Homework 5 Solutions
Fundamental Algorithms, Spring 2008, Professor Yap

Due: Wed Apr 9, in class.
HOMEWORK with SOLUTION, prepared by Instructor and T.A.s

INSTRUCTIONS:

- Please read questions carefully. When in doubt, please ask.
- Please write succinctly, to the point. Every sentence must be an complete English sentence.
- You may post general questions to the homework discussion forum in class website. Also, bring your questions to recitation on Monday.

---

1. (0 Points)
   Exercise 1.1. (All Exercises in this homework refers to Lecture V)

   Suppose you are a cashier at a checkout and has to give change to customers. You want to give out the minumum number of notes and coins.
   (a) What is the greedy algorithm for this?
   (b) Assuming a US cashier giving change less than \$100. You have bills in denominations $\$50, \$20, \$10, \$5, \$1$ and common coins 25¢, 10¢, 5¢, 1¢. Is your greedy algorithm optimal here?
   (c) Give a scenario in which your greedy algorithm is non-optimal.

   SOLUTION: (a) Give out the maximum number of the largest denomination available..
   (b) Yes, it is optimal.
   (c) Suppose we have coins for 10¢, 8¢, 5¢, 1¢. To give back a change of 13¢, the greedy solution would use 4 coins. But 2 coins suffice.

2. (0 Points)
   Exercise 1.2. (Do not hand in problems with 0 Points.)

   Give a counter example to the greedy algorithm in case the w_i's can be negative.

3. (2 Points)
   Exercise 1.3.

   Suppose the weight w_i's can be negative. How bad can the greedy solution be, as a function of the optimal number of bins?

   SOLUTION: Arbitrarily bad! Consider the following input sequence on $n+1$ weights: $\{1, 1, \ldots, 1, -n\}$. The greedy algorithm would use n bins, but 1 bin suffices.

4. (0 Points)
   Exercise 1.4.

   There are two places where our optimality proof for the greedy algorithm breaks down when there are negative weights. What are they?

   SOLUTION: Two places: We cannot assume that $n_1 < m_1$. We cannot assume in the induction that we add additional riders to a car, its weight will increase.

5. (8 Points)
   Exercise 1.5.

Consider the following "generalized greedy algorithm" in case w_i's can be negative. A solution to linear bin packing be characterized by the sequence of indices, $0 = n_0 < n_1 < n_2 < \cdots < n_k = n$ where the ith car holds the weights

$$[w_{n_{i-1}+1}, w_{n_i+2}, \ldots, w_{n_i}].$$

Here is a greedy way to define these indices: let n_1 to be the largest index such that $\sum_{j=1}^{n_1} w_j \leq M$. For $i > 1$, define n_i to be the largest index such that $\sum_{j=n_{i-1}+1}^{n_i} w_j \leq M$. Either prove that this solution is optimal, or give a counter example.

-

SOLUTION: For a counterxample, let $M = 5$ and $w = (5, -6, 5, 5, 1)$. The -optimal answer is two bins, however the greedy algorithm will give -a solution with 3.

6. (10 Points)
Exercise 1.8. [NOTE: This WAS an open-ended problem.]

Consider an extension of linear bin packing where we now load two cars (called front and rear cars) at any one time. For example, for joy rides in a Ferris wheel, this is a realistic scenario. We are allowed to slightly violate the first-come first-serve policy: a rider can be assigned to the rear car, and the next rider in the queue can be assigned to the front car. But this is the worst that can happen (people coming behind in the queue can never be ahead by more than one car). We continue to make the "online requirement", i.e., we must make a decision for the ith rider before the $i + 1$st rider. Let the choice for the ith rider be denote $x_i \in \{1, 2, 3\}$ where $x_i = 1$ (resp., $x_i = 2$) means the ith rider goes into the front (resp., rear) car. Also $x_i = 3$ means we dispatch the front car (so the previous rear car becomes the front car, and we obtain a new empty rear car). Here is a greedy algorithm: *for each i, make x_i to be the smallest possible integer.* I.e., load into the front car if possible, otherwise try to load into the rear car if possible, and otherwise dispatch the front car. car to which we will assign the i-th rider. Prove or disprove that this strategy is at least as good as the original greedy algorithm for loading one car.

SOLUTION: If the input sequence of weights is $w = (w_1, \ldots, w_n)$, let $G_1(w)$ be the number of cars used by the original greedy method and $G_2(w)$ be the number of cars in the 2-car greedy method. We want to show that $G_2(w) \leq G_1(w)$. But let us prove the stronger statement: *if w is any weight sequence and w' is any subseqence of w then*

$$G_2(w') \leq G_1(w). \tag{1}$$

To make the comparison of G_1 with G_2 solutions clearer, we use this trick: let us re-interpret "w' is a subsequence of w" to mean that w' is obtained from w by replacing some of the weights of w by weight 0. E.g., if $w = (1, 2, 2, 1, 3, 1)$ and $w' = (2, 3, 1)$ we re-interpret w' as $(0, 2, 0, 0, 3, 1)$ (or $(0, 0, 2, 0, 3, 1)$ if we like). More generally, w' is a **subsequence** of w iff $w = (w'_1, \ldots, w'_n)$ has the same length as $w = (w_1, \ldots, w_n)$ and for each i, we have $w'_i \leq w_i$.
Our proof uses induction on the number of cars used by the G_1 solution. If $G_1(w) = 1$, then clearly (1) holds. Suppose $G_1(w) \geq 2$, and let the first car load C in the G_1 solution be the multiset $C = \{w_1, w_2, \ldots, w_i\}$ for some $i \geq 1$. So

$$G_1(w) = 1 + G_1(w_{i+1}, w_{i+2}, \ldots, w_n). \tag{2}$$

The first car load C' in the $G_2(w')$ solution must contain the multiset $\{w'_1, \ldots, w'_i\}$. Let w'' be the weight sequence that is obtained from $(w'_{i+1}, w'_{i+2}, \ldots, w'_n)$ by setting to weight 0 any weight that is in C'. Clearly, w'' is a subsequence of $(w'_{i+1}, \ldots, w'_n)$, and hence w'' is a subseqeunce of $(w_{i+1}, \ldots, w_n)$. By induction hypothesis, we conclude that

$$G_2(w'') \leq G_1(w_{i+1}, \ldots, w_n). \tag{3}$$

Thus our main claim (1) follows from (2), (3) and the following:

$$G_2(w') = 1 + G_2(w''). \tag{4}$$

To see this, let j be the largest index such that w'_j is in second car of the G_2 solution. Clearly, G_2 assigned w'_{j+1} to the third car, and it is at this moment that the first car was dispatched. We see that, at this moment, the multiset of weights assigned to the second car is precisely $\{w'_{i+1}, w'_{i+2}, \ldots, w'_j\} \setminus C'$. But this is precisely the set that would be put into the first car of the G_2 solution on input w''. This completes the proof of (4).

7. (10 Points)
Exercise 2.3.

Consider the activities selection problem. We now want to maximize the length $|A|$ of a set $A \subseteq S$ of activities. Define the **length** $|A|$ of a set compatible set S to be the length of all the activities in S (the length of an activity $I = [s, f)$ is just $|I| = f - s$). In case S is not compatible, its length is 0. Let $A_{i,j} = \{I_i, I_{i+1}, \ldots, I_j\}$ for $i \leq j$ and $F_{i,j}$ be an optimal solution for $A_{i,j}$.
(a) Show by a counter-example that the following "dynamic programming principle" fails:

$$F_{i,j} = \overline{\max}_{i \leq k \leq j-1} F_{i,k} \cup F_{k+1,j}$$

where $\overline{\max}\{F_1, F_2, \ldots, F_m\}$ returns the set F_ℓ whose length is maximum. (Recall that the length of F_ℓ is zero if it is not feasible.
(b) Give an $O(n \log n)$ algorithm for this problem. HINT: order the activities in the set S according to their finish times, say,

$$f_1 \leq f_2 \leq \cdots \leq f_n.$$

Consider the set of subproblems $S_i := \{I_1, \ldots, I_i\}$ for $i = 1, \ldots, n$. Use an incremental algorithm to solve $S_1, S_2, \ldots, S_n$ in this order.

SOLUTION: (a) Consider $A_1 = [1, 3), A_2 = [2, 6), A_3 = [3, 7)$.
(b) First lets sort the jobs by finish time so we have $f_1 \leq f_2 \leq - + \ldots f_n$. For $i = 1 \ldots n$, let $r(i)$ denote the largest j such -that $f_j <= s_i$. This is the job we can schedule before job i. -Defined $S_i = \{I_1, I_2, \ldots, I_n\}$, we can define the following -recurrence: $Opt(S_i) = \max\{Opt(S_{i-1}), |I_i| + Opt(S_{r(i)})$.

8. (10 Points)
Exercise 3.3.

Consider the following letter frequencies:

$$a = 5, b = 1, c = 3, d = 3, e = 7, f = 0, g = 2, h = 1, i = 5, j = 0, k = 1, l = 2, m = 0,$$
$$n = 5, o = 3, p = 0, q = 0, r = 6, s = 3, t = 4, u = 1, v = 0, w = 0, x = 0, y = 1, z = 1.$$

Please determine the cost of the optimal tree. NOTE: you may ignore letters with the zero frequency.

SOLUTION: The tree may not be unique, but should have cost of 212.

9. (15 Points)
Exercise 3.10.

For any binary full tree T, we have given two representations: the array A_T and the bit string α_T. Give detailed algorithms for the following conversion problems:
(a) To construct the string α_T from the array A_T.
(b) To construct the array A_T from the string α_T.

SOLUTION: The following solution is from your classmate, Jason Lee.
(a) Assume the input array is $A[1..n]$ encoding A_T, and the output array is $b[1..2n-1]$ encoding the α_T. If $A[i] < 0$, it means node i is a leaf.

```
arrayToBits (A, i)
    if A[i] < 0
        return '1'
    else
        return '0' + arrayToBits(A, i + 1) + arrayToBits(A, A[i]).
```

Note that we use the C++ convention where string concatenation is indicated by '+'.
(b) Assume the input bit string is $b[1..2n-1]$, and the output array is $A[0..2n-2]$. We will use a stack to keep track of the current path from the root to the current node. An integer *index* (initially 0) is used to assign index numbers to nodes. We scan the input $b[1..2n-1]$ sequentially; the output $A[0..2n-2]$ is filled in almost sequentially.
This is the inductive situation: suppose $index = i$, we are about to read $b[j]$. and the stack contains the values $0 \le s_1 < s_2 < \cdots < s_h$. This means the current node is i, and the entries of $A[0..i-1]$ have all been filled in except for $A[s_1], \ldots, A[s_h]$.
Consider the next bit $b[j]$:
If $b[j] = 0$, then we push i on the stack (so the stack now contains $0 \le s_1 < s_2 < \cdots < s_h < i$. That is because we are unable to fill in the array entry $A[i]$, so we just remember this.
If $b[j] = 1$, we assign $A[i] \leftarrow -1$ (i.e., node i is a leaf). We check if the stack is empty. If so, we are done. Otherwise, we pop the value s_h from stack. Now $A[s_h] \leftarrow i + 1$. Conceptually, we move up the tree to the last node s_h whose right child is not yet determined.

```
bitsToArray(b, n)
    Input: bit array b[1..2n − 1]
    Output: integer array A[0..2n − 2]
    S ← empty stack
    j ← 1; index ← 0
    while index ≤ 2n − 2 do
        ch ← b[j++]
        if (ch = 0)
            S.push(index++)
        else
            A[index++] ← −1
            if (S is non-empty)
                A[S.pop()] ← index
```

10. (10 Points)
What binary string would you transmit in order to send the string `now is the time`, under the dynamic Huffman algorithm? Show your working. Note: you would have to transmit ascii codes for the letters `n`, `o`, etc. Just write ASCII(`n`), ASCII(`o`), etc.

SOLUTION:

ASCII(`n`)0*ASCII*(`o`)00*ASCII*(`w`)100*ASCII*(`_`)
000*ASCII*(`i`)1100*ASCII*(`s`)1111000*ASCII*(`t`)0000
ASCII(`h`)11100*ASCII*(`e`)011111111111000*ASCII*(`m`)0011