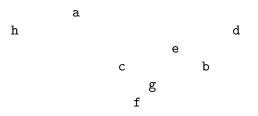
Fundamental Algorithms, Assignment 6 Solutions

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	с	d	е	\mathbf{f}	g	h
parent	Ν	е	е	a	d	g	с	a
left	h	Ν	Ν	е	с	Ν	f	Ν
right	d	Ν	g	Ν	b	Ν	Ν	Ν
parent left right key	80	170	140	200	150	143	148	70

Draw a nice picture of the tree. Illustrate INSERT[i] where key[i]=100. Solution: Here is the picture, without the key values.



For INSERT[i]:

We start at root a with key[a] = 80. As 80 < 100 we replace a by its right child d with key[d] = 200. As 100 < 200 we replace d by its left child e with key[e] = 150. As 100 < 150 we replace e by its left child c with key[c] = 140. As 100 < 140 we replace c by its left child. But its left child is NIL so we make the new vertex i its left child by setting p[i] = c and left[c] = i.

- 2. Continuing with the Binary Search Tree of the previous problem:
 - (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, SUC-CESSOR 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 child 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

3. 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 each would go all the way to the right and you would get a line to the right. In the second case a_n would be to the right of the root a_1 and the remaining would from a line going to the left from a_n . Note that in either case you get a "long stringy" tree which will be very inefficient.

4. Set $N = 2^{K}$. We'll represent integers $0 \le x < N$ by $A[0 \cdots (K-1)]$ with $x = \sum_{i=0}^{k-1} A[i]2^{i}$. (This is the standard binary representation of x, read right to left.) Consider the following algorithms:

```
Procedure JACK[A]

I \leftarrow 0

A[0] + +

WHILE (A[I] = 2 \text{ AND } I < K - 1)

A[I] \leftarrow 0

I + +

A[I] + +

END WHILE
```

and: ANYA[A] FOR J = 1 TO N - 1DO JACK[A]END FOR

If the input to JACK[A] is the binary representation of x with $0 \le x \le N - 2$ describe what the output will be.

Solution: JACK increments by one, the final value of A will be the binary representation of x+1. For example, if A (reading right to left) is 1100111 then it becomes 1100112, 1100120, 1100200, 11001000 and then stops.

- 5. Let T be a binary search tree on nodes $1, \ldots, N$ (in no particular order in the tree) with height H For any vertex v define depth[v] as the distance from v to the root. (The root has depth zero, its children have depth one, grandchildren two, etc.) Let TD denote the sum of depth[v] for all nodes v.
 - (a) Give an algorithm to find any particular depth[i] in time O(H) and TD in time O(HN).
 Solution:For a particular i climb to the top: DEPTH[i]
 d ← 0
 WHILE i ≠ root[T]
 d + +
 end WHILE
 return d
 For TD go through all vertices and take their sum.
 - (b) Modifying In-Order-Tree-Walk give an algorithm that finds all depth[i] and also TD in total time O(N) regardless of the value

of *H*. Solution:Initialize r = root(T), depth[r] = 0, TD = 0. Now run IOTW[r]. When you call IOTW[s] $(s \neq r)$ set depth(s) = $depth(\pi(s)) + 1$ (note the depth of the parent has already been calculated!) and $TD \leftarrow TD + depth[s]$.