Binomial Queues

CSE 326
Data Structures
Lecture 12

Reading

• Reading
  › Section 6.8,

Merging heaps

• Binary Heap is a special purpose hot rod
  › FindMin, DeleteMin and Insert only
  › does not support fast merges of two heaps
• For some applications, the items arrive in prioritized clumps, rather than individually
• Is there somewhere in the heap design that we can give up a little performance so that we can gain faster merge capability?

Binomial Queues

• Binomial Queues are designed to be merged quickly with one another
• Using pointer-based design we can merge large numbers of nodes at once by simply pruning and grafting tree structures
• More overhead than Binary Heap, but the flexibility is needed for improved merging speed

Worst Case Run Times

<table>
<thead>
<tr>
<th></th>
<th>Binary Heap</th>
<th>Binomial Queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>$\Theta(\log N)$</td>
<td>$\Theta(\log N)$</td>
</tr>
<tr>
<td>FindMin</td>
<td>$\Theta(1)$</td>
<td>$\Theta(\log N)$</td>
</tr>
<tr>
<td>DeleteMin</td>
<td>$\Theta(\log N)$</td>
<td>$\Theta(\log N)$</td>
</tr>
<tr>
<td>Merge</td>
<td>$\Theta(N)$</td>
<td>$O(\log N)$</td>
</tr>
</tbody>
</table>

Binomial Queues

• Binomial queues give up $\Theta(1)$ FindMin performance in order to provide $O(\log N)$ merge performance
• A binomial queue is a collection (or forest) of heap-ordered trees
  › Not just one tree, but a collection of trees
  › each tree has a defined structure and capacity
  › each tree has the familiar heap-order property
Binomial Queue with 5 Trees

<table>
<thead>
<tr>
<th>depth</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>number of elements</td>
<td>$2^4 = 16$</td>
<td>$2^3 = 8$</td>
<td>$2^2 = 4$</td>
<td>$2^1 = 2$</td>
<td>$2^0 = 1$</td>
</tr>
</tbody>
</table>

Structure Property

- Each tree contains two copies of the previous tree
  - the second copy is attached at the root of the first copy
- The number of nodes in a tree of depth $d$ is exactly $2^d$

Powers of 2

- Any number $N$ can be represented in base 2
  - A base 2 value identifies the powers of 2 that are to be included

<table>
<thead>
<tr>
<th>$n_2$</th>
<th>$n_1$</th>
<th>$n_0$</th>
<th>Binary</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1002</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1012</td>
<td>5</td>
</tr>
</tbody>
</table>

Numbers of nodes

- Any number of entries in the binomial queue can be stored in a forest of binomial trees
- Each tree holds the number of nodes appropriate to its depth, i.e., $2^d$ nodes
- So the structure of a forest of binomial trees can be characterized with a single binary number
  - $100_2 \rightarrow 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 4$ nodes

Structure Examples

What is a merge?

- 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 that fact to help see how fast merges can be accomplished
**Merge Algorithm**

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

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

**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

Part 1 - Form the carry.

Part 2 - Add the existing values and the carry.

**Exercise**
O(log N) time to Merge

- For N keys there are at most \( \lceil \log_2 N \rceil \) trees in a binomial forest.
- Each merge operation only looks at the root of each tree.
- Total time to merge is \( O(\log N) \).

Insert

- Create a single node queue \( B_0 \) with the new item and merge with existing queue
- \( O(\log N) \) time

DeleteMin

1. Assume we have a binomial forest \( X_0, \ldots, X_m \)
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 forest \( Y_0, Y_1, \ldots, Y_{k-1} \).
5. Merge this new queue with remainder of the original (from step 3)
- Total time = \( O(\log N) \)

Implementation

- Binomial forest as an array of multiway trees
  - FirstChild, Sibling pointers

DeleteMin Example
Why Binomial?

- Why Binomial?
  - B
  - 0
  - B
  - 1
  - B
  - 2
  - B
  - 3
  - B
  - 4
  - (d-1/2)^d

- Why Binomial?
  - Why Binomial?
    - B
    - 0
    - B
    - 1
    - B
    - 2
    - B
    - 3
    - B
    - 4
    - (d-1/2)^d

Other Priority Queues

- Leftist Heaps
  - O(log N) time for insert, deletemin, merge
- Skew Heaps
  - O(log n) amortized time for insert, deletemin, merge
- Fibonacci Heaps
  - O(1) amortized time for findmin, insert, merge
  - O(log n) amortized time for deletemin, delete
- Calendar Queues
  - O(1) average time for insert and deletemin
  - Assuming insertions are suitably "random"
  - Suitable for limited, but important, applications

Exercise Solution

- Exercise Solution
  - Exercise Solution