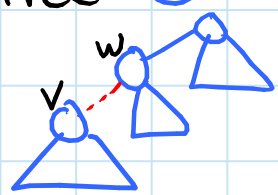
6.851	Lecture 19	May 1, 2012

TODAY: Dynamic graphs I (of 3)
- link-cut trees
- preferred paths (again) [L6]
- heavy-light decomposition

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

maintain forest of rooted (unordered) trees subject to $O(\lg n)$-time operations:
- maketree: return new vertex in new tree
- link(v, w): make v new child of w
  ⇒ adding edge (v, w)



- cut(v): delete edge (v, parent(v))
- findroot(v): return root of tree containing v
- 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 [L6])
- preferred child of node v: differs
  $= \begin{cases} \text{none} & \text{if last access in v's subtree was v} \\ w & \text{if last access was in child w's subtree} \end{cases}$
- preferred path = chain of preferred edges

$\Rightarrow$ partition represented tree into paths

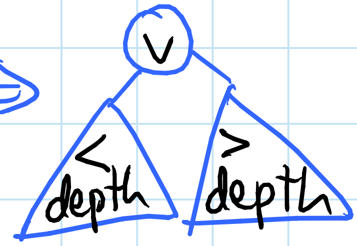
Auxiliary trees: (also like Tango trees [L6])

represent each preferred path by a splay tree keyed on depth

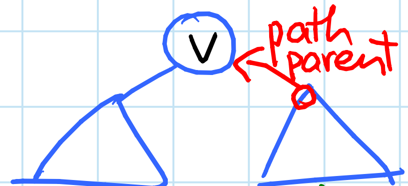
- root of aux. tree stores path parent: path's top node's parent in represented tree (can't easily store path children ~ can be many)
- auxiliary trees + path parent pointers = tree of auxiliary trees
  - potentially high degree
  - goal: balanced

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

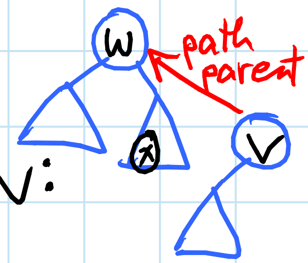
- splay v (within its aux. tree) $\Rightarrow$



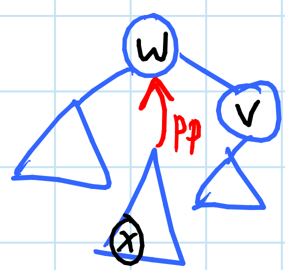
- remove v's preferred child:
  - if v.right { - v.right.pathparent = v
    - .parent = none }
  - v.right = none $\Rightarrow$



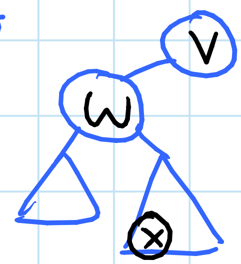
- until v.pathparent = none: (i.e. root aux. tree)
  - w = v.pathparent
  - splay w (within its aux. tree) $\Rightarrow$



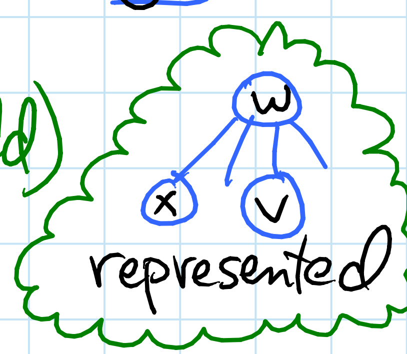
  - switch w's preferred child to v:
    - if w.right { - w.right.pathparent = v
      - .parent = none }
    - w.right = v
    - v.parent = w
      - .pathparent = none $\Rightarrow$



  - splay v = rotate v $\Rightarrow$
    $\Rightarrow$ v.pathparent = w.pathparent

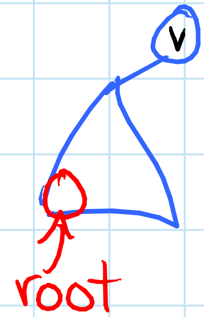


$\Rightarrow$ v has no right child
(deepest node on preferred path because v has no preferred child)



findroot(v):

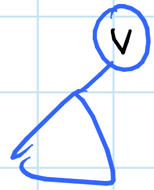
- access(v)
- v = v.left until v.left = none
- splay v → so fast next time
- return v



path aggregate(v): (for vertex weights)

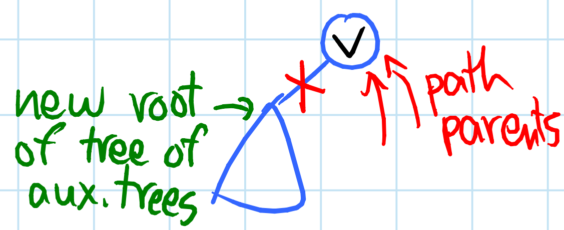
- access(v)
- return v. $\underbrace{\text{subtree sum}}$

augmentation within each aux. tree



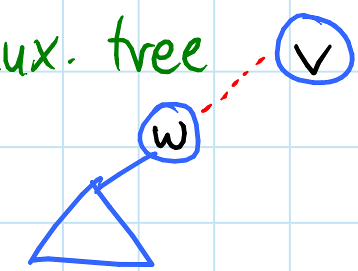
cut(v):

- access(v)
- v.left.parent = none
- v.left = none



link(v, w): (v: root)

- access(v) ⇒ v alone in its aux. tree
- access(w)
- v.left = w
- w.parent = v



⇒ v becomes deepest node in w's preferred path

[OR w.right = v ~ 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 · # preferred child changes
- lemma: splay analysis works in this setting
  (or use balanced BSTs)

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

$\Rightarrow$ m operations cost

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

claim: $O(m \lg n)$

– for this, need a tool:

Heavy-light decomposition: (in represented tree)

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

$\Rightarrow$ ≤ 1 heavy child of a node

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

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

$\Rightarrow$ represented edge can be $\binom{\text{preferred}}{\text{not}}$ & $\binom{\text{heavy}}{\text{light}}$

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

- #changes $\le$ # light preferred edge creations
  \+ # heavy preferred edge destructions
  \+ $\underbrace{n-1}_{\text{\# edges}}$ ~ in case created (heavy) & not destroyed or destroyed (light) & not created

- access(v):
  - creates preferred edges along root-to-v path
  - $\le \lg n$ of them can be light
  - each heavy preferred edge destroyed $\Rightarrow$ light preferred edge created } $\lg n$
    ... except former preferred child of v } 1

  $\Rightarrow \le \lg n + 1$

  $\Rightarrow O(\lg n)$ total

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

  $\Rightarrow \emptyset$

- cut(v): lightens nodes on root-to-v path
  - $\le \lg n$ of path edges can be(come) light
  - also destroy edge (v, parent(v)), possibly heavy

  $\Rightarrow O(\lg n)$

$O(\lg n)$ amortized bound:

- $W(v)$ = # nodes in v's subtree in tree of aux. trees
  $= \sum_{\substack{w \text{ in } v\text{'s subtree} \\ \text{in } v\text{'s aux. tree}}} (1 + \text{size(aux. trees hanging off } w))$
- potential $\Phi = \sum_v \lg W(v)$ ~ splay potential
- <u>access lemma</u>: 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 $\leq 3(\lg \underbrace{W(w)}_{W(v) \text{ in next splay}} - \lg W(v)) + 1$
  - sum telescopes
  - $\Rightarrow \leq \underbrace{3(\lg W(\text{root}) - \lg W(v))}_{O(\lg n)} + O(\text{\# preferred child changes})$
- 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 Φ

Worst-case $O(\lg n)$: [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.