Announcements

• Written HW #1 – due NOW
• Written HW #2 – out today, due next Friday
• Project #2 coming – Part A on Monday
  – Can work in pairs; start figuring out who you’d like to work with or whether you want to go alone
• Final exam – Thur. June 7.  8:30(!) am

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:

  – second attempt: concatenate binary heaps’ arrays and run buildHeap.

  runtime:
Leftist Heaps

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
null path length \( npl \) of a node \( x \) = the number of nodes between \( x \) and a null in its subtree
OR
\[ npl(x) = \min \text{ distance to a descendant with 0 or 1 children} \]

\( npl(\text{null}) = -1 \)
\( npl(\text{leaf}) = 0 \)
\( npl(\text{single-child node}) = 0 \)

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

Leftist Heap Properties
• Heap-order property
  – parent’s priority value is \( \leq \) to childreens’ priority values
  – result: minimum element is at the root

• Leftist property
  – For every node \( x \), \( npl(\text{left}(x)) \geq npl(\text{right}(x)) \)
  – result: tree is at least as “heavy” on the left as the right

Are leftish trees…
   complete?
   balanced?

Are These Leftist?

Every subtree of a leftist tree is leftist!
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: \( D_1 < D_2 \)
Say it diverges from right path at \( x \)

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

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

Leftist property at \( x \) violated!

4/15/07

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 at least \( r \).

1. Right subtree: right path of \( r - 1 \) nodes
\[ \Rightarrow 2^{r-1} - 1 \text{ right subtree nodes (by induction)} \]

2. Left subtree: also right path of length at least \( r - 1 \) (by previous slide)
\[ \Rightarrow 2^{r-1} - 1 \text{ left subtree nodes (by induction)} \]

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

4/15/07

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 \( \log(N+1) \) nodes

Idea – perform all work on the right path

4/15/07

Merge two heaps (basic idea)

• Put the smaller root as the new root,
• Hang its left subtree on the left.
• Recursively merge its right subtree and the other tree.

4/15/07
Merging Two Leftist Heaps

- `merge(T_1, T_2)` returns one leftist heap containing all elements of the two (distinct) leftist heaps `T_1` and `T_2`.

Let’s do an example, but first…

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
Leftest Merge Example

Leftest Heaps: Summary

Sewing Up the Example

Finally…
Random Definition: Amortized Time

Amortized time:
Running time limit resulting from “writing off” expensive runs of an algorithm over multiple cheap runs of the algorithm, usually resulting in a lower overall running time than indicated by the worst possible case.

If $M$ operations take total $O(M \log N)$ time, amortized time per operation is $O(\log N)$

Difference from average time:

Skew Heaps

Problems with leftist heaps
- extra storage for npl
- extra complexity/logic to maintain and check npl
- right side is “often” heavy and requires a switch

Solution: skew heaps
- “blindly” adjusting version of leftist heaps
- merge always switches children when fixing right path
- amortized time for: merge, insert, deleteMin = $O(\log n)$
- however, worst case time for all three = $O(n)$

Merging Two Skew Heaps

Only one step per iteration, with children always switched

Example
Skew Heap Code

```c
void merge(heap1, heap2) {
    case {
        heap1 == NULL: return heap2;
        heap2 == NULL: return heap1;
        heap1.findMin() < heap2.findMin():
            temp = heap1.right;
            heap1.right = heap1.left;
            heap1.left = merge(heap2, temp);
            return heap1;
        otherwise:
            return merge(heap2, heap1);
    }
}
```

Runtime Analysis:
Worst-case and Amortized

- No worst case guarantee on right path length!
- All operations rely on merge

⇒ worst case complexity of all ops =
- Will do amortized analysis later in the course (see chapter 11 if curious)
- Result: $M$ merges take time $M \log n$

⇒ amortized complexity of all ops =

Comparing Heaps

- Binary Heaps
- Leftist Heaps

- d-Heaps
- Skew Heaps

Still scope for improvement!