CSE 326: Data Structures

Priority Queues: Leftist Heaps

Brian Curless

Spring 2008

New Heap Operation: Merge

Given two heaps, merge them into one heap

- first attempt: insert each element of the smaller heap into the larger.

  runtime: $O(n \log n)$ worst

  $O(n)$ average

- second attempt: concatenate binary heaps' arrays and run buildHeap.

  runtime: $O(n)$

Leftist Heaps

Idea:

Focus all heap maintenance work in one small part of the heap

Leftist heaps:

1. Binary trees
2. Most nodes are on the left
3. All the merging work is done on the right

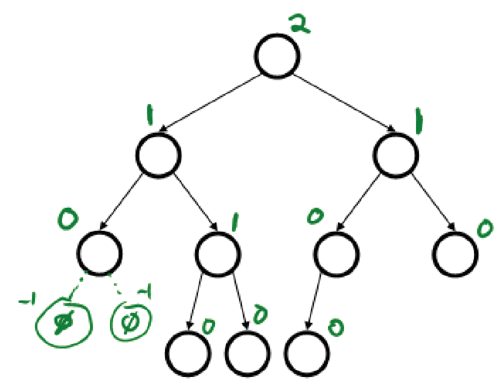
Definition: Null Path Length

null path length (npl) of a node x = the number of nodes between x and a null in its subtree

OR

npl(x) = min distance to a descendant with 0 or 1 children

- *npl*(null) = -1
- *npl*(leaf) = 0
- *npl*(single-child node) = 0



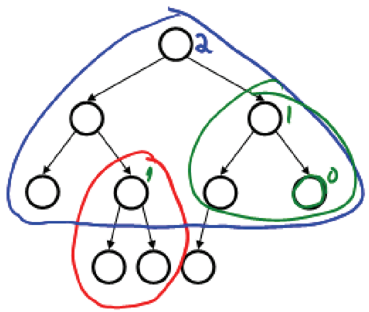
Equivalent definition:

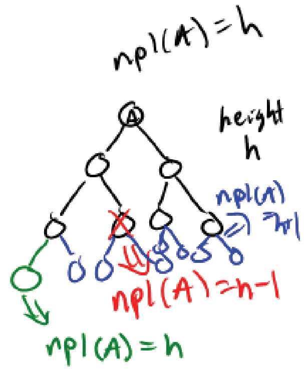
$npl(x) = 1 + \min\{npl(\text{left}(x)), npl(\text{right}(x))\}$

Definition: Null Path Length

Another useful definition:

npl(*x*) is the height of the largest perfect binary tree that is both itself rooted at *x* and contained within the subtree rooted at x.



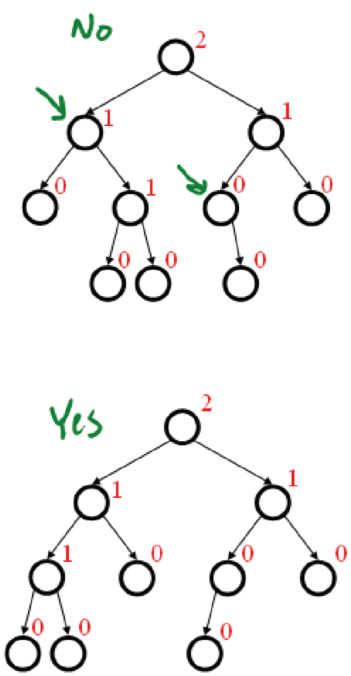


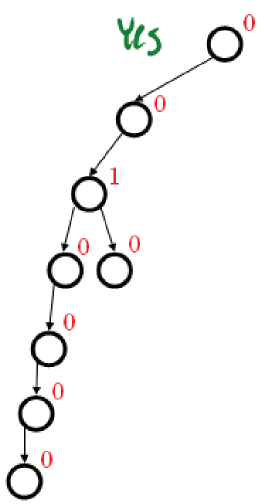
Leftist Heap Properties

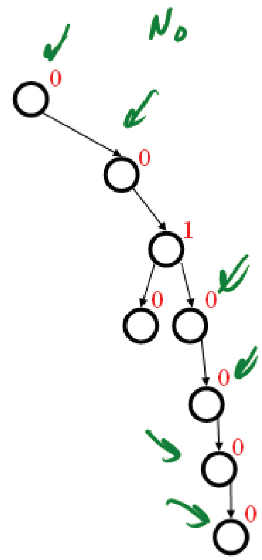
- Order property
  - parent's priority value is ≤ to childrens' priority values
  - <u>result</u>: minimum element is at the root
  - (Same as binary heap)
- Structure property
  - For every node *x*, *npl*(left(*x*)) ≥ *npl*(right(*x*))
  - <u>result</u>: tree is at least as "heavy" on the left as the right

(Terminology: we will say a leftist heap's tree is a leftist tree.)

Are These Leftist?







Observations

Are leftist trees always…
- complete? No
- balanced? NO!

Consider a subtree of a leftist tree…
- is it leftist? Yes

Right Path in a Leftist Tree is Short (#1)

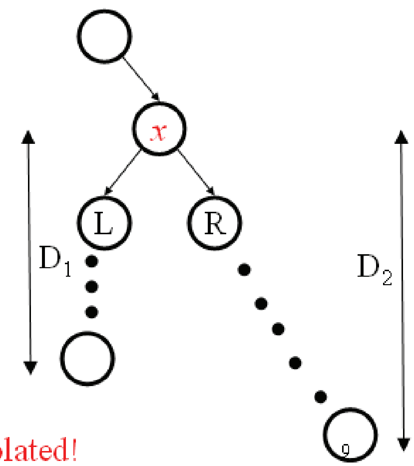
Claim: The right path (path from root to rightmost leaf) is as short as *any* in the tree.

Proof: (By contradiction)

Pick a shorter path: $D_1 < D_2$
Say it diverges from right path at x

$npl(L) \leq D_1-1$ because of the path of length D_1-1 to null

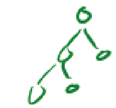
$npl(R) \geq D_2-1$ because every node on right path is leftist



Leftist property at x violated!

Right Path in a Leftist Tree is Short (#2)

Claim: If the right path has `r` nodes, then the tree has at least `2^r-1` nodes.



Proof: (By induction)

Base case : `r=1`. Tree has at least $2^1-1 = 1$ node

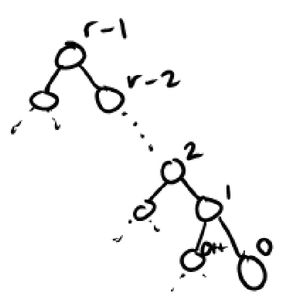
Inductive step : assume true for `r-1`. Prove for tree with right path at least `r`.

1. Right subtree: right path of `r-1` nodes
   $\Rightarrow$ $2^{r-1}-1$ right subtree nodes (by induction)
2. Left subtree: also right path of length at least `r-1` (prev. slide)
   $\Rightarrow$ $2^{r-1}-1$ left subtree nodes (by induction)

$\Rightarrow$ Total tree size: $(2^{r-1}-1) + (2^{r-1}-1) + 1 = 2^r-1$

$npl(x) = 1 + \min\{npl(R), npl(L)\}$

Leftist: $npl(L) \geq npl(R)$



r nodes on right path

$npl(root) = r-1$

perfect bin. tree of $h=r-1$ below root

$2^{h+1}-1 \Rightarrow n \geq 2^r-1$

r is $O(\log n)$

Why do we have the leftist property?

Because it guarantees that:

- the *right path is really short* compared to the number of nodes in the tree
- A leftist tree of N nodes, has a right path of at most $\log_2(N+1)$ nodes

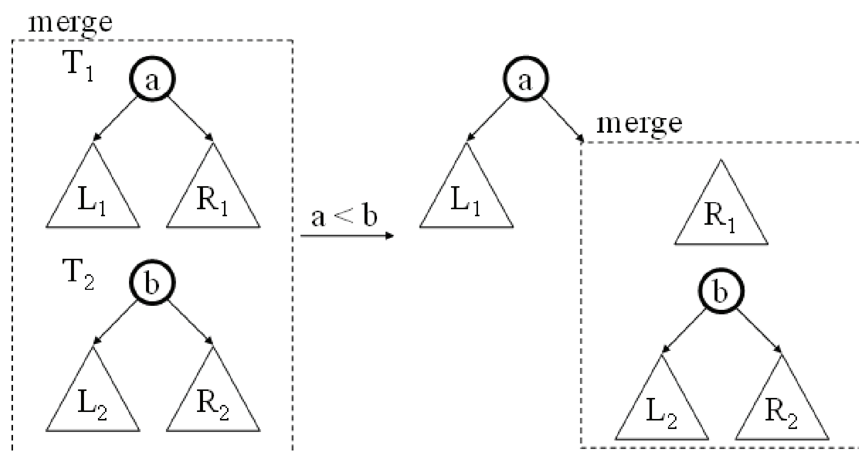
Idea – perform all work on the right path

Merge two Leftist heaps (basic idea)

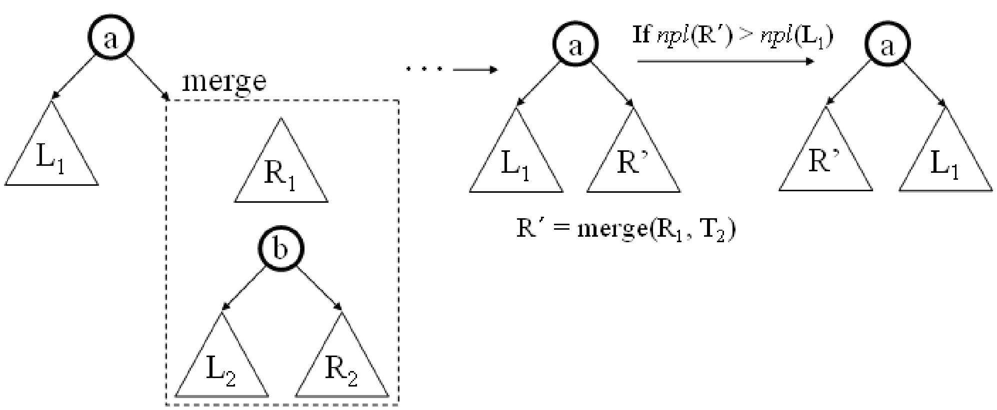
- Put the root with smaller value as the new root.
- Hang its left subtree on the left.
- Recursively merge its right subtree and the other tree.
- Before returning from recursion:
  - Update npl of merged root.
  - Swap left and right subtrees just below root, if needed, to keep leftist property of merged result.

Merging Two Leftist Heaps

Recursive calls to merge(T_1,T_2): returns one leftist heap containing all elements of the two (distinct) leftist heaps T_1 and T_2



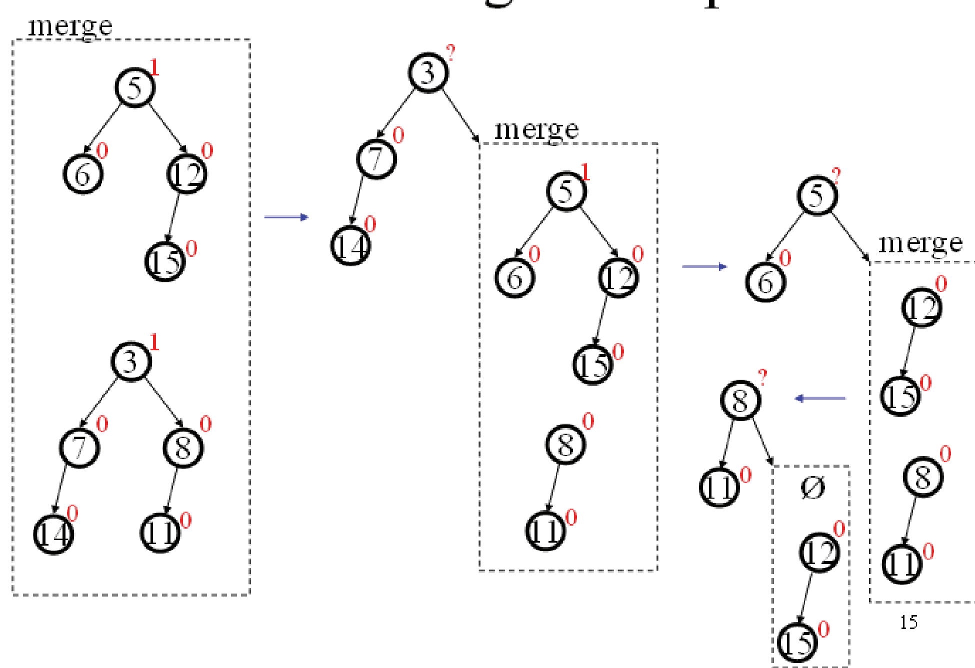
Merge Continued



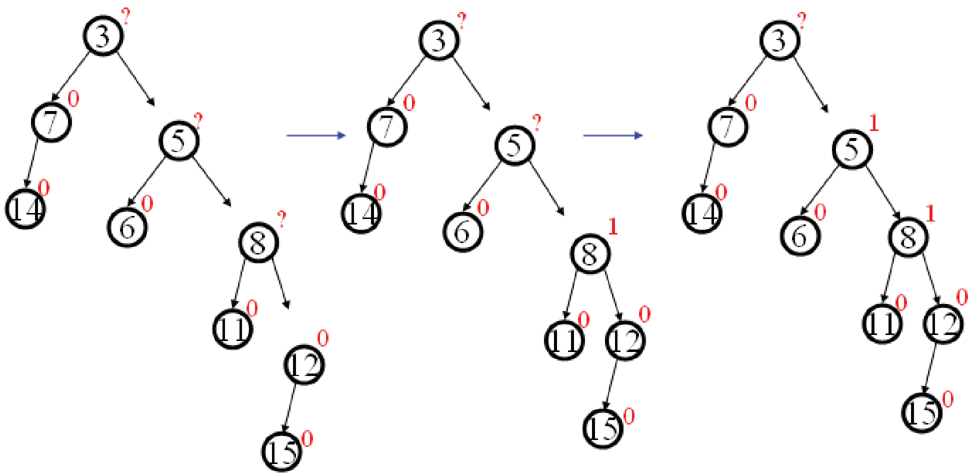
Note special case: merge(null, T) = merge(T, null) = T

runtime:

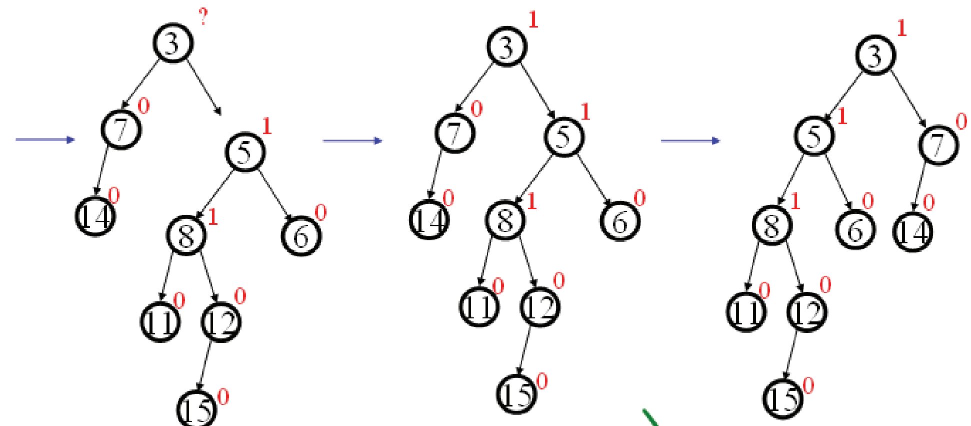
Leftest Merge Example



Sewing Up the Example



Sewing Up the Example



O(log n)

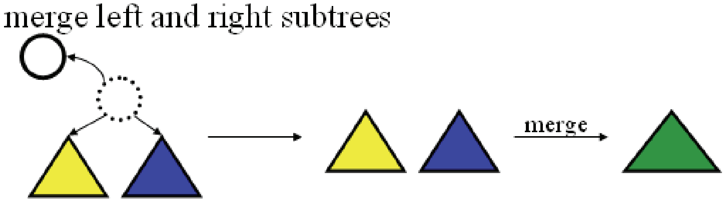
Other Heap Operations

- insert ?
- deleteMin ?

Operations on Leftist Heaps

- merge with two trees of total size n: O(log n)
- insert with heap size n: O(log n)
  - pretend node is a size 1 leftist heap
  - insert by merging original heap with one node heap

- deleteMin with heap size n: O(log n)
  - remove and return root
  - merge left and right subtrees



Leftist Heaps: Summary

<u>Good</u>

- One core operation: merge
- merge is $O(\log n)$

<u>Bad</u>

- Not array-based
- npl field, pointers