Lecture 16

**Dynamic Trees (Link Cut Tree)**

- Data structure to represent a forest of rooted trees

**Operations:**

1. **MAKE-TREE:** Returns a new vertex as a single-vertex tree.
2. **LINK(u, w):** Links the trees of u and w by making u a child of w.
3. **CUT(u):** Remove the edge (u, parent of u) from the tree.
4. **FIND-ROOT(w):** Return the root of w's tree.

*References: [Sleator, Tarjan '83]*
* A vertex was accessed if it was an argument of some OPN.

* Every tree of the forest is represented by a tree in the data structure.

  ▼
  
  Preferred nodes in this case are paths of the original tree stored as Splay-trees (key $v$ = depth of $v$).

  * A preferred child of a node $n$ in the original tree is the child that contains the last accessed node in its subtree within the subtree below $n$.

  ▼
  
  Last access

  preferred child

  * A preferred edge: edge between $n$ and its preferred child.

  * A preferred path: maximal connected component of preferred edges.
Example:

Every pretend path to a node in the ds-tree is a splay tree keyed by depth, which is the auxiliary tree.

- For every auxiliary tree we store a parent-path pointer in the root of the auxiliary tree.
For the example (before accessing)

- Analogous to TANNOID TREES but not binary and not balanced
- CUT/LINK/FIND-ROOT share a common
  scan routine: ACCESS (r)
  assures that the ds-tree is correct

How does it work?:
- Place in the root in its auxiliary tree
- Update the preferred edges along root ds-tree
ACCESS(m):
(1) Splay m in its auxiliary tree
   (update path/parent for auxiliary) m
(2) Remove u's preferred child:
   (2.1) access
         path/parent (right(u)) = m
   (2.2) right(u) = NULL ? MAKE
        parent (right(u)) = NULL
(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) = w
        parent(w) = w
   (3.5) path/parent (w) = NULL
   (3.6) u = w
(4) SPLAY (w)
**FIND-ROOT**(v)

1. **ACCESS**(v)
2. Retrieve the smallest entry of the auxiliary tree of the ds-tree - root (aux-tree contains root if root has min depth) 
   
   \( v \leq \text{left}(v) \), until left(v) = NULL
3. Return v
4. Splay(v)

**CUT**(v)

1. **ACCESS**(v)
2. left(v) \leq NULL
   
   \{ Make \# parent(left(v)) \leq NULL \}

**LINK**(v,w)

1. **ACCESS**(w) \( \leq v \) alone in its aux. tree
2. **ACCESS**(w) \( \leq w \) root of its aux. tree
3. Make \( \{ \text{left}(w) \leq \# \}
   
   \{ \text{parent}(w) \leq w \} \)
ANALYSIS

- Link, cut: Access + O(1)
- FIND-ROOT: Access + find min + split
- ACCESS: # of child changes
- Splay costs: Splay analysis works in tree setting

O(log n) \cdot (m + total # of pref child changes)

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

Heavy-light decomposition

- \text{size}(v) = \text{# nodes in } v's \text{ subtree}
- \text{edge}(u,v) \text{ is heavy, if } (u \text{ is parent})
\quad \text{size}(v) \geq \frac{1}{2} \text{ size}(u)
\quad \text{otherwise light}
- \leq 1 \text{ heavy edge/ node}
- \leq \log(n) \text{ light edges on a path to root}
• # of sprays = # of preferred child changes
  = # of pushed edge constructions

• # of light preferred child creations
  \leq O(\log u) \quad (\text{because we have } \leq \log u \text{ light edges on a root } \rightarrow \text{leaf path})

• # of heavy edge construction
  \leq # of heavy edge destruction + 1
    \quad \text{(every other created edge short & en implies an destruction in prior)}

• destruction of a heavy edge as pc
  \Rightarrow \text{construction of light edge as p}

  \text{only } O(\log u) / \text{Access}

  \Rightarrow \text{total costs } O(\log u) / \text{Access}

\text{bottom line: # of light preferred edge creations / by}
\quad \text{gives the } \frac{1}{50} \quad C \leq O(\log u)$
Do cut/cut interfere?

Cut: lightens ancestors of \( v \)

\[ \rightarrow \text{creation of log}_4 \text{ light pref. edges} \]

(link (W,W)) • heavy nodes on root path, lightens nodes hanging

• these edges

edges gets lighter \( \Rightarrow \)

because they being of root - there are not preferred

\[ \rightarrow \text{ACCESS (W) was executed} \]

• no creation of pc. lighter

\[ \Rightarrow \text{total: } O(\log^2 n) \text{ time worthwhile?} \]

**IMPROVEMENT**

Rewards SP and trees:
**ACCESS Theorem**

\[ W_i = \text{weights for Splay tree nodes} \]

\[ S(x) = \sum_{i : i \text{ is in subtree rooted at } x} \]

\[ \Phi = \sum_{x} \log(x) \quad \text{Potential} \]

**Theorem**

The amortized cost of accessing \( x \)

\[ \leq 3 \left( \log S_{\text{new}}(x) - \log S_{\text{old}}(x) \right) \]

*Condition: \( \max \Phi - \min \Phi \) is small, which is true for general splay trees.*

*Observation 1: A splay only changes the \( S(x) \) in the aux.

\( x \rightarrow \text{Splay tree analysis holds (we refer to } S(x) \text{ to the auxiliary tree.)} \)
Observation 2
\[
S_{\text{new}}(x) = S_{\text{old}}(w)
\]
\[
\Rightarrow \text{costs} \leq 3 \left( \log S(w) - \log S(x) \right)
\]
(amortized for accessing x)

Observation 3
- Assume that we splay \( x_1, x_2, \ldots, x_k \) during an access
\[
\Rightarrow S(x_1) \leq S(x_2) \leq S(x_3) \leq \ldots
\]

Observation 4
- Changing the preferred children does not change the tree of aux trees
- Does not change \( S(x) \) values

Access

\[ \xrightarrow{\text{splay}} \quad \xrightarrow{\text{splay}} \quad \xrightarrow{\text{splay}} \quad \xrightarrow{\text{splay}} \quad \xrightarrow{\text{splay}} \quad \xrightarrow{\text{access cost}} \]

\[ \text{tree of aux trees} \]
\[ \text{access cost} \leq 3 \log S(\text{root}) + 1 \]

- In order to make the amortization work, we have to show that
\[ S(\text{root}) < O(\log n) \]

- this holds for the access open because we only changed inside
  1. Auxiliary tree \to\text{usual splay analysis}
\[(\text{cut}(v)) : \text{decreases } S(x)\]

\[\therefore \Phi \text{ decreases}\]

\[(\text{join}(v,w)) : \text{increase only } S(w)\]

\[\text{by } \leq w\]

\[\Rightarrow \Phi \text{ increases by } \leq \log w\]

**Start configuration:**

- Singlets

\[\therefore \Phi = \sum \log (\cdot)\]

\[= \mathcal{O}\]

**End configuration:**

- \[\Phi \leq \log \text{ of } \text{end joins}\]

\[\Rightarrow \# \text{ of end joins}\]

\[\Rightarrow \text{we can change } \Phi_{\text{max}} - \Phi_{\text{min}}\]

to the joins

- each join works \(= O(\log w)\)

An amortized