Data Structures and Algorithms: Homework #6

Due on June 9, 2015 at 16:20

Instructors Hsuan-Tien Lin, Roger Jang

Tim Liou (b03902028)

6.1 Skip List, Binary Search Tree

(1) Do Exercise C-9.15 of the textbook.

For every node, also store the width of the link. The width is defined as the number of bottom layer links being traversed by each of the higher layer "express lane" links. Following is Example from `http://en.wikipedia.org/wiki/Skip_list#Indexable_skiplist`

```
   1                               10
 o---> o---------------------------------------------------------> o   Top
   1           3              2                    5
 o---> o---------------> o---------> o---------------------------> o
   1         2         1      2                    5
 o---> o---------> o---> o---------> o---------------------------> o
   1     1     1     1     1     1     1     1     1     1     1
 o---> o---> o---> o---> o---> o---> o---> o---> o---> o---> o---> o   Bottom

Head  1st   2nd   3rd   4th   5th   6th   7th   8th   9th   10th  NIL
      Node  Node  Node  Node  Node  Node  Node  Node  Node  Node
```

function MEDIAN
  node ← head
  remain ← $\left\lfloor \frac{n}{2} \right\rfloor$
  while BELOW(node) $\neq$ null **do**
    node ← BELOW(node)
    while remain $\geq$ WIDTH(node) **do**
      remain ← remain − WIDTH(node)
      node ← AFTER(node)
    end while
  end while
  return node
end function

Algorithm 1: Find the median element

We can use **Width** function to get the width in $O(1)$ since we store the value. This **Median** function involes two nested **while** loops. One performs a scan forward, and the other drops down to the next level. Since the height h of S is $O(\log n)$ with high probability, the number og drop-down steps is $O(\log n)$ with high probability. Let n_i be the number of keys examined while scanning forward at level i. We can easily observe that, after the key at the starting postion, each additional key examined in a scan-forward at level i cannot also belong to level $i + 1$. If any of these keys were on the previous level, we would have encountered them in the previous level. Thus, the expected value of n_i is exactly equal to the expected number of times we must flip a fair coin before it comes up heads, That is 2. Hence, the expected

Problem 6.1 continued on next page...

amount of time spent scanning forward at any level i is $O(1)$. Since S has $O(\log n)$ levels with high probability, this **Median** function in S takes expected time $O(\log n)$.

(2) Do Exercise R-10.5 of the textbook.

Consider these two input orders, $\{1, 3, 2, 4, 5\}$ and $\{4, 2, 1, 5, 3\}$. Note that these two sets contain same entries $\{1, 2, 3, 4, 5\}$. We can easily find that they generate different trees.

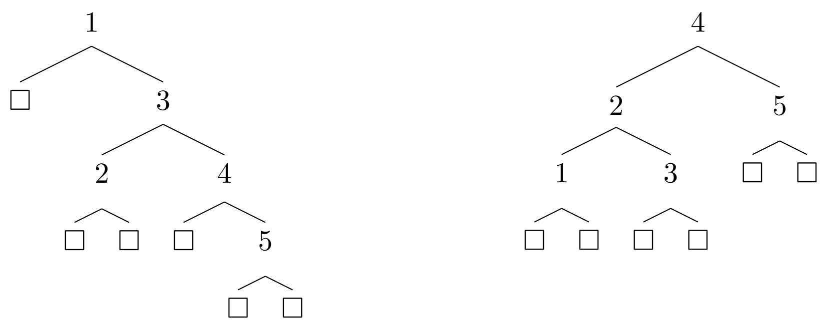

Figure 1: Left tree generates from $\{1, 3, 2, 4, 5\}$, and right tree generates from $\{4, 2, 1, 5, 3\}$.

We conclude that the order of insertion does matter.

(3) Do Exercise C-10.12 of the textbook.

====== **Pending** ======

6.2 Balanced Binary Search Trees

(1) Write a program `hw6_2` that reads 32 strings (of length at most 128 that can be compared lexicographically) line by line (each line containing one string) from `stdin` and inserts them to the AVL tree (`avl.c`), height-bounded binary search tree (`bst.c`), and Red-Black tree (`rb.c`). Please output the resulting trees (pre-order) to `stdout` with specific format.

====== **Pending (code part)** ======

6.3 Disjoint Set

(1) Prove that the disjoint-set forest with this heuristic yields a worst-case running time for find and union within $O(\log n)$.

====== **Pending** ======

(2) Suppose that you only need to output u rather than u and k for this problem. Write down the pseudo-code of an efficient algorithm based on the disjoint forest.

====== **Pending** ======

(3) Suppose that the prices of your friend u's games are stored in a balanced BST as keys, and you have access to the size and the sum of all keys of any subtree of the BST in an $O(1)$ time, write down the pseudo-code of an efficient algorithm for outputting k for the particular u.

====== Pending ======

(4) If we take the same heuristic as (1) and always insert the elements of the smaller BST into the bigger one, prove that processing all incidents of the first kind takes $O(n(\log n)2)$ time.

====== Pending ======

(5) Write a program `hw6_3` to solve the problem efficiently.

====== Pending (code part) ======