ESO 207A: QUIZ - 2

Full time for this Quiz is 45 mts
Mention your name and roll number in each page of the paper in the space provided.
Answer the questions in the spaces provided on the question sheets.

Name: ______________________________

Roll No: ______________________________

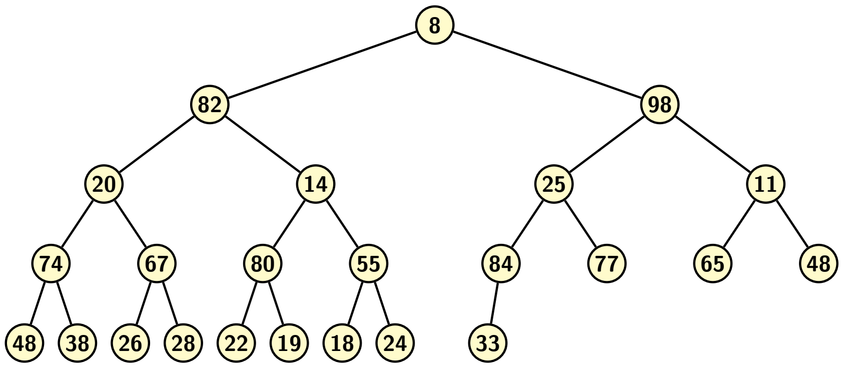
(Feb 8, 2018)

1. (6 points) Consider a d-heap. Each internal node of a d-heap can have at most d children. The tree implementing a d-heap must also satisfy the heap property (element at parent is smaller than elements at children nodes) and a structural property (a complete d-ary tree) like in a binary heap.
   (a) (3 points) What will be the time to perform m heapify up operations in a d-heap having n elements?
   (b) (3 points) What will be time to perform n DeleteMIN operations.

Solution: (a) For `heapifyUp`, just compare the element with the parent and exchange the values only if parent has a value greater than what child have. So, m heapify up operation will still be $O(\log_d n)$. Base of log comes from structural property.

(b) For a single `DeleteMIN` operation on a d-heap, the last element is copied to the root as in the case of binary heap. But `heapifyDown` will have to be performed to put the copied element in correct position in the heap. For pushing the element down it has to be compared with d values now instead of just 2 values in case of binary heap. So the time will $O(dn \log_d n)$.

2. (6 points) A min max heap is binary heap with following heap order properties:
   For any node X at an even depth, the key value stored at X is the smallest in its subtree.
   For any node X at an odd depth, the key value stored at X is the largest in its subtree.
   A min-max heap example is given below.



   (a) (3 points) How do you find the maximum and the minimum element?
   (b) (3 points) How do you insert 5 into the min-max heap of the figure above?

Solution: The minimum value is at the root. So it can be done in O(1) time. Maximum value will be one the two children nodes at odd depth=1. So, it can also be found in O(1) time.

The element 5 will be placed initially as the right child of 84. Since $5 < 84$, it will be always be smaller than all elements at the max levels of heap above. Therefore, 5 need to be compared with the element at the min levels. It implies 5 is compared with 25, and these element exchanged their values. Then 5 is compared next with 8 and these two values exchange positions. After that heap is final.

3. (6 points) given a BST and a path $v_1, v_2, \ldots, v_k$ in that BST. Call the set $\{v_1, v_2, \ldots, v_k\}$ as B. Now study the following pseudo codes computing two sets of nodes A and C resepctively.

```
/* 1. $p = (v_1, ..., v_k)$, $B = \{v_1, v_2, \ldots, v_k\}$ is a path from root to the node in the BST

    2. InorderList give the inorder list of all element under the subtree
       rooted at a node.

   3. ptr->key refers to value stored in the node.
*/
computeA(p) {
        A = {};
        i = 1
        ptr = $v_i$;
        %while(ptr!=NULL && $i \le k$) {
        while(ptr!=NULL && $i \le k$) {
              if(ptr->key < $v_k$) {
                    A = $A \cup \{InorderList(ptr->left)\}$
              }
              i = i + 1;
              ptr = $v_i$;
        }
        return A;
}


/* 1. $p = (v_1, ..., v_k)$, $B = \{v_1, v_2, \ldots, v_k\}$ is a path from root to the node in the BST

    2. InorderList give the inorder list of all element under the subtree
       rooted at a node.

   3. ptr->key refers to value stored in the node.
*/

computeC(p) {
        C = {};
        i = 1
        ptr = $v_i$;
        while(ptr!=NULL && $i \le k$) {
              if(ptr->key > $v_k$) {
                      $C = C \cup \{InorderList(ptr->right)\}$
              }
              i = i + 1;
              ptr = $v_i$;
        }
        return C;
}
```

(a) (3 points) What do you think the set A and C would represent in relation to the nodes on the tree path (i.e, set B)?

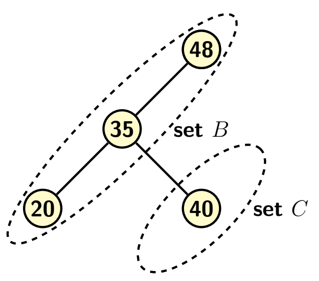
(b) (3 points) Now, the claim is that any three keys $k_a \in A$, $k_b \in B$ and $k_c \in C$ sastisfy the property $k_a \le k_b \le k_c$. Do you agree? If, yes, then give a proof. If, not, give a smallest counter example to the claim.

(Hint: Try out computation of A on a small BST example.)

(a) $A = \{x | x \in LST(v_i \in B), \}$ and $C = \{x | x \in RST(v_i \in B), \}$,

where LST and RST respectively denote the left and the right subtrees of the corresponding nodes.

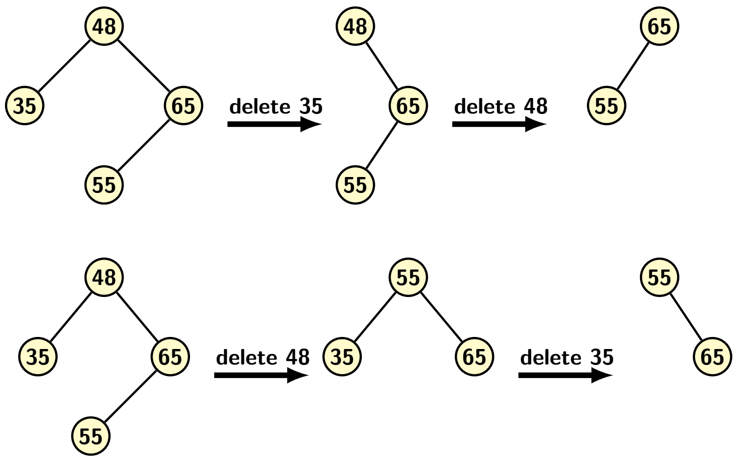
(b) No, the claim is false. The smallest counter example is shown in figure below. In this example, the search is for key 20. Set $A = \phi$, set $B = \{48, 35, 20\}$. set $C = \{40\}$



For any two keys $a \in A$ and $b \in B$ the claim is true because A is empty. But for the keys $b = 48 \in B$ and $c = 40 \in C$, we have $b > c$.

4. (6 points) Suppose two students started with identical binary search trees (BSTs) and were asked to perform a pair of delete operations x and y, where x and y are two fixed values in the BST. The student turn over the results to the TA, who was suprised to see that the results are different. He/she conlcuded that one of them must be wrong. Do you agree? If yes, why? If not, give a smallest counter example to show that both A and B may actually be correct.

Solution: Smallest counter example is given in the figure below.



5. (6 points) Lowest Common Ancestor (LCA) of two unrelated (neither of the nodes is an ancestor of the other) distinct nodes n_1, n_2 in a tree is the deepest node from root r which is an ancestor of both the nodes. How can you use BST property to determine LCA of any pair of node n_1 and n_2? Give a pseudo code clearly identifying the major steps of your algorithm.

Solution: Let x be LCA of $n1$ and n_2. Without loss of generality assume $n_1 < n_2$. Since, n_1 and n_2 are unrelated, n_1 and n_2 belong to two different subtrees of the input tree T. As x is an ancestor of both, n_1 and n_2 must respectively belong to the left and the right subtrees of x. Once we find a key n such that $n_1 < n < n_2$, the node associated with the key n in BST will be the LCA. So, the method to find x can be stated as follows:

```
 node * LCA(node * root, int n1, int n2) {
    1. if (root == NULL)
          return NULL;
    2. if (root->key > n1 &&  root->key > n2)
          return LCA(root->left, n1, n2);
    3. if (root->key < n1 &&  root->key < n2)
          return LCA(root->right, n1, n2);
    4. return root;
}
```

Space for Rough Work