Estructuras de Datos, 2023-2

Para entregar el domingo 3 de septiembre de 2023
A las 23:59 en la arena de programación

Instrucciones para la entrega

- Para esta tarea y todas las tareas futuras en la arena de programación, la entrega de soluciones es *individual*. Por favor escriba claramente su nombre, código de estudiante y sección en cada archivo de código (a modo de comentario). Adicionalmente, cite cualquier fuente de información que utilizó. Los códigos fuente que suba a la arena de programación deben der de su completa autoría.
- En cada problema debe leer los datos de entrada de la forma en la que se indica en el enunciado y debe imprimir los resultados con el formato allí indicado. No debe agregar mensajes ni agregar o eliminar datos en el proceso de lectura. La omisión de esta indicación puede generar que su programa no sea aceptado en la arena de programación.
- Aunque la arena de programación soporta envíos en otros lenguajes de progamación como Python, **todos los problemas de esta tarea deben ser resueltos en C**. Los envíos en Python no serán admitidos.
- Debe enviar sus soluciones a través de la arena. Antes de subir sus soluciones asegurese de realizar pruebas con los casos de pruebas proporcionados para verificar que el programa finalice y no se quede en un ciclo infinito.
- El primer criterío de evaluación será la aceptación del problema en la arena cumpliendo los requisitos indicados en los enunciados de los ejercicios y en este documento. Además, se tendrán en cuenta aspectos de estilo como no usar `break` ni `continue` y que las funciones deben tener únicamente una instrucción `return` que debe estar en la última línea.

Problemas prácticos

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

A - Problem A

Source file name: `bin.c`
Time limit: 1 second

Bin packing, or the placement of objects of certain weights into different bins subject to certain constraints, is an historically interesting problem. Some bin packing problems are NP-complete but are amenable to dynamic programming solutions or to approximately optimal heuristic solutions.

In this problem you will be solving a bin packing problem that deals with recycling glass.

Recycling glass requires that the glass be separated by color into one of three categories: brown glass, green glass, and clear glass. In this problem you will be given three recycling bins, each containing a specified number of brown, green and clear bottles. In order to be recycled, the bottles will need to be moved so that each bin contains bottles of only one color.

The problem is to minimize the number of bottles that are moved. You may assume that the only problem is to minimize the number of movements between boxes.

For the purposes of this problem, each bin has infinite capacity and the only constraint is moving the bottles so that each bin contains bottles of a single color. The total number of bottles will never exceed 2^{31}.

Input

The input consists of a series of lines with each line containing 9 integers. The first three integers on a line represent the number of brown, green, and clear bottles (respectively) in bin number 1, the second three represent the number of brown, green and clear bottles (respectively) in bin number 2, and the last three integers represent the number of brown, green, and clear bottles (respectively) in bin number 3. For example, the line

10 15 20 30 12 8 15 8 31

indicates that there are 20 clear bottles in bin 1, 12 green bottles in bin 2, and 15 brown bottles in bin 3.

Integers on a line will be separated by one or more spaces. Your program should process all lines in the input file.

The input must be read from standard input.

Output

For each line of input there will be one line of output indicating what color bottles go in what bin to minimize the number of bottle movements. You should also print the minimum number of bottle movements.

The output should consist of a string of the three upper case characters `'G'`, `'B'`, `'C'` (representing the colors green, brown, and clear) representing the color associated with each bin.

The first character of the string represents the color associated with the first bin, the second character of the string represents the color associated with the second bin, and the third character represents the color associated with the third bin.

The integer indicating the minimum number of bottle movements should follow the string.

If more than one order of brown, green, and clear bins yields the minimum number of movements then the alphabetically first string representing a minimal configuration should be printed.

The output must be written to standard output.

Sample Input	**Sample Output**
`1 2 3 4 5 6 7 8 9` `5 10 5 20 10 5 10 20 10`	`BCG 30` `CBG 50`

B - Problem B

Source file name: `car.c`
Time limit: 1 second

You are in a car and going at the speed of u m/s. Your acceleration a is constant. After a particular time *t*, your speed is *v* m/s and your displacement is *s*. Now you are given some (not all of them) values for the given variables. And you have to find the missing parameters.

Input

The input file may contain multiple test cases. Each test case can be one of the

1 *u v t*

2 *u v a*

3 *u a s*

4 *v a s*

Input will be terminated by a single `'0'`.

The input must be read from standard input.

Output

For each case of input you have to print one line containing the case number and

If 1 *u v t* are given then print *s* and *a*

If 2 *u v a* are given then print *s* and *t*

If 3 *u a s* are given then print *v* and *t*

If 4 *v a s* are given then print *u* and *t*

Check the samples for more details. You can assume that the given cases will not evaluate to an invalid situation. Use double for all calculations and output all floating point numbers to three decimal places.

The output must be written to standard output.

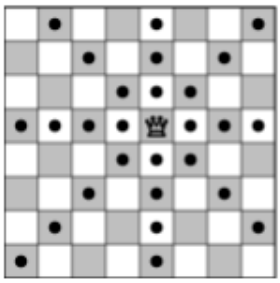
Sample Input	**Sample Output**

`1 10 5 2.0`
`1 5 10.0 2`
`2 10 11 2`
`3 5 1 6`
`4 5.0 -1 6`
`0`
	`Case 1: 15.000 -2.500`
	`Case 2: 15.000 2.500`
	`Case 3: 5.250 0.500`
	`Case 4: 6.083 1.083`
	`Case 5: 6.083 1.083`

C - Problem C

Source file name: `queen.c`
Time limit: 1 second

The game of Chess has several pieces with curious movements. One of them is the Queen, which can move any number of squares in any direction: in the same line, in the same column or in any of the diagonals, as illustrated by the figure below (black dots represent positions the queen may reach in one move):



The great Chess Master Kary Gasparov invented a new type of chess problem: given the position of a queen in an empty standard chess board (that is, an 8×8 board) how many moves are needed so that she reaches another given square in the board?

Kary found the solution for some of those problems, but is having a difficult time to solve some others, and therefore he has asked that you write a program to solve this type of problem.

Input

The input contains several test cases. The only line of each test case contains four integers X_1, Y_1, X_2 and Y_2 ($1 \leq X_1, Y_1, X_2, Y_2 \leq 8$). The queen starts in the square with coordinates (X_1, Y_1), and must finish at the square with coordinates (X_2, Y_2). In the chessboard, columns are numbered from 1 to 8, from left ro right; lines are also numbered from 1 to 8, from top to bottom. The coordinates of a square in line X and column Y are (X, Y).

The end of input is indicated by a line containing four zeros, separated by spaces.

The input must be read from standard input.

Output

For each test case in the input your program must print a single line, containing an integer, indicating the smallest number of moves needed for the queen to reach the new position.

The output must be written to standard output.

Sample Input	**Sample Output**

`4 4 6 2`
`3 5 3 5`
`5 5 4 3`
`0 0 0 0`
	`1`
	`0`
	`2`

D - Problem D

Source file name: `reverse-add.c`
Time limit: 1 second

The "*reverse and add*" method is simple: choose a number, reverse its digits and add it to the original. If the sum is not a palindrome (which means, it is not the same number from left to right and right to left), repeat this procedure.

For example:

```
  195     Initial number
  591
 ----
  786
  687
 ----
 1473
 3741
 ----
 5214
 4125
 ----
 9339     Resulting palindrome
```

In this particular case the palindrome `'9339'` appeared after the 4th addition. This method leads to palindromes in a few step for almost all of the integers. But there are interesting exceptions. 196 is the first number for which no palindrome has been found. It is not proven though, that there is no such a palindrome.

You must write a program that give the resulting palindrome and the number of iterations (additions) to compute the palindrome.

You might assume that all tests data on this problem:

- will have an answer,
- will be computable with less than 1000 iterations (additions),
- will yield a palindrome that is not greater than 4,294,967,295.

Input

The first line will have a number N ($0 < N \leq 200$) with the number of test cases, the next N lines will have a number P to compute its palindrome.

The input must be read from standard input.

Output

For each of the N tests you will have to write a line with the following data : minimum number of iterations (additions) to get to the palindrome and the resulting palindrome itself separated by one space.

The output must be written to standard output.

Sample Input	**Sample Output**
3 195 265 750	4 9339 5 45254 3 6666

E - Problem E

Source file name: `square.c`
Time limit: 1 second

Given a rectangular grid of characters you have to find out the length of a side of the largest square such that all the characters of the square are same and the center [intersecting point of the two diagonals] of the square is at location (r, c). The height and width of the grid is M and N respectively. Upper left corner and lower right corner of the grid will be denoted by $(0, 0)$ and $(M - 1, N - 1)$ respectively.

Consider the grid of characters given below. Given the location (1, 2) the length of a side of the largest square is 3.

```
abbbaaaaaa
abbbaaaaaa
abbbaaaaaa
aaaaaaaaaa
aaaaaaaaaa
aaccaaaaaa
aaccaaaaaa
```

Input

The input starts with a line containing a single integer T ($T < 21$). This is followed by T test cases. The first line of each of them will contain three integers M, N and Q ($Q < 21$) separated by a space where M, N denotes the dimension of the grid. Next follows M lines each containing N characters. Finally, there will be Q lines each containing two integers r and c. The value of M and N will be at most 100.

The input must be read from standard input.

Output

For each test case in the input produce $Q + 1$ lines of output. In the first line print the value of M, N and Q in that order separated by single space. In the next Q lines, output the length of a side of the largest square in the corresponding grid for each (r, c) pair in the input.

The output must be written to standard output.

Sample Input	**Sample Output**
1 7 10 4 abbbaaaaaa abbbaaaaaa abbbaaaaaa aaaaaaaaaa aaaaaaaaaa aaccaaaaaa aaccaaaaaa 1 2 2 4 4 6 5 2	7 10 4 3 1 5 1