CSE 2320 Lab Assignment 2

Due October 20, 2016

Goals:

1. Understanding of heaps and interfacing with a dictionary.

2. Understanding of the five steps for developing a dynamic programming solution.

Requirements:

1. Use C to implement algorithms for (a) approximate *order-preserving* Huffman coding - each phase merging two *adjacent* subtrees whose weights give the smallest sum, and (b) exact *order-preserving* Huffman coding - using the dynamic programming formulation described in Notes 7.C.

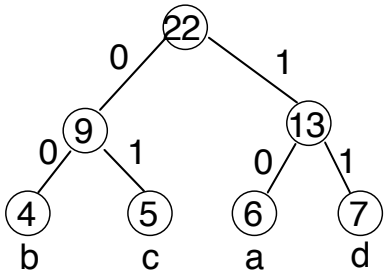
   The input is 1) a positive integer n and 2) a sequence of n positive integers giving the frequency counts (weights) for symbols in an ordered character set. To simplify output, the character set will be referenced numerically as $0 \ldots n - 1$.

   For both types of subtrees, your program should output the bit code assigned to each symbol and the weighted sum $\left(\sum_i length_i \bullet count_i \right)$ based on the generated code tree and the frequency counts.

2. Submit your program on Blackboard by 1:45 p.m. on October 20, 2016. One of the comment lines should include the compilation command used on OMEGA.

Getting Started:

1. Suppose the input frequency counts are: `6 4 5 7`. The following tree is for the listed conventional Huffman code.

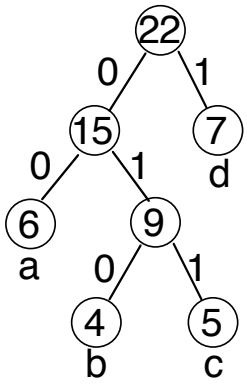



```
6   10
4   00
5   01
7   11
```

The weighted sum is 6•2 + 4•2 + 5•2 + 7•2 = 44

Now suppose the ordered character set is {a, b, c, d} with the indicated frequency counts. If the strings "abc" and "bad" are compressed to "`100001`" and "`001011`", respectively, they *do not compare* the same way as their uncompressed counterparts. Order preservation is guaranteed *only* when the leaf order is consistent with the order of the character set.

2. For the same input sequence, the following tree is for the approximate *order-preserving* Huffman code. At each step in its construction, we greedily merge the two *adjacent* trees whose weights have the smallest sum.

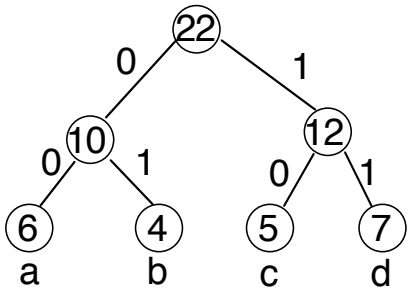



```
6   00
4   010
5   011
7   1
```

The weighted sum is 6•2 + 4•3 + 5•3 + 7•1 = 46.

Now, the strings "abc" and "bad" will be compressed to "`00010011`" and "`010001`", respectively, but the weighted sum has not been minimized.

3. For the same input sequence, the following tree is for the exact *order-preserving* Huffman code. It was constructed using dynamic programming to determine optimal order-preserving subtrees.




```
6   00
4   01
5   10
7   11
```

The weighted sum is 6•2 + 4•2 + 5•2 + 7•2 = 44.

4. It is not difficult to see that the weighted sums for the three different approaches have the following relationship: conventional ≤ exact ≤ approximate.

5. Your approximate solution must use a heap to achieve $\Theta(n \log n)$ time. Submissions taking $\Theta\left(n^2\right)$ time will be severely penalized.

6. Input is to be read from standard input. Do not prompt for a file name.

7. Heap-based code (`http://ranger.uta.edu/~weems/NOTES2320/huffman.freq.c`) for conventional Huffman coding with frequencies is available on the course webpage. It may be modified *significantly* to achieve part (a). You will want each heap entry to correspond to *two* adjacent subtrees that could be merged. After a *single* `heapExtractMin()` determines the merge to apply, you will need `minHeapDelete()` to discard unneeded candidate(s) (due to the merge) and a `minHeapInsert()` to include new candidate(s) (also resulting from the merge). Handles (Notes 5) facilitate this.

   Extra tables will be helpful when implementing part (a).