Heaps and Binary Tree Implementation

3-14-2005

Opening Discussion

- What did we talk about last class? Do you have any code to show?
- Do you have any questions about the assignment?
- We want a faster way to do a priority queue. How might we do this? We have used a list, would a tree work?

Code

- Let’s finish our code to implement a sorted binary tree. We should include some traversals in the code.

Heaps

- The only implementation we have for a priority queue right now is to use a sorted linked list. However, the O(n) adds make that inefficient when n gets really large.
- A heap is a data structure like a binary tree that can be used for an efficient priority queue.
- Instead of using the ordering of a sorted tree, it uses "heap ordering", heap trees also have to be complete.

Heap Order and Complete Trees

- Instead of putting lesser things to the left, in a heap, the children of a node all have a “lower priority” than that node. In your game, lower priority means a larger update time.
- A complete tree is a tree that is filled from left to right and each level is completely filled before it starts putting anything in the next level.

Complete Trees as Arrays

- When a tree is complete we can easily represent it as an array. In a complete tree, the left most nodes in a generation get their children first, and nodes get a left child before a right child.
- Note that for a binary tree, the path to a leaf can be expressed as a binary number. If we prefix it with a 1 we get a unique encoding where the first element is 1. This also allows us to quickly find

Inserting to a Heap

- To insert into a binary heap, we put a “bubble” node at the next open space and let it move up the heap until it moves into place.
- At each step it gets swapped with a higher priority object, so the heap-order property is maintained.
- This operation always takes O(log n) time because the heap is a complete tree.

Removing from a Heap

- To remove the smallest element we simply make the "bubble" node at the top and this time the last element in the heap is what will wind up filling it. We let it "sink" down through the tree. At each step we swap it with the smaller of the two children assuming that it is larger than them. If it isn't it can stop there.

Minute Essay

- Do you have any questions about the heap? Show me what your heap would look like if you added 5,3,7,1,6 then pulled off two elements. Draw a total of 6 tree structures to show it.
- The semester is winding down and we are talking about more diverse topics. Read up on java.io for next class. Quiz #6 is next class.