Augmented Data structures, and Dynamic Order Statistic

Original slides courtesy of Erik Demaine and Carola Wenk

Dynamic order statistics

OS-SELECT\((i, S)\): returns the \(i\) th smallest element in the dynamic set \(S\).

OS-RANK\((x, S)\): returns the rank of \(x \in S\) in the sorted order of \(S\)’s elements.

(\(\text{the rank of the smallest element is 1}\))

IDEA: Use your favorite balance search tree (AVL, Red-Black, B-Trees, 2-3 trees, SkipList etc etc) for the set \(S\), but keep subtree sizes in the nodes.

Notation for nodes:

\[
\text{key} \quad \text{size}
\]

Example of an OS-tree

\[
\text{size}[x] = \text{size}[\text{left}[x]] + \text{size}[\text{right}[x]] + 1
\]
Selection

Implementation trick: Use a sentinel (dummy record) for NIL such that size[NIL] = 0.

OS-SELECT(x, i) \( \triangleright \) returns the \( i \)th smallest element in the subtree rooted at \( x \)

\[ k \leftarrow \text{size}[\text{left}[x]] + 1 \quad \triangleright \quad k = \#\text{keys} \leq x \text{ in the subtree rooted at } x \]

if \( i = k \) then return \( x \)

if \( i < k \)
    then return OS-SELECT( left[x], i )
    else return OS-SELECT( right[x], i - k )

\( \triangleright \) when turning right, we skip \( k \) keys, all smaller than \( x \).

Implementation trick:

Use a sentinel (dummy record) for NIL such that size[NIL] = 0.

Example

OS-SELECT(root, 5)

Running time = \( O(h) = O(\log n) \) for a balanced tree, since this is the height of the tree.

Finding the rank of \( x \)

OS-Rank(\( T, x \))

\( \triangleright \) Assume \( x \) already found, and the path from the root is known

\[ r \leftarrow \text{size}[\text{left}[x]] + 1 \]

\( \triangleright \) Recall that if \( \text{left}[x] \) is NIL then its size=0

\[ y \leftarrow x \]

While ( \( y \neq \text{root}(T) \) ) {
    do if \( y \) is the right child of \( \text{parent}[y] \)
        then \( r \leftarrow \text{size}[\text{left}[\text{parent}[y]]] + 1 \)
    \[ y \leftarrow \text{parent}[y] \]
}

Return \( r \)
**Data structure maintenance**

**Q.** Why not keep the ranks themselves in the nodes instead of subtree sizes?

**A.** They are hard to maintain when the tree is modified.

**Modifying operations:** INSERT and DELETE.

**Strategy:** Update subtree sizes when inserting or deleting.

---

**Example of insertion**

**INSERT(“K”)**

---

**Handling rebalancing**

If balancing is done via rotation, INSERT and DELETE may also need to modify the tree in order to maintain balance.

*Rotations:* fix up subtree sizes in $O(1)$ time.

**Example:**

$C \rightarrow E$

$7 \rightarrow 3 \rightarrow 4$

$7 \rightarrow 16 \rightarrow 4$

$7 \rightarrow 16 \rightarrow 3$

$7 \rightarrow 3 \rightarrow 4$

$\therefore$ INSERT and DELETE still run in $O(\log n)$ time.
Data-structure augmentation

**Methodology:** (e.g., order-statistics trees)

1. Choose an underlying data structure (red-black trees).
2. Determine additional information to be stored in the data structure (subtree sizes).
3. Verify that this information can be maintained for modifying operations (RB-INSERT, RB-DELETE — don’t forget rotations).
4. Develop new dynamic-set operations that use the information (OS-SELECT and OS-RANK).

These steps are guidelines, not rigid rules.