A. Habrard

Master CPS2/DSC/MLDM

Advanced Algorithms

Examination – 19th December 2018

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	6 pts	12.5 pts	1.5 pts

Part 1 Complexity (6 points)

1. Order increasingly the following functions, according to their growth rate $O(\cdot)$:
$(\sqrt{2})^n$, $\sqrt{n^3}$, $\cos(n)$, $n\log(n)$, $n+\sqrt{n}$.

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) $g(n) \in O(f_1(n))$ and $h(n) \in O(f_2(n))$ then $g(n)-h(n) \in O(f_1(n)-f_2(n))$.

3. Consider the following two functions f_1 and f_2:

$$f_1(n) = \begin{cases} n, \text{if } n \text{ is odd} \\ \sqrt{n}, \text{if } n \text{ is even and } n \leq 256 \\ n^3, \text{if } n \text{ is even and } n > 256 \end{cases}$$

$$f_2(n) = \begin{cases} n, \text{if } n \leq 250 \\ n^2, \text{if } 250 < n \leq 255 \\ n^3, \text{if } n > 255. \end{cases}$$

Give the true assertion(s) among the following ones:

   (a) f_1 is $O(f_2)$,

   (b) f_2 is $O(f_1)$,

   (c) f_1 and f_2 are incomparable.

4. Solve the following recurrences:

   - $T(n) = 3T(n/2) + 2n^2$, for $n \geq 2$ and $T(1) = 1$
   - $T(n) = T(\sqrt{n}) + n$, for $n \geq 4$ and $T(2) = 1$.

Part 2 Traveling in the desert (12.5 points)

A traveler in a desert wants to go from one oasis to another until a final destination without taking the risk of dying from thirst (*i.e.* lack o f water). He knows the position of all the oases from his starting point to his final destination. The traveler needs to drink one liter of water for each kilometer made. When he leaves his starting point, he has a bottle full of water. When he reaches an oasis, he has two possibilities: (i) continue his trip or (ii) stop and refill his bottle of water to the max. In this latter case, the traveler first empties completely the bottle and then refills it to always have a bottle of fresh water. Additionally, the traveler empties completely his bottle at his final destination. The bottle has a capacity of C liters, we assume that the capacity is enough to reach two consecutive oases.

1. The traveler wants to plan a trop with the minimum number of stops. Propose a greedy approach and prove that it is optimal. Give the complexity.

2. Now, the traveler wants to empty the minimum number of liters of water. Show that the previous greedy approach is still optimal for this problem.

3. Now, we assume that at each oasis, the traveler has to pay (say in euros) to a watchmen the quanitty of water lost in each oasis. More precisely, he has to pay an amount equivalent to the squared of the number of liters of water emptied from the bottle when he stops and refills his bottle. We also need to take into account the quantity of water emptied at the final destination. The problem now is then to find a solution in order to pay the minimum amount of money.

   (a) Propose a greedy approach. Is it optimal?

   (b) We propose to build a solution based on the dynamic programming paradigm. Your solution must consider the following quantities (note accessible points depend on the value of C):

   - $paid(i)$: minimum amount of money paid from the starting point to point i when the traveler empties his bottle at point i - **this the quantity we want to optimize.**
   - $d(i, j)$: number of kilometers between point i and point j.
   - C: the capacity of the bottle.

   Questions:

   i. Give the structure of an **optimal solution** (and if possible prove its optimality)
   ii. Give a recursive definition of the optimal value *paid*.
   iii. Provide an algorithm for finding the solution in polynomial time. You must provide a complete pseudo-code for all solution. Give the complexity (time and space).
   iv. Provide an approach for computing the solution (*i.e.* the trip).
   v. Apply your algorithm to the following example: the bottle has a capacity of 10 liters, and source points located at 8, 9, 16, 18, 24 and 27 kilometers from the starting point and such that the final point is at 32 kilometers from the starting point.

Part 3 Branch and Bound (1.5 points)

Recall the definition of an NP-hard problem, and then present the general principle of Branch and Bound methods used to solve these problems. Your description can be accompanied with an illustrative example.

① $T(n) = 3T\left(\frac{n}{2}\right) + 2n^2$

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

$T(2^K) = 3T(2^{K-1}) + 2(2^K)^2$

$= 3T(2^{K-1}) + 2^{2K+1}$

$= 3T(2^{K-1}) + 2 \cdot 2^{2K}$

$= 3^2 T(2^{K-2}) + 3 \cdot 2(2^{K-1})^2 + 2 \cdot 2^{2K}$

$= 3^2 T(2^{K-2}) + 3 \cdot 2^{2K-1} + 2^{2K+1}$

$= 3^3 T(2^{K-3}) + 3^2 2^{2(K-2)+1} + 3 \cdot 2^{2K-1} + 2^{2K+1}$

$= 3^3 T(2^{K-3}) + 3^2 2^{2K-3} + 3 \cdot 2^{2K-1} + 2^{2K+1}$

$\leq K \cdot 2^{2K+1}$

$\leq n^2 \log n \cdot 2$

$T(n) = O(n^2 \log n)$

② $T(n) = T(\sqrt{n}) + n$

$\therefore n = 2^K$

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

$\therefore F(K) = F\left(\frac{K}{2}\right) + 2^K$

$\Rightarrow F(K) = F\left(\frac{K}{2^2}\right) + 2^{\frac{K}{2}} + 2^K$

$= 2^K + 2^{\frac{K}{2}} + 2^{\frac{K}{2^2}} + \cdots + 2^{\frac{K}{2^{\log K}}}$

$\leq 2^K + 2^{K-1} + 2^{K-2} - 1 - \cdots + 1$

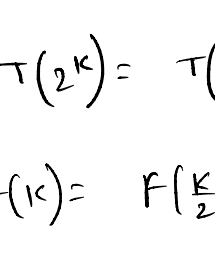
$\leq \frac{1(2^K - 1)}{2-1} = (2^K - 1) = (n-1)$

$F(K) = O(n)$

Part 2

Let = $\{a_1, a_2, \cdots, a_n\}$ be all the oasis.

Drink 1 liter of water per kilometer.



① For each step, we will check the distance of the next step

If it's > C, refill

else move to the next step **and**

$C = C - d(i, i+1)$

① $X_a = \{a_1, a_2, \ldots a_n\}$ be the greedy approach.

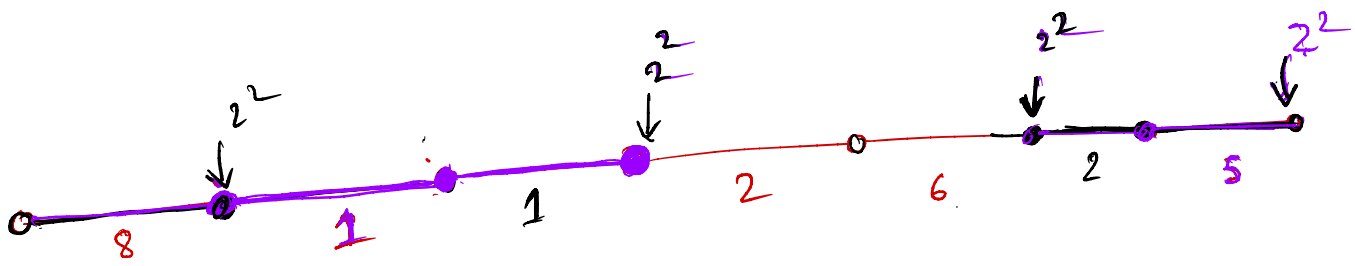
$X_o = \{b_1, b_2, \ldots b_m\}$ be the optimal approach

Let j = the max index X_a & X_o agree.

We will replace

③ So, the previous greedy algorithm won't work

16



Paid (i)

d(i,j)

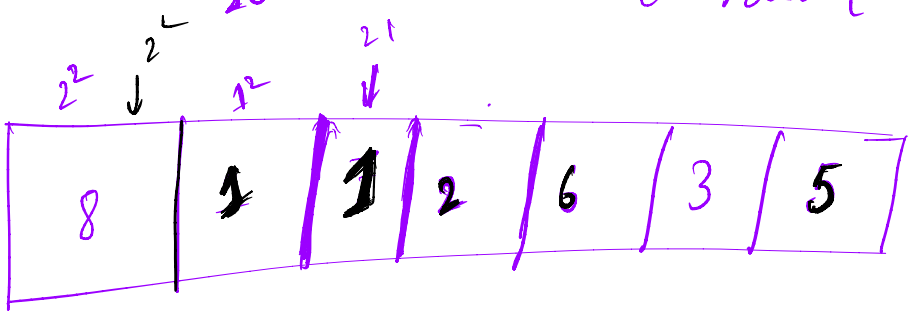
$9 + 16 + 9 = 20$

C

③ⓑ

ⓘⓘ Paid(i) = min { Paid(i-1) + $(c - d(i, i-1))^2$ → c' ← d[i,j-1]

Paid(i-2) + $(c - d(i, i-1) - c')^2$ → c' ← (d[i,j-1] + c') if d[i,i-1] ≤ 10)




```
def recursion_min_cost:

    if n == 1:
        c_hat = A[1]
        cost = sq(c - A[n])
        return c_hat, cost, n

    else:
        prev = recursion_min_cost(A, n-1, c)
        cost_1 = square(c - A[n])
        cost_2 = square(c - A[n] - prev[0])   if ≤ 10
        if cost_1 > cost_2:
            c_hat = A[n] + prev[0]
            return c_hat, cost_2

        else:
            c_hat = A[n]
            return c_hat, cost_1, n
```

Km	0	8	1	7	2	6	3	5
c								
1	0	∞	∞	∞	∞	∞	∞	∞
2	0	∞						
3								
4								
5								
6								
7								
8								
9								
10								

① Paid = [0]

```
def find_min:
    if n = 1:
        add lg(C-A[1]) in Paid
    else:
        for j in 2 to n:
            cur_cost = A[n]
            min = inf
            i=n-1
            while cur_cost ≤ C:
                cost = Paid[i-1] + (C-(cur_cost + A[i]))^2
                If cost ≤ min:
                    min = cost
        add min to Paid.
```