CSE 326: Data Structures  
Lecture #22

Mergeable Heaps  
Henry Kautz  
Winter Quarter 2002

Summary of Heap ADT Analysis

- Consider a heap of N nodes
- Space needed: O(N)
  - Actually, O(MaxSize) where MaxSize is the size of the array
  - Pointer-based implementation: pointers for children and parent
  - Total space = 3N + 1 (3 pointers per node + 1 for size)
- FindMin: O(1) time; DeleteMin and Insert: O(log N) time
- BuildHeap from N inputs: What is the run time?
  - N Insert operations = O(N log N)
  - O(N): Treat input array as a heap
  - and fix it using percolate down
  - Thanks, Floyd!

Other Heap Operations

- Find and FindMax: O(N)
- DecreaseKey(P, ΔH): Subtract Δ from current key value at position P and percolate up. Running Time: O(log N)
- IncreaseKey(P, ΔH): Add Δ to current key value at P and percolate down. Running Time: O(log N)
  - E.g.: Schedulers in OS often decrease priority of CPU-hogging jobs
- Delete(P, H): Use DecreaseKey (to 0) followed by DeleteMin. Running Time: O(log N)
  - E.g.: Delete a job waiting in queue that has been preemptively terminated by user

But What About...

- Merge(H1, H2): Merge two heaps H1 and H2 of size O(N).
  - E.g.: Combine queues from two different sources to run on one CPU.
  1. Can do O(N) Insert operations: O(N log N) time
  2. Better: Copy H2 at the end of H1 (assuming array implementation) and use Floyd’s Method for BuildHeap.
  Running Time: O(N)
  Can we do even better? (i.e. Merge in O(log N) time?)

Binomial Queues

- Binomial queues support all three priority queue operations
  - Merge, Insert and DeleteMin in O(log N) time
- Idea: Maintain a collection of heap-ordered trees
  - Forest of binomial trees
- Recursive Definition of Binomial Tree (based on height k):
  - Only one binomial tree for a given height
  - Binomial tree of height 0 = single root node
  - Binomial tree of height k = Bk = Attach Bk-1 to root of another Bk-1

Building a Binomial Tree

- To construct a binomial tree Bk of height k:
  1. Take the binomial tree Bk-1 of height k-1
  2. Place another copy of Bk-1 one level below the first
  3. Attach the root nodes
- Binomial tree of height k has exactly 2^k nodes (by induction)

B0  B1  B2  B3
Building a Binomial Tree

• To construct a binomial tree $B_k$ of height $k$:
  1. Take the binomial tree $B_{k-1}$ of height $k-1$
  2. Place another copy of $B_{k-1}$ one level below the first
  3. Attach the root nodes
• Binomial tree of height $k$ has exactly $2^k$ nodes (by induction)

$B_0$ $B_1$ $B_2$ $B_3$

Building a Binomial Tree

• To construct a binomial tree $B_k$ of height $k$:
  1. Take the binomial tree $B_{k-1}$ of height $k-1$
  2. Place another copy of $B_{k-1}$ one level below the first
  3. Attach the root nodes
• Binomial tree of height $k$ has exactly $2^k$ nodes (by induction)

$B_0$ $B_1$ $B_2$ $B_3$
Why Binomial?

- Why are these trees called binomial?
  - Hint: how many nodes at depth d?

Number of nodes at different depths $d$ for $B_k = [1], [1 1], [1 2 1], [1 3 3 1], ...$

Binomial coefficients of $(a + b)^k = k!/(k-d)!d!$.

Definition of Binomial Queues

Binomial Queue = “forest” of heap-ordered binomial trees

<table>
<thead>
<tr>
<th>$B_0$</th>
<th>$B_1$</th>
<th>$B_2$</th>
<th>$B_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>②</td>
<td>② 1</td>
<td>② 1</td>
<td>② 1</td>
</tr>
</tbody>
</table>

Binomial queue $H_1$: 5 elements = 101 base 2
$\rightarrow B_1 B_0$

Binomial queue $H_2$: 11 elements = 1011 base 2
$\rightarrow B_1 B_1 B_0$

Binomial Queue Properties

Suppose you are given a binomial queue of $N$ nodes

1. There is a unique set of binomial trees for $N$ nodes
2. What is the maximum number of trees that can be in an $N$-node queue?
   - 1 node $\rightarrow 1$ tree $B_1$; 2 nodes $\rightarrow 1$ tree $B_1$; 3 nodes $\rightarrow 2$ trees $B_0$ and $B_1$; 7 nodes $\rightarrow 3$ trees $B_0$, $B_1$, and $B_2$...
   - Trees $B_0$, $B_1$, ..., $B_k$ can store up to $2^k + 2^{k-1} + ... + 2^0 = 2^{k+1} - 1$ nodes $= N$.
   - Maximum is when all trees are used. So, solve for $(k+1)$.
   - Number of trees is $\leq \log(N+1) = O(\log N)$

Binomial Queues: Merge

- Main Idea: Merge two binomial queues by merging individual binomial trees
  - Since $B_{k+1}$ is just two $B_k$’s attached together, merging trees is easy
- Steps for creating new queue by merging:
  1. Start with $B_k$ for smallest k in either queue.
  2. If only one $B_k$, add $B_k$ to new queue and go to next $k$.
  3. Merge two $B_k$’s to get new $B_{k+1}$ by making larger root the child of smaller root. Go to step 2 with $k = k + 1$.

Example: Binomial Queue Merge

- Merge $H_1$ and $H_2$

$H_1$: ② ② ② ② ②

$H_2$: ④ ④ ④ ④ ④

Final merged queue: ④ ④ ④ ④ ④ ④ ④
Example: Binomial Queue Merge

- Merge \( H_1 \) and \( H_2 \)

\( H_1: \quad \begin{array}{c}
\circ \circ \\
\circ \\
\circ \\
\circ \\
\circ
\end{array} \\
\quad \\
\quad \\
\quad \\
\quad \\
\quad
\)

\( H_2: \quad \begin{array}{c}
\circ \circ \\
\circ \\
\circ \\
\circ \\
\circ
\end{array} \\
\quad \\
\quad \\
\quad \\
\quad \\
\quad
\)

Example: Binomial Queue Merge

- Merge \( H_1 \) and \( H_2 \)

\( H_1: \quad \begin{array}{c}
\circ \\
\circ \\
\circ \\
\circ \\
\circ
\end{array} \\
\quad \\
\quad \\
\quad \\
\quad \\
\quad
\)

\( H_2: \quad \begin{array}{c}
\circ \\
\circ \\
\circ \\
\circ \\
\circ
\end{array} \\
\quad \\
\quad \\
\quad \\
\quad \\
\quad
\)

Example: Binomial Queue Merge

- Merge \( H_1 \) and \( H_2 \)

\( H_1: \quad \begin{array}{c}
\circ \\
\circ \\
\circ \\
\circ \\
\circ
\end{array} \\
\quad \\
\quad \\
\quad \\
\quad \\
\quad
\)

\( H_2: \quad \begin{array}{c}
\circ \\
\circ \\
\circ \\
\circ \\
\circ
\end{array} \\
\quad \\
\quad \\
\quad \\
\quad \\
\quad
\)

Example: Binomial Queue Merge

- Merge \( H_1 \) and \( H_2 \)

\( H_1: \quad \begin{array}{c}
\circ \\
\circ \\
\circ \\
\circ \\
\circ
\end{array} \\
\quad \\
\quad \\
\quad \\
\quad \\
\quad
\)

\( H_2: \quad \begin{array}{c}
\circ \\
\circ \\
\circ \\
\circ \\
\circ
\end{array} \\
\quad \\
\quad \\
\quad \\
\quad \\
\quad
\)
Example: Binomial Queue Merge

• Merge H1 and H2

Binomial Queues: Merge and Insert

• What is the run time for Merge of two O(N) queues?
• How would you insert a new item into the queue?

Binomial Queues: Merge and Insert

• What is the run time for Merge of two O(N) queues?
  – O(number of trees) = O(log N)
• How would you insert a new item into the queue?
  – Create a single node queue B₀ with new item and merge with existing queue
  – Again, O(log N) time
• Example: Insert 1, 2, 3, ..., 7 into an empty binomial queue

Insert 1, 2, ..., 7

H1: H2:
**Binomial Queues: DeleteMin**

- **Steps:**
  1. Find tree $B_k$ with the smallest root
  2. Remove $B_k$ from the queue
  3. Delete root of $B_k$ (return this value); You now have a new queue made up of the forest $B_0, B_1, \ldots, B_{k-1}$
  4. Merge this queue with remainder of the original (from step 2)

- **Run time analysis:** Step 1 is $O(\log N)$, step 2 and 3 are $O(1)$, and step 4 is $O(\log N)$. Total time = $O(\log N)$

- **Example:** Insert 1, 2, …, 7 into empty queue and DeleteMin

---

**Insert 1,2,…,7**

**DeleteMin**

**Merge**

**Merge**
**Implementation of Binomial Queues**

- Need to be able to scan through all trees, and given two binomial queues find trees that are same size
  - Use array of pointers to root nodes, sorted by size
  - Since is only of length \(\log(N)\), don’t have to worry about cost of copying this array
  - At each node, keep track of the size of the (sub) tree rooted at that node
- Want to merge by just setting pointers
  - Need pointer-based implementation of heaps
- DeleteMin requires fast access to all subtrees of root
  - Use First-Child/Next-Sibling representation of trees

**Other Mergeable Priority Queues: Leftist and Skew Heaps**

- **Leftist Heaps**: Binary heap-ordered trees with left subtrees always “longer” than right subtrees
  - Main idea: Recursively work on right path for Merge/Insert/DeleteMin
  - Right path is always short \(\Rightarrow\) has \(O(\log N)\) nodes
  - Merge, Insert, DeleteMin all have \(O(\log N)\) running time (see text)
- **Skew Heaps**: Self-adjusting version of leftist heaps (\(a\ la\ splay\ trees\))
  - Do not actually keep track of path lengths
  - Adjust tree by swapping children during each merge
  - \(O(\log N)\) amortized time per operation for a sequence of \(M\) operations
- We will skip details… just recognize the names as mergeable heaps!

**Coming Up**

- Some random randomized data structures
  - Treaps
  - Skip Lists
  - FOR MONDAY: Read section on randomized data structures in Weiss. Be prepared, if called on, to explain in your own words why we might want to use a data structure that incorporates randomness!