CS21003 Algorithms I, Autumn 2011–12

Mid-Semester Test

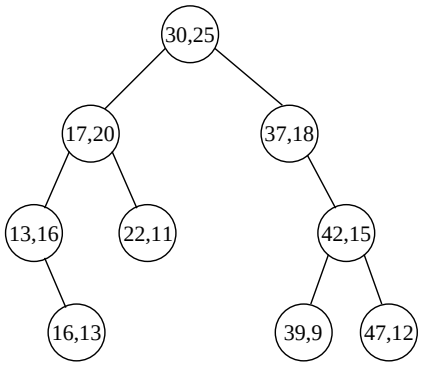
Maximum marks: 30 | Date: September 27, 2011 | Duration: Two hours

---

Roll no: _____ **Name:** _____

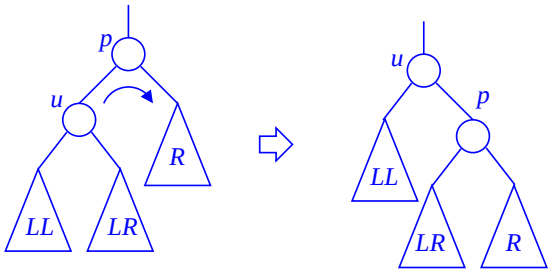
[*Write your answers in the question paper itself. Be brief and precise. Answer all questions.*]

1. A *treap* T is a binary search tree with each node storing (in addition to a value) a priority. The priority of any node is not smaller than the priorities of its children. The root is the node with the highest priority. Unlike heaps, a treap is not forced to satisfy the heap-structure property. An example of a treap is given in the adjacent figure, where the pair (x, y) stored in a node indicates that x is the value of the node, and y is the priority of the node. The x values satisfy binary-search-tree ordering, and the y values satisfy heap ordering.

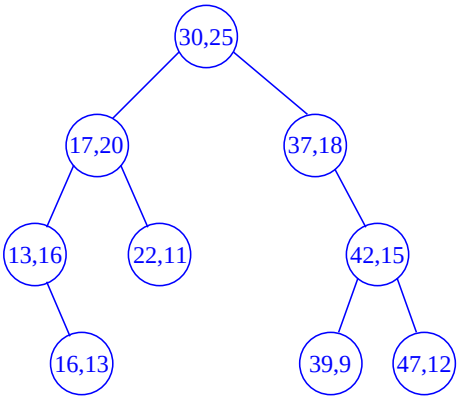


(a) Design an $O(h(T))$-time algorithm to insert a value x with priority y in a treap. (**Hint:** Use rotations.) **(6)**

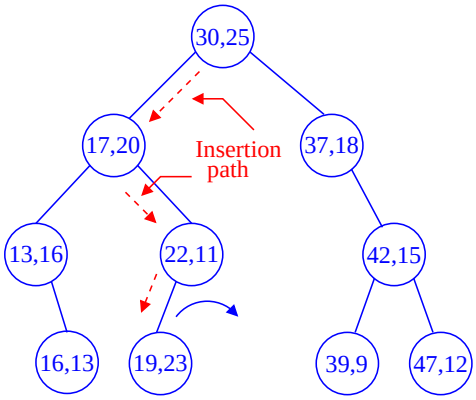
Solution First, insert x in the tree using the standard BST-insertion procedure. This insertion may violate heap ordering, that is, we may encounter a situation where a node u has a priority larger than the priority of its parent p. Depending upon whether u is the left or the right child of p, we make a right or left rotation making u the parent of p. Such a situation is demonstrated in the figure below. The larger priority value moves by one level up the tree, and may again be larger than the priority of its new parent. This violation of heap ordering is repaired by another rotation. This process is repeated until heap ordering is restored, or the node with a large priority reaches the root of the treap.



(b) Demonstrate how your algorithm inserts the value 19 with priority 23 in the treap of Page 1. Show clearly all the major steps of the algorithm (instead of drawing only the final result). **(6)**

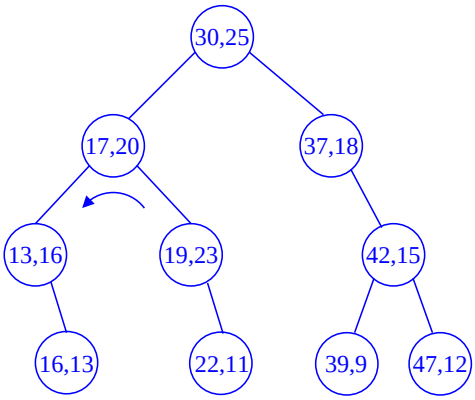


(a) The original treap

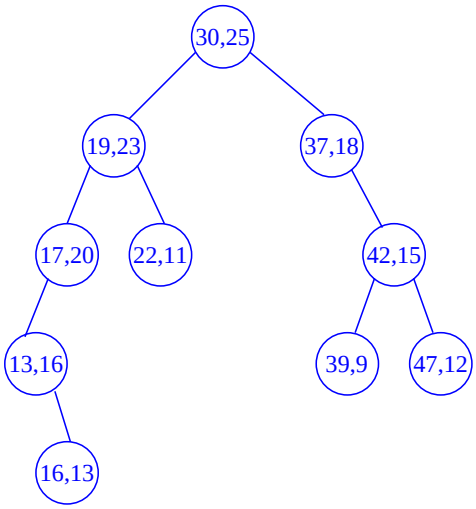


(b) After BST insertion (rotation needed)

Solution



(c) Another rotation necessary

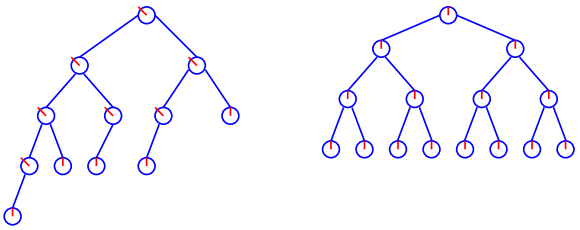


(d) Heap ordering restored

Roll no: _____ **Name:** _____

2. Prove or disprove: For all integers $h \geqslant 0$, any AVL tree of height $h+1$ contains strictly more nodes than any AVL tree of height h. **(6)**

Solution *False*: A Fibonacci tree of height four consists of only $F_{4+3} - 1 = F_7 - 1 = 13 - 1 = 12$ nodes, whereas a full binary tree of height three contains $1 + 2 + 4 + 8 = 15$ nodes.



(a) Fibonacci tree of height 4 (b) Complete tree of height 3

AVL trees

3. You are given an array of n distinct integers $a_0, a_1, a_2, \ldots, a_{n-1}$. You are also given an integer t. Your task is to find out whether $t = a_i + a_j$ for distinct indices i, j ($0 \leqslant i < j \leqslant n-1$). Propose an algorithm to solve this problem. Your algorithm must have an expected running time of $\mathrm{O}(n)$. **(6)**

Solution Insert $a_0, a_1, a_2, \ldots, a_{n-1}$ in a hash table. Subsequently, for each $i = 0, 1, 2, \ldots, n-1$, if $t \neq 2a_i$, search for $t - a_i$ in the hash table. If any of the searches succeeds, return *true*. If all these n searches fail, return *false*.

Each insertion and searching in the hash table takes $\mathrm{O}(1)$ expected running time, so the expected running time of this algorithm is $\mathrm{O}(n)$.

Roll no: _____ **Name:** _____

4. You have a collection of r three-Rupee coins, s seven-Rupee coins, t eleven-Rupee coins, and u sixteen-Rupee coins. (These need not be actual coins, but tokens worth these values.) Given an integer $n \geqslant 0$, your task is to determine whether a sum of n Rupees can be exchanged exactly by coins from your collection. As an example, let $r = 1$, $s = 2$, $t = 3$, and $u = 4$. Thirty Rupees can be exchanged as $7 + 7 + 16$ or as $3 + 11 + 16$ (but not as $3 + 3 + 3 + 3 + 7 + 11$, since you do not have so many three-rupee coins), whereas 31 Rupees cannot at all be exchanged by the coins in this collection (try it!).

Describe an $O(n^2)$-time algorithm to solve this problem. Your algorithm does not have to find a change (when it exists). It suffices to find out only whether a change is possible or not. **(6)**

Solution

```
int existsChange ( int r, int s, int t, int u, int n )
{
   int i, j, A[MAX], B[MAX];

   /* O(n)-time initialization */
   for (i=0; i<=n; ++i) A[i] = B[i] = 0;

   /* Check possibilities with 3- and 7-Re coins in O(n^2) time */
   for (i=0; i<=min(r,n/3); ++i) {
      for (j=0; j<=min(s,n/7); ++j) {
         if (3*i + 7*j > n) break; else A[3*i + 7*j] = 1;
      }
   }

   /* Check possibilities with 11- and 16-Re coins in O(n^2) time */
   for (i=0; i<=min(r,n/11); ++i) {
      for (j=0; j<=min(s,n/16); ++j) {
         if (11*i + 16*j > n) break; else B[11*i + 16*j] = 1;
      }
   }

   /* Combine the possibilities in O(n) time */
   for (i=0; i<=n; ++i) {
      if ((A[i] == 1) && (B[n-i] == 1)) { return 1; }
   }

   /* No possibilities located */
   return 0;
}
```