CSE 326 Data Structures

CSE 326 : Dave Bacon

Priority Queues : Leftist Heaps, Skew Heaps, Binomial Queues
Logistics

• Updated Due Dates
  – Project 2, Phase A, due Friday, January 26
  – Homework 3, due Monday, January 29 in class

• Project 2A
  – Work in partners! Easier for you, good experience for “real” world. See webpage for instructions…don’t forget to email about your partnership (or, less desirably that you’re working alone.)
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 $T_2$.

merge

$T_1$

```
    a
   / \
L_1   R_1
```

$T_2$

```
    b
   / \nL_2   R_2
```

$a < b$
Leftist Merge Continued

If $npl(R') > npl(L_1)$

$R' = \text{Merge}(R_1, T_2)$

runtime: $O(\log n)$
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

  ![Diagram of merge operation]

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

  ![Diagram of deleteMin operation]
Random Definition:
Amortized Time

am-or-tized 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: $\neq$ each step is $\log N$!

Average - still might be bad sequences
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

Always swap (special case of BST)
Example
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
  \[ \Rightarrow \text{worst case complexity of all ops} = \mathcal{O}(\log n) \]

- Will do amortized analysis later in the course (see chapter 11 if curious)

- Result: \( M \) merges take time \( M \log n \)
  \[ \Rightarrow \text{amortized complexity of all ops} = \mathcal{O}(\log n) \]
Comparing Heaps

- Binary Heaps
  - Memory efficient (no pointers)
  - Fast & simple
  - ins: $O(\log N)$, Avg $O(1)$
  - del: $O(\log N)$
  - merge: bad $O(N)$
- d-Heaps
  - Fancy binary heaps
  - ins: $O(\log an)$
  - del: $O(d \log an)$
  - Slower math
- Leftist Heaps
  - Fast merge, ins, del $O(\log N)$
  - Complicated memory cost (links, np)
- Skew Heaps
  - Less storage
  - Good amortized time
  - simple

Still scope for improvement!
Yet Another Data Structure: Binomial Queues

- Structural property
  - Forest of binomial trees with at most one tree of any height

- Order property
  - Each binomial tree has the heap-order property

What’s a forest? bunch of trees
What’s a binomial tree?
The Binomial Tree, $B_h$

- $B_h$ has height $h$ and exactly $2^h$ nodes.
- $B_h$ is formed by making $B_{h-1}$ a child of another $B_{h-1}$.
- Root has exactly $h$ children.
- Number of nodes at depth $d$ is binomial coeff. $\binom{h}{d}$.
  - Hence the name; we will not use this last property.

$B_0$, $B_1$, $B_2$, $B_3$ with $h=0$, $h=1$, $h=2$, $h=3$ respectively.

$B_0$: $2^0$ nodes

$B_1$: $2^1$ nodes

$B_2$: $2^2$ nodes

$B_3$: $2^3$ nodes
Binomial Queue with \( n \) elements

Binomial Q with \( n \) elements has a *unique* structural representation in terms of binomial trees!

Write \( n \) in binary: \( n = 1101 \) (base 2) = 13 (base 10)
Properties of Binomial Queue

- At most one binomial tree of any height

- $n$ nodes $\Rightarrow$ binary representation is of size $\Omega(n)$
  $\Rightarrow$ deepest tree has height $\Theta(\log n)$
  $\Rightarrow$ number of trees is $\Omega(\log n)$

Define: $\text{height(forest } F) = \max_{\text{tree } T \in F} \{ \text{height}(T) \}$

Binomial Q with $n$ nodes has height $\Theta(\log n)$
Operations on Binomial Queue

- Will again define *merge* as the base operation
  - insert, deleteMin, buildBinomialQ will use merge

- Can we do increaseKey efficiently? decreaseKey?

- What about findMin?
Merging Two Binomial Queues

Essentially like adding two binary numbers!

1. Combine the two forests
2. For $k$ from 1 to maxheight {
   a. $m \leftarrow$ total number of $B_k$'s in the two BQs
   b. if $m=0$: continue;
   c. if $m=1$: continue;
   d. if $m=2$: combine the two $B_k$'s to form a $B_{k+1}$
   e. if $m=3$: retain one $B_k$ and combine the other two to form a $B_{k+1}$

Claim: When this process ends, the forest has at most one tree of any height
Example: Binomial Queue Merge

H1:

H2:
Example: Binomial Queue Merge

H1:

H2:
Example: Binomial Queue Merge

H1:

H2:
Example: Binomial Queue

H1: H2:
Example: Binomial Queue Merge

H1: H2:
Example: Binomial Queue Merge

H1:  

H2:  

```
-1
 / 
2  1
 / 
8 11 5
 / 
7 3
 / 
6
 / 
21 9 6
 / 
7
```
Complexity of Merge

Constant time for each height
Max height is: \( \log n \)

\[ \Rightarrow \text{worst case running time} = \Theta(\quad) \]
Insert in a Binomial Queue

Insert\(x\): Similar to leftist or skew heap

runtime

Worst case complexity: same as merge
\[O(\quad)\]

Average case complexity: \[O(1)\]

Why??  Hint: Think of adding 1 to 1101
deleteMin in Binomial Queue

Similar to leftist and skew heaps....
deleteMin: Example

find and delete smallest root

merge BQ (without the shaded part) and BQ'
deleteMin: Example

Result:

runtime:
Tree Calculations

*Recall:* height is max number of edges from root to a leaf

Find the height of the tree...

runtime:
Tree Calculations Example

How high is this tree?
More Recursive Tree Calculations: Tree Traversals

A traversal is an order for visiting all the nodes of a tree.

Three types:

- **Pre-order:** Root, left subtree, right subtree
- **In-order:** Left subtree, root, right subtree
- **Post-order:** Left subtree, right subtree, root

(an expression tree)
void traverse(BNode t) {
    if (t != NULL)
        traverse (t.left);
    print t.element;
    traverse (t.right);
}
}
Binary Trees

- Binary tree is
  - a root
  - left subtree (*maybe empty*)
  - right subtree (*maybe empty*)

- Representation:
Binary Tree: Representation

```
A
  /   \
B     C
  /   /\  \
D    E  F
```

- `A` has left and right pointers.
- `B` has left and right pointers.
- `C` has left and right pointers.
- `D` has left and right pointers.
- `E` has left and right pointers.
- `F` has left and right pointers.
Binary Tree: Special Cases

- **Complete Tree**
- **Perfect Tree**
- **Full Tree**