COS 423
Spring, 2009

Implicit Heaps

These notes discuss the classic *implicit heap* data structure [1, 2].

A *heap* (or *priority queue*) is a data structure that contains a set of items, each with a real-valued *key*, and that supports (at least) the following operations:

- *make-heap*: return a new, empty heap.
- *insert*(x, H): insert item x, with predefined key, into heap H.
- *find-min*(H): return an item in heap H of minimum key.
- *delete-min*(H): delete from heap H an item of minimum key, and return it; return null if the heap is empty.

An *implicit heap* is an implementation of a heap consisting of a complete binary tree whose nodes contain the heap items, one node per item. The items are arranged in *heap order*: every node contains an item with a key no smaller than the key of the item in its parent, if it has a parent. Thus the root contains an item of minimum key, so a *find-min* takes $O(1)$ time. We number the nodes level-by-level: the root is node 1, its children are nodes 2 and 3, the children of nodes 2 and 3 are nodes 4 and 5, and 6 and 7, respectively, and in general the children of node k are nodes $2k$ and $2k + 1$. Thus the parent of node k if $k \neq 1$ is $\lfloor k/2 \rfloor$. This numbering allows us to represent the tree without using pointers: we store the heap in an array, indexed by node number. We also maintain a count n of the number of items in the heap.

To insert an item into the heap, add one to n. Node n is now empty. Compare the key of the new item to the key of the item in the parent of the empty node. If the former is no smaller, insert the new item in the empty node and stop. Otherwise, move the item in the parent into the empty node, making the parent empty. Continue in this way until the new item is inserted or the root is empty; in the latter case, insert the new item in the root.

To delete a minimum item from the heap, remove the item in the root. This creates an empty node. Compare the keys of the items in its children and move the item of smaller key into the empty node, making one of its children empty; break a tie arbitrarily. Fill the newly empty node from one of its children in the same way, and repeat this step until a leaf becomes empty. If this leaf is node n, subtract one from n. Otherwise, remove the item in node n, subtract one from n, and fill the remaining empty node as in an insertion, by comparing the key of the item removed from node n with the key of the item in the parent of the empty node, putting the item of larger key in the empty node (favoring the item removed from node n), and continuing if the parent is now empty. Once there is no empty node, return the item deleted from the root.

Both *insert* and *delete-min* take $O(\log n)$ time in the worst case, at most $\lg n$ binary comparisons for an insertion, $2 \lg n$ for a deletion. For an appropriate model of the average case, the number of comparisons is $O(1)$ for an *insert* and $\lg n + O(1)$ for a *delete-min*, because a newly inserted item is likely to rise to only a constant height in the tree, as is the item moved from node n in

a deletion. This implementation of *delete-min* differs from the classical implementation in that the latter begins by swapping the item in node n with the item in the root, and then moves this item down through the tree until heap order is restored. This takes two comparisons per level, and the item moved to the root is likely to sink to a constant height, so the average number of comparisons for a *delete-min* is $2 \lg n - O(1)$, a factor of two worse than the bound for the method given here.

Exercise: Propose a model for the average case, and verify the claims in the preceding paragraph.

Exercise: Devise methods to do an *insert* in $\lg \lg n + O(1)$ comparisons in the worst case and a *delete-min* in $\lg n + \lg \lg n + O(1)$ comparisons in the worst case.

Exercise: Devise a method to do n consecutive insertions into an initially empty implicit heap in $O(n)$ time.

Exercise: Implement the following operation, arbitrary deletion, so that it takes $O(\log n)$ time on an n-node implicit heap:

delete(x, H): delete item x from the heap H containing it, given the node containing x.

Exercise: Consider using complete trees of degree four instead of degree two as implicit heaps. What are the advantages and disadvantages of doing this?

References

[1] R. W. Floyd. Algorithm 245 : Treesort 3. *Comm. ACM*, 7:701, 1964.

[2] J. W. J. Williams. Algorithm 232: Heapsort. *Comm. ACM*, 7:347–348, 1964.