Fundamental Algorithms, Assignment 6 Solutions

1. Consider a Binary Search Tree T with vertices a, b, c, d, e, f, g, h and $ROOT[T] = a$ and with the following values (N means NIL)

vertex	a	b	c	d	e	f	g	h
parent	N	e	e	a	d	g	c	a
left	h	N	N	e	c	N	f	N
right	d	N	g	N	b	N	N	N

   (a) Which is the successor of c. Illustrate how the program `SUCCESSOR` will find it.
   `Solution:`The successor of c is f. As c has a right child g, SUCCESSOR will call `MIN[g]` which will go to the left as long as possible, ending (in one step) at f.

   (b) Which is the minimal element? Illustrate how the program `MIN` will find it.
   `Solution:`h. Start at root a. Go to left: h. Go to left: NIL. Return h.

   (c) Illustrate the program `DELETE[e]`
   `Solution:`There are two approaches (equally correct) to `DELETE[x]` when x has two children. One can effectively replace x by the maximum of its left tree or the minimum of its right tree.
   `Solution 1:` e has a left child c. Applying `MAX[c]` gives g. g has a left child f. So we splice f into g's place by resetting $right[c] = f$ and $p[f] = c$ and we put g in e's place, setting $left[d] = g$, $left[g] = c$, $right[g] = b$. and $p[g] = d$.
   `Solution 2:` e has right chils b. Applying `MIN[b]` gives b itself. We splice b into e's place by resetting $p[c] = b$ and $left[b] = c$ and $p[b] = d$ and $left[d] = e$

2. Draw binary search trees of height 2,3,4,5,6 on the set of keys {1,4,5,10,16,17,21}.
`Solution:`For height 2 with 7 it must be balanced so 10 is the root with children $4, 17$ which in turn have children $1, 5$ and $16, 21$. For height 6 it must be a line. There are two ways: 1 as root and then taking right children go dwon $4, 5, 10, 16, 17, 21$. (Or take 21 as root

and go in reverse.) For the other heights there are many solutions. E.g.: height 3: 10 has left child 5 with left child 4 with left child 1 and 10 has right child 17 which has two children $16, 21$. Height 4: root 16 has left child 10 with left child 5 with left child 4 with left child 1 and 16 has right child 17 with right child 17. Height 5 would be similar with 17 as root.

3. What is the difference between the binary-search property and the heap property? (*) Can the heap property be used to print out the keys of an n-node tree in sorted order in $O(n)$ time? Explain how or why not.

   `Solution:`Actually there is not that much the same. In min-heap the parent is less than both children. In BST the parent is bigger than the leftchild and less than the right child and moreover the parent is bigger than all of the left subtree and less than all of the right subtree.

   There is no way to get a sorted order from a min-heap in time $O(n)$. Recall you can build a min-heap from scratch in time $O(n)$. So is you could then get a sorted order in further time $O(n)$ then in total $O(n)+O(n) = O(n)$ you would sort an array from scratch. We can't do that – sorts take time $O(n \lg n)$ unless there is additional information about the data.

4. You are given an array $A[1 \cdots n]$, whose values come from a universe Ω. (In application, the values would be the keys of records.) You want to test if there are any duplicates, if there are any $1 \le i < j \le n$ such that $A[i] = A[j]$. You are given a hash function $h : \Omega \to \{1, \ldots, n\}$ and a table $T[1 \cdots n]$ of linked lists, initially all empty. Using the hash function, give an algorithm that returns `BAD` if there is a duplicate and `GOOD` if there is no duplicate. Discuss the time of the algorithm under the assumption that calculating the hast function takes unit time.

   `Solution:`For $1 \le i \le n$ calculate $w = h(A[i])$. Go through the linked list $T[w]$ to see if any of the values are $A[i]$. If so, end the procedure and returen `BAD`. If not, add $A[i]$ to the head of the linked list $T[w]$. If the FOR loop ends then return `GOOD`.

5. What would the BST tree look like if you start with the root a_1 with $key[a_1] = 1$ (and nothing else) and then you apply $INSERT[a_2], \ldots, INSERT[a_n]$ in that order where $key[a_i] = i$ for each $2 \le i \le n$? Suppose the same assumptions of starting with a_1 and the key values but the INSERT commands were done in *reverse* order $INSERT[a_n], \ldots, INSERT[a_2]$.

   `Solution:`In the first case it would be like a line to the right, with each a_i the right child of the previous a_{i-1}. In the second case a_n would be the right child of a_1 and then it would look like a line to the left (but always staying to the right of a_1) with a_{i-1} being the left child of a_i.