CSE 326: Data Structures

Binomial Queues

Steve Seitz
Winter 2009

Announcements

HW #2 due on Friday at beginning of class.

Comparing Heaps

<table>
<thead>
<tr>
<th>Worst case [avg/amort. case]</th>
<th>Binary Heap</th>
<th>Skew Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>findMin</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>build</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>merge</td>
<td>O(n)</td>
<td>O(n) [O(log n)]</td>
</tr>
<tr>
<td>insert</td>
<td>O(log n) [O(1)]</td>
<td>O(n) [O(log n)]</td>
</tr>
<tr>
<td>deleteMin</td>
<td>O(log n) [O(log n)]</td>
<td>O(n) [O(log n)]</td>
</tr>
</tbody>
</table>

Our Last Priority Queue

Skew heaps give fast merges, but we lose the fast (average case) inserts of binary heaps.

We’ll look at one more priority queue data structure that doesn’t make this trade-off: binomial queues.

But first, a brief diversion into binomial trees…
Binomial Trees

- A binomial tree $B_h$ has height $h$ and exactly $2^h$ nodes.
- $B_h$ is formed by making $B_{h-1}$ a child of another $B_{h-1}$.

<table>
<thead>
<tr>
<th>Height ($h$)</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of elements</td>
<td>$2^3 = 8$</td>
<td>$2^2 = 4$</td>
<td>$2^1 = 2$</td>
<td>$2^0 = 1$</td>
</tr>
</tbody>
</table>

Binomial Trees

- The root has exactly $h$ children.
- Every subtree of a binomial tree is also a binomial tree.

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?
**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)

<table>
<thead>
<tr>
<th>height (( h ))</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of elements</td>
<td>( 2^3 = 8 )</td>
<td>( 2^2 = 4 )</td>
<td>( 2^1 = 2 )</td>
<td>( 2^0 = 1 )</td>
</tr>
</tbody>
</table>

Each binomial tree obeys heap order.

Write \( n \) in binary: \( n = 1101 \) (base 2) = 13 (base 10)

<table>
<thead>
<tr>
<th>height (( h ))</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of elements</td>
<td>( 2^3 = 8 )</td>
<td>( 2^2 = 4 )</td>
<td>( 2^1 = 2 )</td>
<td>( 2^0 = 1 )</td>
</tr>
</tbody>
</table>

**Properties of Binomial Queue**

- At most one binomial tree of any height.
- \( n \) nodes
  - \( \Rightarrow \) # of bits in binary representation:
  - \( \Rightarrow \) number of trees:
  - \( \Rightarrow \) deepest tree has height:
- Is each subtree a binomial queue?

**A Few More Examples**

- \( N = 2_{10} = 101_2 \)
  - \( 2^2 = 4 \) \( 2^1 = 2 \) \( 2^0 = 1 \)
- \( N = 4_{10} = 100_2 \)
  - \( 2^2 = 4 \) \( 2^1 = 2 \) \( 2^0 = 1 \)
- \( N = 3_{10} = 11_2 \)
  - \( 2^2 = 4 \) \( 2^1 = 2 \) \( 2^0 = 1 \)
- \( N = 5_{10} = 101_2 \)
  - \( 2^2 = 4 \) \( 2^1 = 2 \) \( 2^0 = 1 \)
How to merge queues?

- There is a direct correlation between
  - the number of nodes in the tree
  - the representation of that number in base 2
  - and the actual structure of the tree
- When we merge two queues, the number of nodes in the new queue is the sum of $N_1 + N_2$
- We can use these facts to help see how fast merges can be accomplished
- E.g., add 3+3 in base 2 arithmetic:

Example 1.
Merge BQ.1 and BQ.2

Easy Case.
There are no comparisons and there is no restructuring.

Example 2.
Merge BQ.1 and BQ.2
This is an add with a carry out.
It is accomplished with one comparison and one pointer change: $O(1)$

Example 3.
Merge BQ.1 and BQ.2

= carry
### Merge Algorithm

- Just like binary addition algorithm
- Assume trees $X_0, \ldots, X_k$ and $Y_0, \ldots, Y_k$ are binomial queues
  - $X_i$ and $Y_i$ are of type $B_i$ or null

```plaintext
C_0 := null; //initial carry is null/
for i = 0 to k do
  combine $X_i$, $Y_i$, and $C_i$ to form $Z_i$ and new $C_{i+1}$
  $Z_{k+1} := C_{k+1}$
```

### Complexity of Merge

Constant time for each tree.

Max number of trees is:

\[ \Rightarrow \text{worst case running time} = \]
**Insert**

- Create a single node queue $B_0$ with the new item and merge with existing queue.

- Total time (worst case) =

- Total time (average case) =
  - Hint: Think of adding 1 to 1101

**DeleteMin**

1. Assume we have a binomial queue $X_0, \ldots, X_m$
2. Find tree $X_k$ with the smallest root
3. Remove $X_k$ from the queue
4. Remove root of $X_k$ (return this value)
   - This yields a binomial queue $Y_0, Y_1, \ldots, Y_{k-1}$.
5. Merge this new queue with remainder of the original (from step 3)

- Total time (worst case) =
More Operations on Binomial Queue

- buildBinomialQ can be done with repeated inserts in $O(n)$ time.

- Can we do decreaseKey efficiently? increaseKey?

- What about findMin?

### Comparing Heaps

<table>
<thead>
<tr>
<th></th>
<th>Binary Heap</th>
<th>Skew Heap</th>
<th>Binomial Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>findMin</strong></td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td><strong>build</strong></td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td><strong>merge</strong></td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td><strong>insert</strong></td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td><strong>deleteMin</strong></td>
<td>$O(\log n)$</td>
<td>$O(n)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>