ESO207A: Data Structures and Algorithms	
Homework *3: Red-Black Trees*	HW Due Date: Oct 12, 2018

Instructions.

1. Start each problem on a new sheet. For each problem, write your name, Roll No., the problem number, the date and the names of any students with whom you collaborated. Remember that you must write the answer and the algorithm in your own words.
2. For questions in which algorithms are asked for, first summarize the problem you are solving and your results (including time/space complexity as appropriate). The body of the write-up should provide the following:
   (a) A clear description of the algorithm in English and/or pseudo-code, where, helpful.
   (b) At least one worked example or diagram to show more precisely how your algorithm works.
   (c) A proof/argument of the correctness of the algorithm.
   (d) An analysis of the running time of the algorithm.

   Remember, your goal is to communicate. *Full marks will be given only to correct solutions which are described clearly.* Convoluted and unclear descriptions will receive *low marks.*

Problem 1. *Insertion.* Consider the red-black tree given in Figure 1. Insert a new node with key 37 and show in a step-by step manner how the insertion and fixup of the tree happens. Please explain your transformations and relate them to the cases in the fixup routine.

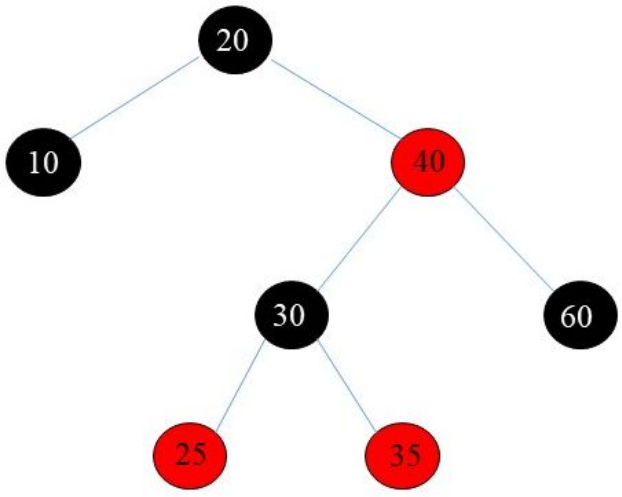


Figure 1: Insert a node in this RB-tree with key value 37

Problem 2. *Deletion.* Consider the red-black tree given in Figure 2. Delete the node with key 20. Show in a step-by step manner how the insertion and fixup of the tree happens. Please explain your transformations and relate them to the cases in the fixup routine.

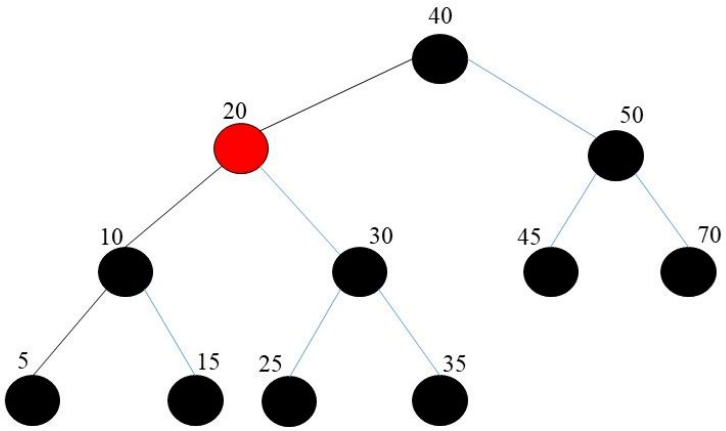


Figure 2: Delete the node with key 20 in this RB-tree

Problem 3. *2-3 search trees: Long* A 2-3 search tree is an (probably a simpler) alternative to red-black search trees and is defined as follows. A non-NIL 2-3 search tree consists of

1. 2-nodes: which has one key and a left child link and a right child link, as in a normal BST. That is, the left child link points to a 2-3 search tree with smaller keys and the right child link points to a 2-3 search tree with larger keys. Or,
2. 3-nodes: that has two keys and *three* links, a left link to a 2-3 search tree with smaller keys, a middle link to a 2-3 search tree with keys between the node's keys, and a right link to a 2-3 search key with larger keys.
3. For any node, the length of the downwards simple path from the node to any leaf node is the same.

An example 2-3 search tree is given in Figure 3.

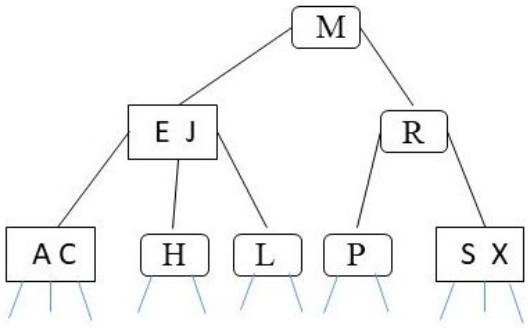


Figure 3: An example 2-3 search tree. The links at the bottom point to NIL nodes

(a) Let h be the height of a 2-3 search tree and n be the number of non-NIL nodes in the tree. Derive a relation of the form $f(h) \leq n \leq g(h)$ and hence argue that $h = \Theta(\log n)$.

Search. The search algorithm for a key directly generalizes the search algorithm for BSTs. Consider the 2-3 search tree in Figure 3. Suppose we wish to search for the key H. Comparing H with the key M of the 2-node root, we have $H < M$ and so we take the left child link to get to the 3-node (E, J). Since, $E < H < J$, we now take the middle link to go the 2-node with key H. Now, H is found and the search is a success.

(b). Write pseudo-code for a general search routine $Search(T, v)$, where, T is a 2-3 search tree and v is a key value that runs in $O(\log n)$ time. Design your own structure for the node of a 2-3 tree.

Insert. Let us consider insertion of a new key in a given 2-3 search tree. The main idea of BST insertion is used, namely, we first do an unsuccessful search. In a BST, we insert the new node as a child of the leaf node. Here, we will insert the new key into the leaf node where the unsuccessful search ended. If the leaf node is a 2-node, we insert the new key and make it a 3-node. Here is an example.

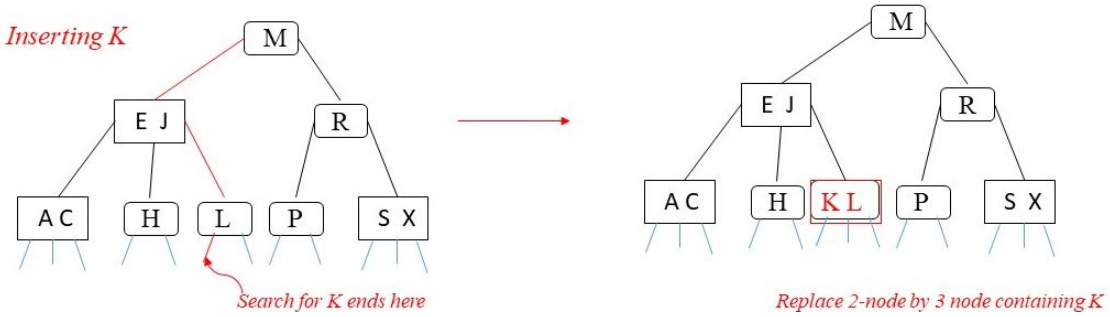
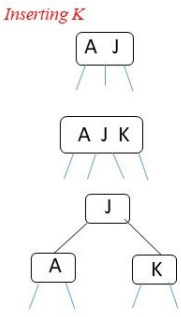


Figure 4: Inserting into 2-3 search tree: add into a 2-node: easy case

Now suppose you have a 2-3 tree whose root node is a 3-node with keys (AJ). Suppose we wish to insert K. Conceptually and temporarily we form (the illegal) *4-node* (AJK) with 3 keys. Now take the median of these keys, which is J and split this *4-node* as shown.



The question now asks you to figure the general case. Say we insert the new key into a 3-node whose parent is a 2-node. The 3-node temporarily becomes a 4-node, which we split into 2 2-nodes (as above) and the median is promoted as the parent of these 2-nodes. This parent can then be inserted into the 2-node parent to make it a 3-node. In the most general case, the leaf node into which the initial insertion is made is a 3-node and the path from the leaf upwards to the root all consist of 3 nodes. In that case, successively, from leaf upwards to root, the median from the lower

level node is inserted (which has two pointers to the left and right children obtained from splitting) into the new node, making it a 4-node and causing a split, and so on it propagates upwards. What happens at the root? Explain this with figures in detail as answer to the following question.

(c) Design an algorithm for inserting in a general 2-3 search tree. Show that it runs in time $O(\log n)$.