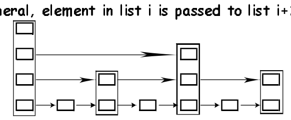
Skip List

- Simple data structure, easier to implement than red/black trees.
- Sorted linked list with "skip" pointers.
- Construction:
  - » Every element in the "bottom list" (list 0)
  - » Element passed to list 1 with probability 1/2.
  - » In general, element in list i is passed to list i+1 with prob. 1/2.



101

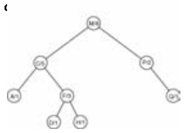
Skip Lists

- Analysis:
  - » Expected O(log n) lists.
  - » Expected total number of nodes is n+n/2+n/4+… = O(n).
  - » Search: find range in last list, examine this range next one down, etc.
  - » Each range is expected to be constant length: O(1) work per range, O(log n) total.
- Works only if deletions are not malicious.

102

Extending Red-Black Trees

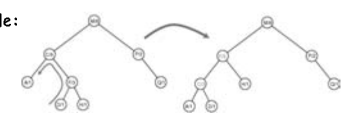
- Dynamic order statistics:
  - » Select(x,i) - i-th smallest in tree rooted
  - » Rank(T,x) - rank of x in tree T.
- New field: Size of subtree
- Idea: sufficient in order to know whether to go right or left.
- Ex: Rank: go down, each right - add size of left subtree +1. When elements found, add 1 + left subtree. For H in the figure: size(A)+1 +size(D)+1 +1+0 =5.



103

Dynamic selection continued

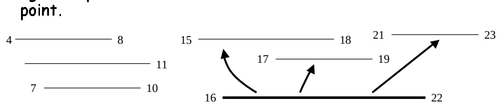
- In order to be able to claim O(log n) time, need to be able to update extra fields during:
  - » Insertion/deletion into red/black tree
  - » Rotations.
- Easy to see that this is trivial in this case: only local data needed.
- Example:



104

Interval intersection

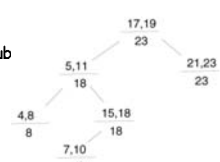
- Goal:
  - » Maintain collection of intervals (support deletion/insertion)
  - » Given an interval I, produce interval J from data structure, such that I intersects J.
- Ignore open vs closed, each interval starts at a new point.



105

Interval data structure

- Interval = Node
  Low endpoint = key
  New info: max endpoint in sub
- Easy to maintain during insertion/deletion/rotation



106

Using the data structure

```
• x=root(T)
  while x≠ NIL & I ∩ int(x)=∅
    if left(x) ≠ NIL && max(left(x))> low(I)
       then x = left(x)
       else x = right(x)
```

```
• Theorem: if overlap exists, and search goes left,
           then there exists an overlap on left.
    Same for “goes right”
```

```
• Proof:
  Case 1: goes right.
     Case 1a: overlap in right subtree - done.
     Case 1b: no overlap in right subtree.
       we went right - left(x)=NIL or max(left(x))<low(i)
       In both cases there can not be overlap on the left !!
```


Case 2

```
• Case 2: we go left.
    Case 2a: overlap on left - done.
    Case 2b: no overlap on left.
      we went left -- left(x) != NIL, max(left(x)) >= low(i)
      no overlap -- exists interval I’ s.t low(I’) > high(I)
      but tree ordered by low --- all intervals in right tree have lows
                                 above high(I) ! -- no overlap there.
```