CS29003 Algorithms Laboratory

Assignment No: 3

Last date of submission: 30–January–2018

End-semester tests are approaching. An international student Mr. Pigro Faulenzer from the Department of Foobarnautic Engineering discovers that he has n pending foomatics assignments. He has only M minutes (a positive integer) to solve the assignments. He works out that the assignments will take $m_1, m_2, \ldots, m_n$ minutes of effort (each m_i is a positive integer). Also, suppose that the points (maximum marks) for these assignments are $p_1, p_2, \ldots, p_n$ (positive integers again). There is no part marking, that is, Pigro has to solve an assignment totally, or leave it.

Pigro's primary goal is to solve those assignments such that the total point of the solved assignments is maximized. He wants to preserve energy for the end-semester tests. So his secondary goal is to attempt the maximum achievable total point with the minimum possible effort. Help Pigro achieve his objectives.

Part 1: In this part, assume that all points are equal, that is, $p_1 = p_2 = \cdots = p_n$. Therefore the primary goal of Pigro reduces to solving as many assignments as possible. The obvious greedy strategy of solving the easiest assignments first works in this case, and addresses his secondary goal too.

Write an $O(n \log n)$-time sorting algorithm (do not use any built-in library function) to sort the array $m_1, m_2, \ldots, m_n$ in the ascending order. Keep on solving assignments from the easy end until M minutes are over. Print this optimal solution. Write a function *solveGRD*() for solving this part.

Part 2: Now, assume that the points array $p_1, p_2, \ldots, p_n$ contains arbitrary integer values. There need not be any correlation between the efforts and the points. Pigro's professor may have rated the assignments using metrics other than raw effort (like novelty, trickiness, depth and breadth of knowledge of the students, and so on). Greedy strategies do not work now. Dynamic programming helps you instead.

Let the total point in all the n assignments be

$$P = p_1 + p_2 + \cdots + p_n.$$

Build a two-dimensional table T of size $(n+1) \times (P+1)$ such that $T[i][p]$ is intended to store the minimum possible effort to attempt ***exactly*** p points from the assignments $1, 2, \ldots, i$. If exactly p points cannot be achieved from $p_1, p_2, \ldots, p_i$, you should store $T[i][p] = \infty$.

First, notice that $T[i][0] = 0$ for all i (irrespective of the number of assignments, if Pigro does not solve any assignment, his attempted point is 0). Another boundary condition pertains to $i = 0$, which means that no assignment is considered. Therefore, $T[0][0] = 0$, whereas $T[0][p] = \infty$ whenever $p > 0$.

Now, take $1 \leqslant i \leqslant n$ and $1 \leqslant p \leqslant P$. If $p < p_i$, Pigro cannot attempt the i-th assignment, so $T[i][p] = T[i-1][p]$. If $p \geqslant p_i$, Pigro has two possibilities: if he does not solve the i-th assignment, then take $M_0 = T[i-1][p]$, whereas if he solves the i-th assignment, then take $M_1 = m_i + T[i-1][p-p_i]$. If $M_1 > M$, the second option is not valid, so set $T[i][p] = M_0$, otherwise set $T[i][p] = \min(M_0, M_1)$.

Write a function *solveDP*() to build the table T using the approach just mentioned. The maximum achievable point is the largest p such that $T[n][p] \neq \infty$. To achieve this p, the minimum effort needed is $T[n][p]$.

Part 2 (Extended): Augment the function *solveDP*() of Part 2 to compute a choice of the assignments achieving the optimal solution. Print the solution (both effort-wise and point-wise breakups). Do not write a new function. Just update the function *solveDP*().

The *main*() function

- Read n (the number of assignments) and M (the total time Pigro has) from the user.
- Read only the efforts $m_1, m_2, \ldots, m_n$ from the user. Assume that $p_1 = p_2 = \cdots = p_n$ (the exact value need not be specified). Call *solveGRD*() to find and print the greedy solution of Part 1.

- In Part 1, you have already sorted the effort array. In Part 2, you should work with an arbitrary array. So re-read a new set of efforts $m_1, m_2, \ldots, m_n$ from the user. Also read the points $p_1, p_2, \ldots, p_n$ from the user. Call *solveDP*() to compute the optimal solution in this case. Print the solution (maximum point and minimum effort). After you augment the function as specified in Part 2 (Extended), the point-wise and effort-wise breakups for the optimal solution are also printed.

Sample output

```
n = 10
M = 100

+++ Efforts :  23  23  28  28  13   5  21  25  31   5

+++ Part 1 (Greedy)
    Minimum effort =    90 =  5 +  5 + 13 + 21 + 23 + 23

+++ Points  :   8   2   7   4   5   3   8   2   2  10
+++ Efforts :  15   9  22  25  25  30  26   4  29  39

+++ Part 2 (DP)
    Maximum points =    30 =  8 +  7 +  5 +  8 +  2
    Minimum effort =    92 = 15 + 22 + 25 + 26 +  4
```

Submit a single C/C++ source file. Do not use global/static variables.