# Binomial Heaps

2-2-2006

## Opening Discussion

- Let's collect the tests.  Do you have any questions on it?
- Do you have any questions about the reading?
- Do you have any questions about the assignment?

## Review Binomial Trees

- Last time we looked at the structure of binomial trees. They are quite different from other trees you might be used to. Most importantly, they are far more limited.
- The binomial tree $B_k$ has $2^k$ elements in it. The root has k children and each child is a binomial tree of size k-1..0.
- The binomial tree $B_k$ also has a root with height k.

## Structure of a Binomial Heap

- To build a binomial heap we will use multiple binomial trees. Each tree will have proper heap-ordering, just like a binary heap. For each order k, there can be no more than 1 binomial tree of that order.
- To see what trees we have, simply picture the binary representation of the number for the size of the heap. If our heap has 10 items in it, it will have a $B_1$ and a $B_3$ in it.
- We link the binomial trees together in a linked list from smallest to largest.

## Heap Union

- The primary operation on a binomial heap is the union operation. Our other operations will be generally defined in terms of it.
- To merge two heaps we merge their lists in sorted order then walk the combined list. If we ever have two (and only two) trees of the same size, we merge them by adding one as the child of the other (the one with the highest priority root will be the root of the merged tree) and we repeat the process on that merged node. Any other situation leads to simply stepping to the next node.
- CLR gives nice tight code for this.

## Structure in Memory

- Inevitably there are many ways that we could store binomial heaps in memory, but the form CLR uses is optimal for keeping everything or the right order.
- They use a tree with the “first-child, next-sibling” structure and keep a parent pointer in each node. The list for the heap itself uses the next-sibling pointer.
- Each node also needs to store it's level, which is equivalent to it's height.

## Add Element

- To add an element we simply create a “heap” with the one element in it (a $B_0$ tree), then do a union between that and the heap we are adding to.
- Since union works in O(log n) time, this will as well.

## Remove Element

- To find the next element in the heap to remove (a peek) we simply walk the list of trees and take the root with the highest priority.
- To remove that element we take the list of children from it, reverse their order and set their parent pointer to null. This turns it into a little binomial heap. Simply union this heap with the full one where that tree has been taken out.
- Any element can be removed by “bubbling” that element to the top of its tree and doing the remove on that root.

## Reminders

- Remember that assignment #2 is due on Tuesday. We will start talking about greedy algorithms at that time.