Exercise Set 4 - Solutions

March 27, 2009

Heaps and Priority Queues

R-8.2

$(1, D)$, $(3, J)$, $(4, B)$, $(5, A)$, $(2, H)$, $(6, L)$.

R-8.11

Yes, tree T is a heap. It is a complete binary tree and each node stores a key value greater than the key of its parent, except for the root.

R-8.13

With a preorder traversal, a heap that produces its entries in increasing order is that which is represented by the array list $[x, 1, 2, 5, 3, 4, 6, 7]$. There does not exist a heap for which an inorder traversal produces the keys in order. This is because in a heap the parent is always less than all of its children or greater than all of its children. The heap represented by $[x, 1, 5, 2, 7, 6, 4, 3]$ is an example of one which produces its keys in decreasing order during a postorder traversal.

R-8.16

Imagine the heap which is represented by the array list $[x, 1, 5, 2, 8, 9, 7, 6]$. This heap will not produce keys in nondecreasing order when a preorder traversal is used.

R-8.17

Imagine the heap which is represented by the array list $[x, 1, 5, 2, 8, 9, 7, 6]$. This heap will not produce keys in nonincreasing order when a postorder traversal is used.

C-8.4

Maintain a variable m initialized to 0. On a push operation for element e, call **insert**(m, e) and decrement m. On a pop operation, call **remove** and increment m.

C-8.5

Maintain a `maxKey` variable initialized to 0. On an enqueue operation for element e, call **insertItem** (`maxKey`, e) and increment `maxKey`. On a dequeue operation, call **removeMinElement** and decrement `maxKey`.

C-8.16

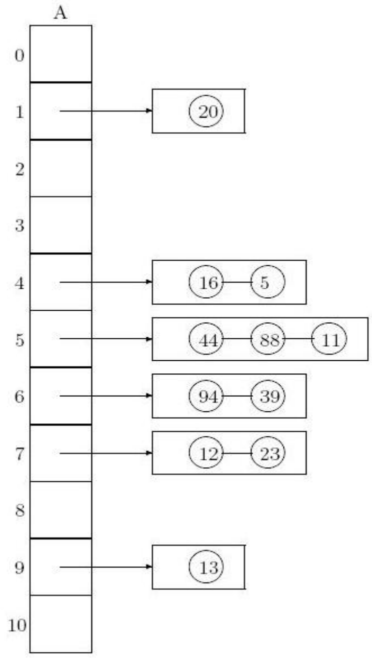
Build a heap storing the frequent flyers and their mileage, using bottom-up heap construction. This takes $O(n)$ time. Next, call **removeMin** $\log n$ times, which takes $O(\log n \cdot \log n)$ time, to determine the top $\log n$ flyers. Thus, the total time is $O(n)$.

C-8.17

Construct a heap, which takes $O(n)$ time. Then call **removeMin** k times, which takes $O(k \log n)$ time.

Hashing

R-9.5



Extra Question

1. The worst case search time is $O(N)$. It is possible that these N keys produce same $h(K)$ value, cause all of them are inserted into the same table entry and an unsorted linked list with N keys is built. It takes $O(N)$ time to perform a search in that unsorted linked list. For example, let $M = 13$ and the keys are $14, 27, 92, 40, 1, 53, 66, 79, \ldots$ will have the worst case search time.

2. No! If the application is time-critical, it is not a good choice to use a hash table (with chaining technique to solve collisions) because we can't guarantee the worst case will not happen.