Homework 5
Fundamental Algorithms, Fall 2001, Professor Yap

UPDATES ON HOMEWORK:

- There is a bug in my description of the cost of adding two counters C and C'.

  The cost should be defined to be the number of bits of C and C' that you need to look at (and possibly change) when you add the 2 binary numbers.

  Of course, this cost is AT MOST the sum of the bit lengths of C and C' (which was what I mistakenly told you IS the cost).

  Let me illustrate: if $C = 1001,1101$ and $C' = 110$, then $C + C' = 1010,0011$ and the cost is 9. This is because you need to look at 6 bits of C and 3 bits of C'.

- DUE: Mon Nov 26, in class.

1. [**10+20+10 POINTS**] Consider the following "Grouping Problem". Suppose we are given an input of the form (M, w) where
$$w = (w_1, w_2, , \ldots, w_n),$$
with M and the w_i's being non-negative numbers. A **solution** to this problem is any subdivision of the sequence w into some number of "groups", where each group has "size" at most M. Here a group G is a subsequence of **consecutive** numbers from w, of the form
$$G = (w_i, w_{i+1}, w_{i+2}, \ldots, w_j), \qquad 1 \le i \le j \le n.$$
The size of G is simply the sum $\sum_{\ell=i}^{j} w_\ell$. You are guaranteed that each $w_i \le M$ so that a solution exists. A subdivision of w into k groups can be represented by a sequence of k integers,
$$1 \le n_1 < n_2 < n_3 < \cdots < n_k \le n$$
where the ith group is given by $G_i = (w_{n_{i-1}+1}, w_{n_{i-1}+2}, \ldots, w_{n_i}\}$. Let us denote this subdivision as "$[n_1, n_2, \ldots, n_k]$". When $i = 1$ in this definition, we let n_0 be 0. You want a solution that is **optimal** in the sense that it has the smallest possible number of groups.

   Example. If $M = 5$ and $w = (1, 1, 1, 3, 2, 2, 1, 3, 1)$, one solution is the the subdivision $[n_1, n_2, n_3, n_4, n_5] = [2, 4, 6, 8, 9]$. This corresponds to the 5 groups $(1, 1), (1, 3), (2, 2), (1, 3), (1)$. But we can get 4 groups, using $[n_1, n_2, n_3, n_4] = [2, 4, 7, 9]$. This is also optimal.
   (i) Give a simple greedy algorithm for the Grouping Problem.
   (ii) Prove that your algorithm is optimal.
   (iii) Suppose the w_i's may be negative as well. Either prove that your algorithm is still optimal or show a counter example.

   (i) The greedy algorithm is to start consider each w_i in turn, from $i = 1, 2, \ldots, n$. At each moment, we have a current group. Initially, the current group is empty. We add w_i to the current group as long as the weight of the current group is $\le M$. If this is not possible, we start a new current group containing w_i as its only element.

   (ii) Why is this optimal? Suppose there are k groups in the greedy solution. Recall that this can be represented as $s = [n_1, n_2, \ldots, n_k]$. Let any other $S = [N_1, N_2, \ldots, N_K]$ be any other solution with K groups. We need to prove that $k \le K$. The argument is actually simple: we need to argue that $N_i \le n_i$ for each $i \le \min(k, K)$. This is true for $i = 1$. For $i > 1$, assume the result for $i - 1$. Then if $N_i > n_i$ this means the sum $w_{N_{i-1}+1} + \cdot + w_{N_i}$ is at least the value $w_{n_{i-1}+1} + \cdot + w_{n_i} + w_{n_i+1}$ which is $> W$, contradiction.

   (iii) It is clearly false: the greedy algorithn on input $W = 1$ and $w = (1, 1, -1)$ will produce three groups, but one group suffices.

2. [**15 POINTS**] Compute the optimal Huffman code for the first two sentences of President Lincoln's Gettysburg address. The full address is reproduced below, but ONLY encode the first two sentences. You will need to encode spaces, commas and full stop, but NOT the newlines.

   In fact, define "white space" to be any maximal contiguous sequence of one or more space, newline and tab characters. The rule is to ignore the initial and terminal white spaces (if any), and to replace any remaining white space by a single space character.

   Note that small and capital letters are distinguished. You **must**
   (i) state the overall bit length of your coded string, and
   (ii) display the collection of intermediate code trees which are obtained in your computation.

3. [**10+15+10+10 POINTS**] Exercise 16.3-5, page 392. Representation of an optimal prefix code tree. However, instead of using $2n-1$ bits to specify the tree, we allow you to use up to $4n-4$ bits, so that the overall representation should use at most $4n-4+n\lceil \lg n \rceil$ bits. Note that the alphabet C is the binary representation of the numbers 0 to $n-1$. You **must** do 3 things:
   (i) Prove that a Huffman tree with n leaves has exactly $2n-2$ edges. (Use induction of the structure of the tree)
   (ii) Tell us how to construct the representation from any Huffman tree for the set $C = \{0, \ldots, n-1\}$.
   (iii) Apply your code to the tree in Figure 16.4(b), assuming that $a = 0, b = 1, c = 2, \ldots, f = 5$.
   (iv) Describe how to reconstruct the Huffman code tree from your representation.

   MORE HINTS: You need to exploit the structure of the set C. How many edges does a Huffman tree with n leaves have?

   (i) A Huffman tree on n leaves have $2n-2$ edges. We can specify this tree using $4n-4$ bits (starting at the root, '0' means go down and '1' means go up. Going up is unambiguous, and going down is also unambiguous if we assume that we always go down the left branch first). Let α denote this binary string of length $2n-2$. E.g., for the binary tree in figure 16.4(b), page 387, we have $\alpha = 01000, 10110, 00101, 10111$ (the commas are a visual aid).

   Next, we would like to attach the binary strings representing elements of $C = \{0, \ldots, n-1\}$ to the leaves of this tree. We can simply list these according to the in-order traversal of the Huffman tree. This takes $n\lceil \lg n \rceil$ bits. Let this string be denote β. Finally, we simply put them together as the string $\alpha\beta$. E.g., in figure 16.4(b), we have $\beta = 000, 010, 001, 101, 100, 011$ (again, the commas are visual aid).

   (ii) This is already done above.

   (iii) How can we decode this? (1) First, how can we know we reached the end of α? Well, whatever the first bit of β, when we read it, it will cause us to attempt to go above the root, or to go down from the root for the third time. (2) Next, we need to extract the in-order listing of the elements of C. We need to know the value of $\lceil \lg n \rceil$. But this is easy to compute since we know n from part (1). Thus we can get the individual code words from the string β.

4. [**20 POINTS**] We generalize the example of incrementing binary counters. Suppose we have a collection of binary counters, all initialized to 0. We want to perform a sequence of operations. Each operation is one of two types:

$$\texttt{inc}(C), \quad \texttt{add}(C, C')$$

   where C, C' are names of counters. The operation $\texttt{inc}(C)$ increments the counter C by 1 as before. The operation $\texttt{add}(C, C')$ adds the contents of C' to C while simultaneously set the counter C' to zero. Assume that the cost to perform $\texttt{add}(C, C')$ is equal to the sum of the lengths of the binary numbers stored in C and C'. Show that this problem has an amortized cost that is constant per operation.
   HINT: the potential of a counter C should take into account the number of 1's as well as the bit-length of the counter.

5. [**15+15+15 POINTS**] Problem 17-2, page 426. Dynamic binary search. Our T.A. Sean had discussed this problem in recitation.

GETTYSBURG ADDRESS (for Problem 2):

```
Four score and seven years ago our fathers brought forth on this
continent a new nation, conceived in liberty and dedicated to the
proposition that all men are created equal.  Now we are engaged in
a great civil war, testing whether that nation or any nation so
conceived and so dedicated can long endure. We are met on a great
battlefield of that war. We have come to dedicate a portion of
that field as a final resting-place for those who here gave their
lives that that nation might live. It is altogether fitting and
proper that we should do this. But in a larger sense, we cannot
dedicate, we cannot consecrate, we cannot hallow this ground.
The brave men, living and dead who struggled here have consecrated
it far above our poor power to add or detract. The world will
little note nor long remember what we say here, but it can never
forget what they did here.  It is for us the living rather to be
dedicated here to the unfinished work which they who fought here
have thus far so nobly advanced. It is rather for us to be here
dedicated to the great task remaining before us--that from these
honored dead we take increased devotion to that cause for which
they gave the last full measure of devotion--that we here highly
resolve that these dead shall not have died in vain, that this
nation under God shall have a new birth of freedom, and that
government of the people, by the people, for the people shall
not perish from the earth.

-- Gettysburg, Pennsylvania.  November 19, 1863
```

The following exercises are NOT to be handed in, but we encourage you to try to solve them.

1. Problem 16.3-6, p.392. Generalize Huffman code to ternary codewords.

   Here is a sketch of what is needed: in the following, the "degree" of a node in a tree is the number of its children. We note two facts:

   (a) In an optimal 3-ary code tree, any node of degree 2 must have leaves as both its children. Moreover, the depth of its children must be the height of the tree.
   (b) There are either no degree 2 nodes (if $|\Sigma|$ is odd) or one degree 2 node (if $|\Sigma|$ is even).

   Each of the above is easy to show. It is now easy to give an algorithm, using a priority queue structure just as in original Huffman tree algorithm.

2. Give an algorithm for the Grouping problem when the numbers may be negative.

   Let (M, w) be input. Assume that our goal is to compute the optimum cost for all $(M, w(0, i))$, $i = 2, \ldots, n$, where $w(i, j) = (w_{i+1}, \ldots, w_j)$ for $0 \leq i < j \leq n$. To compute $Cost(M, w)$ we simply choose it to be $1 + \min_j\{Cost(M, w(0, j))\}$ where j range over all those "candidate values" where $|w(0, j-1)| \leq M$. These candidate values can be determined in linear time. Hence $T(n) = n + T(n-1) = O(n^2)$.

3. Problem 17.4-3. Question on Dynamic Tables.
   Just follow the analysis in the text, case by case!