Today's Outline

- Finish Binary Heaps
- d-heaps from last week's notes
- Leftist Heaps

New Operation: Merge

Given two heaps, merge them into one heap:
1. First attempt: insert each element of the smaller heap into the larger.
   - runtime: 
2. Second attempt: concatenate binary heaps' arrays and run buildHeap.
   - runtime:

How about \( O(\log n) \) time?

Idea: Hang Heaps From a New Root

Idea:
- Focus all heap maintenance work in one small part of the heap

Leftist Heaps:
1. Most nodes are on the left.
2. All the merging work is done on the right.
**Definition: Null Path Length**

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

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

Equivalent definitions:
1. npl(x) is the height of the largest complete subtree rooted at x.
2. npl(x) = 1 + max{npl(left(x)), npl(right(x))}

**Leftist Heap Properties**

- Heap-order property
  - Parent's priority is at most children's priority
  - smallest inimum element is at the root

- Leftist property
  - For every node x, npl(left(x)) \(\geq\) npl(right(x))
  - A tree is at least as "heavily" on the left as the right.

**Are leftist trees complete? balanced?**

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

**Claim:** The right path is as short as any in the tree.

**Proof:** (By contradiction)

Pick a shorter path:
- D1 < D2
- npl(L1):
- npl(R1):

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 \(r' < r\). Prove for tree with right path of length 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 \(r-1\) (by previous slide)
   \(\Rightarrow\) \(2^{r-1} - 1\) left subtree nodes (by induction)

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

**Merging Two Leftist Heaps**

- m(merge(T1, T2)) returns one leftist heap containing all elements of the two (distinct) leftist heaps T1 and T2.
Let's do an example, but first...

Other Heap Operations

- Insert?
- DeleteMin?
- BuildHeap?

Operations on Leftist Heaps

- **merge** with two trees of total size \( n \): \( \Theta(\log n) \)
- **insert** with heap size \( n \): \( \Theta(\log n) \)
  - pretend node is a size 1 leftist heap
  - insert by merging original heap with one node heap
- **deleteMin** with heap size \( n \): \( \Theta(\log n) \)
  - remove and return root
  - merge left and right subtrees

buildHeap:
- options are
  1. Use Floyd's method
     - but need pointer-based implementation
     - unclear how to traverse right-to-left, bottom-up
  2. Do no inserts
     - Takes \( \Theta(n \log n) \) time
  3. Use merge in a smart way!
     - (Exercise in your next homework)

Merge Example

Sewing Up the Example

\[ \text{Done?} \]
Finally...

Iterative Leftist Merging

1. Downward pass: merge right paths

Iterative Leftist Merging

2. Upward pass: fix right path

What do we need to implement this iteratively?

Leftist Heaps: Summary

Good
•
•
•

Bad
•
•
•

To Do

• Continue project #1
• First written homework will be out Wednesday
• Finish reading chapter 6