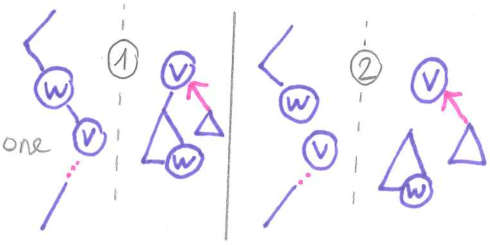
link(v, w) ← root of its tree

- ① • access(v)
- • access(w)
- ② • v.left = w
- w.parent = v



cut(v)

- ① • access(v)
- ② • v.left.parent = none
- v.left = none

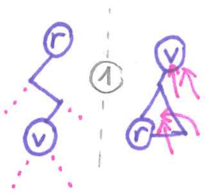


„Cool" Operations

path aggregate(v)

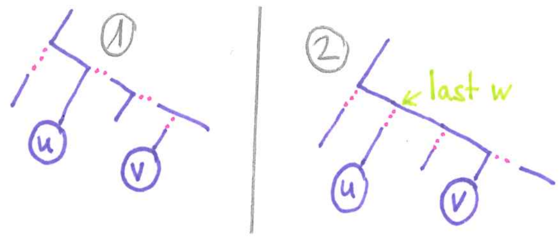
- ① • access(v)
- • return v.value

↑ subtree min/max/sum aggregate within the aux. splay tree (modify access, link, cut accordingly)



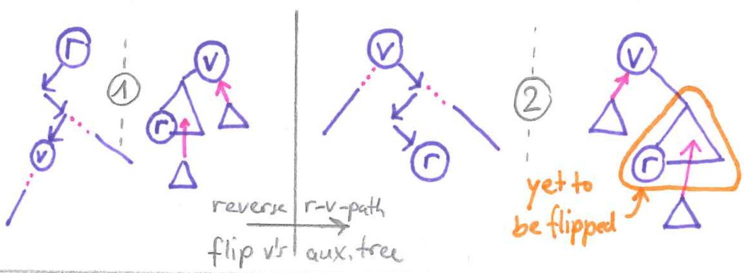
LCA(u, v)

- ① • access(u)
- ② • access(v)
- • return last w in the access-loop



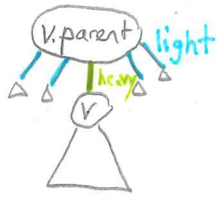
reroot(v) / evert(v) // reverse the direction of all the edges on the root-v-path

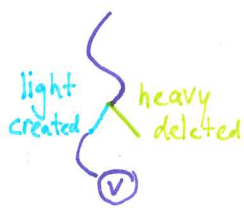
- ① • access(v)
- • swap(v.left, v.right) ← modify 'access' accordingly to first check for lazy bit.
- ② • flip lazy-reverse bit of v.right



Quick Analysis $O(\log^2 n)$ per operation

- #splays = #preferred child changes + m $\in O(m \log n)$
- #changes $\leq$ #light pref. edges created + #heavy pref. edges deleted + $(n-1)$ ← heavy pref. created but never deleted
- Heavy-light decomposition (useful tool for unbalanced trees)
  - (v.parent, v) is heavy iff $size(v) > \frac{1}{2} size(v.parent)$ (w.r.t. represented tree)
  - $lightdepth(v) \leq \log_2(n)$
- check the operations
  - access(v): weights unchanged, only preferredness
    - new root-v-pref. path: $\leq \log n$ light edges
    - deleted pref edges: $\leq \log n$ heavy edges
  - link(v, w): weight of root-w path increases
    - new pref. heavy & non-pref light → 0 changes
  - cut(v): root-v path lightens
    - $\leq \log n$ light pref. edges on this path created
    - ≤ 1 heavy pref. edge (v.parent, v) deleted





Continuations

- $O(\log n)$ amortization
- $O(\log n)$ worst case
- applications: Dinic in $O(nm \log n)$
- Dyn. connectivity on general graphs
- Tango trees