ADA - Análisis y Diseño de Algoritmos, 2024-1
Tarea 3: Semanas 5, 6 y 7

Para entregar el domingo 10 de marzo de 2024

Problemas conceptuales a las 23:59 por BrightSpace

Problemas prácticos a las 23:59 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

La siguiente colección de ejercicios, tomados del libro de Cormen et al. es para repasar y afianzar conceptos, pero no deben ser entregados como parte de la tarea.

14.1-2, 14.1-3, 14.2-1, 14.2-4, 14.4-1, 14.4-2, 14.4-5, 14.5-1.

Problemas conceptuales

1. Ejercicio 2: *Text Segmentation* (Erickson, página 124).
2. Ejercicio 6: *Suffle of Strings* (Erickson, página 126).
3. Ejercicio 6.8: *The City of Zion* (Kleinberg & Tardos, página 319).

Problemas prácticos

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

A - On a Diet

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

Alarm bells are ringing: Summer is rapidly approaching and our worst enemy is our mirror.

We've got a kitchen recipe book including the calories associated to each course. We want to select some courses, without repeating, that match the amount of calories given or exceed them minimaly.

Input

The first line of the input contains an integer, t, indicating the number of test cases. For each test case, three lines appear, the first one contains a number n, $100 \leq n \leq 2500$, representing the minimun amount of calories we want to eat (n is multiple of 10). The second line contains a number p, $5 \leq p \leq 100$, representing the number of courses we have. The third line of each test case contains p numbers, representing the amount of calories of the p courses (these p numbers are larger or equal to 50, smaller or equal to 2500, and multiple of 10).

The input must be read from standard input.

Output

For each test case the output should contain a single line, that consists of the amount of calories of our selection (i.e., the selection that matches the amount of calories given or exceeds them minimaly) or the string `'NO SOLUTION'` if no solution is possible.

The output must be written to standard output.

Sample Input	**Sample Output**
4 2480 5 1230 1050 820 890 1150 2140 4 450 150 120 50 1200 5 320 570 610 1560 890 1810 6 2340 780 940 310 660 790	2760 NO SOLUTION 1210 1880

B - Philip J. Fry Problem

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

It's been a few months since Bender solved his famous bending problem. A special request has arrived to Planet Express Inc., the interplanetary courier company where Bender works. Professor Farnsworth (founder, CEO, and chairman of the board) immediately prepared the mission and summoned his crew: Philip J. Fry, Turanga Leela, and Bender. To succeed in the mission, the crew has to make n trips $\tau_1, \tau_2, \ldots, \tau_n$, in the exact order established by Professor Farnsworth. He has also provided the crew with a notebook that specifies how many minutes each trip in the spaceship will take.

Nibbler, Leela's stupid pet, is also a member of the crew and a critical element to succeed in the mission. Nibbler's feces, which it expels every now and then, consist of spheres of dark matter that can be used to increase the speed of the spaceship. In fact, a sphere used during a trip halves its duration. Bender's role in the mission is to pick up the heavy spheres and put them in the reactor of the spaceship. However, he will never put two spheres during the same trip: the accumulation of dark matter is extremely dangerous and may destroy the spaceship.

In each trip of the mission, Nibbler expels a certain number of dark matter spheres that can be used to decrease the duration of any of the upcoming trips. In other words, a sphere of dark matter produced during the trip τ_i can be used to duplicate the speed of the spaceship in one of the trips τ_j , with $i < j \leq n$.

Fry is responsible for planning how to use the spheres to reduce the total travel time. Your task is to help Fry in determining what is the minimum duration of the mission if he uses Nibbler's spheres cleverly.

Input

The input consists of several test cases. The first line of each case contains an integer n indicating the number of trips ($1 \leq n \leq 100$). Then, n lines follow describing each of the trips $\tau_1, \tau_2, \ldots, \tau_n$: each line contains two integers t_i and b_i separated by blanks, where t_i ($2 \leq t_i \leq 1000$) indicates the duration in minutes specified by Professor Farnsworth for the trip τ_i (t_i is always even) and b_i ($0 \leq bi \leq n$) indicates the number of dark matter spheres that Nibbler will expel during the trip τ_i. The last test case is followed by a line containing a single `'0'`.

The input must be read from standard input.

Output

For each test case, print a line with an integer number indicating the minimum time required to complete the mission.

The output must be written to standard output.

Sample Input	**Sample Output**
2 24 1 10 0 2 10 1 24 0 3 10 0 24 0 38 0 3 10 1 24 0 14 0 3 10 1 24 0 38 0 3 10 1 24 1 38 0 3 10 3 24 0 38 1 0	29 22 72 36 53 41 41

C - The Poor Giant

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

On a table, there are n apples, the i-th apple has the weight $k + i$ ($1 \leq i \leq n$). Exactly one of the apples is sweet, lighter apples are all bitter, while heavier apples are all sour. The giant wants to know which one is sweet, the only thing he can do is to eat apples. He hates bitter apples and sour apples, what should he do?

For examples, $n = 4$, $k = 0$, the apples are of weight 1, 2, 3, 4. The giant can first eat apple #2.

- if #2 is sweet, the answer is #2
- if #2 is sour, the answer is #1
- if #2 is bitter, the answer might be #3 or #4, then he eats #3, he'll know the answer regardless of the taste of #3

The poor giant should be prepared to eat some bad apples in order to know which one is sweet. Let's compute the total weight of apples he must eat in all cases.

- #1 is sweet: 2
- #2 is sweet: 2
- #3 is sweet: 2 + 3 = 5
- #4 is sweet: 2 + 3 = 5

The total weights = 2 + 2 + 5 + 5 = 14.

This is not optimal. If he eats apple #1, then he eats total weight of 1, 3, 3, 3 when apple #1, #2, #3 and #4 are sweet respectively. This yields a solution of 1 + 3 + 3 + 3 = 13, beating 14.

What is the minimal total weight of apples in all cases?

Input

The first line of input contains a single integer t ($1 \leq t \leq 100$), the number of test cases. The following t lines each contains a positive integer n and a non-negative k ($1 \leq n + k \leq 500$).

The input must be read from standard input.

Output

For each test case, output the minimal total weight in all cases as shown in the sample output.

The output must be written to standard output.

Sample Input	**Sample Output**
5 2 0 3 0 4 0 5 0 10 20	Case 1: 2 Case 2: 6 Case 3: 13 Case 4: 22 Case 5: 605

D - Shopping

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

You have just moved into a new apartment and have a long list of items you need to buy. Unfortunately, to buy this many items requires going to many different stores. You would like to minimize the amount of driving necessary to buy all the items you need.

Your city is organized as a set of intersections connected by roads. Your house and every store is located at some intersection. Your task is to find the shortest route that begins at your house, visits all the stores that you need to shop at, and returns to your house.

Input

The first line of input contains a single integer, the number of test cases to follow. Each test case begins with a line containing two integers N and M, the number of intersections and roads in the city, respectively. Each of these integers is between 1 and 100000, inclusive. The intersections are numbered from 0 to $N - 1$. Your house is at the intersection numbered 0. M lines follow, each containing three integers X, Y, and D, indicating that the intersections X and Y are connected by a bidirectional road of length D. The following line contains a single integer S, the number of stores you need to visit, which is between 1 and ten, inclusive. The subsequent S lines each contain one integer indicating the intersection at which each store is located. It is possible to reach all of the stores from your house.

The input must be read from standard input.

Output

For each test case, output a line containing a single integer, the length of the shortest possible shopping trip from your house, visiting all the stores, and returning to your house.

The output must be written to standard output.

Sample Input	**Sample Output**
1 4 6 0 1 1 1 2 1 2 3 1 3 0 1 0 2 5 1 3 5 3 1 2 3	4

E - Towards Zero

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

Have you ever heard of this game? The player jumps in a special game board under certain rules, so that the numbers he jumps on, after being linked up by plus or minus signs, get closest to zero.

The game board looks like the one shown in Figure 1.1. Its size is determined by the number of squares in the middle row N. (Figure 1.1 is an example where $N = 4$.) The player starts at the bottom-most square, then jumps in any of the directions shown in Figure 1.2. The game ends when the player reaches the topmost square. During the game, the player cannot jump out of the game board. Finally we write down the $2N - 1$ numbers in order, then insert plus or minus signs between each pair of adjoining numbers such that the result is closest to zero.

Let us look at the game board in Figure 1.1 as a example. We should get: $7 + 8 + (-5) + (-2) - 5 - 1 - 2 = 0$, $7 + 10 + (-7) - 6 + (-3) - 3 + 2 = 0$, $7 + 10 + (-5) - 10 - 5 + 1 + 2 = 0$, or $7 + 10 + (-5) + (-2) - 5 - 3 - 2 = 0$.

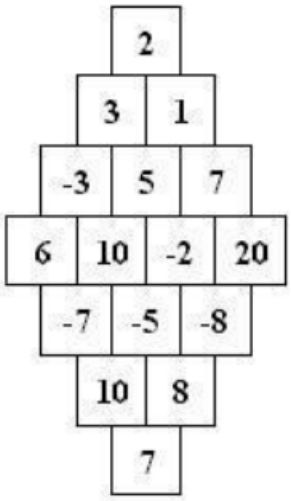

Figure 1.1

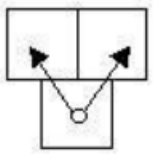

Figure 1.2

Input

The first line of input contains N ($1 \leq N \leq 30$). The following $2N - 1$ lines give the numbers in the squares in the game board. The j-th number in the $(i + 1)$-th line corresponds to the j-th number in the i-th row of the game board. (The numbers are all ≥ -50 and ≤ 50)

Input contains multiple test cases, and it ends with a case where $N = 0$.

The input must be read from standard input.

Output

For each case, your output should print the absolute value of the result you get for each game board.

The output must be written to standard output.

Sample Input	**Sample Output**
4 2 3 1 -3 5 7 6 10 -2 20 -7 -5 -8 10 8 7 0	0