CCSS Programming Contest 2015

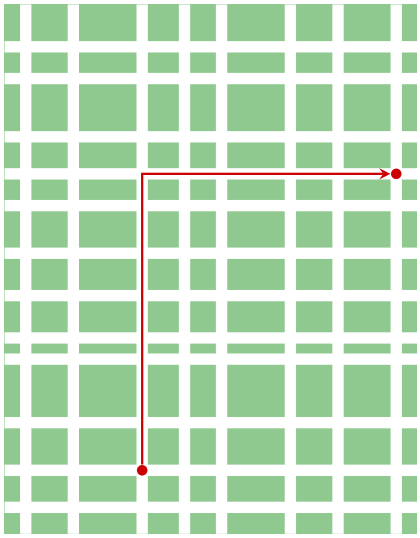
Exclusively for students attending the
Challenges in Computer Science Seminar
2014–2015 at Leiden University

Problems

A Alternate Routes
B Burning CDs
C Competitive Programming
D Dice
E European Energy Union

This is not a blank page

A Alternate Routes



Andrew lives in downtown Manhattan (New York), where the streets are layed out in blocks of long galleries separated by parallel streets. In order to be able to move around easily, Andrew and his friends decided to live close to an intersection. Whenever Andrew visits one of his friends, he first travels the right number of blocks in the north–south direction and then in the east–west direction. For example, in the example depicted on the right he first travels seven blocks to the north and then another five blocks to the east.

Andrew recently befriended a philosopher named Brian. After talking to Andew about his travel routine, Brian noticed that there are many alternate routes that Andrew might take, many of which have the same length as his usual route. However, being a philosopher, Brian is not so good at math. Can you help the two of them calculate the number of shortest routes from Andrew's to Brian's?

Input

The input starts with a line containing an integer T, the number of test cases. Then for each test case:

- One line with two space separated integers N and E satisfying $-1000 \leq N \leq 1000$ as well as $-1000 \leq E \leq 1000$. These denote the number of blocks Andrew has to travel to the north and to the east, respectively. A negative number is used if Andrew has to travel to the south (in case $N < 0$) or to the west (in case $E < 0$).

Output

For each test case, output one line with a single integer R, the number of shortest routes from Andrew's to Brian's. Since this can become quite large, you must reduce your answer modulo 123456789. (No answer greater than or equal to 123456789 or strictly smaller than 0 will be accepted.)

Sample input and output

Input	Output
4	792
7 5	1
0 -10	35
-3 -4	69229809
123 456	

This is not a blank page

B Burning CDs

Good old fashioned uncle Bernard relies on his good old fashioned CD player to play music in his living room. He has selected some of his favourite good old fashioned songs and wants to burn them onto his good old fashioned CDs from the 1990s, using his good old fashioned CD burner from that same era, because—as he keeps telling at family dinners—sure, you can fit days and days of that new crappy music on a flash card the size of your finger nail, but you just cannot do that to good old fashioned music. Good old fashioned music needs space. If you put good old fashioned music on a CD that fits 80 minutes or more, you can just hear that it feels crammed.

Needless to say, uncle Bernard's CDs do not fit 80 minutes. Uncle Bernard's computer, however, is entirely up-to-date. "You have to fight fire with fire," your uncle says, "new crappy viruses need new crappy software to stop them. They are nothing like the good old fashioned chain e-mails that Bill Gates used to send." In particular, his new crappy CD burning software assumes that CDs can fit 80 minutes of music, and as such keeps ruining his good old fashioned CDs.

To solve this issue, uncle Bernard has asked you to do some good old fashioned programming to fix his CD burning software, in exchange for a good old fashioned balloon. You have narrowed the problem down to this: Given the track list and the playing time of the CD, you have to determine how much time is left on the CD.

Input

The input starts with a line containing an integer T, the number of test cases. Then for each test case:

- One line with an integer N, the number of tracks, satisfying $1 \leq N \leq 99$. Then on the same line the playing time of the CD is shown, which is at least 60 minutes and at most 80 minutes.
- One line with N times, the durations of each of the tracks from the track list. Each song lasts at least 4 seconds and at most 80 minutes.

Each time is formatted as m:s, that is: the number of minutes immediately followed by a colon and the number of seconds. The number of minutes m never starts with additional leading zeroes. On the other hand, the number of seconds s is always formatted as a two-digit number. This is illustrated in the samples below.

Output

For each test case, output one line indicating the time remaining (in case the playlist fits on the CD) or the time overrun (in case it doesn't fit). Follow the format from the samples below.

Sample input and output

Input	Output
4 3 74:00 6:41 5:28 3:02 6 67:43 30:00 20:00 10:00 5:00 2:00 0:15 3 75:00 20:43 25:09 30:12 5 65:43 11:52 13:37 9:09 12:59 18:06	58:49 remaining 0:28 remaining 1:04 too long 0:00 remaining

This is not a blank page

C Competitive Programming

You find yourself participating in a programming contest. Using your psychic abilities, for each problem you know in advance:

- the exact moment when you will think of the solution (measured from the beginning of the contest);
- the amount of time it will take you to implement the solution in code.

Time spent programming does not affect the first prediction: by using teamwork you can think of solutions while implementing another problem in code. Since your team has only one computer at its disposal, you can only write code for one problem at a time. Obviously you cannot start working on the code for a particular problem before you figure out how to solve it.

For each problem you have already written down the time predictions. (This will be the input.) Now all you have to do is decide which problems to implement in code and which problems to ignore. The goal is of course to get the best possible final score. (A brief summary of the scoring procedure can be found on the next page.)

Input

The input starts with a line containing an integer T, the number of test cases. Then for each test case:

- One line with two integers D and N, denoting the contest duration (in minutes) and the number of problems. These satisfy $60 \leq D \leq 600$ and $1 \leq N \leq 15$.
- N lines with two integers S_i and C_i, where S_i denotes the moment in time when you think of the solution to problem i (measured in minutes after the start of the contest) and C_i denotes the time it takes to implement problem i in code (also in minutes). These satisfy $0 \leq S_i < D$ and $1 \leq C_i \leq D$.

Output

For each test case, output one line containing two space separated integers K and P, where K is the number of problems you solve before the end of the contest and P is the total penalty time. You should choose your strategy so as to maximise K and minimise P.

Sample input and output

Input	Output
3	2 170
300 2	3 100
0 100	2 100
50 10	
300 3	
0 10	
0 20	
0 30	
60 3	
10 30	
30 30	
40 20	

Scoring

Scoring in ICPC-style programming contests is done in the following way. The score for a team consists of two numbers:

- First of all, the number of problems solved.
- Secondly, the total *penalty time*. Whenever you solve a problem, the penalty time for that problem will be the number of minutes that have passed since the start of the contest, plus an additional 20 minutes for every wrong submission you made before solving the problem. The total penalty time is the sum of the penalty times for the individual problems. No penalty time is incurred for a problem that isn't solved by the end of the contest, even if you make various unsuccessful attempts.

Teams are ranked according to these numbers, first by increasing number of problems solved (more problems is better) and then by decreasing total penalty time (less penalty time is better). Thus, when optimising the final score, you must first make sure to solve as many problems as possible, and then solve them in an order such that the total penalty time is as small as possible.

In the first sample test case above, you must divide your computer time in the following way:

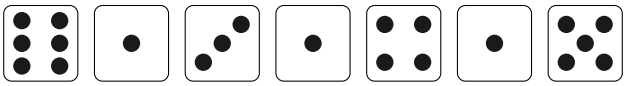
minutes	0 – 50	50 – 60	60 – 110	110 – 300
work on problem	1	2	1	(none)

Then the penalty time for problem 1 is 110 minutes, and for problem 2 it is 60 minutes. Thus, the total penalty time is 170 minutes, which is optimal. In this problem we never make wrong submissions, so we never incur the additional 20 minutes penalty time for wrong submissions. However, submissions might still be wrong in the contest you are currently participating in!

D Dice

The Centre for Concoction of Silly Stratagems (CCSS) has been playing a new game that has recently been rising in popularity among students: Dice Linkage Face-off (DLF). The game is played as follows: two players A and B are supplied with N identical dice, each of which has 6 sides denoting various integers from the interval $\{1, \ldots, 9\}$. Different sides may depict the same number.

The players then take turns placing a piece at the front of a line of dice. A line of dice is said to be *forbidden* if it has a subline consisting of two or more consecutive dice that splits into two smaller consecutive sublines with the same sum. For example, the line of dice in the example below is forbidden because it contains the subline (1, 3, 1, 4, 1) that splits into $1 + 3 + 1 = 4 + 1$.



Note that in particular no two consecutive dice may have the same value. The player to win the game is the last player to place a dice without making the line forbiddden. Players must therefore avoid forbidden configurations for as long as possible.

Daniel and Dominique are tired of playing the game, so they start analysing it. They wonder if it's possible for the game not to have a winner. Can you run out of dice before the line becomes forbidden? This turns out to be a difficult question to solve by hand, so they have asked for your help. Can you figure out how many lines consisting of N dice are allowed (that is, not forbidden)?

Input

The input starts with a line containing an integer T, the number of test cases. Then for each test case:

- One line with a single integer N, satisfying $1 \leq N \leq 10$, which denotes the number of dice.
- One line with six (not necessarily distinct) integers between 1 and 9, denoting the values on the sides of the dice.

Output

For each test case, output a single line containing an integer L, denoting the number of lines of N dice that are not forbidden.

Sample input and output

Input	Output
4	0
3 1 1 1 1 1 1	2
3 1 1 1 2 1 1	1334
5 1 2 3 4 5 6	451132
10 3 4 5 6 7 8	

Note that two lines are considered to be equal if they show the same values in the same order. If multiple sides have the same value, we don't distinguish between those sides. In the second example, there are two allowed lines: $(1, 2, 1)$ and $(2, 1, 2)$. Therefore the answer is 2, even though we might turn one of the dice showing 1 to a different side which also shows 1.

This is not a blank page

E European Energy Union

The European Union wants to be less dependent on Russia for its natural gas supply. Therefore a European Energy Union is emerging, with the goal of building gas pipes between all European cities so that gas can flow from any city to any other city. Some cities already have gas pipes between them, others don't. You have been hired to calculate the minimum number of gas pipes that need to be added so that gas can flow from any city to any other city, possibly via other cities. All gas pipes are bidirectional.

Input

The input starts with a line containing an integer T, the number of test cases. Then for each test case:

- One line with two space-separated integers C and P, denoting the number of cities and the number of existing pipes respectively. These satisfy $1 \leq C \leq 2\,000$ and $0 \leq P \leq 1\,000\,000$.
- P lines with two space-separated integers X and Y, denoting that there is a bidirectional gas pipe between cities X and Y. These satisfy $1 \leq X < Y \leq C$. No pair of cities will be listed more than once.

Output

For each test case, output the minimum number of bidirectional gas pipes that need to be added so that all cities are connected.

Sample input and output

Input	Output
4 3 1 1 3 3 3 1 2 1 3 2 3 4 0 8 6 1 2 3 4 5 6 7 8 1 7 2 8	1 0 3 2

This is not a blank page