Augmenting Data Structures

- Sometimes a “textbook” data structure is sufficient to solve a problem exactly as it is
- However, there will be times when augmenting an existing data structure by adding more data will be required
- Rarely will you invent a brand new data structure
Dynamic Order Statistic

- OS-SELECT(S, i):

- OS-RANK(S, x):

Example

- S: {6, 3, 74, 23, 84, 8, 19, 21}
- What’s the result of OS-SELECT(S, 4)
- What’s the result of OS-RANK(S, 23)

Order Statistics

- We have previously seen that any order statistic can be determined in O(n) from an unordered set
- How?
- Today we'll speed this up to O(lg n) time
Idea

- Augment a red-black tree
- The red-black tree will represent the set
- The size of every subtree will be stored in the node
- Notation for nodes

key
size

Order Statistic Tree

- Example

- $x.size = x.left.size + x.right.size + 1$
- $T.nil.size = 0$
OS-SELECT(x, i)

OS-SELECT(x, i)
\[ r = x . \text{left.size} + 1 \]
if \( i == r \)
\[ \text{return } x \]
else if \( i < r \)
\[ \text{return } \text{OS-SELECT}(x . \text{left}, i) \]
else\text{ return } \text{OS-SELECT}(x . \text{right}, i - r)

Goal: Find \( i^{th} \) smallest element in subtree rooted at \( x \).

Rank \( r \) is number of nodes with keys less than \( x \) in walk of tree by key.

Example: OS-SELECT(T.root, 1) returns (a pointer to) the node with the smallest key in the tree

Example

• What’s the result of OS-SELECT(T.root, 17)?
Running Time

- What's the running time of OS-SELECT?

OS-Rank(T, x)

\[ r = x.left.size + 1 \]
\[ y = x \]

while \( y \neq T.root \)
  if \( y == y.p.right \)
    \[ r = r + y.p.left.size + 1 \]
  \[ y = y.p \]

return \( r \)

Idea: Only add to size if right child in family tree. If parent is right child, need to add need to add uncle’s size to running total. Walk up the tree by parent until \( y \) is \( T.root \).
Example

- What is the result of OS-RANK(T, 38)?

- What is the running time of OS-RANK?

Maintaining Subtree Sizes

- Can the sizes be efficiently maintained?
Your Turn

• OS-SELECT(T.root, 5) on the following tree
  • Note that you will need to calculate the sizes
• INSERT(“K”) into the tree

Methodology for Augmentation

1. Choose an underlying data structure
2. Determine additional information to be stored in the data structure
3. Verify that this information can be maintained for modifying operations
4. Develop new dynamic set operations that use the information
Interval Trees

- Goal: Maintain a dynamic set of intervals (closed), such as time intervals

- Query: for a given interval \( i = [i.low, i.high] \), find an interval \( i' \) in the set that overlaps \( i \) (i.e. intersects). Need:

Following the Methodology

1. Choose an underlying data structure
   - Red-black tree so that each node \( x \) is keyed on the low endpoint \( x.int.low \)

2. Determine additional information to be stored in the data structure
   - Store in each node \( x \) the largest value \( x.max \) in the subtree rooted at \( x \), as well as the interval \( x.int \) (that is \([x.int.low, x.int.high]\)) corresponding to the key
Example

- Inorder tree traversal?

Modifying Operations

3. Verify that this information can be maintained for modifying operation.
   - Insert: fix x.max's on the way down
     - x.max = max (x.int.high, x.left.max, x.right.max)
     - O(lg(n))
   - Rotation and fixup: O(1)
New Operations

4. Develop new dynamic set operations that use the new information

\[ \text{INTERVAL-SEARCH}(T, i) \]
\[ x = T.\text{root} \]
\[ \text{while } x \neq T.\text{nil} \text{ and } i \text{ does not overlap } x.\text{int} \]
\[ \quad \text{if } x.\text{left} \neq T.\text{nil} \text{ and } x.\text{left}.\text{max} \geq i.\text{low} \]
\[ \quad \quad x = x.\text{left} \]
\[ \quad \text{else } x = x.\text{right} \]
\[ \text{return } x \]

Example

- \text{INTERVAL-SEARCH}(T, [14, 16])
Another Example

- INTERVAL-SEARCH(T, [12, 14])