Problem A. Sequence maker - 3pts

Input file:	`stdin`
Output file:	`stdout`
Time limit:	2 seconds
Memory limit:	64 megabytes

You have a sequence of integers $a_1, a_2, ..., a_n$. You can decrease or increase any number by 1. Count the minimum number of steps (each decrement or increment counts as a step), needed to build an ideal sequence. Ideal sequence is a sequence, which contains all the numbers from 1 to n (in some order), where n is the length of this sequence.

Input

Input starts with a single integer t ($1 \le t \le 100$) - the number of testcases. Each of the testcases starts on a new line and contains an integer n ($1 \le n \le 10^4$), the number of elements in the sequence followed by n integers $-10^6 \le a_i \le 10^6$, elements of the sequence.

Output

For each testcase you need to output a single integer - the minimum number of steps to make an ideal sequence.

Examples

stdin	stdout

2
2 3 0
3 -1 -1 2
	2
	6

Problem B. Morse Enumeration - 5pts

Input file:	`stdin`
Output file:	`stdout`
Time limit:	2 seconds
Memory limit:	64 megabytes

Morse code was a very widespread communication format before the use of telephones and computers. In fact, Morse code is still in widespread use simply because of its simplicity and clarity on noisy signals. However, one problem can arise with this method of communication: exact meaning. Consider the table on the next page (retrieved from http://en.wikipedia.org/wiki/Morse_code).

Notice that Morse code specifies spacing between letters, and a different spacing between words.If those spaces were lost, it would be impossible to determine the correct meaning of the message without examining all possible transmitted messages. For instance, consider the string ".-". The two characters together could encode the single letter "A"or the string "ET". As strings grow longer, the number of potential meanings grows as well. For instance, the string ".-..."has 15 possible interpretations: "AEEE "AEI "AIE "AS "EB "EDE "ENEE "ENI "ETEEE "ETEI "ETIE "ETS "LE "REE and "RI".

You job is, given a message in International Morse Code, determine the number of strings of letters it could represent.

Input

The input will consist of a number of lines with length no greater than 30 characters. The end of input will consist of line with a "#"at the beginning.

Output

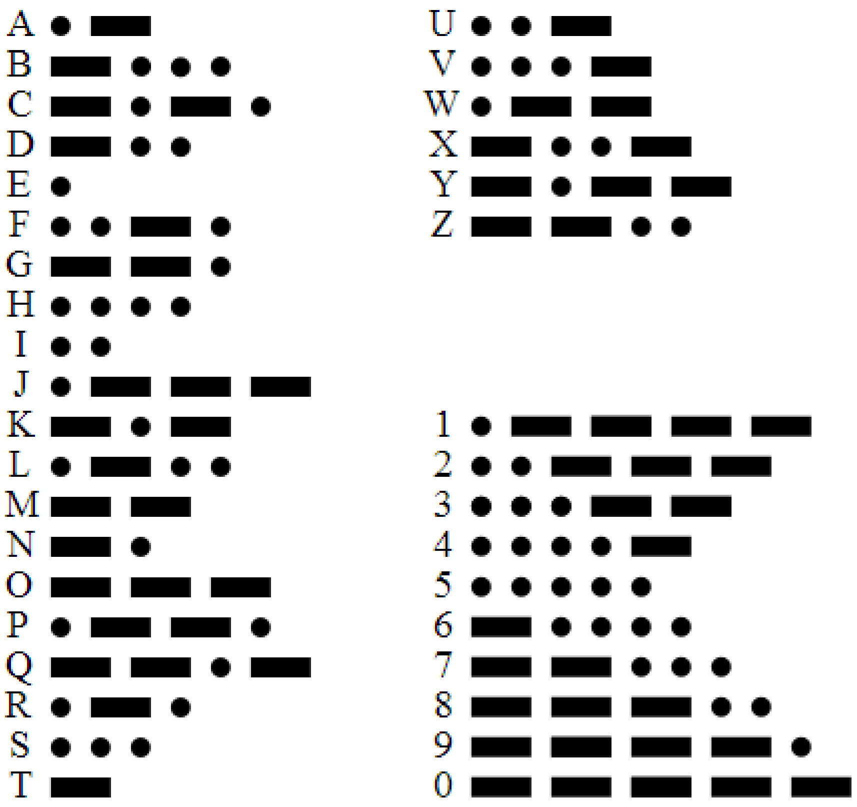
For each test case, print the number of possible messages that can encoded with the given series of dashes and dots on a separate line.

Examples

stdin	stdout
`.-`	`2`
`....`	`8`
`.-...`	`15`
`.-.-.-.-.-.-.-.-.-.-.-.-.-.-.-`	`104099605`
`#`	

Note

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.



A	● ▬	U	● ● ▬
B	▬ ● ● ●	V	● ● ● ▬
C	▬ ● ▬ ●	W	● ▬ ▬
D	▬ ● ●	X	▬ ● ● ▬
E	●	Y	▬ ● ▬ ▬
F	● ● ▬ ●	Z	▬ ▬ ● ●
G	▬ ▬ ●		
H	● ● ● ●		
I	● ●		
J	● ▬ ▬ ▬		
K	▬ ● ▬	1	● ▬ ▬ ▬ ▬
L	● ▬ ● ●	2	● ● ▬ ▬ ▬
M	▬ ▬	3	● ● ● ▬ ▬
N	▬ ●	4	● ● ● ● ▬
O	▬ ▬ ▬	5	● ● ● ● ●
P	● ▬ ▬ ●	6	▬ ● ● ● ●
Q	▬ ▬ ● ▬	7	▬ ▬ ● ● ●
R	● ▬ ●	8	▬ ▬ ▬ ● ●
S	● ● ●	9	▬ ▬ ▬ ▬ ●
T	▬	0	▬ ▬ ▬ ▬ ▬

Problem C. Hotdog Vendors - 2 + 3pts

Input file:	`stdin`
Output file:	`stdout`
Time limit:	15 seconds
Memory limit:	64 megabytes

Last year, several hot dog vendors were lined up along a street, and they had a tricky algorithm to spread themselves out. Unfortunately, the algorithm was very slow and they are still going. All is not lost though! The hot dog vendors have a plan: time to try a new algorithm!

The problem is that multiple vendors might be selling too close to each other, and then they will take each other's business. The vendors can move along the street at 1 meter/second. To avoid interfering with each other, they want to stand so that every pair of them is separated by a distance of at least D meters.

Remember that the street is really long, so there is no danger of running out of space to move in either direction. Given the starting positions of all hot dog vendors, you should find the minimum time they need before all the vendors are separated (each two vendors are at least D meters apart from each other).

Input

Each point of the street is labeled with a number, positive, negative or zero. A point labeled p is |p| meters east of the point labeled 0 if p is positive, and |p| meters west of the point labeled 0 if p is negative. We will use this labeling system to describe the positions of the vendors in the input file.

The first line of the input file contains the number of cases, T. T test cases follow. Each case begins with a line containing the number of points C that have at least one hot dog vendor in the starting configuration and an integer D – the minimum distance they want to spread out to. The next C lines each contain a pair of space-separated integers P, V, indicating that there are V vendors at the point labeled P.

$1 \leq T \leq 50$

All the values P are integers in the range $[-10^5, 10^5]$. Within each test case all P values are distinct and given in an increasing order. The limit on the sum of V values is listed below. All the V values are positive integers. $1 \leq D \leq 10^6$ $1 \leq C \leq 200$. The sum of all V values does not exceed 10^6

Output

For each test case, output one line containing "Case x: y where x is the case number (starting from 1) and y is the minimum amount of time it will take for the vendors to spread out apart on the street. Answers should be printed with 6 decimal digits.

Examples

stdin	stdout

2
3 2
0 1
3 2
6 1
2 2
0 3
1 1
	Case #1: 1.000000
	Case #2: 2.500000