Colombian Collegiate Programming League

CCPL 2017

Round 3 – April 08

Problems

This set contains 11 problems; pages 1 to 17.

(Borrowed from several sources online.)

Official site `http://programmingleague.org`

Follow us on Twitter `@CCPL2003`

A - Ardenia

Source file name: `ardenia.c`, `ardenia.cpp`, `ardenia.java`, *or* `ardenia.py`

Welcome to Ardenia. Ardenia is a mythical land, filled with adventure and danger, dwarves and dragons, mages and rouges. And puzzles. Lots of puzzles. Actually, the love for puzzles is the most important life ingredient of the inhabitants, and the only part they have in common. This month, the people of Ardenia wonder what is the distance between two line segments in three dimensional space. (The distance between segments is defined as the minimum among distances between two points of different segments.) Actually, this problem had originally some motivation, but as nobody from Ardenia cares about motivations, neither should you.

Input

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 10^5$, denoting the number of test cases. Then Z test cases follow. The first line of the input instance contains six space-separated integers $x_1, y_1, z_1, x_2, y_2, z_2 \in [-20, 20]$. Points (x_1, y_1, z_1) and (x_2, y_2, z_2) are the (different) endpoints of the first segment. The second line contains six integers in the same format, describing the second segment.

The input must be read from standard input.

Output

For each test case you should output a single line consisting of two co-prime integers l and $m > 0$, such that $\frac{l}{m}$ is the squared distance between the given two segments.

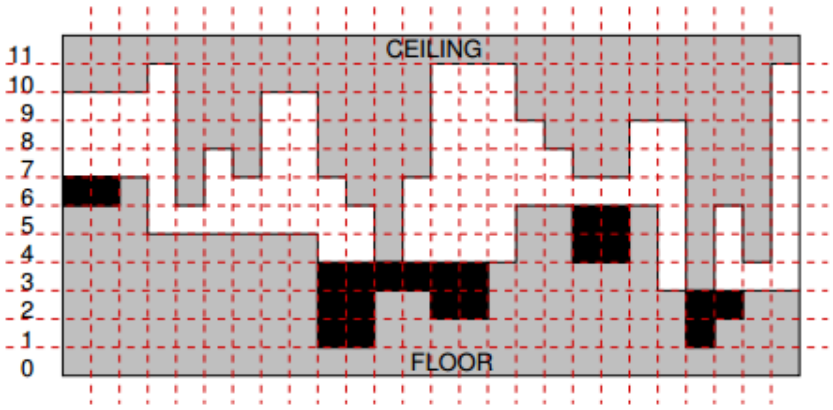
The output must be written to standard output.

Sample Input	Sample Output
2 0 0 0 1 1 1 1 1 1 2 2 2 1 0 0 0 1 0 1 1 0 2 2 0	0 1 1 2

B - Cave

Source file name: `cave.c`, `cave.cpp`, `cave.java`, *or* `cave.py`

As an owner of a land with a cave you were delighted when you last heard that underground fuel tanks are great business. Of course, the more volume one can store, the better. In case of your cave, the effective volume is not easy to calculate, because the cave has a rather sophisticated shape (see figure). Thank heavens it is degenerate in one dimension!



The cave. All ponds that can be flooded with fuel are marked black.

Furthermore, there is some electrical wiring on the ceiling of the cave. You can never be sure if the insulation is intact, so you want to keep the fuel level just below the ceiling at every point. You can pump the fuel to whatever spots in the cave you choose, possibly creating several ponds. Bear in mind though that the fuel is a liquid, so it minimises its gravitational energy, e.g., it will run evenly in every direction on a flat horizontal surface, pour down whenever possible, obey the rule of communicating vessels, etc. As the cave is degenerate and you can make the space between the fuel level and the ceiling arbitrarily small, you actually want to calculate the maximum possible area of ponds that satisfy aforementioned rules.

Input

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 15$, denoting the number of test cases. Then Z test cases follow. In the first line of an input instance, there is an integer n ($1 \leq n \leq 10^6$) denoting the width of the cave. The second line of input consists of n integers $p_1, p_2, \ldots, p_n$ and the third line consists of n integers $s_1, s_2, \ldots, s_n$, separated by single spaces. The numbers p_i and s_i satisfy $0 \leq p_i < s_i \leq 1000$ and denote the floor and ceiling level at interval $[i, i + 1)$, respectively.

The input must be read from standard input.

Output

For each input instance your program is to print out one integer: the maximum total area of admissible ponds in the cave.

The output must be written to standard output.

Sample Input	Sample Output
1 15 6 6 7 5 5 5 5 5 5 1 1 3 3 2 2 10 10 10 11 6 8 7 10 10 7 6 4 7 11 11	14

C - Casting Spells

Source file name: `casting.c`, `casting.cpp`, `casting.java`, *or* `casting.py`

Casting spells is the least understood technique of dealing with real life. Actually, people find it quite hard to distinguish between a real spells like "abrahellehhelleh" (used in the battles and taught at the mage universities) and screams like "rachelhellabracadabra" (used by uneducated witches for shouting at cats).

Finally, the research conducted at the Unheard University showed how one can measure the power of a word (be it a real spell or a scream). It appeared that it is connected with the mages' ability to pronounce words backwards. (Actually, some singers were burned at the stake for exactly the same ability, as it was perceived as demonic possession.) Namely, the power of a word is the length of the maximum subword of the form ww^Rww^R (where w is an arbitrary sequence of characters and w^R is w written backwards). If no such subword exists, then the power of the word is 0. For example, the power of `abrahellehhelleh` is 12 as it contains `hellehhelleh` and the power of `rachelhellabracadabra` is 0. Note that the power of a word is always a multiple of 4.

Input

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 40$, denoting the number of test cases. Then Z test cases follow. Each test case is one line containing a word of length at most $3 \cdot 10^5$, consisting of (large or small) letters of the English alphabet.

The input must be read from standard input.

Output

For each test case you should output one integer k being the power of the word.

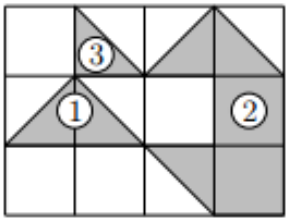
The output must be written to standard output.

Sample Input	Sample Output
2 abrahellehhelleh rachelhellabracadabra	12 0

D - Decision

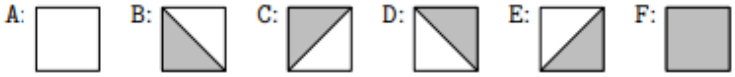
Source file name: `decision.c`, `decision.cpp`, `decision.java`, *or* `decision.py`

In a galaxy not so far away, in a time where men were real men, women were real women, and small furry creatures from Alpha Centauri were real small furry creatures from Alpha Centauri, an astronom called Mr. Gorsky discovered a small inhabited planet. After an initial enthusiasm (*yes, we are not alone!*), all the living Nobel Peace Prize winners gathered in one place, formed a committee, and discussed options of invading the planet. Long story short, in order to decide on this important topic, they need to know the number of the cities on the remote planet. The quality of photos delivered by Mr. Gorsky were unfortunately quite bad: on a rectangular grid, each grid element was either blank (no city there) or (partially) dark, which meant a city or a part of it. If two dark parts share a common edge, they are a part of the same city. The committee then said plainly: "You have to count the number of the cities. Good luck, Mr. Gorsky". An example map containing three cities is presented below.



Input

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 40$, denoting the number of test cases. Then Z test cases follow. The first line of an input instance contains two integers n and m, the photo dimensions, such that $1 \leq n, m \leq 1000$. The following n lines contain the description of the photo. Each line contains m characters from the set {`A`,`B`,`C`,`D`,`E`,`F`} encoding the grid elements in the following way:



The input must be read from standard input.

Output

For each input instance, your program should output one line containing the number of cities on a given map.

The output must be written to standard output.

Sample Input	Sample Output
4 1 2 DD 2 2 FB DF 2 3 FAA AFB 4 4 AACB CAFD AFCE AACA	2 1 2 6

E - Furry Nuisance

Source file name: `furry.c`, `furry.cpp`, `furry.java`, *or* `furry.py`

In order to protect himself from evil bunnies, Freddy decided to install an automatic system to detect them in pictures from surveillance cameras. Sophisticated software detects important points in the picture and lines between them. Unfortunately, the terrain in the pictures is quite varied and lot of the points and lines are actually not bunnies.

You have made the following observation: Each bunny has four paws and a body joining them. Based on this observation, write a program to decide whether a given picture can possibly contain a bunny.

Input

The input contains several test cases. The first line of each test case contains two integers n and m ($0 \leq n \leq 10\,000, 0 \leq m \leq 20\,000$), giving the number of points and lines in the image, respectively. Each of the m following lines contains two distinct integers x and y ($1 \leq x, y \leq n$), indicating that the points x and y are directly joined by a line. You may assume that each pair of points is joined by at most one direct line and that no point is directly joined with itself.

The input must be read from standard input.

Output

For each input instance, output "`YES`" if the picture can contain a bunny, and "`NO`" otherwise. The picture can contain a bunny if it is possible to remove some of the points and lines so that the resulting image is connected and has exactly 4 paws. The image is said to be connected if (and only if) each two points are joined with each other by one or more successive lines. A paw is a point which is directly joined with exactly one other point.

The output must be written to standard output.

Sample Input	**Sample Output**
2 1 1 2 5 4 1 2 1 3 1 4 1 5	NO YES

F - Fractional Lotion

Source file name: `fractional.c`, `fractional.cpp`, `fractional.java`, *or* `fractional.py`

Freddy practices various kinds of alternative medicine, such as homeopathy. This practice is based on the belief that successively diluting some substances in water or alcohol while shaking them thoroughly produces remedies for many diseases.

This year, Freddy's vegetables appear to have caught some disease and he decided to experiment a little bit and investigate whether homeopathy works for vegetables too. As Freddy is also a big fan of mathematics, he does not strictly insist that the substances have small concentrations, but he instead requires the concentrations to be reciprocals of integers $\left(\frac{1}{n}\right)$. In experiments, some of the vegetables really got much better.

Seeing Freddy's successes, a fellow gardener also wants to try one of these potions and asks for a flask. Freddy has one flask of the potion in concentration $\frac{1}{n}$ and does not want to give it all out. Your task is to find out in how many ways the potion can be split into two flasks and diluted so that the resulting potions both have the same volume as the original one and the resulting concentrations also are reciprocals of integers – we do not want to end up with useless fluid, do we?

Input

Each line of the input describes one test case. The line contains the expression "`1/n`" representing the original concentration. You are guaranteed that $1 \leq n \leq 10\,000$. There are no spaces on the line.

The input must be read from standard input.

Output

For each test case, output a single line with the total number of distinct pairs $\{x, y\}$ of positive integers satisfying $\frac{1}{x} + \frac{1}{y} = \frac{1}{n}$. Pairs differing only in the order of the two numbers are not considered different.

The output must be written to standard output.

Sample Input	Sample Output
1/2 1/4 1/1 1/5000	2 3 1 32

G - Garlands

Source file name: `garlands.c`, `garlands.cpp`, `garlands.java`, *or* `garlands.py`

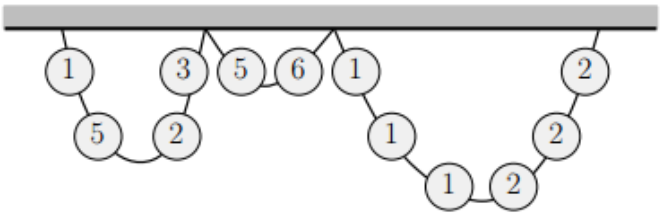
Garland decorator is a profession which recently gained in importance, especially during Christmas time. Any kid can decorate a christmas tree, any parent can put gifts in sockets, and even anyone can start believing in Santa Claus, but hanging christmas garlands is a completely different story. As you will learn, it is an extremely important, responsible and tough job.

A garland consists of n pieces of equal length. Due to decorations like christmas balls attached to garlands, piece i has its own weight w_i. The garland has to be attached to the ceiling in m spots, where the very beginning of the garland should be attached to spot 1 and its end to spot m. The garland should also be hooked to the remaining spots, which divides it into segments, each consisting of several consecutive pieces. There are, however, several rules that every respectable garland decorator should keep in mind.

i Each segment should contain a positive even number of pieces. Due to this condition, we may divide a segment into two *half-segments*.

ii To minimize the chance that a guest hits your precious garland with their head (and tears it into pieces), the garland cannot hang too low: each half-segment can contain at most d pieces.

iii Finally, to keep the ceiling from falling on people heads, the decorator should minimize the weight of the heaviest half-segment.



Input

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 50$, denoting the number of test cases. Then Z test cases follow. The description of each garland consists of two lines. The first line describing a particular garland contains three positive integers n, m, and d ($1 \leq n \leq 40\,000, 2 \leq m \leq 10\,000, 1 \leq d \leq 10\,000$) separated by single spaces and described above. The second line contains n positive integers $w_1, w_2, ..., w_n$ ($1 \leq w_i \leq 10\,000$), being the weights of the corresponding pieces.

The input must be read from standard input.

Output

For each garland, your program should output a single line containing one integer, being the weight of the heaviest half-segment in an optimal attachment of the garland. If it is not possible to hang the garland satisfying conditions (i) and (ii), then your program should output word `BAD`.

The output must be written to standard output.

Sample Input	Sample Output
4 4 3 10 10 10 20 20 6 4 10 1 1 100 100 1 1 6 3 10 1 1 100 100 1 1 1 2 2 333	20 100 200 BAD

H - False Sense of Security

Source file name: `security.c`, `security.cpp`, `security.java`, *or* `security.py`

Freddy discovered a new procedure to grow much bigger cauliflowers. He wants to share this finding with his fellow gardener Tommy but he does not want anyone to steal the procedure. So the two gardeners agreed upon using a simple encryption technique proposed by M. E. Ohaver.

The encryption is based on the Morse code, which represents characters as variable-length sequences of dots and dashes. The following table shows the Morse code sequences for all letters:

A	.-	H	....	O	---	V	...-
B	-...	I	..	P	.--.	W	.--
C	-.-.	J	.---	Q	--.-	X	-..-
D	-..	K	-.-	R	.-.	Y	-.--
E	.	L	.-..	S	...	Z	--..
F	..-.	M	--	T	-		
G	--.	N	-.	U	..-		

Note that four possible dot-dash combinations are unassigned. For the purposes of this problem we will assign them as follows (note these are not the assignments for actual Morse code):

underscore ("_")	..--	period (".")	---.
comma (",")	.-.-	question mark ("?")	----

In practice, characters in a message are delimited by short pauses, typically displayed as spaces. Thus, the message ACM GREATER NY REGION is encoded as:

```
.- -.-. -- ..-- --. .-. . .- - . .-. ..-- -. -.-- ..-- .-. . --. .. --- -.
```

The Ohaver's encryption scheme is based on mutilating Morse code, namely by removing the pauses between letters. Since the pauses are necessary (because Morse is a variable-length encoding that is not prefix-free), a string is added that identifies the number of dots and dashes in each character. For example, consider the message "`.--.-.--`". Without knowing where the pauses should be, this could be "`ACM`", "`ANK`", or several other possibilities. If we add length information, such as "`.--.-.-- 242`", then the code is unambiguous.

Ohaver's scheme has three steps, the same for encryption and decryption:

i Convert the text to Morse code without pauses but with a string of numbers to indicate code lengths.

ii Reverse the string of numbers.

iii Convert the dots and dashes back into the text using the reversed string of numbers as code lengths.

As an example, consider the encrypted message "`AKADTOF IBOETATUK IJN`". Converting to Morse code with a length string yields:

```
.--.-.--..----..-...--..-...---.-.--..--.-..--...----. 232313442431121334242
```

By reversing the numbers and decoding, we get the original message "`ACM GREATER NY REGION`". The security of this encoding scheme is not too high but Freddy believes it is sufficient for his purposes. Will you help Freddy to implement this encoding algorithm and to protect his sensitive information?

Input

The input will consist of several messages encoded with Ohaver's algorithm, each of them on one line. Each message will use only the twenty-six capital letters, underscores, commas, periods, and question marks. Messages will not exceed 1 000 characters in length.

The input must be read from standard input.

Output

For each message in the input, output the decoded message on one line.

The output must be written to standard output.

Sample Input	Sample Output
FENDSVTSLHW.EDATS,EULAY TRDNWPLOEF NTTTGAZEJUIIGDUZEHKUE QEWOISE.EIVCAEFNRXTBELYTGD. ?EJHUT.TSMYGW?EJHOT DSU.XFNCJEVE.OE_UJDXNO_YHU?VIDWDHPDJIKXZT?E ADAWEKHZN,OTEATWRZMZN_IDWCZGTEPION	FALSE_SENSE_OF_SECURITY CTU_PRAGUE TWO_THOUSAND_THIRTEEN QUOTH_THE_RAVEN,_NEVERMORE. TO_BE_OR_NOT_TO_BE? THE_QUICK_BROWN_FOX_JUMPS_OVER_THE_LAZY_DOG ADAPTED_FROM_ACM_GREATER_NY_REGION

I - Islands

Source file name: `islands.c`, `islands.cpp`, `islands.java`, *or* `islands.py`

Deep in the Carribean, there is an island even stranger than the Monkey Island, dwelled by Horatio Torquemada Marley. Not only it has a rectangular shape, but is also divided into an $n \times m$ grid. Each grid field has a certain height. Unfortunately, the sea level started to raise and in year i, the level is i meters. Another strange feature of the island is that it is made of sponge, and the water can freely flow through it. Thus, a grid field whose height is at most the current sea level is considered *flooded*. Adjacent unflooded fields (i.e., sharing common edge) create unflooded areas. Sailors are interested in the number of unflooded areas in a given year. An example of a 4×5 island is given below. Numbers denote the heights of respective fields in meters. Unflooded fields are darker; there are two unflooded areas in the first year and three areas in the second year.

Year 1:

1	2	3	3	1
1	3	2	2	1
2	1	3	4	3
1	2	2	2	2

Year 2:

1	2	3	3	1
1	3	2	2	1
2	1	3	4	3
1	2	2	2	2

Input

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 20$, denoting the number of test cases. Then Z test cases follow. The first line of each test case contains two numbers n and m separated by a single space, the dimensions of the island, where $1 \leq n, m \leq 1000$. Next n lines contain m integers from the range $[1, 10^9]$ separated by single spaces, denoting the heights of the respective fields. Next line contains an integer T ($1 \leq T \leq 10^5$). The last line contains T integers t_j , separated by single spaces, such that $0 \leq t1 \leq t2 \leq \ldots \leq t_{T-1} \leq t_T \leq 10^9$.

The input must be read from standard input.

Output

For each test case your program should output a single line consisting of T numbers r_j separated by single spaces, where r_j is the number of unflooded areas in year t_j.

The output must be written to standard output.

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

J - Game

Source file name: `game.c`, `game.cpp`, `game.java`, *or* `game.py`

Tic-tac-toe is the third most popular activity to kill a lazy afternoon in Ardenia (right after solving puzzles and insulting your neighbors). Arthum and Breece are not fans of this game, but their mother told them to play, so they sit at a 5×5 board. Both have a large pile of marbles: marbles of Arthum have an *A* written on them and that of Breece have a *B*. However, as they are both two years old, they have no concept of rounds. Instead, they just toss their marbles as quick as possible, so after a while each board field has either marble *A* or marble *B*.

At that point, they would like to determine the winner, but counting is not their strong point, either. (They are two years old, remember?) Recall that the goal of tic-tac-toe is to have three own marbles in a row, i.e., lying at three consecutive fields horizontally, vertically or diagonally. If both Arthum and Breece have their three marbles in a row, or neither of them has it, we call it a draw.

Input

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 10^5$, denoting the number of test cases. Then Z test cases follow The input instance describes marbles placed on the board in a single game. The instance consists of 5 rows and each of them consists of 5 letters: *A* or *B*.

The input must be read from standard input.

Output

For each test case you should output one line describing the outcome of the game, i.e., one of the three possible strings: `A wins`, `B wins`, or `draw`.

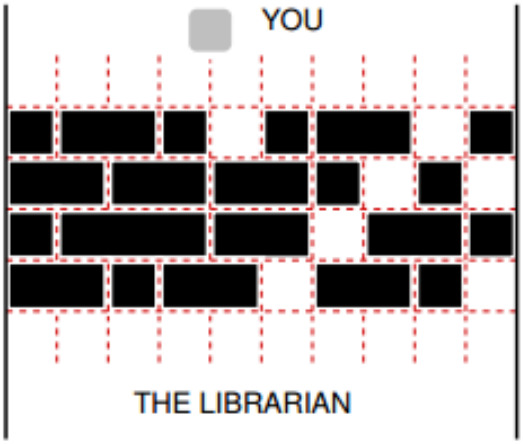
The output must be written to standard output.

Sample Input	**Sample Output**
2 AABBA BAAAB AAABA ABAAB BAAAB AAAAA AAAAA BAAAA ABAAA AABAA	A wins draw

K - Knowledge for the Masses

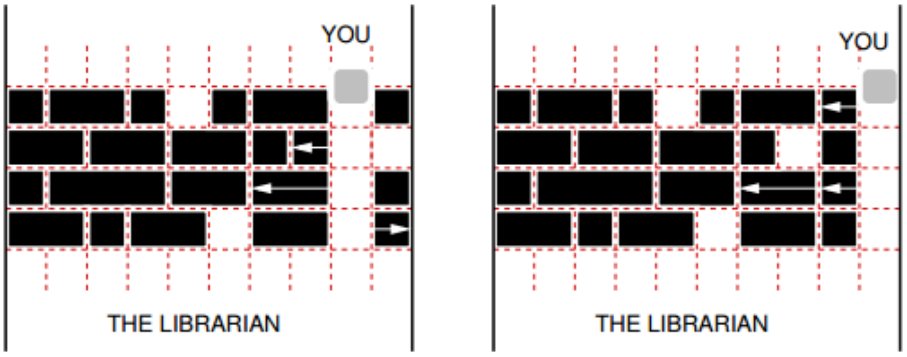
Source file name: `knowledge.c`, `knowledge.cpp`, `knowledge.java`, *or* `knowledge.py`

You are in a library equipped with bookracks that move on rails. There are many parallel rails, i.e., the bookracks are organized in several rows, see figure:



The boockracks in the library. There is no passage to the librarian at the moment.

To borrow a book, you have to find the librarian, who seems to hide on the opposite side of the bookracks. Your task then is to move the racks along the rails so that a passage forms. Each rack has a certain integer width, and can be safely positioned at any integer point along the rail. (A rack does not block in a non-integer position and could accidentally move in either direction). The racks in a single row need not be contiguous – there can be arbitrary (though integer) space between two successive bookracks. A passage is formed at position k if there is no bookrack in the interval $(k, k + 1)$ in any row (somehow you don't like the idea of trying to find a more sophisticated passage in this maze.)



The passages formed in the library: at position 8 (the left figure) and at position 9 (the right figure). Both attained at cost 3 by moving the bookracks marked with arrows.

Moving a rack requires a certain amount of effort on your part: moving it in either direction costs 1. This cost does not depend on the distance of the shift, which can be explained by a well known fact that static friction is considerably higher than kinetic friction. Still, you are here to borrow a book, not to work out, so you would like to form a passage (at any position) with as little effort as possible.

Input

The input contains several test cases. The first line of the input contains a positive integer $Z \leq 15$, denoting the number of test cases. Then Z test cases follow. Each test case starts with two space separated integers R and L ($1 \leq R, 1 \leq L \leq 10^6$). They denote the number of rows and the width of each and every row, respectively. Then R lines with rows descriptions follow. Each such line starts with an integer n_i, followed by n_i integers $a_{i,1}, a_{i,2}, \ldots, a_{i,n_i}$, all separated by single spaces. Number $a_{i,j}$ denotes either the width of a bookrack when $a_{i,j} > 0$ or a unit of empty space when $a_{i,j} = 0$. Note that for any row i, $\sum_j a_{i,j}$ equals L minus the number of $a_{i,j}$ that are equal to zero. You may assume that $n_1 + n_2 + \cdots + n_R \leq 2 \cdot 10^7$. Moreover, there will be at least one 0 in the description of each row, which means that creating a passage is always possible.

The input must be read from standard input.

Output

In the first line, your program should output the minimum cost of making a passage through the bookracks. In the second line, it should print out the increasing sequence of all the positions at which a minimum cost passage can be formed.

The output must be written to standard output.

Sample Input	Sample Output
1 4 10 8 1 2 1 0 1 2 0 1 7 2 2 2 1 0 1 0 6 1 3 2 0 2 1 7 2 1 2 0 2 1 0	3 8 9