Algorithmic Problem Solving

MoSIG 2018–2019 Exam — Semester 1 — Session 1

December 7th, 2018

This exam has 2 exercises, for a total of 23 points. Obtaining 20 points is enough to get the best possible grade. The marking scheme is there to help you in discriminating the important questions from the easy ones and is subject to slight changes.
All 2 exercises are independent and can be solved in any order, however, **you must use a different paper sheet to answer both exercises.**

All documents forbidden except one A4 sheet handwritten. All other materials forbidden (including: books and electronic devices).

Exam duration: 3h.

I The Problem of Mushroom Pickers [11 points]

You manage a mushroom-selling company that employs mushroom pickers. Your task is to maximize the number of mushrooms you get at the end of the day. To that end, you are given a list of mushroom pickers that you need to call to let them know whether or not they must work. For each of them, you know when she begins working and when she stops. At the end of her working time, a mushroom picker always gets 42 mushrooms in her bag, **independently** of the duration of the number of working hours. To get these 42 mushrooms, the picker needs the help of a mushroom dog, and unfortunately, you only have one such single dog. Said differently, only one mushroom picker may work at a time.

More formally, we note the list mushroom pickers $MPs = mp_1, mp_2, ..., mp_n$. Each mushroom picker mp_i has a start working time s_i and a finish working time f_i. We assume that this list is **always sorted in ascending order** of their finish working time so that $f_1 \leqslant f_2 \leqslant ... \leqslant f_n$. Because of the single dog constraint, two mushroom pickers mp_i and mp_j are *compatible* only if they never work at the same time, that is, if $f_i \leqslant s_j$.

An example of a list of mushroom pickers is given below:

Mushroom picker id	mp_1	mp_2	mp_3	mp_4	mp_5	mp_6	mp_7	mp_8	mp_9	mp_{10}	mp_{11}
Start working time (s_i)	6am	8am	5am	10am	8am	9am	11am	1pm	1pm	7am	4pm
Finish working time (f_i)	9am	10am	11am	12am	2pm	2pm	3pm	4am	5pm	7pm	8pm

I.1 Understanding the Problem

1. For the example above, give a list of 3 compatible mushroom pickers. [½ pt]

2. For the example above, give a list of compatible mushroom pickers that maximize the total number of mushrooms you have by the end of the day. [½ pt]

I.2 The Recursive Definition of the Problem

Given a list of mushroom pickers, we now focus on the problem of **computing the maximum number** of mushrooms we can get by the end of the day. To that end, lets define $MPs_{i,j}$ as the list of mushroom pickers starting their work after (or at the same time) picker mp_i finished to work and finishing their work before (or at the same time) picker mp_j starts working. Lets also note $n(i, j)$ the optimal, that is the maximum number of mushrooms we can get from $MPs_{i,j}$. Because this problem has optimal substructure, the recursive definition below allows computing $n(i, j)$:

$$n(i,j) = \begin{cases} 0, & \text{if } MPs_{ij} \text{ is empty.} \\ \max\limits_{mp_k \in MPs_{ij}} \{n(i,k) + n(k,j) + 42\}, & \text{otherwise.} \end{cases} \quad (1)$$

3. For the example above what is the set $MPs_{4,11}$? [½ pt]

4. For the example above what is the set $MPs_{3,4}$? [½ pt]

5. Draw the recursion tree for computing $n(2, 11)$ using the recursive definition (1) for the example above until you find an interesting property for an implementation. What is this property? [1½ pts]

I.3 A Dynamic Programming Solution

6. The above property and the optimal substructure of the problem suggests the use of dynamic programming. What data structure would you use to perform memoization? Explain what it contains, how and when it is filled and how and when it is accessed to get a value. [1½ pts]

7. What is the complexity of this dynamic programming solution? Justify. [1 pt]

I.4 A Better Solution

8. It is possible to have a better algorithm than the dynamic programming solution. How? [1 pt]

9. Give the pseudo code for your solution. [3 pts]

10. What is the complexity of your improved solution? [1 pt]

II The Most Beautiful Road [12 points]

You have just bought a magnificent castle in the middle of nowhere. Sadly, there is no road to access it, so you decide to build one yourself, and make it as beautiful as possible. Since you have a poor understanding of beauty, and you don't have much money remaining, you decide to buy second-hand road segments on the internet (yes, such things exist...).

You have found a list of many road segments, and you need to choose which ones you will use for your road. The inputs for your problem are the following:

- L is the length of your road;
- n is the number of road segments you found;
- l_i for $1 \leqslant i \leqslant n$ is the length of road segment i;
- v_i for $1 \leqslant i \leqslant n$ is the (objective!) aesthetic value of road segment i.

Note that you can cut a road segment, but then its value becomes 0 because it breaks the harmony of that segment. You can suppose it is always possible to complete your road (i.e., $\sum l_i \geqslant L$) and that no segment is longer than your road (i.e., $l_i \leqslant L$). Finally, there is only one copy of each segment (but two segments $i \neq j$ might have the same value and length).

11. Propose a naive algorithm that provides a complete road, but which is not an α-approximation for any α constant. You must prove it. [2 pts]

12. You order the road segments by decreasing ration v_i/l_i. So road segment 1 has the highest ratio. Propose a greedy algorithm based on this ordering. What can you say about the algorithm in terms of optimality? [3 pts]

13. Propose a solution based on dynamic programming that finds the optimal solution for your problem. Recommendations: [5 pts]

- Give the recursion equation;
- State clearly what is a subproblem;
- Give an example;
- Explain how you define and use your cache;
- (Hint) each road segment can either be taken or not.

14. **Difficult.** This problem is NP-complete, however, the dynamic programming solution seems to be polynomial. Make a complexity analysis of your algorithm that shows that, actually, it is exponential. [2 pts]