ADA - Análisis y Diseño de Algoritmos, 2019-1
Tarea 5: Semanas 11 y 12

Para entregar el viernes 12 de abril/domingo 14 de abril de 2019

Problemas conceptuales a las 16:00 (12 de abril) en el Departamento de Electrónica y Ciencias de la Computación

Problemas prácticos a las 23:59 (14 de abril) 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

No hay.

Problemas conceptuales

1. *Famous alphametic*[A.Levitin: Introduction to the Design and Analysis of Algorithms. 3rd Edition, 2012.]

   A puzzle in which the digits in a correct mathematical expression, such as a sum, are replaced by letters is called *cryptarithm*. If, in addition, the puzzle's words make sense, it is said to be an *alphametic*. The most well-known alphametic was published by the renowned British puzzlist Henry E. Dudeney (1857–1930):

   ```
     S E N D
   + M O R E
   ---------
   M O N E Y
   ```

   Two conditions are assumed: first, the correspondence between letters and decimal digits is one-to-one, i.e., each letter represents one digit only and different letters represent different digits. Second, the digit zero does not appear as the left-most digit in any of the numbers. To solve an alphametic means to find which digit each letter represents. Note that a solution's uniqueness cannot be assumed and has to be verified by the solver.

   (a) Specify the given problem and design an algorithm for solving cryptarithms by exhaustive search. Assume that a given cryptarithm is a sum of two words.

   (b) Solve Dudeney's puzzle the way it was expected to be solved when it was first published in 1924.

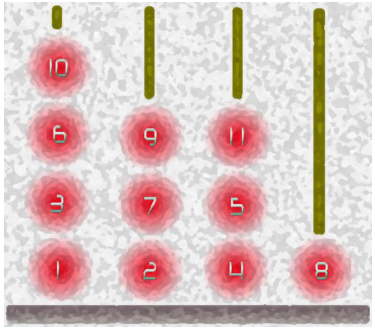
Problemas prácticos

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

A - Hanoi Tower Troubles Again!

Source file name: `hanoi.py`
Time limit: 1 second

People stopped moving discs from peg to peg after they know the number of steps needed to complete the entire task. But on the other hand, they did not stopped thinking about similar puzzles with the Hanoi Tower. Mr. S invented a little game on it. The game consists of N pegs and a LOT of balls. The balls are numbered $1, 2, 3 \ldots$. The balls look ordinary, but they are actually magic. If the sum of the numbers on two balls is NOT a square number, they will push each other with a great force when they're too closed, so they can NEVER be put together touching each other.



The player should place one ball on the top of a peg at a time. He should first try ball 1, then ball 2, then ball 3, $\ldots$. If he fails to do so, the game ends. Help the player to place as many balls as possible. You may take a look at the picture above, since it shows us a best result for 4 pegs.

Input

The first line of the input contains a single integer T, indicating the number of test cases ($1 \leq T \leq 50$). Each test case contains a single integer N ($1 \leq N \leq 50$), indicating the number of pegs available.

The input must be read from standard input.

Output

For each test case in the input print a line containing an integer indicating the maximal number of balls that can be placed. Print '`-1`' if an infinite number of balls can be placed.

The output must be written to standard output.

Sample Input	**Sample Output**
2 4 25	11 337

B - Knuth's Permutation

Source file name: `knuth.py`
Time limit: 2 seconds

There are some permutation generation techniques in Knuth's book "The Art of Computer Programming - Volume 1". One of the processes is as follows:

For each permutation $A_1A_2 \ldots A_{n-1}$ form n others by inserting a character n in all possible places obtaining

$$nA_1A_2 \ldots A_{n-1}, A_1nA_2 \ldots A_{n-1}, \ldots, A_1A_2 \ldots nA_{n-1}, A_1A_2 \ldots A_{n-1}n$$

For example, from the permutation 231 inserting 4 in all possible places we get 4231 2431 2341 2314.

Following this rule you have to generate all the permutation for a given set of characters. All the given characters will be different and there number will be less than 10 and they all will be alpha numerals. This process is recursive and you will have to start recursive call with the first character and keep inserting the other characters in order. The sample input and output will make this clear. Your output should exactly mach the sample output for the sample input.

Input

The input contains several lines of input. Each line will be a sequence of characters. There will be less than ten alpha numerals in each line. The input will be terminated by "End of File".

The input must be read from standard input.

Output

For each line of input generate the permutation of those characters. The input ordering is very important for the output. That is the permutation sequence for '`abc`' and '`bca`' will not be the same. Separate each set of permutation output with a blank line.

The output must be written to standard output.

Sample Input	**Sample Output**
abc bca dcba	cba bca bac cab acb abc acb cab cba abc bac bca abcd bacd bcad bcda acbd cabd cbad cbda acdb cadb cdab cdba abdc badc bdac bdca adbc dabc dbac dbca adcb dacb dcab dcba

C - Sum It Up

Source file name: `sum.py`
Time limit: 2 seconds

Given a specified total t and a list of n integers, find all distinct sums using numbers from the list that add up to t. For example, if $t = 4$, $n = 6$, and the list is $[4, 3, 2, 2, 1, 1]$, then there are four different sums that equal 4: 4, 3 + 1, 2 + 2, and 2 + 1 + 1. (A number can be used within a sum as many times as it appears in the list, and a single number counts as a sum). Your job is to solve this problem in general.

Input

The input will contain one or more test cases, one per line. Each test case contains t, the total, followed by n, the number of integers in the list, followed by n integers $x_1, \ldots, x_n$. If $n = 0$ it signals the end of the input; otherwise, t will be a positive integer less than 1000, n will be an integer between 1 and 12 (inclusive), and $x_1, \ldots, x_n$ will be positive integers less than 100. All numbers will be separated by exactly one space. The numbers in each list appear in nonincreasing order, and there may be repetitions.

The input must be read from standard input.

Output

For each test case, first output a line containing '`Sums of`', the total, and a colon. Then output each sum, one per line; if there are no sums, output the line '`NONE`'. The numbers within each sum must appear in nonincreasing order. A number may be repeated in the sum as many times as it was repeated in the original list. The sums themselves must be sorted in decreasing order based on the numbers appearing in the sum. In other words, the sums must be sorted by their first number; sums with the same first number must be sorted by their second number; sums with the same first two numbers must be sorted by their third number; and so on. Within each test case, all sums must be distinct; the same sum cannot appear twice.

The output must be written to standard output.

Sample Input	**Sample Output**

`4 6 4 3 2 2 1 1`
`5 3 2 1 1`
`400 12 50 50 50 50 50 50 25 25 25 25 25 25`
`0 0`
	`Sums of 4:`
	`4`
	`3+1`
	`2+2`
	`2+1+1`
	`Sums of 5:`
	`NONE`
	`Sums of 400:`
	`50+50+50+50+50+50+25+25+25+25`
	`50+50+50+50+50+25+25+25+25+25+25`

D - Budget Travel

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

An American travel agency is sometimes asked to estimate the minimum cost of traveling from one city to another by automobile. The travel agency maintains lists of many of the gasoline stations along the popular routes. The list contains the location and the current price per gallon of gasoline for each station on the list.

In order to simplify the process of estimating this cost, the agency uses the following rules of thumb about the behavior of automobile drivers.

- A driver never stops at a gasoline station when the gasoline tank contains more than half of its capacity unless the car cannot get to the following station (if there is one) or the destination with the amount of gasoline in the tank.
- A driver always fills the gasoline tank completely at every gasoline station stop.
- When stopped at a gasoline station, a driver will spend $2.00 on snacks and goodies for the trip.
- A driver needs no more gasoline than necessary to reach a gasoline station or the city limits of the destination. There is no need for a "safety margin".
- A driver always begins with a full tank of gasoline.
- The amount paid at each stop is rounded to the nearest cent (where 100 cents make a dollar).

You must write a program that estimates the minimum amount of money that a driver will pay for gasoline and snacks to make the trip.

Input

Program input will consist of several data sets corresponding to different trips. Each data set consists of several lines of information. The first 2 lines give information about the origin and destination. The remaining lines of the data set represent the gasoline stations along the route, with one line per gasoline station. The following shows the exact format and meaning of the input data for a single data set.

Line 1: One real number – the distance from the origin to the destination

Line 2: Three real numbers followed by an integer

- The first real number is the gallon capacity of the automobile's fuel tank.
- The second is the miles per gallon that the automobile can travel.
- The third is the cost in dollars of filling the automobiles tank in the origination city.
- The integer (less than 51) is the number of gasoline stations along the route.

Each remaining line: Two real numbers

- The first is the distance in miles from the origination city to the gasoline station.
- The second is the price (in cents) per gallon of gasoline sold at that station.

All data for a single data set are positive. Gasoline stations along a route are arranged in nondescending order of distance from the origin. No gasoline station along the route is further from the origin than the distance from the origin to the destination. There are always enough stations appropriately placed along the each route for any car to be able to get from the origin to the destination.

The end of data is indicated by a line containing a single negative number.

The input must be read from standard input.

Output

For each input data set, your program must print the data set number and a message indicating the minimum total cost of the gasoline and snacks rounded to the nearest cent. That total cost must include the initial cost of filling the tank at the origin. Sample input data for 2 separate data sets and the corresponding correct output follows.

The output must be written to standard output.

Sample Input	**Sample Output**
475.6 11.9 27.4 14.98 6 102.0 99.9 220.0 132.9 256.3 147.9 275.0 102.9 277.6 112.9 381.8 100.9 516.3 15.7 22.1 20.87 3 125.4 125.9 297.9 112.9 345.2 99.9 -1	Data Set #1 minimum cost = $27.31 Data Set #2 minimum cost = $38.09