Western Europe Regional Contest 🗝

's-Hertogenbosch, 13–14 November 1999



General Remarks

All languages

- ?? Input must be read from a file called 'x.in', where x stands for the letter of the problem you are solving. That is, input for problem A should be read from file 'a.in', while input for problem E should be read from file 'e.in'.
- ?? Output must be written to a file called 'x.out', where x stands for the letter of the problem you are solving. That is, output for problem A should be written to file 'a.out', while output for problem E should be written to file 'e.out'.
- ?? Submitted program files must have as name the single letter of the problem for which they are submitted. For extension naming see below.
- ?? Printing under DOS is not supported. Use for example notepad for printing under Windows.
- ?? If you encounter any print-errors, please notify the runners instead of submitting a clarification request.
- ?? Emacs and Win Vi have been installed, you will find shortcuts on your desktop.

Pascal submissions

- ?? Solutions submitted must have file extensions '.pas'. That is, submissions for problem E should be named 'e.pas'.
- ?? You can use the following command to compile from a DOS box: bpc.
- ?? Your problem will be compiled with the following compiler switches: -\$M65000, 0, 650000.

C submissions

- ?? Solutions submitted must have file extensions '. c'. That is, submissions for problem E should be named 'e.c'.
- ?? You can use the following command to compile from a DOS box: icc.
- ?? Your problem will be compiled with the following compiler switches: /Sa.

C++ submissions

- ?? Solutions submitted must have file extension '. cpp'. That is, submissions for problem E should be named 'e.cpp'.
- ?? You can use the following command to compile from a DOS box: icc.
- ?? Your problem will be compiled with no compiler switches.

Java submissions

- ?? Solutions submitted must have **no** file extension (!). That is, submissions for problem E should be named 'e'.
- ?? From a Java solution an object from the class 'solution' will be executed in order to verify the correctness of your submission. Therefore your submission must always contain this class.
- ?? You can use the following command to compile from a DOS box: javac.
- ?? You can execute your Java program with: java solution.
- ?? Your problem will be compiled with no compiler switches.

Contest Contents

	Problem Name	Pages
A.	Splitting Peas	3-4
B.	Love You, Disney World	5-6
C.	Chess x Tetris	7-8
D.	Dungeon of Doom	9-10
E.	Papergirl Paradise	11-12
F.	Basically Boolean	13-14
G.	Bits of Trinity	15-16
H.	Land Lords	17-18



Problem A Splitting Peas Input: a.in

Splitting peas are a remarkable phenomenon. Every so often they all jump up and split themselves in two. Now imagine a straight line of cups, each one containing a certain number of splitting peas. To the left of the line of cups, there is a wall (we will get back to it later). When it is splitting time, all the peas jump up out of their cups, and split themselves in midair. From each splitted pea, the two resulting peas land in the two neighboring cups (one in each cup).

An example:



In this configuration we have five cups, and one pea in the cent er cup. We will denote this configuration as (0,0,1,0,0). After one step (i.e. one splitting of the peas), we get the following configuration (0,1,0,1,0):



One step later the configuration is (1,0,2,0,1), we leave it up to you to draw this one if you like. In the next step some special things happen, since there are peas in the leftmost cup and in the rightmost cup. The peas in the leftmost cup jump up, split, and one pea will go to the second cup from the left (as usual), but the other pea will bounce against the wall and land in the leftmost cup again. The peas in the rightmost cup will also jump up, split, and one pea will go to the second cup from the right (as usual), but the other pea will go to the second cup from the right (as usual), but the other pea will also jump up, split, and one pea will go to the second cup from the right (as usual), but the other pea will fall off the right side of the table and disappear. So the configuration after the next step will be (1,3,0,3,0):



Another example is the configuration (1,2,3,4,5), which becomes (3,4,6,8,4) in one step. We say that the *predecessor* of (3,4,6,8,4) is (1,2,3,4,5). Most configurations have a predecessor (which is always unique), some configurations however, do not have a predecessor: these configurations are called *root configurations*. In the first example we gave, (0,0,1,0,0) is obviously a root configuration, and in our second example (1,2,3,4,5) is also a root configuration.

You are given a configuration that resulted after a certain number of steps from some root configuration. It is up to you to calculate the number of steps that took place, in going from the unknown root configuration to the given input configuration.

Input

The first line of the input contains the number of runs R(1 = R = 10,000). For each run there is one line containing the number of cups p (2 = p = 1,000), followed by one line containing p integers $n_i(0 = n_i = 1,000,000)$, representing the number of peak in cup i (from left to right).

Output

For each run, the output should consist of one line containing the number of steps that have been taken to reach the given input configuration from a root configuration.

Sample Input	Output for the Sample Input
4	3
5	1
1 3 0 3 0	4
5	10
3 4 6 8 4	
6	
4 1 6 0 4 0	
15	
10 45 1 120 0 210 0 252 0 210 0 120 0 44 0	

Western Europe Regional Contest 🗝

's-Hertogenbosch, 13-14 November 1999



Problem B Love You, Disney World Input: b.in

A lot of people work behind the scenes at Disney World, the world's largest theme park, located in Orlando, USA. During the day all these people have to work fulltime. They are allowed to take only one break per day, but only when absolutely no attractions or rides are active in the part of the park they are assigned to.

Two of the people, Donald and Daisy, are madly in love with each other. Fortunately they work in the same area of the park, so they are able to take their breaks at the same time. They want to spend as much time together as possible. So at the beginning of each day they look at the schedule for the rides for that day and figure out when they can have the longest break.

You are to write a program that reads in the scheduled start and end times of all the rides, and tell Donald and Daisy when to take their break in order for them to be able to spend as much time together as possible. Note that management forbids taking a break before the first and after the last ride of the day.

Input

The first line of the input contains the number of runs R(1 = R = 1,000). For each run there is one line with a single integer n, 2 ? n ? 1,000. This number denotes the number of rides during the day. The next *n* lines each contain 2 numbers s_n and e_n (0 ? $s_n < e_n$? 10,000) denoting the start and end time of ride *n*. These times are inclusive, i.e. a brake can only start in the time slot *after* a ride has ended and must end in the time slot *before* the next ride starts. Note that the scheduled times may overlap (they may be different attractions).

Output

For each test case, you must output a line containing the start time b_s and the end time e_s of the longest break Donald and Daisy will be able take. Make certain that for each time $t(b_s ? t ? b_e)$ no ride is active. If there is more than one solution, output the earliest one (with the smallest b_s). If there is no brake possibility for Donald and Daisy, output a single number 0.

Sample Input	Output for the Sample Input
2	8 11
5	0
13 18	
22 40	
3 7	
1 5	
12 17	
2	
7 12	
13 20	

(This page intentionally left blank)



One of the members of the Jury is both a good chess player as well as a fanatical computer game player. As he also works professionally for a computer game company, he is constantly thinking up new combinations of games to market (that's his third hobby: earning money). One day he thought up the following combination of Chess and Tetris. The player is presented with a Chessboard (8x8 fields, the coloring is irrelevant for this game) and a set of Tetris blocks, i.e. 0 or more of each of the following 5 blocks:



From left to right these Tetris blocks are called an O-block, a Z -block, a T -block, an I-block and an L-block. The task in the game is to find a tiling of the complete Chessboard with a given set of Tetris blocks within a certain amount of time. To do so the player must select a block, rotate and/or flip it multiple times and then position it somewhere on the Chessboard, until all the Tetris pieces are placed on the Chessboard and the Chessboard is completely filled up. Note that flipping a Tetris block is the same as mirroring the Tetris block in either the vertical or the horizontal axis.

He asked a programmer to come up with a first draft version, and also with a list of input sets that could be used on different levels. The programmer had to make sure that each of the input sets the program would use has a least one solution for the tiling problem.

Unfortunately, the programmer has not yet had enough time to come up with a sufficiently long list of input sets. Therefore our member of the Jury has decided to ask you to help him. He has generated a random list of input sets, and you are asked to write a program that determines if a given input set can be used to completely tile the Chessboard.

Input

The first line of the input contains the number of runs R(1 = R = 100). For each run there is one line with 5 integers b_i (0 ? b_i ?16). Each integer gives the number of a specific type of Tetris block that are in the input set to be tested, in the order as they appear in the picture above: number of O-blocks, number of Z-blocks, number of T-blocks, number of I-blocks and number of L-blocks.

The sum of the number of blocks in an input set is always 16.

Output

For each test case, you must output a line containing the string 'Tiling possible.' when it is possible to find a complete tiling of the Chessboard with the given input set. If it is not possible to completely tile the Chessboard using the input set, you must output the string 'Tiling not possible.'.

Sample Input	Output for the Sample Input
4	Tiling possible.
16 0 0 0 0	Tiling not possible.
4 4 4 4 0	Tiling not possible.
0 16 0 0 0	Tiling possible.
4 3 2 4 3	

Western Europe Regional Contest 💀

's-Hertogenbosch, 13-14 November 1999



Problem D Dungeon of Doom Input: d.in

The Dungeon of Doom is a game, which is loosely based on the legend of the Minotaur. This legend describes how, in the ancient city of Thebe, there was once a great maze, and in this maze lived a big monster: the Minotaur. Every year maids were sacrificed to the Minotaur: they were led to some point in the maze and released there, and had to find their way back out of the maze. Most of the time the Minotaur would find the poor girls before they found the exit, and the girl was never seen outside of the maze again.

In this problem you're a given a maze with not just one Minotaur, but possibly many more than one Monster, and you have to find a way out of the maze from a given starting point. But, you have a number of advantages over the people in the legend:

- You know the layout of the maze, as it will be given to you as input;
- You know where the exit is located (it's also marked in the input);
- At every turn you have 5 choices: stand still, take one step north, take one step west, take one step south or take on step east. Of course, you can only take a step if there is nothing blocking you in that direction, i.e. there is no wall or monster directly adjacent to you in the direction in which you want to move.
- You alternate your moves with the moves of all the Monsters (they all move at the same time), and you get to begin with the first move;
- Although there are many more Monsters, they are all of the exact same type, which is the 'not very bright'-type. Each Monster has a fixed set of rules that completely describes how it thinks and reacts, based on its position in the maze and the direction it is facing. It is actually pretty simple, for each move a Monster does only the following:
 - 1. If the Monster sees you, it goes into an absolute feeding frenzy, and runs towards you. If this ever happens you are utterly lost and will die. However, the Monster only sees you if you are standing in its exact line of sight in the direction it is looking: any deviation of the direct line of sight will take you out of harms way. Also, the Monster can not see you if you are standing behind another Monster, or behind a wall.
 - 2. If the Monster does not see you, it will try to take a step in the direction it is facing. However, if it sees another Monster in the direction it is looking in, it will not move in that direction, and neither will it do so if there is a wall directly in front of it. If neither of these two conditions is met, the Monsters takes one step forward and its move ends. If it does not take a step, the Monster always turns 90 degrees clock-wise (to the right), and its move then also ends.

All the Monsters take their move in parallel: they all first look, and then decide what to do. This means that the decision of what a monster does is only based on the start position of the other Monsters at the beginning of the move, and more than one Monster can end up in the same position at the end of a turn!

Note that from these rules it follows that you are much more agile than your opponents, since you can turn while running (e.g. when facing north, you can immediately take a step east or even south). A Monster will always lose a move when it turns to face another direction.

You are to determine the shortest time it will take for the player to reach the exit from a given start position of the player and start positions of all the Monsters in the maze. Stepping onto the exit is enough to win the game (but this final step also counts as a step). So, if you start next to the exit, you will always win in 1 step, since you get to make the first move.

Input

The first line of the input contains the number of runs R(1 = R = 50). For each run there is one line with 2 integers h (3 ? h ? 25) and w (4 ? w ? 80), giving the height and width of the maze that follows. Then exactly h lines will follow, where each line will contain exactly w characters. Each of these characters can only be one of the following:

- + These are used to denote walls in the maze (there is no distinction between these walls, it just makes for a more pleasing maze to have different characters for the walls);
- This is used to denote your start position;
- X This is used to denote the exit;
- M This is used to denote a monster;
- . This is used to denote an open space in the maze.

There will always be exactly one @ and exactly one X in the maze. There will also be at least one M, and no more than 100 M's in the input. All the monsters in the maze start facing east (that is to the right).

The surrounding of the maze will always be completely walls (i.e. any character on the first and last line, and the first and last character of all the lines in between will be a wall type character). Note that 'rooms' (large open areas in the maze) may exist!

Output

For each test case, you must output a line containing the minimal number of steps it takes to reach the exit. If you cannot reach the exit, or cannot reach it within 10,000 steps, you must output the number 0.

Sample Input	Output for the Sample Input
5	23
10 10	5
+++	0
++	12
	1
+- + . M	
$ \ldots @ \ldots $	
· · · + · · ·	
-+	
$ \ldots X. $	
++	
5 5	
++	
X	
M	
@MM	
++	
5 5	
++	
@FM4 ++	
++	
M.X	
М	
@MM	
++	
5 5	
++	
M@X	
MMM	
MMM	
++	

Western Europe Regional Contest 🔤

's-Hertogenbosch, 13–14 November 1999



Problem E Papergirl Paradise Input: e.in

The life of a papergirl is not always easy. Suburbs often have a maze of little streets, and subscribers to the newspaper may live quite far apart. Compared to that, a papergirl who makes her round in the city center of Skyscraper Sity has a far simpler task. There are often enough subscribers in one single skyscraper from which to make a living, so the round of a papergirl rarely consists of more than one flat. A papergirl just starts at the ground floor of the flat and works her way up to the top, delivering the newspaper on each floor as she goes along. The only thing she has to take care of is that each subscriber gets her/his newspaper before 8:00 in the morning.

Jill is one of the papergirls in Skyscraper Sity, and she is given a number of skyscrapers to choose from for her round. Since she wants to earn money, but doesn't want to get up too early in the morning, she tries to figure out which skyscraper takes the least amount of time in which to deliver the newspapers. For each skyscraper she knows the layout of the skyscraper, including the position of the entrance on the ground floor and where each of the subscribers is located. She knows that in each flat the stairs are always at the left end and the right end of the flat, and they always go up from the ground floor all the way to the top. She has set one ground rule for herself: she doesn't want to take more stairs than strictly necessary with all those newspapers on her back, so she will always deliver all the newspapers on a floor before moving on to the next floor.

You are asked by Jill to help her determine the time it would take for her to deliver all the newspapers in a given skyscraper.

Input

The first line of the input contains the number of skyscrapers S (1 = S = 100). For each skyscraper there follows one line with 2 integers: f (1 ? f ?30) and w (4 ? w ?80), giving the number of floors and the width of each floor in the skyscraper. Then exactly f+1 lines will follow giving the layout of the skyscraper, where every line will contain exactly w characters.

The first line is the roof of the skyscraper, which is a '+', followed by w-2 '-' characters and a '+' again. Each of the following lines starts and ends with a '%' (representing the stairs at the left and right side of the flat), and in between are by w-2 characters from the following set:

- * The apartment of a subscriber to the newspaper
- . The apartment of someone who is not a subscriber to the newspaper, and therefor irrelevant for you.

On the ground floor (that is the last floor in the input of a skyscraper), there is also exactly one '@' character between the stair symbols, which denotes the entrance to the skyscraper. The number of subscribers on a floor is only limited by the width of the floor. There will always be at least one subscriber on the top floor.

A newspaper is delivered by moving past a subscriber apartment. Moving up a stair takes the same amount of time as moving one step in a horizontal direction.

Take a look at the next example input, with 5 floors (f=5) and 10 apartments per floor (w=12):

5 12	
++	
8**.	Fourth floor = top floor
^{ol} ^{ol}	Third floor
8*******8	Second floor
8*.*.***.*.8	First floor
8@.**8	Ground floor

[Note: the names of the floors are not in the real input!]

In this example the entrance is at position 3 on the ground floor. There are 2 subscribers on the ground floor (at locations 5 and 9), 6 at the first floor, 8 at the second floor but none at the third floor, while there are 3 subscribers at the top floor.

Output

For each test case, you must output a line containing the least number of steps it takes to deliver the last newspaper.

Sample Input	Output for the Sample Input
2	40
5 12	93
++	
8**8	
88	
8******	
8*.*.***.*.8	
8@.**8	
10 10	
++	
8**8	
8***8	
8.***8	
8.***.8	
8*	
8**.*8	
Š ^{***} Š	
<u> </u>	
5 [*] [*] 5	
٥٣٠٠٠٠٠	

Western Europe Regional Contest 🔤

's-Hertogenbosch, 13–14 November 1999



Problem F Basically Boolean Input: f.in

Boolean expressions of more than two or three variables can become very complex. Sometimes they become so complex, it is no longer simple to see how the outcome of the expression depends on the value of a single variable. It may even be so that the outcome of an expression no longer depends on the value of one of the variables in it, i.e. that the outcome is the same whether the variable is true or false. If an expression *E* no longer depends on a variable *X*, we say that *X* is *not important* for the expression, and conversely, if a variable *X* is important for an expression *E*, then the outcome of *E* will depend on the value of *X*. If none of the variables in an expression are important, then the expression is either a tautology (always true), or a contradiction (always false).

You are asked to write a program that will, given an expression E, determine all the important variables in the expression.

Input

The first line of the input contains the number of runs R(1 = R = 1,000). Then R lines appear, with each line containing one formula. Each formula consists of one or more of the following elements:

- A variable, denoted by a capital letter A through J (each variable may occur multiple times, or may not occur at all)
- The unary operator not
- The binary operators **or**, **and**, or => (*implies*)
- Brackets: (and), always matching

A formal description of an input line is given by the following grammar (where 'space' denotes exactly one space, and FORMULA is the start symbol, as shown by the underscoring):

<u>FORMULA</u> =	VARIABLE
	UNARY_OP space FORMULA
	(FORMULA space BINARY_OP space FORMULA)
VARIABLE =	A B C D E F G H I J
$UNARY_OP =$	not
BINARY_OP =	or and =>

Note: the **not** operator binds stronger than the binary operators, and between the binary operators no precedence needs to be defined as every formula is fully bracketed. The Boolean operators are to be interpreted as usual:

x or y is true, if and only if x is true, or y is true, or both are true; x and y is true, if and only if x is true and y is true;

 $x \Rightarrow y$ is true, if and only if x is false or y is true.

Output

For each test case, you must output a line with the variables from the input expression that are important for that expression, preceded by the string 'Important variables: '. Multiple variables are to be separated by single spaces. There must also be exactly one space between the ':' and the first variable. If no variables are important for the evaluation of the formula, you must print the string 'No variables are important.'

Sample Input

4 H (not A and not not A) (((B and C) or (D and B)) => (A or B)) ((not (not E or (B or C)) or G) and not (((not E or B) or C) and G))

Output for the Sample Input

Important variables: H No variables are important. No variables are important. Important variables: B C E

Western Europe Regional Contest 🗝

's-Hertogenbosch, 13–14 November 1999



TRM event

Problem G Bits of Trinity Input: g.in

In the Trinity Galaxy, spaceships use an elaborate method of communication to receive messages from the Home World. Scientists have created a numbered list of 1,000,000 different Messages. In order to send out a message the sender first looks up the index number of the selected message. This number is then encoded into a binary bit stream of exactly 200 bits and sent into space with a highly focussed solar impulse laser. Because the energy requirements for sending out a pulse (i.e. a 1-bit) are very high the president of the energy committee has decided that a bit stream may contain at most 3 ones. All the allowed bit streams are ordered in a large ascending list, where the ordering is based on the normal binary representation of the number. The allowed bit streams are numbered consecutively starting with the bit stream consisting of all 0's, which has index 1. You see the start of the big list in the box below (note that the dots represent a leading string of 0's).

Recently, the Admiralty has decided that the risk of carrying a list like this around on all the spaceships is too large it might fall into the hands of the wrong people, and they could decipher all the messages between the space ships and the Home World. Therefore they request a computer program that will do the conversion from index number to corresponding Allowed Bit Stream.

You are asked to write this program and help to protect the spaceships and the Home World.

Input

The first line of the input contains the number of runs R(1 = R = 10,000). This line is followed by exactly one line for each of the runs. Each line will contain one number, the index number i(1 = i = 1,000,000) for which you must find the corresponding Allowed Bit Stream.

Output

For each test case, you must output a line containing the corresponding Allowed Bit Stream, without leading 0's. You will always print at least 1 bit, i.e. in the special case where the index number is 1, you must print a single 0.

The List Of Allowed Solar Impulse Laser Bit Streams (TOP SECRET)	
bit stream index 000000 1 000001 2 000011 4 000100 5 000101 6 000110 7 000111 8 001000 9 001001 10 001011 12 001001 11 001101 14 001101 14 001101 15 010000 16 010010 17 010010 18 010010 18 010011 19 010101 21 010101 21	
011000 23 011001 24	

Sample Input	Output for the Sample Input
6	0
1	1
2	10
3	11
4	100
5	100000110
100	

Western Europe Regional Contest 💀

's-Hertogenbosch, 13–14 November 1999



Problem H Land Lords Input: h.in

In the history of the world many stories are known of the arguments of large families about the distribution of the lands of the parents after their deaths. In some societies, the eldest living son would always receive the entire estate, making the position of eldest living son sometimes a dangerous one. In other societies, the estate would be divided up into equal parts, and each child would receive one part of the estate. This meant that a sizeable estate would be reduced to a large collection of small patches of land within a few generations.

In the land of Dukeswood a very rich farmer lived with a number of children, and he wanted to distribute his land over his children. He had a number of maps of his farmstead, which were fairly simple to make since his farmstead was a rectangular sized piece of land, and on each such map he drew a number of lines, thereby dividing the land into pieces. When drawing lines, he would always start on one of the boundaries of his land and draw a line to one of the other boundaries. The end point of this line was then the starting point of the next line, which would again lead to another boundary. The farmer would never draw a line that would a cross another line at a position where this other line was already crossing a different line (i.e. there would never be more than two lines crossing at any single point at once). One of the maps he drew is shown below:



The start point for this division can be either of the two points marked with a star. After he made a number of maps like this, the farmer counted the number of pieces of land in the map, to see if he had created enough pieces of land for each of his children. For instance, the map above contained 18 pieces of land, as the numbering clearly shows. However, the farmer would often lose count when he counted the pieces of land on a map, and that is where your help is requested.

You are asked to give the number of pieces of land on a map with a rectangular size and a number of lines drawn on it.

Input

The first line of the input contains the number of maps M (1 = M = 1,000). The description of one map starts with a line with two integers, the width w (1 = w = 1,000) and the height h (1 = h = 1,000) of the farmers land. The next line contains the number of lines L (1 = L = 1,000) the farmer has drawn on the map. Finally there are L+1 lines, each containing a co-ordinate (X_i , Y_i) on a boundary of the land (i.e. either X_i is 0 or w, or Y_i is 0 or h). The lines on the map are then formed by the points (X_i , Y_i) and (X_{i+1} , Y_{i+1}), for all i from 1 up to L (inclusive).

Output

For each test case, you must output a line containing the number of separate pieces of land on the map.

Sample Input	Output for the Sample Input
2	18
18 12	13
8	
2 0	
6 12	
10 0	
18 9	
15 12	
0 6	
14 0	
10 12	
0 9	
7 6	
б	
2 0	
5 6	
7 3	
0 3	
3 0	
3 6	
0 5	