ECS122A Homework Assignment #4 **Due: 4:00pm, February 8, 2016**

1. Suppose that we have a set of activities to schedule among a large number of lecture halls. We wish to schedule all the activities using as few lecture halls as possible. Given an efficient greedy algorithm to determine which activity should use which lecture hall. Provide the required time in the worst case, and justify the efficiency of your algorithm.

   Solution 1: (*we take this as a correct answer for grading*) An acceptable solution is to use the Greedy-Activitiy-Selection algorithm we learned in the class:

   - We first find a maximum-size set S_1 of compatible activities from S for the first lecture hall,
   - then using it again to find a maximum-size set S_2 of compatible activities from $S - S_1$ for the second hall,
   - and so on until all the activities are assigned.

   This requires $\Theta(n^2)$ time in the worst case.

   Solution 2: There is a better algorithm. *The general idea is to go through the activities in order of start time, assigning each to any hall that is available at that time.* To do this, move through the set of events consisting of activities starting and activities finishing, in order of event time. Maintain two lists of lecture halls. Halls that are busy at the current event-time t (because they have been assigned an activity i that started at $s_i \leq t$ but won't finish until $f_i > t$) and halls that are free at time t. [1] When $t =$ the start time of some activity, assign that activity to a free hall and move the hall from the free list to the busy list. When $t =$ the finish time of some activity, move the acitivity's hall from the busy list to the free list. (The activity is certainly in some hall, because the event times are processed in order and activity must have started before its finish time t, hence must have been assigned to a hall.)

   To avoid using more halls than necessary, always pick a hall that has already had an activity assigned to it, if possible, before picking a never-used hall. (This can be done by always working at the front of the free-halls list – putting freed halls onto the front of the list and taking halls from the front of the list – so that a new hall doesn't come to the front and get chosen if there are previously-used halls.)

   This guarantees that the algorithm uses as few lecture halls as possible: The algorithm will terminate with a schedule requiring $m \leq n$ lecture halls. Let activity i be the first activity scheduled in lecture hall m. The reason that i was put in the mth lecture hall is that the first $m - 1$ lecture halls were busy at time s_i. So at this time there are m activites occuring simultaneously. Therefore any schedule must use at least m lecture halls, so the schedule returned by the algorithm is optimal.

   Run time:

   - Sort the $2n$ activity-starts/activity-ends events. (In the sorted order, an activity-ending event should precede an activity-starting event at the same time.) $O(n \lg n)$ time for arbitrary times, possibly $O(n)$ if the times are restricted (e.g. to small integers).
   - Process the events in $O(n)$ times: Scan the $2n$ events, doing $O(1)$ work for each (moving a hall from one list to the other and possibly associating an activity with it).

[1] As in the activity-selection problem in section 16.1 we are assuming that activity time intervals are half open, i.e., that if $s_i \geq f_j$, activities i and j are compatible.

Total: $O(n + \text{time to sort})$.

2. We are given n jobs $j_1, j_2, \ldots, j_n$, with known running time $t_1, t_2, \ldots, t_n$, respectively. We have a single processor. We want to schedule these jobs so as to minimize the avarage completion time? Give a greedy algorithm for the job scheduling problem, and verify the optimality. (*We assume nonpreemptive scheduling, i.e., once a job is started, it must run to completion.*)

   Solution: The greedy solution is to arrange the jobs by shortest job first.

   We can show that this yields an optimal schedule. Let the running time $\{t_i\}$ of n jobs be sorted such that

   $$t_{i_1} \le t_{i_2} \le t_{i_3} \le \cdots \le t_{i_n}.$$

   Then the first job finishes in time t_{i_1}, and the second job finishes after $t_{i_1} + t_{i_2}$, and the third job finishes after $t_{i_1} + t_{i_2} + t_{i_3}$, and so on. Hence the *average completion time* is

   $$\begin{aligned} A &= \frac{1}{n}\left[t_{i_1} + (t_{i_1} + t_{i_2}) + (t_{i_1} + t_{i_2} + t_{i_3}) + \cdots + (t_{i_1} + t_{i_2} + \cdots + t_{i_n})\right] \\ &= \frac{1}{n}\left[n\, t_{i_1} + (n-1)t_{i_2} + (n-2)t_{i_3} + \cdots + t_{i_n}\right] \\ &= \frac{1}{n}\sum_{k=1}^{n}(n-k+1)t_{i_k} \\ &= (1+\frac{1}{n})\sum_{k=1}^{n} t_{i_k} - \frac{1}{n}\sum_{k=1}^{n} k \cdot t_{i_k}. \end{aligned}$$

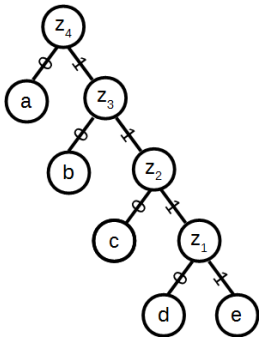
   Note that the first term is independent of the job ordering, so only the second term affects the average completion time. Let $t_{i_\ell} < t_{i_m}$ for some ℓ and m. Then it is easy to see that by swapping t_{i_ℓ} and t_{i_m}, the second term decreases, increasing the total average completion cost. Therefore, A is optimal.

3. Suppose the symbols a, b, c, d, e occur with frequencies 1/2, 1/4, 1/8, 1/16, 1/16, respectively.

   (a) What is the Huffman codes of the alphabet?

   (b) If this coding is applied to a file consisting 1,000,000 characters with the given frequencies, what is the length of the encoded file in bits?

   Answer: (a) By the Huffman coding, we have the following full binary tree T and the corresponding codewords.



char	codes
a	0
b	10
c	110
d	1110
e	1111

   (b) The number of bits

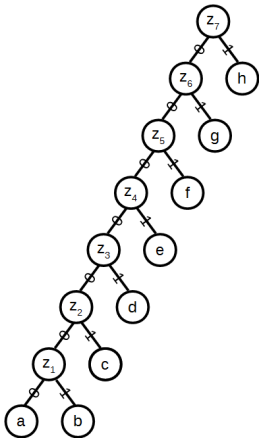
   $$B(T) = (1000000/2)\cdot 1 + (1000000/4)\cdot 2 + (1000000/8)\cdot 3 + 2\cdot(1000000/16)\cdot 4 = 1875000$$

4. (a) What is an optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci number?

`a:1, b:1, c:2, d:3, e:5, f:8, g:13, h:21`

(b) Generalize your answer to find the optimal code when the frequencies are the first n Fibonacci numbers?

Solution: (a) We can construct the full binary tree and corresponding codewords as the following:



char	frequency	code
h	21	1
g	13	01
f	8	001
e	5	0001
d	3	00001
c	2	000001
b	1	0000001
a	1	0000000

(b) the codeword for the ith character in a sequence of n characters where the frequencies are the first n Fibonacci numbers, is "$\underbrace{00\cdots0}_{n-1}$" for $i = 1$, and "$\underbrace{00\cdots0}_{n-i}1$" for $2 \le i \le n$

Note that it is also correct if the answer is "$\underbrace{11\cdots1}_{n-1}$" for $i = 1$, and "$\underbrace{11\cdots1}_{n-i}0$" for $2 \le i \le n$

5. We use Huffman's algorithm to obtain an encoding of alphabet $\{a, b, c\}$ with frequencies f_a, f_b and f_c. In each of the following cases, either give an example of frequencies $\{f_a, f_b, f_c\}$ that would yield the specified code, or explain why the code cannot possibly be obtained (no matter what the frequencies are)

(a) Code: $\{0, 10, 11\}$ (b) Code: $\{0, 1, 00\}$ (c) Code: $\{10, 01, 00\}$

Solution.

(a) the frequencies $(2/3, 1/6, 1/6)$ gives the code $\{0, 10, 11\}$

(b) This encoding is not possible, since the code "0" for the letter "a" is a prefix of the code "00" for the letter c.

(c) This code is not optimal since the code $\{1, 01, 00\}$ gives a shorter encoding. Also, it does not correspond to a **full** binary tree and hence cannot be given by the Huffman algorithm.

6. The 0-1 knapsack problem. Given six items $\{(v_i, w_i)\}$ for $i = 1, 2, \ldots, 6$ as follows:

i	v_i	w_i
1	40	100
2	35	50
3	18	45
4	4	20
5	10	10
6	2	5

and the total weight $W = 100$, where v_i and w_i are the value and weight of item i, respectively. Find the greedy solutions by using following strategies:

(a) Greedy by value, i.e., at each step select from the remaining items the one with the highest value

(b) Greedy by weight, i.e., at each step select from the remaining items the one with the least weight.

(c) Greedy by value density, i.e., at each step select from the remaining items with the largest value per pound ratio v_i/w_i.

Are these greedy solutions optimal? Comment your findings.

Solution: The following table shows the greedy solutions:

				Greedy by		
item i	value v_i	weight w_i	density v_i/w_i	value	weight	density
1	40	100	0.4	1	0	0
2	35	50	0.7	0	0	1
3	18	45	0.4	0	1	0
4	4	20	0.5	0	1	1
5	10	10	1.0	0	1	1
6	2	5	0.4	0	1	1
Total value				40	34	51
Total weight				100	80	85

Note that all three approaches generate feasible solutions, but **none of them generate the optimal solution**.

The optimal solution, which can be found by dynamic programming (to do in homework 5), is to take items $\{2, 3, 6\}$. Then the total value is 55, and total weight is 100.