Data Structures and Algorithms

Session 12. March 2, 2009

Instructor: Bert Huang

http://www.cs.columbia.edu/~bert/courses/3137

Announcements

- Homework 3 is out. Due 3/9
- Sample midterm problems on Courseworks
- Midterm review March 9th
- Midterm Exam March 11th
- No work during break

Review

- Note about Young Tableaux *
- Visualization of Splay Trees
- Tries

- Definition of Priority Queues
  - Heap implementation

Today’s Plan

- Solving the Young Tableaux Recurrences
- buildHeap description and analysis
- HW2 solutions

Young Tableaux

- Analysis of recursive solution to HW1's sorted 2d array problem is related to MergeSort and buildHeap
- MergeSort splits array into two subproblems, linear cost to merge
- We'll look at buildHeap later in today's class

Young Tableaux

- Using simple linear search, the running time is

$$\begin{aligned} T(N) &= 2T(N/2) + cN \\ &= \sum_{i=0}^{\log N} 2^i c \frac{N}{2^i} \\ &= \sum_{i=0}^{\log N} cN \\ &= cN \log N \end{aligned}$$

Young Tableaux with Binary Search

- Using binary search, the running time is:

$$T(N) \quad = \quad 2T(N/2) + \log N = \sum_{i=0}^{\log N} 2^i c \log \frac{N}{2^i}$$

$$= \quad \sum_{i=0}^{\log N} 2^i \left(\log N - \log 2^i\right) = \sum_{i=0}^{\log N} 2^i \left(\log N - i\right)$$

- Let **h = log N**

$$T(2^h) \quad = \quad \sum_{i=0}^{h} 2^i \left(h - i\right)$$

Young Tableaux with Binary Search

$$T(2^h) \quad = \quad \sum_{i=0}^{h} 2^i \,(h-i) \qquad\qquad 2T(N) = \sum_{i=0}^{h} 2^{i+1}(h-i)$$

$$T(N) = 2T(N) - T(N) =$$

$$\begin{array}{l} \quad 2^1(h-0) + 2^2(h-1) + 2^3(h-2) + \ldots + 2^h(1) + 2^{h+1}(0) \\ \underline{-\left[2^0(h-0) + 2^1(h-1) + 2^2(h-2) + \ldots + 2^{h-1}(1) + 2^h(0)\right]} \end{array}$$

Young Tableaux with Binary Search

$$T(2^h) \quad = \quad \sum_{i=0}^{h} 2^i \, (h-i) \qquad\qquad 2T(N) = \sum_{i=0}^{h} 2^{i+1}(h-i)$$

$$T(N) = 2T(N) - T(N) =$$

$$
\begin{array}{l}
\qquad\qquad 2^1(h-0) + 2^2(h-1) + 2^3(h-2) + \ldots + 2^h(1) + \\
-\ \underline{\left[2^0(h-0) + 2^1(h-1) + 2^2(h-2) + \ldots + 2^{h-1}(1) + 2^h(0)\right]} \\
\quad -h \quad + \quad 2^1 \quad + \quad 2^2 \quad + \quad \ldots\ldots \quad + 2^h
\end{array}
$$

$$= -h + \sum_{i=1}^{h} 2^i = 2^{h+1} - 2 - h$$

$$= 2^{\log N + 1} - 2 - \log N = 2N - 2 - \log N$$

Heap operations

- Recall the basic two heap operations: insert, deleteMin
  - Use percolateUp and percolateDown
- If we want to change a key, we can also just use percolateUp and percolateDown
- The cost of each is constant + cost of percolate up/down

Building a Heap from an Array

- How do we construct a binary heap from an array?
- Simple solution: insert each entry one at a time
- Each insert is worst case **O(log N)**, so creating a heap in this way is **O(N log N)**
- Instead, we can jam the entries into a full binary tree and run **percolateDown** intelligently

buildHeap

- Start at deepest non-leaf node
  - in array, this is node N/2
- **percolateDown** on all nodes in reverse level-order
  - for i = N/2 to 1
    percolateDown(i)

Analysis of buildHeap

- **N/2** percolateDown calls: **O(N log N)**?
  - But calls to deeper nodes are much cheaper
- Percolate Down costs the height of the node
- Let **h** be height of tree. 1 node at height **h**
  - 2 nodes at **(h-1)**, 4 nodes at **(h-2)**...
  - 2^h nodes at height 0

Analysis of buildHeap

- Recall that **h = log N**
- Total height of all nodes in heap is:

$$T(N) = \sum_{i=0}^{h} 2^i (h - i)$$

- We solved this earlier today: **T(N) = O(N)**

HW2 Solutions

- Up on Courseworks

Assignments

- Homework 3
- Look at practice problems
- Weiss 6.4