ADA - Análisis y Diseño de Algoritmos, 2019-1
Tarea 2: Semanas 3 y 4

Para entregar el viernes 15 de febrero/domingo 17 de febrero de 2019

Problemas conceptuales a las 16:00 (15 de febrero) en el Departamento de Electrónica y Ciencias de la Computación

Problemas prácticos a las 23:59 (17 de febrero) en la arena de programación

---

Tanto los ejercicios como los problemas deben ser resueltos, pero únicamente las soluciones de los problemas deben ser entregadas. La intención de los ejercicios es entrenarlo para que domine el material del curso; a pesar de que no debe entregar soluciones a los ejercicios, usted es responsable del material cubierto en ellos.

Instrucciones para la entrega

Para esta tarea y todas las tareas futuras, la entrega de soluciones es *individual*. Por favor escriba claramente su nombre, código de estudiante y sección en cada hoja impresa entregada o en cada archivo de código (a modo de comentario). Adicionalmente, agregue la información de fecha y nombres de compañeros con los que colaboró; igualmente cite cualquier fuente de información que utilizó.

¿Cómo describir un algoritmo?

En algunos ejercicios y problemas se pide "dar un algoritmo" para resolver un problema. Una solución debe tomar la forma de un pequeño ensayo (es decir, un par de párrafos). En particular, una solución debe resumir en un párrafo el problema y cuáles son los resultados de la solución. Además, se deben incluir párrafos con la siguiente información:

- una descripción del algoritmo en castellano y, si es útil, pseudo-código;
- por lo menos un diagrama o ejemplo que muestre cómo funciona el algoritmo;
- una demostración de la corrección del algoritmo; y
- un análisis de la complejidad temporal del algoritmo.

Recuerde que su objetivo es comunicar claramente un algoritmo. Las soluciones algorítmicas correctas y descritas *claramente* recibirán alta calificación; soluciones complejas, obtusas o mal presentadas recibirán baja calificación.

Ejercicios

4.1-1, 4-1-2, 4.1-3 (página 74), **4-2** (página 107), 15.1-2, 15.1-3, 15.1-4, 15.1-5 (página 370).

Problemas conceptuales

1. Problema 15-4: *Printing Neatly* (Cormen et. al. página 405).
2. Problema 15-9: *Breaking a String* (Cormen et. al. página 410).
3. Ejercicio 9: *High Performance Computing* (Kleinberg & Tardos página 320).

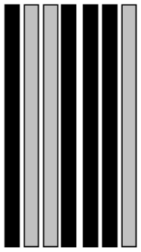
Problemas prácticos

Hay cinco problemas prácticos cuyos enunciados aparecen a partir de la siguiente página.

A - Barcodes

Source file name: `barcodes.py`
Time limit: x seconds

A bar-code symbol consists of alternating dark and light bars, starting with a dark bar on the left. Each bar is a number of units wide. The figure below shows a bar-code symbol consisting of 4 bars that extend over 1 + 2 + 3 + 1 = 7 units.



In general, the bar code BC(n, k, m) is the set of all symbols with k bars that together extend over exactly n units, each bar being at most m units wide. For instance, the symbol in the figure above belongs to BC(7, 4, 3) but not to BC(7, 4, 2). The table below shows all 16 symbols in BC(7, 4, 3). Each '1' represents a dark unit, each '0' a light unit.

0:	1000100	4:	1001110	8:	1100100	12:	1101110
1:	1000110	5:	1011000	9:	1100110	13:	1110010
2:	1001000	6:	1011100	10:	1101000	14:	1110100
3:	1001100	7:	1100010	11:	1101100	15:	1110110

Input

Each input will contain three positive integers n, k, and m ($1 \leq n, k, m \leq 50$).

The input must be read from standard input.

Output

For each input print the total number of symbols in BC(n, k, m).

The output must be written to standard output.

Sample Input	**Sample Output**
`7 4 3` `7 4 2`	`16` `4`

B - Boxes

Source file name: `boxes.py`
Time limit: x seconds

We have some boxes numbered 1 to N. The dimensions of all boxes are identical. Now we have to stack up some of the boxes, subject to the following constraints:

1. One cannot put more than one box directly upon a box;
2. Boxes with lower serial numbers are not to be put upon one with a higher number;
3. The weight and maximum load for each box are given. The total weight of all boxes upon a box should not exceed its maximum load.

Please write a program that finds the maximum number of boxes that can be stacked up according to the above constraints.

Input

The first line of each set of input is an integer N ($1 \leq N \leq 1\,000$). This is followed by N lines, each with two integers, both at most 3 000, representing the weight and maximum load of each box respectively.

Input ends with a case where $N = 0$.

The input must be read from standard input.

Output

Each line of your output should give the maximum number of boxes that can be stacked up.

The output must be written to standard output.

Sample Input	**Sample Output**
5 19 15 7 13 5 7 6 8 1 2 0	4

C - Making Change

Source file name: `change.py`
Time limit: x seconds

Given an amount of money and unlimited (almost) numbers of coins (we will ignore notes for this problem) we know that an amount of money may be made up in a variety of ways. A more interesting problem arises when goods are bought and need to be paid for, with the possibility that change may need to be given. Given the finite resources of most wallets nowadays, we are constrained in the number of ways in which we can make up an amount to pay for our purchases —assuming that we can make up the amount in the first place, but that is another story.

The problem we will be concerned with will be to minimise the number of coins that change hands at such a transaction, given that the shopkeeper has an adequate supply of all coins (the set of New Zealand coins comprises 5c, 10c, 20c, 50c, $1 and $2). Thus, if we need to pay 55c and we do not hold a 50c coin, we could pay this as $2 * 20c+10c+5c$ to make a total of 4 coins. If we tender $1 we will receive 45c in change which also involves 4 coins, but if we tender $1.05 ($1 + 5c), we get 50c change and the total number of coins that changes hands is only 3.

Write a program that will read in the resources available to you and the amount of the purchase and will determine the minimum number of coins that change hands.

Input

Input will consist of a series of lines, each line defining a different situation. Each line will consist of 6 integers representing the numbers of coins available to you in the order given above, followed by a real number representing the value of the transaction, which will always be less than $5.00. The file will be terminated by six zeroes (`0 0 0 0 0 0`). The total value of the coins will always be sufficient to make up the amount and the amount will always be achievable, that is it will always be a multiple of 5c.

The input must be read from standard input.

Output

Output will consist of a series of lines, one for each situation defined in the input. Each line will consist of the minimum number of coins that change hands right justified in a field 3 characters wide.

The output must be written to standard output.

Sample Input	**Sample Output**

`2 4 2 2 1 0 0.95`
`2 4 2 0 1 0 0.55`
`0 0 0 0 0 0`
	`  2`
	`  3`

D - Injured Queen Problem

Source file name: `queen.py`
Time limit: x seconds

Chess is a two-player board game believed to have been played in India as early as the sixth century. However, in this problem we will not discuss about chess, rather we will talk about a modified form of the classic n- queens problem.

I know you are familiar with plotting n-queens on a chess board with the help of a classic backtracking algorithm. If you write that algorithm now you will find that there are 92 ways of plotting 8 queens in an 8×8 board provided no queens attack each other.

In this problem we will talk about injured queens who can move only like a king in horizontal and diagonal direction from current position but can reach any row from current position like a normal chess queen. You will have to find the number of possible arrangements with such injured queens in a particular $(n \times n)$ board (with some additional constraints), such that no two queens attack each other.

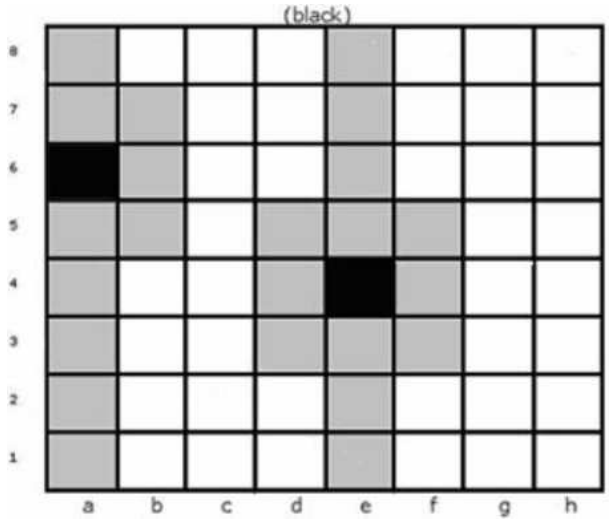


Fig: Injured Queen at $a6$ can reach the adjacent grey squares. Queen at $e4$ can reach adjacent grey squares. The injured queen positions are black and the reachable places are grey.

Input

Input file contains several lines of input. Each line expresses a certain board status. The length of these status string is the board dimension n $(0 < n \leq 15)$. The first character of the string denotes the status of first column, the second character of the string denotes the status of the second column, and so on.

So, if the first character of the status string is 2, it means that we are looking for arrangements (no two injured queen attack each other) which has injured queen in column a, row 2. The possible numbers for rows are $1, 2, 3, \ldots, D, E, F$ which indicates row $1, 2, 3 \ldots 13, 14, 15$. If any column contains '?' it means that in that column the injured queen can be in any row. So a status string '`1?4??3`' means that you are asked to find out total number of possible arrangements in a (6×6) chessboard which has three of its six injured queens at $a1$, $c4$ and $f3$. Also note that there will be no invalid inputs. For example, '`1?51`' is an invalid input because a (4×4) chessboard does not have a fifth row.

The input must be read from standard input.

Output

For each line of input produce one line of output. This line should contain an integer which indicates the total number of possible arrangements of the corresponding input status string.

The output must be written to standard output.

Sample Input	**Sample Output**

`??????`
`??????????????`
`???8?????`
`43?????`
	`2642`
	`22696209911206174`
	`2098208`
	`0`

E - Shopping Trip

Source file name: `trip.py`
Time limit: x seconds

For some reason, Daniel loves to collect and watch operas on DVD. He can find and order all the operas he wants from Amazon, and they will even deliver them right to his door, but he can usually find a better price at one of his favourite stores. However, with the cost of gas nowadays, it is hard to tell whether or not one would actually save money by driving to the stores to purchase the DVDs.

Daniel would like to buy some operas today. For each of the operas he wants, he knows exactly one store that is selling it for a lower cost than the Amazon price. He would like to know if it would actually be worth it to go out and buy the operas from the stores.

Daniel only knows the road system connecting his favourite stores, and will only use those roads to get around. He knows at least one route, if only an indirect one, to every store.

An example diagram of Daniel's house, his favourite stores, and the roads connecting them.

In his shopping trip, Daniel begins at his house, drives from store to store in any order to purchase his operas, then drives back to his house. For any particular opera, he can opt not to drive to the store to buy it, since he can just order it from Amazon.

For convenience, Daniel assigned his house the integer 0, and numbered each of his favourite stores with integers starting at 1. You are given a description of the road system and the exact amount it would cost for Daniel to drive each road. For each opera Daniel wants, you are given the number of the store it is available at, and the amount he would save if he bought that particular opera at that store.

Your task is to determine the greatest amount of money Daniel can save by making the shopping trip

Input

The first line of input contains a single number indicating the number of scenarios to process. A blank line precedes each scenario.

Each scenario begins with line containing two numbers: N ($1 \leq N \leq 50$), the number of stores, and M ($1 \leq M \leq 1000$), the number of roads. The following M lines each contain a description of a road. Each road is described by two integers indicating the house or stores it connects, and a real number with two decimal digits indicating the cost in dollars to drive that road. All roads are two-way.

The next line in the scenario contains a number P ($1 \leq P \leq 12$), the number of opera DVDs Daniel wants to buy. For each of the P operas, a line follows containing an integer indicating the store number at which the opera is available, and a real number with two decimal digits indicating the difference between the Amazon price and the price at that store in dollars.

The input must be read from standard input.

Output

For each scenario in the input, write one line of output indicating the largest amount of money, in dollars and cents, that Daniel can save by making his shopping trip. Follow the format of the sample output; there should always be two digits after the decimal point to indicate the number of cents. If Daniel cannot save any money by going to the stores, output a single line saying '`Don't leave the house`'.

The output must be written to standard output.

Sample Input	**Sample Output**
2 4 5 0 1 1.00 1 2 3.00 1 3 2.00 2 4 4.00 3 4 3.25 3 2 1.50 3 7.00 4 9.00 1 1 0 1 1.50 1 1 2.99	Daniel can save $3.50 Don't leave the house