EC 504 – Fall 2020 – Homework 5

Due Thursday, Oct 29, 2020 in the beginning of class. Coding problems submitted in the directory `/projectnb/alg504/yourname/HW3` on your SCC account by Thursday Octo 29, 11:59PM.

Reading Assignment: CLRS Chapters 12, Sec 16.3, B.2 (relations) and Introduction to Graphs: Chapter 22 and B.4

1. (15 puts) Determine whether the following statements are true or false, and explain briefly why.

   (a) A heapsort algorithm for a given list first forms a max-heap with the elements in that list, then extracts the elements of the heap one by one from the top. This algorithm will sort a list of n elements in time of O(log(n)).

   Solution: False. Forming the heap is $\Theta(n)$ and extracting each element is $O(\log(n))$. Altogether that gives you a runtime of $O(n\log(n))$ Also the fastest sorting algorithm is $O(n\log(n))$ so this should've been obvious.

   (b) A connected undirected acyclic graph with N nodes and N-1 edges is a tree.

   Solution: True. Start with N=1 a single node and zero linka. Each time you add a link it has one more node if it does not make a cycle.

   (c) A binary heap of n elements is a full binary tree for all possible values of n.

   Solution: False. It is a complete binary tree.

   (d) The following tree is a valid min-heap.

```
         4
       /   \
      10    5
     / \
    12 14
```

   Solution: True. It is a complete binary tree satisfying the min heap property.

   (e) The following tree can appear in a binomial min-heap. .

```
         4
       /   \
      10    5
     / \
    12 14
```

   Solution: False. It is not a binomial tree, as the number of elements is not a power of 2.

   (f) Given a array of positive integers $a[i]$, maximizing $\sum i * a[i]$ over permutations of the array requires sorting $a[i]$ in assigning order.

   Solution: True. This is the same argument used for the Huffman coding proof. One way is to use induction. Consider n = 2. Have two possibilites. If $a[1] < a[2]$ but

$$(2-1)(a[2]-a[1]) > 0 \implies 1a[1] + 2a[2] > 1a[2] + 2a[1]$$

Now assume for n-1 and add another element. Put the max into the last position $a[n]$.

$$Max[\sum_{i=1}^{n} i * a[i]] = Max[\sum_{i=1}^{n-1} i * a[i]] + na[n]$$

The n-element list must sorted by assumption and $na[n]$ is as large as it can be. q.e.d.

2. (20 Pts) Given the following list of elements,

$$35, 22, 11, 98, 55, 66, 77, 80, 85, 90, 95$$

(a) Draw the sequence of binary min-heaps which results from inserting the following values in the order in which they appear into an empty heap.

Solution: Final Solution Shown:

```
                  11
                /    \
            35          22
        /       \     /     \
    80          55  66      77
    / \         / \
98      85      90  95
```

(b) Compare this answer with inserting them into the BST tree. **Solution:** Final Solution Shown:

```
                    35
                /       \
            22              98
        /               /
    11              55
                        \
                        66
                            \
                            77
                                \
                                80
                                    \
                                    85
                                        \
                                        90
                                            \
                                                95
```

(c) Compare this answer with inserting them into the AVL tree. **Solution:** Final Solution Shown:

```
                        55
                    /       \
                22              85
            /       \       /       \
        11          35    77          95
```

```
              /    \          /    \
            66          80   90      98
```

3. (20 Pts) You are interested in compression. Given a file with characters, you want to find the binary code which satisfies the prefix property (no conflicts) and which minimizes the number of bits required. As an example, consider an alphabet with 8 symbols, with relative weights (frequency) of appearance in an average text file give below:

alphabet:	A	L	G	O	R	I	T	H
weights:	68	20	5	30	18	15	19	12

(a) Determine the Huffman code by constructing a tree with **minimum external path length**: $\sum_{i=1}^{8} w_i d_i$. (Arrange tree with smaller weights to the left.) **Solution:** Solution Shown:

```
                       187
                /               \
           69                      118
       /       \                 /       \
    32           37            50          68(A)
  /    \       /     \      /    \
15(I)      17    18(R)  (T)19  20(L)  30(O)
         /    \
       5(G)    12(H)
```

(b) Identity the code for each letter and list the number of bits for each letter and compute the average number of bits per symbole in this code. Is it less than 3? (You can leave the answer as a fraction since getting the decimal value is difficult without a calculator.) **Solution:** Solution Shown:

```
LETTER | WEIGHT | CODE | BITS
A            68       11       2
O            30       101      3
L            20       100      3
T            19       011      3
R            18       010      3
I            15       000      3
H            12       0011     4
G            5        0010     4
AVG BITS PER CHAR < 3
```

(c) Give an example of weights for these 8 symbols that would saturate 3 bits per letter. What would the Huffman tree look like? Is a Huffman code tree always a full tree? **Solution:** Solution Shown: If all letters had the same weight it would saturate the 3 bits because all 8 letters would need 3 bits to represent it, any example is a complete full binary tree.

4. (20 Pts) Given the following list of $N = 10$ elements.

28 14 7 4 6 30 36 33 10 40

(a) Insert them sequentially into a BST (Binary Search Tree). Compute the total height $T_H(N)$ and the total depth $T_D(N)$, where H is the height of the root. (Note the height of a node is the longest path length to a leaf whereas its depth is its unique distance from the root.) **Solution:** Final Solution Shown:

```
                        28
                    /       \
                14              30
              /                  \
           7                      36
          / \                        /   \
         4   10                       33     40
          \
            6
```

(b) Find H, $T_H(N)$, $T_D(N)$ and check the sum rule:

$$T_H(N) + T_D(N) \geq HN$$

Solution:

```
H=4
To(N) = 22
Th(N) = 13
To(N) + Th(N) > HN
35 < 40
```

(c) Insert them sequentially into an empty AVL tree, restoring the AVL property after each insertion. Show the AVL tree which results after each insertion and name the type of rotation (RR or LL zig-zig or versus RL or LR zig-zag). **Solution:** Final Solution Shown:

```
                        14
                    /       \
                6              30
            /       \       /       \
        4            7    28          36
                        \           /   \
                          10       33    40
```

(d) Has the final AVL tree decreased the total height $T_H(N)$ and the total depth $T_D(N)$? What are the new values? What is the new value of $T_H(N) + T_D(N)$? **Solution:** Yes

```
Th(N) = 9
To(N) = 19
Both values decreased with the AVL tree, new value of To(N) + Th(N) = 28
```

5. **Coding Exercise:** (25 Pts) You are give a code that runs a binary search tree (BST). The BST code is has the following function

```
        SearchTree MakeEmpty( SearchTree T );
        Position Find( ElementType X, SearchTree T );
        Position FindMin( SearchTree T );
        Position FindMax( SearchTree T );
        SearchTree Insert( ElementType X, SearchTree T );
        SearchTree Delete( ElementType X, SearchTree T );
        ElementType Retrieve( Position P );
```

declared in `tree.h` and defined in `tree.cpp`

As in the case of HW2 HeapSort the main program `mainBST.cpp` has the following function allows you to insert into the binary search tree from either an input file or form generate internally a random sequence. You should use the input list `List100.txt` and `List100K.txt`. It should work on any random input list. You can generate more list using `makeSingleList.cpp` Note `n = InputList[0]` is the size of the list NOT an element to be used in the BST which are `InputList[i] for i = 1, ..., n`

The problem is to

```
(1)Read into InputList[]  the files List100.txt first and then  List100K.txt
(2)Build the BST  from the first half (i = 1,...n/2)
(3)Delete the even ones (i= 2,4,.., n/2)
(4)Insert the second half i = n/2 +1,..., n
(5) Find the Max and Min in this BST
(6) Implement a function that calculate the height of each node in the BST.
```

The implementation of the height should be included as a new field as follows:

```
struct TreeNode
{
  HeightType  Height;
  ElementType Element;
  SearchTree  Left;
  SearchTree  Right;
};
```

keeping it up to date as you build and change the tree with Delete and Insert. This is a step toward implementing an AVL balances tree which maybe used in the future.

Analysis stage: Do some analysis of algorithmic performance. So do a small step on the following (doesn't have to be fancy unless you are seeking extra credit) do the following.

```
(1) Rum the BST code  starting with an internal random for range
    of sizes: n = 8, 32, 128, 512, ....
(2) Plot the heights averaged over set random permutation of one large list.
(3) For n = 1024  show after  random deletion/insertions the BST
height increases.
```

I have suggested sizes that increase by a 4x. It is usually good to have an exponentially increasing sequence but this is an option to explore in any scaling study.

Extra Credit: For extra credit you do a bit of error analysis. When you measure set of height $(h_1, h_2, \cdots, h_N)$ for a give size (with in our case different permutations) you can not only estimate average (or mean *overlineh*!) but the error (or standard deviation σ) by the well know formulae:

$$\overline{h} = \frac{1}{N}\sum_{i=1}^{N} h_i \quad , \quad \sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(h_i - \overline{h})^2 \tag{1}$$

The mean and the width are the first and second moment that fix the form of a Gaussian (or Normal) distribution. (Gnuplot lets use σ in your plots and fits.) Of course as we discussed in class the distribution may not be – in fact almost never is – a perfect Gaussian. Higher moments called `Skewness` and `Kurtosis` look for non-Gaussian features. For fun see:

https://www.itl.nist.gov/div898/handbook/eda/section3/eda35b.htm