Reading

• Reading
  › Section 6.8,
Merging heaps

- Binary Heap has limited (fast) functionality
  - 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>Operation</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>$O(\log N)$</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>Number of Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>$2^4 = 16$</td>
</tr>
<tr>
<td>3</td>
<td>$2^3 = 8$</td>
</tr>
<tr>
<td>2</td>
<td>$2^2 = 4$</td>
</tr>
<tr>
<td>1</td>
<td>$2^1 = 2$</td>
</tr>
<tr>
<td>0</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 $2^d$

<table>
<thead>
<tr>
<th>depth</th>
<th>number of elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$2^2 = 4$</td>
</tr>
<tr>
<td>1</td>
<td>$2^1 = 2$</td>
</tr>
<tr>
<td>0</td>
<td>$2^0 = 1$</td>
</tr>
</tbody>
</table>
Powers of 2 (one more time)

- Any number $N$ can be represented in base 2: $\sum_{i=0}^{n-1} a_i 2^i$
  - A base 2 value identifies the powers of 2 that are to be included

<table>
<thead>
<tr>
<th></th>
<th>$2^3$</th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
<th>Hex$_{16}$</th>
<th>Decimal$_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2^3$</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$2^2$</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>$2^1$</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>$2^0$</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>3</td>
<td>3</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

\[ 101_2 \rightarrow 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 5 \text{ nodes} \]
### Structure Examples

<table>
<thead>
<tr>
<th>$N$</th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$2_{10}$ = $10_2$</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$3_{10}$ = $11_2$</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>$4_{10}$ = $100_2$</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>$5_{10}$ = $101_2$</td>
<td>5</td>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
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 of sizes $N_1$ and $N_2$, 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
Example 1.

Merge BQ.1 and BQ.2

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

\[
\text{BQ.1} + \text{BQ.2} = \text{BQ.3}
\]

\[
\begin{array}{cc|c}
N = 1_{10} = 1_{2} & 2^2 = 4 & 2^1 = 2 & 2^0 = 1 \\
+ & 4 & 8 & \\
= & 4 & 8 & 9 \\
N = 2_{10} = 10_2 & 2^2 = 4 & 2^1 = 2 & 2^0 = 1 \\
\end{array}
\]
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.

\[
\begin{array}{c|c|c}
\text{BQ.1} & 1 & 7 \\
\hline
2^2 = 4 & 2^1 = 2 & 2^0 = 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{BQ.2} & 4 & 8 \\
\hline
2^2 = 4 & 2^1 = 2 & 2^0 = 1 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\text{= carry} & 7 & 8 \\
\hline
2^2 = 4 & 2^1 = 2 & 2^0 = 1 \\
\end{array}
\]
Example 3.

Part 2 - Add the existing values and the carry.

\[ \text{carry} \]

\[ N_{10}=2 \quad 2^2 = 4 \quad 2^1 = 2 \quad 2^0 = 1 \]

\[ + \text{BQ.1} \]

\[ N_{10}=3 \quad 2^2 = 4 \quad 2^1 = 2 \quad 2^0 = 1 \]

\[ + \text{BQ.2} \]

\[ N_{10}=3 \quad 2^2 = 4 \quad 2^1 = 2 \quad 2^0 = 1 \]

\[ = \text{BQ.3} \]

\[ N_{10}=6 \quad 2^2 = 4 \quad 2^1 = 2 \quad 2^0 = 1 \]
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

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

\[
\begin{array}{c|c|c}
N_{10} = 3_{10} = 11_2 & 2^2 = 4 & 2^1 = 2 \\
4 & 20 = 1 & 9
\end{array}
\]

\[
\begin{array}{c|c|c}
N_{10} = 7_{10} = 111_2 & 2^2 = 4 & 2^1 = 2 \\
2 & 21 = 2 & 13 \\
7 & 22 = 4 & 15 \\
12 & & 1
\end{array}
\]
O(log N) time to Merge

• For N keys there are at most $\left\lceil \log_2 N \right\rceil$ 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

FindMin

0 1 2 3 4 5 6 7

5
2
1
9
4
7
10
13
8
12

15

Remove min

0 1 2 3 4 5 6 7

5
2
9
13
8
15
4
7
12
10

1

Return this

FindMin

Remove min

0 1 2 3 4 5 6 7

5
2
9
13
8
15
4
7
12
10

1

Return this
Why Binomial?

\[
\binom{d}{k} = \frac{d!}{(d-k)!k!}
\]

<table>
<thead>
<tr>
<th>tree depth (d)</th>
<th>nodes at depth (k)</th>
<th>(B_0)</th>
<th>(B_1)</th>
<th>(B_2)</th>
<th>(B_3)</th>
<th>(B_4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1, 4, 6, 4, 1</td>
<td>1</td>
<td>1</td>
<td>1, 2, 1</td>
<td>1, 3, 3, 1</td>
<td>1, 4, 6, 4, 1</td>
</tr>
</tbody>
</table>
Other Priority Queues

- **Leftist Heaps**
  - $O(\log N)$ time for insert, deletemin, merge
  - The idea is to have the left part of the heap be long and the right part short, and to perform most operations on the left part.

- **Skew Heaps** ("splaying leftist heaps")
  - $O(\log N)$ amortized time for insert, deletemin, merge
Exercise Solution

\[\begin{align*}
4 & \quad 9 \\
8 & \\
\hline
7 & \quad 10 & 12 & 2 & 13 & 15 \\
\hline
4 & \quad 8 & 13 & 15 & 1 & 9 \\
\hline
\end{align*}\]