Assignment 4
COSC 3P03: Design and Analysis of Algorithms
Winter 2017

Due: April 7, Friday, 5:00 PM.

1. (20) For the chained matrix multiplication problem where we are supposed to find an optimal order by which to calculate $M = M_1 \times M_2 \times \cdots \times M_n$, we know that we can use the technique of dynamic programming to solve the problem. For each of the following suggested greedy ideas, provide a counter example where the greedy solution does not work:
(a) First multiply the matrices M_i and M_{i+1} whose common dimension r_i is smallest, and continue in the same way (note that M_i is a r_{i-1} by r_i matrix);
(b) First multiply the matrices M_i and M_{i+1} whose common dimension r_i is largest, and continue in the same way;
(c) First multiply the matrices M_i and M_{i+1} that minimize the product $r_{i-1}r_ir_{i+1}$, and continue in the same way;
(d) First multiply the matrices M_i and M_{i+1} that maximize the product $r_{i-1}r_ir_{i+1}$, and continue in the same way.

Note that when applying a greedy strategy, the idea is to be carried out through the entire process until an ordering of the multiplications has been found (in other words, you don't just apply it once).

(a) Consider $M_{2\times1} \times M_{1\times2} \times M_{2\times3}$:

- Greedy solution: $((M_{2\times1} \times M_{1\times2}) \times M_{2\times3})$: cost 16
- Another solution $(M_{2\times1} \times (M_{1\times2} \times M_{2\times3}))$: cost 12

(b) Consider $M_{1\times2} \times M_{2\times3} \times M_{3\times4}$:

- Greedy solution: $(M_{1\times2} \times (M_{2\times3} \times M_{3\times4}))$: cost 32
- Another solution $((M_{1\times2} \times M_{2\times3}) \times M_{3\times4})$: cost 18

(c) see (a).

(d) see (b).

2. (20) We have unlimited number of bins each of capacity 1, and n objects of sizes s_1, s_2, ..., s_n, where $0 < s_i \leq 1$. Our job is to pack these objects into bins using the fewest bins possible (optimization). The decision version can be stated as follows: given n objects and k, is there a packing using no more than k bins. Show how to solve one version if you know how to solve the other version, i.e., show how to solve the decision version by solving the optimization version and show how to solve the optimization version by solving the decision version.

Optimization to Decision: Solve optimization, with m bins solution. If $m \leq k$, yes to decision, else, no to decision.

Decision to Optimization: try decision version with $k = 1$, $k = 2$, ... The k for the first yes

answer is the answer.

3. (10) Consider the activity selection problem we studied. Suppose that instead of always selecting the first activity to finish (the strategy we used in class), we instead select the last activity to start that is compatible with all previously selected activities. Prove that it yields an optimal solution.

This algorithm is symmetric to the one we did in class where we always select, among the compatible activities, the one with the smallest finish time. So in order to prove its optimality, all we have to do is to show that there is one optimal solution that includes activity a_n, assuming that all activities are sorted by their start times and w.l.o.g, we have $s_1 \leq s_2 \leq \cdots \leq s_n$. Thus, the proof is similar to the one we did in class.

All we need to do is to show that there is an optimal solution that contains the last activity to start. Let A be an optimal solution and a_k the activity with the latest start time. If a_k is a_n, namely, $k = n$, we are done. Otherwise, we replace a_k from A with a_n. Clearly, a_n is compatible with all other activities in A since $s_k \leq s_n$. Thus the new solution is also optimal.

4. (20) Sequences x_1, x_2, ..., x_n is superincreasing if each number exceeds the sum of the preceding numbers. That is, for $j = 2, 3, \cdots, n$,

$$x_j > \Sigma_{i=1}^{j-1} x_i.$$

For the subset sum problem $\{x_1, x_2, \cdots, x_n\}$ and b, find an efficient algorithm for a solution if sequence x_1, x_2, ..., x_n is superincreasing.

We will start from x_n. If $x_n \leq b$, we know x_n has to be in the solution if the solution exists because otherwise, $\Sigma_{i=1}^{n-1} < b$ and there won't be a solution. In this case, we include x_n in the solution and create a new $b = b - x_n$. If $x_n > b$, we consider our problem without x_n. We repeat the above all the way to x_1.

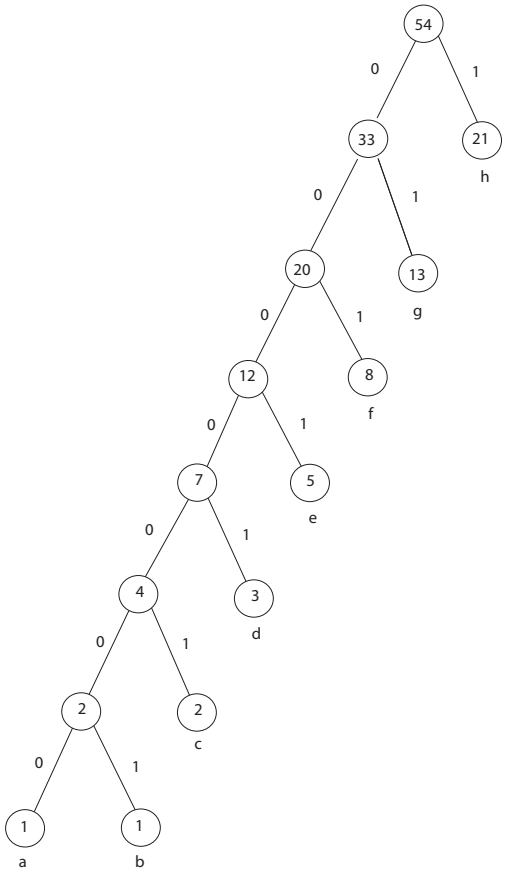
The running time is $O(n)$.

5. (20) What is an optimal Huffman code for the following set of frequencies, $a : 1$, $b : 1$, $c : 2$, $d : 3$, $e : 5$, $f : 8$, $g : 13$, $h : 21$, based on the first 8 Fibonacci numbers? Use the convention that a left edge is always labeled with a 0 and a right edge with 1. What is the expected (average) length of the code? Generalize your answer to find the optimal code when the frequencies are the first n Fibonacci numbers.

What is the average length of the Huffman code for the first n characters whose frequencies are the first n Fibonacci numbers. Note that we want a closed-form (explicit) solution. (Hint: $f(1) + f(2) + ...f(k) = f(k + 2) - 1$).

Actual code:

a : 0000000
b : 0000001
c : 000001
d : 00001
e : 0001
f : 001
g : 01
h : 1



Average code length: Let t be 1+1+2+3+5+8+13+21, i.e., $t = 54$. Then we have $\Sigma_{i=1}^{8} p_i l_i = (1*7 + 1*7 + 2*6 + 3*5 + 5*4 + 8*3 + 13*2 + 21*1)/t = 132/54 = 22/9 = 2.444...$

It is not hard to generalize the above to n characters: C_1, C_2, ...,, C_n with frequencies $f(1)$, $f(2)$, ..., $f(n)$.

Actual code:

$$\begin{aligned}
&C_1 : 000...000 \quad (n-1 \ bits)\\
&C_2 : 000...001 \quad (n-1 \ bits)\\
&C_3 : 000...01 \quad (n-2 \ bits)\\
&\vdots\\
&C_{n-1} : 01 \ (2 \ bits)\\
&C_n : 1 \ (1 \ bit)
\end{aligned}$$

The corresponding Huffman tree is similar to the one above with n leaves.

We now compute the average length of the above Huffman code:

The formula is given: $f(1) + f(2) + ... + f(k) = f(k+2) - 1$. The average length is:

$$[(n-1)f(1) + (n-1)f(2) + (n-2)f(3) + \cdots + 2f(n-1) + 1f(n)]/[f(1) + f(2) + \cdots + f(n)] = x/[f(n+2) - 1],$$

where

$$\begin{aligned}
x &= (n-1)f(1) + (n-1)f(2) + (n-2)f(3) + ... + 2f(n-1) + f(n)\\
&= f(1) + f(2) + f(3) + ...f(n-1) + f(n) +\\
&\quad f(1) + f(2) + f(3) + ...f(n-1) +\\
&\quad ..... +
\end{aligned}$$

$$f(1) + f(2) + f(3) + f(4) +$$
$$f(1) + f(2) + f(3) +$$
$$f(1) + f(2)$$

which is (by using the given formula):

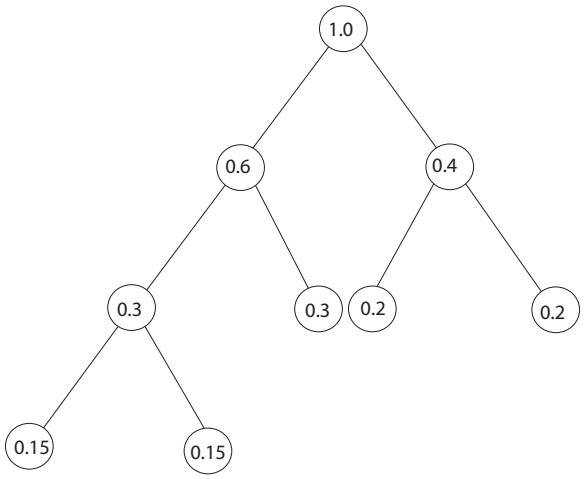
$$\begin{aligned}
& [f(n+2)-1] + [f(n+1)-1] + ...[f(6)-1] + [f(5)-1] + [f(4)-1] \\
= \quad & f(4) + f(5) + ... + f(n+1) + f(n+2) - (n-1) \\
= \quad & [f(1) + f(2) + f(3) + f(4) + f(5) + ... + f(n+1) + f(n+2)] - (n-1) - [f(1) + f(2) + f(3)] \\
= \quad & f(n+4) - 1 - (n-1) - (f(5) - 1) \\
= \quad & f(n+4) - f(5) - n + 1
\end{aligned}$$

Therefore, the average length is $[f(n+4) - f(5) - n + 1]/[f(n+2) - 1]$. When $n = 8$, we have average length:

$$\begin{aligned}
& [f(12) - f(5) - 8 + 1]/[f(10) - 1] \\
& = (144 - 5 - 8 + 1)/(55 - 1) \\
& = 132/54 \\
& = 22/9 \\
& = 2.444...
\end{aligned}$$

6. (10) Fill in the nodes with frequencies (numbers larger than 0 and less than 1) so that it is a legal Huffman tree:

One possible solution is as follows:



7. (20) Suppose that in a 0-1 knapsack problem, the order of the items when sorted by increasing weight is the same as their order when sorted by decreasing value. Give an efficient algorithm to find an optimal solution to this variant of the knapsack problem, and argue (i.e., prove) that your algorithm is correct. What is the running time of your algorithm?
W.l.o.g.,assume that the input is such that

$$\begin{aligned}
objects: \quad & b_1 \quad b_2 \quad \cdots \quad b_{n-1} \quad b_n \\
values: \quad & v_1 \geq v_2 \geq \cdots \geq v_{n-1} \geq v_n \\
weights: \quad & w_1 \leq w_2 \leq \cdots \leq w_{n-1} \leq w_n
\end{aligned}$$

We also assume that the total weight is more than W, otherwise we can just pack everything into the knapsack.

The greedy algorithm is to select the objects in the given order 1, 2, 3, ..., until k such that

$$w_1 + w_2 + \cdots w_k \leq W$$
$$w_1 + w_2 + \cdots w_k + w_{k+1} > W.$$

The running time is $O(n)$ if the input is already sorted (by values or by weights), $O(n \log n)$ otherwise. The proof of its optimality is given as follows:

Let $A : b_{i_1}, b_{i_2}, \cdots, b_{i_j}$ be an optimal solution sorted by weights such that $w_{i_1} \leq w_{i_2} \leq \cdots \leq w_{i_j}$. Compare A with our greedy solution $G : b_1, b_2, \cdots b_k$. If $A \neq G$, then there exists m, $1 \leq m \leq n$, such that

$$\begin{aligned} i_1 &= 1 \\ i_2 &= 2 \\ \cdots i_{m-1} &= m-1 \\ i_m &\neq m. \end{aligned}$$

In other words, we compare A with G object by object until we find the first object where they differ. Also, it must be that $i_m > m$. By the greedy algorithm and the condition, we have $w_{i_m} \geq w_m$ and $v_{i_m} \leq v_m$. So we can replace b_{i_m} with b_m in A without increasing the total weight. If $v_{i_m} < v_m$, we get a solution better than the optimal A, a contradiction. If $v_{i_m} = v_m$, we just obtained another optimal solution which has one more object in common with G. So we can repeat the same argument until we get an optimal solution which is the same as G or a contradiction shown above.

Another proof would proceed as follows: w.l.o.g., assume that objects in A are sorted:

$$\begin{aligned} objects: &\quad b_{i_1} \quad b_{i_2} \quad \cdots \quad b_{i_j} \\ values: &\quad v_{i_1} \geq v_{i_2} \geq \cdots \geq v_{i_j} \\ weights: &\quad w_{i_1} \geq w_{i_2} \geq \cdots \geq w_{i_j} \end{aligned}$$

Since $v_k \geq v_{i_k}$ and $w_i \leq w_{i_k}$, $1 \leq k \leq j$, we know immediately that $v_1 + v_2 + \cdots + v_j \geq v_{i_1} + v_{i_2} + \cdots + v_{i_j}$ and $w_1 + w_2 + \cdots + v_j \leq w_{i_1} + w_{i_2} + \cdots + w_{i_j}$.