Today: Dynamic graphs I (of 3)
- link-cut trees
- preferred paths (again) [LG]
- heavy-light decomposition

Link-cut trees: [Sleator & Tarjan – JCSS 1983; Tarjan – book]
maintain forest of rooted (unordered) trees subject to $O(\lg n)$-time operations:
- maketree: return new vertex in new tree
- $\text{link}(v, w)$: make $v$ new child of $w$
  $\Rightarrow$ adding edge $(v, w)$
- $\text{cut}(v)$: delete edge $(v, \text{parent}(v))$
- $\text{findroot}(v)$: return root of tree containing $v$
- $\text{path aggregate}(v)$: compute sum/min/max/etc.
of node/edge weights on $v$-to-root path

Idea: represent unbalanced trees using balanced trees
Preferred path decomposition: (like Tango trees [16])
- preferred child of node v:
  = \{ w \text{ if last access was in child w's subtree} \\
  \text{none if last access in v's subtree was } v \}
- preferred path = chain of preferred edges
\Rightarrow partition represented tree into paths

Auxiliary trees: (also like Tango trees [16])
represent each preferred path by a Splay tree keyed on depth
- root of aux tree stores path parent: path's top node's parent in represented tree (can't easily store path children ~can be many)
- auxiliary trees + path parent pointers = tree of auxiliary trees
  - potentially high degree
  - goal: balanced
\text{access}(v): \text{make root-to-}v\text{ path preferred} \\
& \text{make } v \text{ the root of its aux. tree} \\
\Rightarrow v \text{ is the root of tree of aux. trees}

- splay } v (\text{within its aux. tree})
- remove } v \text{'s preferred child:}
  - v. \text{right.pathparent} = v \\
  \quad \text{.parent} = \text{none}
- v. \text{right} = \text{none}
- \text{until } v. \text{pathparent} = \text{none: (i.e. root aux. tree)}
  - w = v. \text{pathparent}
  - splay } w (\text{within its aux. tree})
  - switch } w \text{'s preferred child to } v:
    \text{if }
    - w. \text{right.pathparent} = v \\
    \quad \text{.parent} = \text{none}
    - w. \text{right} = v
    - v. \text{parent} = w \\
    \quad \text{.pathparent} = \text{none}
  - \text{splay } v = \text{rotate } v
\Rightarrow v. \text{pathparent} = w. \text{pathparent}

\Rightarrow v \text{ has no right child}
(\text{deepest node on preferred path}
\because v \text{ has no preferred child})
findroot(v):
  - access(v)
  - v = v.left until v.left = none
  - splay v \rightarrow so fast next time
  - return v

path aggregate(v): (for vertex weights)
  - access(v)
  - return v.subtree sum
    augmentation within each aux. tree

Cut(v):
  - access(v)
  - v.left, parent = none
  - v.left = none
    new root \rightarrow
    of tree of aux. trees

link(v, w):
  - access(v)
  - access(w)
  - v.left = w
  - w.parent = v
  \Rightarrow v becomes deepest node in w's preferred path
  [or w.right = v \sim similar analysis]
$O(lg^2 n)$ amortized bound:
- link & cut & path-aggregate cost $O(1 + \text{access})$
- findroot costs access + find/splay min
- access costs splay \cdot \# \text{preferred child changes}
- \textbf{Lemma} : splay analysis works in this setting (or use balanced BSTs)

$\Rightarrow O(lg n)$ amortized/splay

$m$ operations cost:
$O(lg n) \cdot (m + \text{total } \# \text{preferred child changes})$

\textbf{Claim} : $O(m \cdot lg n)$

- for this, need a tool:

\underline{Heavy-light decomposition} (in represented tree):
- $\text{size}(v) =$ \# nodes in $v$'s subtree
- call edge $(v, \text{parent}(v))$:
  - heavy if $\text{size}(v) > \frac{1}{3} \text{size}(\text{parent}(v))$
  - light otherwise

$\Rightarrow \leq 1$ heavy child of a node

$\Rightarrow$ heavy edges form heavy paths
  which partition the nodes

- $\text{light depth}(v) =$ \# light edges on root-to-$v$ path
  $\leq lg n$ (size halves each time)

$\Rightarrow$ represented edge can be (preferred) \& (heavy) not (light)
\(O(m \lg n)\) preferred child changes:
- \#changes \leq \# light preferred edge creations + \# heavy preferred edge destructions + \(n-1\)
  \#edges \sim \text{in case created & not destroyed or destroyed & not created}

- \text{access}(v):
  - creates preferred edges along root-to-v path
  - \(\leq \lg n\) of them can be light
  - each heavy preferred edge destroyed \(\geq \frac{3}{2} \lg n\)
    \Rightarrow \text{light preferred edge created}
    \ldots \text{except former preferred child of } v \geq \frac{3}{2} 1
  \Rightarrow \leq \lg n + 1
  \Rightarrow O(\lg n) \text{ total}

- \text{link}(v, w): \text{"heavens" nodes on root-to-w path}
  \Rightarrow \text{some of these edges might become heavy}
  \& \text{some edges off path might become light}
  (\Rightarrow \text{create light edges & destroy heavy edges})
  - but former preferred & latter not, by access
  \Rightarrow \emptyset

- \text{cut}(v): \text{lightens nodes on root-to-v path}
  - \(\leq \lg n\) of path edges can be\(\text{come}\) light
  - also destroy edge \((v, \text{parent}(v))\), possibly heavy
  \Rightarrow O(\lg n)
$O(lg n)$ amortized bound:
- $W(v) = \# \text{ nodes in } v\text{'s subtree in tree of aux. trees}$
- $= \sum_{w \text{ in } v\text{'s subtree in } v\text{'s aux. tree}} (1 + \text{size(aux. trees hanging off } w))$
- potential $\Phi = \sum_{v} lg W(v) \sim \text{splay potential}$
- access lemma: amortized cost of splay($v$)
  $\leq 3(lg W(\text{root of } v\text{'s aux. tree}) - lg W(v)) + 1$
- splay($v$) affects $W$'s only within $v$'s aux. tree
  $\Rightarrow$ standard splay analysis applies:
    - amortized cost of one splay step
      $\leq 3(lg W_{\text{after}}(v) - lg W_{\text{before}}(v))$
      (some checking & concavity of lg)
  $\Rightarrow$ telescopes, +1 for final rotation
- amortized cost of access($v$)
  $= O(lg n) + O(\# \text{ preferred child changes})$
  $\Rightarrow O(lg n)$ amortized
- changing preferred children doesn't affect $W$ (tree of aux. trees remains the same)
- $W(v) \leq W(\text{root of } v\text{'s aux. tree}) \leq W(w)$
- splay($v$) costs $\leq 3(lg W(w) - lg W(v)) + 1$
- sum telescopes
  $\Rightarrow \leq 3(lg W(\text{root}) - lg W(v)) + O(\# \text{ preferred child changes})$
  $O(lg n)$
- cut($v$) only decreases $W$'s $\Rightarrow \Phi$ only decreases
- link($v, w$) increases only $W(v)$, by $\leq n$
  $\Rightarrow \leq lg n$ increase in $\Phi$
Worst-case $O(\log n)$: [Sleator & Tarjan]
- store heavy paths in aux. trees
- aux. tree = globally biased search tree
  [Bent, Sleator, Tarjan - SICOMP 1985]
- similar to weight-balanced trees in L16
  but dynamic with careful split/concat.