6.046 (1)

Lecture 11

Augmenting Data Structures
Dynamic Order Statistics
Interval Trees

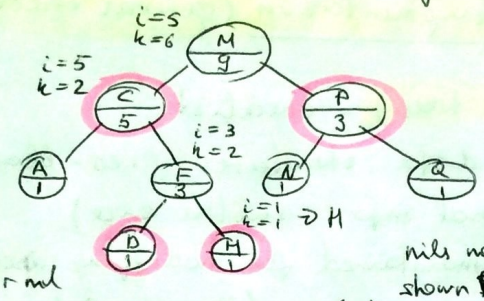
Dynamic order Statistics

the set is dynamic, not static ← inserts and deletes

OS-Select(i) – returns ith smallest item in dynamic set

OS-Rank(x) – returns rank of x in sorted order

Idea: keep subtree sizes in nodes of r-b tree.



A, C, D, F, H, M, N, P, Q

size[x] = size[left[x]] + size[right[x]] + 1

Trick: use sentinel (dummy record) for nil s.t. size[nil] = 0

nils not shown but are counted

```
OS-Select(x, i)  // ith smallest in subtree rooted at x
  k ← size[left[x]] + 1
  if i = k then return x    // k = rank(x)
  if i < k then return OS-Select(left[x], i)
           else return OS-Select(right[x], i-k)
```

my comment: building a tree = O(n lg n). given an array of numbers (static case) • Rand-select or BFPRT are Θ(n), in expectation and worst case respectively.

Ex: OS-Select(root, 5) → pointer to "H"

Analysis: O(lg n)

OS-Rank also O(lg n)

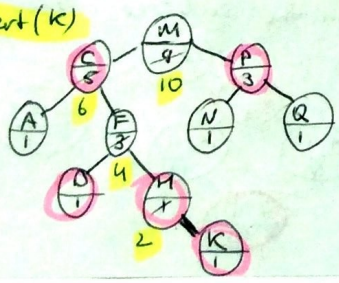
Q why not keep ranks in nodes instead of subtree sizes?

A → hard to maintain after modifications

Modifying ops: Insert, Delete

Strategy: Update subtree sizes when inserting or deleting

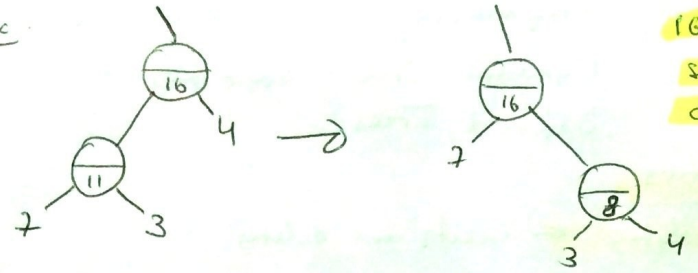
OS-Insert(K)



But must handle rebalancing

- r-b color changes: no effect
- rotations, look at children in nodes and fix up in O(1) time

Ex



16 → 16

size of subtree does not change after rotation

other properties (not subtree size) may change after rotation and require upward propagation.

Insert, Delete still $O(\lg n)$ time since $\Theta(1)$ rotations

Data Structure Augmentation (general checklist)

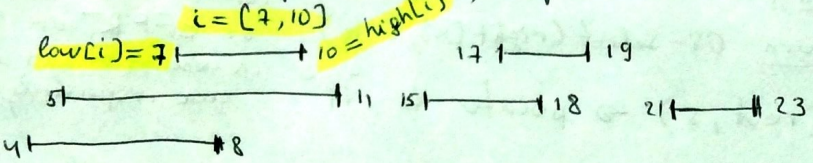
Methodology (Ex. OS trees) – checklist

1. Choose underlying data structure (red-black tree)
2. Determine additional info (subtree size)
3. Verify info can be maintained for modifying operations (Insert, Delete → rotations)
4. Develop new operations that use info (OS-Select, OS-Rank)

Usually, must play with interactions between steps.

Ex: Interval trees

Maintain a set of intervals, e.g. time intervals.

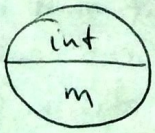


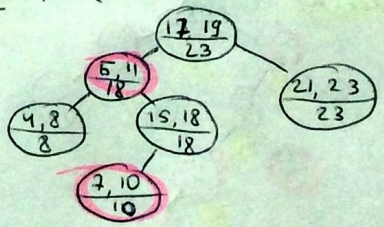
Query: Find an interval in the set that overlaps a given query interval. (have to return one interval)

Methodology:

1. red-black tree keyed on low endpoint
2. Store in node the largest value m in the subtree rooted at that node

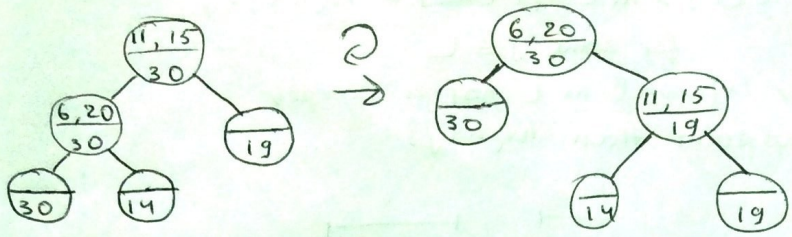
$$m[x] = \max \begin{cases} high[int[x]] \\ m[left[x]] \\ m[right[x]] \end{cases}$$





3. Modifying ops:

Insert: BST insert, fix m's on way down
But, also need to handle rotations.



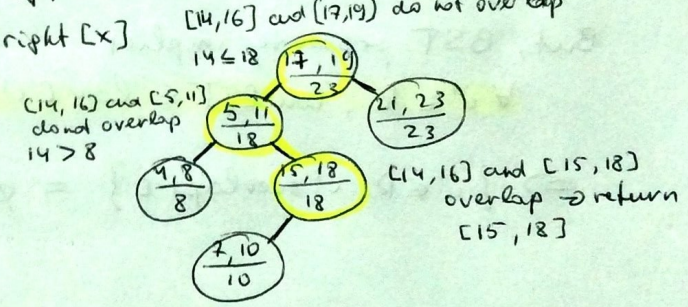
Fixing up m's during rotations takes O(1) time.
Insert time = $O(\lg n)$

Delete: similar

4. **Interval-Search (i)** // Finds an interval that overlaps i

```
x ← root
while x ≠ nil and (low[i] > high[int[x]]
                   or low[int[x]] > high[i])
    do // i and int[x] don't overlap, otherwise just return
        if left[x] ≠ nil and low[i] ≤ m[left[x]]
            then x ← left[x]
            else x ← right[x]
return x
```

Ex [14, 16] → [15, 18]
i
[12, 14] → nil



Time $O(\lg n)$

List all overlaps: $O(k \lg n)$ "output sensitive"

after finding, delete, put on [redacted] temp. storage then insert

Best to date $= O(k + \lg n)$

Correctness

Theorem Let $L = \{ i' \in \text{left}[x] \}$
$R = \{ i' \in \text{right}[x] \}$

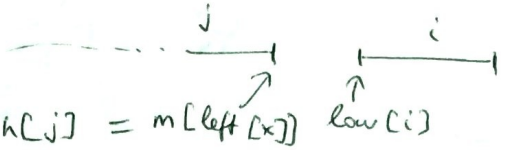
(1) - If search goes right, then $\{ i' \in L : i' \text{ overlaps } i \} = \emptyset$
(2) - If search goes left, then $\{ i' \in L : i' \text{ overlaps } i \} = \emptyset$
$\Rightarrow \{ i' \in R : i' \text{ overlaps } i \} = \emptyset$

Pf: Suppose search goes right.

① - If left[x] = nil, done since $L = \emptyset$

- Otherwise $low[i] > m[left[x]] = high[j]$, for some $j \in L$.

No other interval in L has a larger high endpoint than high[j].

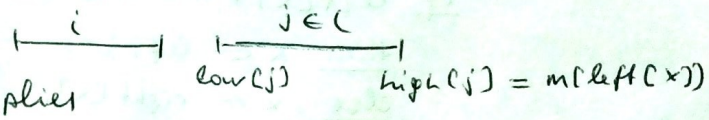


$\Rightarrow \{ i' \in L : i' \text{ overlaps } i \} = \emptyset$

② Suppose search goes left and $\{ i' \in L : i' \text{ overlaps } i \} = \emptyset$

Then, $low[i] \le m[left[x]] = high[j]$ for some $j \in L$.

Since $j \in L$, j does not overlap i $\Rightarrow high[i] < low[j]$



But, BST property implies

$\forall i' \in R$, $low[j] \le low[i']$ ← red-black tree keyed on low endpoint

$\Rightarrow \{ i' \in R : i' \text{ overlaps } i \} = \emptyset$