Assignment 4
COSC 3P03: Design and Analysis of Algorithms
Winter 2016

Due: April 1, Friday, 5:00 PM.

1. (10) 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. (20) (a) Let P_1, P_2, ..., P_n be a set of n programs that are to be stored on a memory chip of capacity L. Program P_i requires a_i amount of space. Note that L and a_i's are all integers. Clearly, if $a_1 + a_2 + ...+ a_n \leq L$, then all the programs can be stored on the chip. So assume that $a_1 + a_2 + ... + a_n > L$. The problem is to select a maximum subset Q of the programs for storage on the chip. A maximum subset is one with the maximum number of programs in it. Give a greedy algorithm that always finds a maximum subset Q such that

$$\sum_{P_i \in Q} a_i \leq L.$$

You also have to prove that your algorithm always finds an optimal solution. Note that a program is either stored on the chip or not stored at all (in other words, you do not store part of a program).
(b) Suppose that the objective now is to determine a subset of programs that maximizes the amount of space used, that is, minimizes the amount of unused space. One greedy approach for this case is as follows: As long as there is space left, always pick the largest remaining program that can fit into the space. Prove or disprove that this greedy strategy yields an optimal solution.

(a) Greedy: always pick a smallest program.
Proof: All we need to do is to show that there is an optimal solution that includes the smallest program. Let $\{P_{i_1}, P_{i_2}, \cdots, P_{i_k}\}$ be an optimal solution with $a_{i_1} \leq a_{i_2} \leq \cdots \leq a_{i_k}$. If a_{i_1} is the smallest, done. Else, replaceP_{i_1} with the smallest and we still have an optimal solution.

(b) No. Example: $P_1 : 3$, $P_2 : 2$, $P_3 : 2$, $L = 4$. Greedy: P_1 with unused space 1; Optimal: P_2 and P_3 with unused space 0.

4. (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

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

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 $w_1 + w_2 + \cdots + v_j \leq w_{i_1} + w_{i_2} + \cdots + w_{i_j}$.

5. (10) Consider the machine scheduling problem studied in class where we have n jobs with start and finish times for job i being s_i and f_i, $s_i < f_i$ and the goal is to schedule them using fewest machines possible. Show that the greedy idea that always selects the shortest job first does not always give an optimal solution.

An easy way to represent a job with a starting time s and finish time f is to use horizontal line segment of length $f - s$.

In the following example, shortest first strategy uses three machines and the optimal one uses two machines.

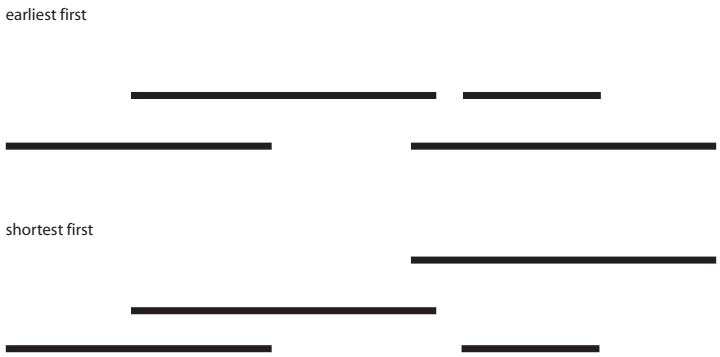

Figure 1: A Counter Example.

6. (20) The greedy algorithm to solve the fractional knapsack problem optimally selects objects in decreasing value to weight ratios (discussed in class). For the 0-1 knapsack problem:
(a) show that the greedy algorithm does not work;
(b) show that the greedy method not only does not work, it yields arbitrarily bad solutions, that is, given any $0 < \epsilon < 1$, construct an example where the ratio of the greedy solution to the optimal solution is $< \epsilon$. Note that if you are able to (b), then your answer will also serve as an answer to (a). In this case, there is no need to do (a). On the other hand, if you are unable to do (b), you should try to do (a).

(a) One of the many such simple examples is as follows: We have three objects with values 2, 2, and 3.1 whose weights are 2, 2, and 3 with the knapsack capacity 4. The greedy solution would pick object 3 with value 3.1 while the optimal solution picks first two objects with a total value of 4.

(b) The capacity W is 3, and w.l.o.g, assume $\epsilon < 1$.

$Objects$	1	2
$Values$	2ϵ	3
$Weights$	ϵ	3
value to weight ratio	2	1

The greedy solution's value is 2ϵ and the optimal solution has value 3. The ratio is $2\epsilon/3 < \epsilon$.