A. Habrard   Master CPS2/DSC/MLDM

Advanced Algorithms

Examination – 15th December 2020

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, according to their growth rate $O(\cdot)$: e^n, $\frac{n}{3}$, $\sin(n) * \cos(n)$, $n/\log(n)$.

2. Let $T(n) = 6T(n/2) + n^2$ and $T'(n) = aT'(n/4) + n^2$. What is the greatest value of a that makes T' asymptotically faster than T? Justify your answer.

3. Give the complexity of the algorithms defined by the following recurrences:

   (a) $T(n) = 2T(\sqrt{n}) + \log_2(n)$, with $n \geq 4$ and $T(2) = 1$.

   (b) $T(n) = 2T(\frac{n}{2}) + \frac{n}{2}$, $T(2) = 1$.

Part 2 Assignment of duration to tasks (10 points)

We consider n tasks $T_1, \ldots, T_n$, each of which can be achieved in different durations (a duration is encoded as an integer between 1 and k). The cost of each task varies with respect to the duration taken for finalizing it. $c(i,d)$ represents the cost for the task i when it is executed in d units of time - this cost is expressed as integers between 10 and 200. For each task i, the cost $c(i,d)$ decreases as the duration allocated to the task increases, except when the task cannot be completed in the allocated time. If a task cannot be completed for a given duration, then it cannot be achieved for any longer duration. As an illustration consider the following example with $n = 4$ tasks and up to $k = 5$ units of time given in Table 1.

duration d:	1	2	3	4	5
Task T_1	110	90	66	65	∞
Task T_2	120	90	70	50	40
Task T_3	90	70	65	60	∞
Task T_4	65	60	55	∞	∞

Table 1: Illustration of the problem on 4 tasks and a maximum of 5 units of time per task.

The value ∞ indicates that the corresponding task cannot be completed with this duration. The tasks 1, 2 and 4 cannot be completed in 5 units of time and task 4 can even not be achieved in 4 time units.

The objective is to find an optimal assignment of the duration for each task such that for a fixed global duration to be shared for all the tasks: $D = \sum_{i=1}^{n} d_i$, the sum of the costs $SC = \sum_{i=1}^{n} c(i, d_i)$ is minimum. We denote *scopt*(i, d) the cost associated to the optimal assignment using a total of d time units share between the tasks T_1 to T_i, where $d \leq D$ and $1 \leq i \leq n$.

1. Given a fixed global duration of D time units to be shared among all tasks, propose a greedy algorithm for finding an optimal assignment for the available tasks. Give its complexity.
2. If you think that your greedy algorithm is optimal give the proof, otherwise justify that it is not optimal.
3. We now want to design a dynamic programming algorithm. Give a recursive definition for *scopt*(i, d) and justify its optimality.
   Hint: for a given task i you have to find the time units allowing to minimize to tradeoff between the cost for the task and the costs for the remaining tasks. Note also that each task needs at least one time unit and cannot use more than k time units.
4. Design a dynamic programming algorithm for finding an optimal assignment Give the time and space complexities of your algorithm.
5. Illustrate the behavior of your algorithm on the instance given in Table 1 and using $D = 10$.

Part 3 A Branch-and-Bound algorithm (5 points)

We want to solve a constraint satisfaction problem over **integers** with a *branch and bound* algorithm. Let $X1$ and $X2$ be two variables defined over the set of positive integer between 0 and 7 ($\{0, 1, 2, 3, 4, 5, 6, 7\}$). We want to **maximize** the following function with respect to $X1$ and $X2$:

$$f(X1, X2) = 3 * X1 + 2 * X2,$$

such that the next two constraints are satisfied:

$$X1 \geq 2 - X2 \quad \text{and} \quad X1 \leq 6 - 1.5 * X2.$$

1. Represent graphically in a 2-dimensional plot the possible solutions for $X1$ and $X2$ satisfying the two constraints defined previously. (the quality of the plot will be taken into account).
2. First, we want to design a *brute-force* approach for finding the best values for $X1$ and $X2$. The algorithm can take as input the value of the best current solution, the constraints and the considered domain for each of the two variables. The search tree considered by the algorithm could be built such that internal nodes can try to reduce the size of the domains of the variables.

   For example, the root of the tree can start with $X1 \in [0, \dots, 7]$ and $X2 \in [0, \dots, 7]$, and children of the root can consider possible domain reductions such as $X1 \in [0, \dots, 3]$ and $X2 \in [4, \dots, 7]$), the final values of the variables being affected at the leaves of the tree.
   The domain reduction can be done in a dichotomous way by trying to reduce the size of the domains by two. In the search tree internal nodes can admit up to four children (the four possible ways to reduce the two domains by dichotomous splits).
   Try to represent graphically the search tree that your algorithm must consider. Then propose an algorithm to solve the problem.
3. Improve the previous algorithm by adding a *branch and bound* strategy. You could try to compare the two lower bounds and the two upper bounds of the domains of the two variables, *i.e.* check that the values in the domains can still satisfy the constraints.