Lab04-Programming and Amortized Analysis

Algorithm and Complexity, Xiaofeng Gao, Spring 2022.

∗ If there is any problem, please contact TA Jialin Lyu.

∗ Name: Zhenran Xiao Student ID: 520030910281 Email: xiaozhenran@sjtu.edu.cn

1. Assume we have a set of arrays $A_0, A_1, A_2, \cdots$, where the i^{th} array A_i has a length of 3^i. Whenever an element is inserted into the arrays, we always intend to insert it into A_0. If A_0 is full, we should first pop the element in A_0 off and insert it into A_1, and then insert the new element in A_0. (Thus, if A_i is already full, we should recursively pop all its members off and insert them into A_{i+1} until we find an empty array to store the new element.) An illustrative example is shown in Figure 1. Inserting or popping an element takes $O(1)$ time.

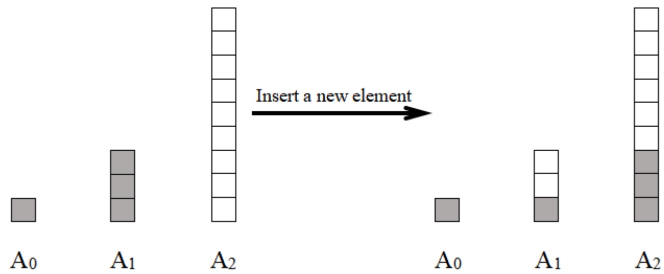

Figure 1: An example of making room for one new element in the set of arrays.

(a) In the worst case, how long does it take to add a new element into the set of arrays containing n elements?

(b) Prove that the amortized cost of adding an element is $O(\log n)$ by *Aggregation Analysis*.

(c) If each array A_i is required to be sorted but elements in different arrays have no relationship with each other, what is the amortized cost of adding an element if the comparison between two elements also takes $O(1)$ time?

Solution.

(a) The worst case is the old n elements fill up the first i arrays. $n = \sum_{k=0}^{i} 3^i = \frac{3^{i+1}-1}{2}$. If we want to add a new element into the set of arrays, for each k from 0 to i, we should pop all the elements in A_k off and insert them into A_{k+1}. Finally insert the new element into A_0.

$$\sharp pop + \sharp insert = n + (n+1) = 2n+1 \sim O(n)$$

So it takes $O(n)$ time.

(b) Assume that the cost of kth adding is C_k, the kth element has been operated for p_k times.

$$\sum_{k=1}^{n} C_k = \sharp pop + \sharp insert$$
$$= \sum_{k=1}^{n} p_k$$

When there are n elements in the set of arrays. The fisrt added one is in the A_i, $i = \log_3(2n+1) - 1$ or $\log_3(2n+1)$. So

$$p_1 = \sharp insert_1 + \sharp pop_1 = i + i - 1 \leq 2\log_3(2n+1)$$

Therefore,

$$\begin{aligned}\sum_{k=1}^{n} C_k &= \sharp pop + \sharp insert \\ &= \sum_{k=1}^{n} p_k \\ &\leq n \cdot 2\log_3(2n+1)\end{aligned}$$

The amortized cost of adding a new element is

$$T(n) = \frac{\sum_{k=1}^{n} C_k}{n} \leq 2\log_3(2n+1) \sim O(\log n)$$

(c) After adding, sorting each array takes $O(3^k \log 3^k)$ time. Sorting all the arrays takes $O(\sum_{k=0}^{i} \log 3 \cdot k3^k) \sim O(n \log n)$ time.
So the amortized cost is

$$T(n) \leq O(\log n) + O(n \log n) \sim O(n \log n)$$

□

2. *Machine Assignment.* A company intends to import 100 machines at the beginning of 2022, with a proper combination of the following 2 production modes:

- **High workload mode:** When the machine runs under high load, the annual profit of each machine is 100 thousand yuan, and the machine damage rate is 0.25 per year;
- **Low workload mode:** When the machine runs under low load, the annual profit of each machine is 80 thousand yuan, and the machine damage rate is 0.1 per year;

(a) Consider a 2-year short-term production plan, please design a scheme of assignment from 2022 to 2023 which maximizes the overall profit at the end of 2023, formulate a linear programming model and give its solving process and optimal solution with necessary explanations.

(b) Transform your LP model in (a) into its standard form and slack form.

(c) Transform your LP model in (a) into its dual form.

Solution.

(a) Suppose in 2022 x_1 machines work under high load, x_2 machines work under low load; in 2023 x_3 machines work under high load, x_4 machines work under low load.
According to the settings, we have

- At the beginning of 2022, there are 100 machines: $x_1 + x_2 \leq 100$
- At the beginning of 2023, there are $100 - 0.25x_1 - 0.1x_2$ machines: $x_3 + x_4 \leq 100 - 0.25x_1 - 0.1x_2$
- Nonnegative machine: $x_1, x_2, x_3, x_4 \geq 0$

Maximizeing the overall profit:

$$max \quad f(x_1, x_2, x_3, x_4) = 100x_1 + 80x_2 + 100x_3 + 80x_4$$

Linear programming model:

$$\begin{aligned} max \quad & f(x_1, x_2, x_3, x_4) = 100x_1 + 80x_2 + 100x_3 + 80x_4 \\ s.t. \quad & \qquad x_1 + x_2 \le 100 \\ & \qquad x_3 + x_4 \le 100 - 0.25x_1 - 0.1x_2 \\ & x_1, x_2, x_3, x_4 \ge 0 \end{aligned}$$

If we want maximuize the profit, we should make all machines busy. So we can let

$$\begin{aligned} x_1 + x_2 &= 100 \\ x_3 + x_4 &= 100 - 0.25x_1 - 0.1x_2 \end{aligned}$$

We put $x_2 = 100 - x_1$ and $x_4 = 100 - 0.25x_1 - 0.1(100 - x_1) - x_3$ into the origin LP model. Then we can obtain a new LP model:

$$\begin{aligned} max \quad & f(x_1, x_3) = 8x_1 + 20x_3 + 15200 \\ s.t. \quad & \qquad x_1 \le 100 \\ & 0.15x_1 + x_3 \le 90 \\ & \qquad x_1, x_3 \ge 0 \end{aligned}$$

We can establish Coordinate system $x_3 - x_1$. Constrains form the feasible region:

$$\left\{ \begin{array}{l} x_1 \ge 0 \\ x_1 \le 100 \\ x_3 \ge 0 \\ x_3 \le 90 - 0.15x_1 \end{array} \right\}$$

Let p represent the profit. We can use the line $x_3 = -0.4x_1 + 0.05p - 760$ to find the optimum point. It is $(100, 75)$.
Therefore the optimal scheme is that in 2022 100 machines work under high load, 0 machine works under low load; in 2023 75 machines work under high load, 0 machine works under low load. The overall profit is $100 \times 100 + 100 \times 75 = 17500$ thousand yuan.

(b) Transform to standard form:

$$\begin{aligned} max \quad & f(x_1, x_2, x_3, x_4) = 100x_1 + 80x_2 + 100x_3 + 80x_4 \\ s.t. \quad & \qquad x_1 + x_2 \le 100 \\ & x_3 + x_4 + 0.25x_1 + 0.1x_2 \le 100 \\ & \qquad x_1, x_2, x_3, x_4 \ge 0 \end{aligned}$$

Transform to slack form:

$$\begin{aligned} max \quad & f(x_1, x_2, x_3, x_4) = 100x_1 + 80x_2 + 100x_3 + 80x_4 \\ s.t. \quad & \qquad x_1 + x_2 + x_5 = 100 \\ & x_3 + x_4 + 0.25x_1 + 0.1x_2 + x_6 = 100 \\ & \qquad x_1, x_2, x_3, x_4, x_5, x_6 \ge 0 \end{aligned}$$

x_5, x_6 are slack variables.

(c) Transfrom to dual from:
Set the multiplier y_1 and y_2. $y_1, y_2 \geq 0$ to ensure no flipping from $\leq$ to $\geq$ in constraints. Multiply horizontally and add vertically, we obtain

$$(y_1 + 0.25y_2)x_1 + (y_1 + 0.1y_2)x_2 + y_2x_3 + y_2x_4 \leq 100y_1 + 100y_2$$

We want the left-hand side to look like out objective function $100x_1+80x_2+100x_3+80x_4$, so that the right-hand side becomes an upper bound on the objective function.

$$100x_1 + 80x_2 + 100x_3 + 80x_4 \leq 100y_1 + 100y_2 \quad if \begin{Bmatrix} y_1, y_2 \geq 0 \\ y_1 + 0.25y_2 \geq 100 \\ y_1 + 0.1y_2 \geq 80 \\ y_2 \geq 100 \\ y_2 \geq 80 \end{Bmatrix}$$

The dual form:

$$\begin{aligned} min \quad & g(y_1, y_2) = 100y_1 \\ s.t. \quad & y_1 + 0.25y_2 \geq 100 \\ & y_1 + 0.1y_2 \geq 80 \\ & y_2 \geq 100 \\ & y_2 \geq 80 \\ & y_1, y_2 \geq 0 \end{aligned}$$

□

3. *Collect Jingye Fu.* Collecting *Five Fortune Cards* was considered as a new tradition of most of Chinese people during Spring Festival. Little Gyro is planning to collect *Five Fortune Cards* on Alipay in 2022.

   As the product manager Hua Guan explains, aside from scanning the specific Chinese Character *Fu* to gain *Jingye Fu*, which is one of *Fortune Cards* usually hard to get, users also can using a *Sticky Card* to copy your friend¡¯s *Jingye Fu*. For each friend, users only have **one** chance to use the *Sticky Card*. After using a *Sticky Card*, you will obtain the *Fortune Card* which you stick with, and it's always the same type *Fortune Card* as your friend's already have. But whether you get *Jingye Fu* or not, this card will disappear. To enhance the possibility of getting *Jingye Fu*, Hua Guan suggests that you should find your friend who has many *Fortune Cards*, because the system will accumulate your *Fortune Value*, which is calculated by adding the total number of your friends *Fortune Cards* who you stick with. So Hua Guan considers that, the more *Fortune Cards* your friends have, the more *Fortune Value* you will accumulated, and the more possibility you will get *Jingye Fu*.

   After known these regulations, Little Gyro thinks that he wants to get **one** *Jingye Fu* **not under than** the possibility P, as well as get more *Fortune Value* as possible. So Little Gyro makes a list and collects his friends¡¯ information about the amount of *Fortune Card* and *Jingye Fu* they already have. But the amount of *Sticky Card* was limited, Little Gyro wants to know which friend he should stick with. Can you help him?

   It¡¯s guaranteed that Little Gyro will definitely get a *Fortune Card* when using a *Sticky Card*.

   Input Specification:

   There are multiple test cases. The first line of the input is an integer $T(1 \leq T \leq 10)$, indicating the number of test cases. Then T test cases follow.

Figure 2: The picture of *Jingye Fu*.

The first line of each test case contains two integers $n, m (1 \leq n \leq 1000, 1 \leq m \leq 10)$ and one floating point number $P (0 \leq P \leq 1)$, indicating the number of friends, the number of *Sticky Card* and the least possibility Little Gyro will get one *Jingye Fu*, respectively.

The following n lines describe the information of Little Gyro¡¯s friends, numbered from 1 to n. The $(i+1)$-th line contains two positive integers t_i and $h_i (1 \leq h_i \leq t_i \leq 100)$, representing the total number of *Fortune Card* and the number of *Jingye Fu* the i-th friend has, respectively.

Output Specification:

For each test case output two lines, the first line consists one integer, indicating the maximum *Fortune Value*. The second line consists at most m numbers, indicating the number of Little Gyro¡¯s friends (within the ascending order) who stuck with.

If there exists more than one proper solutions, output the solution with the maximum *Fortune Value*. And if there is no solution, output ”No Solution” (without quotes) in one line instead.

It¡¯s guaranteed that the proper solutions with the maximum *Fortune Value* is unique.

Sample Input:	**Sample Output:**
2	15
3 2 0.75	2 3
5 2	No Solution
10 4	
5 3	
3 2 0.50	
20 3	
10 2	
5 1	

Remark: The input data `Lab04-JingyeFu.in` (sample test cases only) and the template code `Lab04-JingyeFu.cpp` are attached on the course webpage. Please include your `Lab04-JingyeFu.cpp` file in your uploaded .rar or .zip file.

Hint:

In the first sample, Little Gyro can choose 1 and 3 or 2 and 3 to stick with in order to achieve the possibility 0.75. So the maximum *Fortune Value* is 15.

In the second sample, Little Gyro can not find any ways to achieve the possibility 0.50.

(a) Please briefly describe your algorithm and analyze its time complexity and space complexity. Is the greedy algorithm solvable? If it can be solved by greedy algorithm, please explain the reason. If not, please give a counterexample.

(b) Try to write a C/C++ code to solve this problem, you only need to complete the `TODO` part in `Lab04-JingyeFu.cpp`. Your program will be judged by the online judge system, including several test cases, half for the test data which equivalent to the sample test case, and another half for other corner and huge test cases.

Solution.

(a) My alogrithm: For all the ways of picking m friends out of n friends, calculate the Fortune Value and the possiblility P. Use the bigger Fortune Value with eligible P to update the solution set. Finally we obtain the optimal solution.
Use DFS to obtain all the ways of picking m friends out of n, namely using recursion to realize combinatorial enumeration.
Time complexity: $O(m\binom{n}{m})$.
Space complexity: $O(n+m)$

About greedy algorithm:

- If the greedy algorithm is: Find the combination with the biggest Fortune Value, check whether its P is eligible. If not, find the combination with the second biggest Fortune Value. Until it find a combination with an eligible P. This combination is the optimal solution.
Then the greedy algorithm is solvable. But it is not easy to find the combination with the k biggest Fortune Value.
- If the greedy algorithm is just choosing m friends with m biggest Fortune Values, then it is not solvable. The second sample in the sample input can be a counterexample.

(b) Please check it in `Lab04-JingyeFu.cpp`.

$\square$

Remark: You need to include your .pdf and .tex files in your uploaded .rar or .zip file.