Madhubhani

A. Habrard — Master DSC/CPS2/MLDM

Advanced Algorithms

Final Exam – 18th December 2019

Duration: 2 *h* — Number of page(s): 2.

Only documents from the course or personal notes are permitted. Before you begin, it is recommended to read all the document.

Grading scale (temporary)

Part	1	2	3
on	5 pts	10 pts	5 pts

Part 1 Complexity (5 points)

1. Order increasingly the following functions with respect to their growth rate $O(\cdot)$: $n!, \sqrt{n^3}, cos(n), n/log(n)$
2. True or false? Justify your answers.
   (a) If $f_1(n) \in O(f_2(n))$, $g(n) \in O(f_1(n))$ and $h(n) \in O(f_2(n))$ then $g(n) + h(n) \in O(f_2(n))$.
   (b) $(n+1)! \in O(n!)$.
3. Consider the following two functions: $f_1(n) = \begin{cases} n, \text{if } n \text{ is odd} \\ n^3, \text{if } n \text{ is even} \end{cases}$ and $f_2(n) = \begin{cases} n, \text{if } n \leq 100 \\ n^3, \text{if } n > 100. \end{cases}$
   Compare them according to the $O(\cdot)$ and $\Omega(\cdot)$ relationships.
4. Solve the following recurrence: $T(n) = 2T(\sqrt{n}) + 2\log_2(\sqrt{n})$, with $n \geq 4$ and $T(2) = 1$.

Part 2 Problem: Music Copy (10 points)

We wish to copy some musical pieces on two USB keys. We assume that each key can record 100 minutes of music. We have on a laptop n musical pieces corresponding to M minutes of music with $M > 200$, thus we need to select which pieces to copy on each key (one piece can be copied only once).

In order to simplify the problem, we assume that the duration of each piece is an integer corresponding to its duration expressed in minutes, each piece has a duration of 1 minute at least. We denote by $t = 100$ minutes, the duration available on each key, and by d_i the duration of the piece number i $(1 \leq i \leq n)$ on the laptop. We aim at **maximizing the total duration of music recorded on each key** and we assume that the music pieces cannot be broken. We consider two approaches.

Exercice 2.1 A greedy approach

1. Propose a greedy algorithm to solve the problem.
2. Give the complexity of your approach.
3. Show that your approach is not optimal (you can change t to a smaller value if it helps).
4. Can you give another setting where your greedy approach could work?

Scanned with CamScanner

Exercice 2.2 A dynamic programming approach

1. Let $duration(i, t_1, t_2)$ be the total music duration that can be copied from the first i pieces ($0 \leq i \leq n$) with t_1 minutes available on the first key and t_2 minutes available on the second one. Note that $duration(n, t, t)$ gives then the solution to the problem.

   Define the values of $duration(\cdot, \cdot, \cdot)$ for an **optimal solution** in a recursive manner.

   Hint: In an optimal solution, the i^{th} piece is recorded on the first key, or on the second, or on none of them.
2. Justify that the optimal substructure property exists.
3. Give a recursive definition of *duration* to compute the optimal solution.
4. Deduce a dynamic programming approach for computing this solution.
5. Provide a way to know for each song if it has to be copied or not and where.
6. Give the complexity (time and space) of your algorithm, justify that it is polynomial or not.

Part 3 Heavier and clever (5 points)

We consider a set of $n \geq 2$ elephants. Each elephant i is represented by a triplet $(weight(i), intelligence(i), cost(i))$ which gives respectively its weight, its level of intelligence and its cost. We assume here that all the weights and the intelligence values are different and we look for the subset S of elephants that satisfies the following 2 conditions:

a. for all $i, j \in S$: $(weight(i) < weight(j)) \Leftrightarrow (intelligence(i) < intelligence(j))$

b. the sum $\sum_{i \in S} cost(i)$ is maximum, *i.e.* no other subset of elephants satisfying the condition [a.] just above has a higher subset cost.

For example, consider the 6 elephants presented at the bottom of the page: 8 subsets of at least 2 elephants fulfill the condition [a.] above: $\{1,3\}, \{1,6\}, \{3,4\}, \{3,5\}, \{3,6\}, \{5,6\}, \{1,3,6\}, \{3,5,6\}$. Among them, the optimal is $\{3,4\}$ because its total cost of 90 is the maximum among the 8 solutions.

1. Try to define an approach finding the optimal solution by enumerating all the solutions with a branch and bound method. What is the complexity?
2. Try to define another approach with **dynamic programming**. Hint: this problem can be formulated as the search of a common subsequence of maximum cost between two sequences:
   - A sequence X of the elephants (*i.e.* their index i) in increasing order of intelligence.
   - A sequence Y of the elephants (*i.e.* their index i) in increasing order of weights.

   You can here re-adapt the formulation of the longest common subsequence: define the recursive formula for computing an optimal solution of max cost, give the pseudo-code and the complexity.

<u>Bonus</u> Apply the dynamic programming solution to the example provided above, describe how you obtain the optimal solution.

elephant i	1	2	3	4	5	6
weight	2300	2000	2800	2100	2500	2600
intelligence	7	14	13	11	6	9
cost	10	80	40	50	20	15

Scanned with CamScanner

(1) $f_1 = n!$

$f_2 = \sqrt{n^3}$

$f_3 = \cos n$

$f_4 = \frac{n}{\log n}$

$$\lim_{n\to\infty} \frac{n!}{\sqrt{n^3}} = \lim_{n\to\infty} \frac{n(n-1)(n-2)\cdots 1}{n^{\frac{3}{2}}}$$

$$= \infty$$

$O(n!) > O(\sqrt{n^3})$

$n! > \sqrt{n^3} > \frac{n}{\log n} > \cos n$

2) a) $f_1(n) \in O(f_2(n))$, $f_1(n) \le c_1 f_2(n) \ \forall n \ge n_1$

$g(n) \in O(f_1(n))$, $g(n) \le c_2 f_1(n) \ \forall n \ge n_2$

$h(n) \in O(f_2(n))$, $h(n) \le c f_2(n) \ \forall n \ge n_3$

$\therefore g(n) + h(n) \le c_1 f_2(n) + c_2 f_1(n) \ \forall n \ge \max\{n_1, n_2\}$

$\le c f_2(n) \ \forall n \ge \max\{n_1, n_2\}$

$\therefore g(n) + h(n) = O(f_2(n))$

(b) $(n+1)! = (n+1) \cdot n(n-1) \cdots 1 > n(n-1) \cdots 1$
$= n!$

$\therefore (n+1)! \le c n!$

$\Rightarrow c \ge (n+1)$ Then, there's no c.

(3) $f_1(n) = \begin{cases} n & \text{if } n \text{ is odd.} \\ n^3 & \text{if } n \text{ is even.} \end{cases}$

$f_2(n) \ge f_1(n) \ \forall n \ge 100$

So, $f_1(n) = O(f_2(n))$

$f_2 = \Omega(f_1)$

$f_2(n) = \begin{cases} n & \text{if } n \le 100 \\ n^3 & \text{if } n \ge 100 \end{cases}$

$\textcircled{4}$ $T(n) = 2T(\sqrt{n}) + 2\log_2(\sqrt{n})$, $n \geq 4, T(2) = 1$

$\therefore n = 2^K \Rightarrow K = \log n$

$\therefore T(2^K) = 2T(2^{\frac{K}{2}}) + 2\log_2(2^{\frac{K}{2}})$

$\Rightarrow T(2^K) = 2T(2^{\frac{K}{2}}) + \cancel{2} \cdot \frac{K}{\cancel{2}} \ln 2$

$\Rightarrow T(2^K) = 2T(2^{\frac{K}{2}}) + K$

$\therefore F(K) = 2F(\frac{K}{2}) + K$ $\qquad \therefore 2^n = K$

$\qquad \therefore n \doteq \log K$

$= 2^2 \cdot F(\frac{K}{2^2}) + \frac{K}{2} + K$

$= 2^3 \cdot F(\frac{K}{2^3}) + \frac{K}{2^2} + \frac{K}{2} + K$

$= 2^{\log K} + K(1 + \frac{1}{2} + \frac{1}{2^2} + \text{——} + \frac{1}{2^{\log K}})$

$= K + K \cdot \left(\frac{1 - (\frac{1}{2})^{\log K + 1}}{1 - \frac{1}{2}} \right) = K + 2K(1 - \frac{1}{2^{\log K + 1}})$

$= \theta(\log n)$