Team Practice 1

14 October 2012: 1:00 – 6:00 PM

Contest Problem Set

The ten problems on this contest are referred to, in order, by the following names:

stones, birdtree, money, duke1, dull, maze, howbig, pizza, duke2, antimono

All problem submissions will be done via the PC^2 system. Any questions about the problems should be addressed through the clarification system. Be sure that you are submitting your code under the correct problem name!

No matter what the individual problem statements say, your program should read its input from standard in and print its output to standard out.

You may print to the printer at any point during the contest.

Printed code libraries and language references are permitted; all other aids, including other internet resources and any code you may have typed before the contest, are not.

Good Luck!

A Game of Stones

A game is played with a single pile of stones. Two players, Alice and Bob, start with N stones in the pile and a rational number $c \geq 1$. Alice goes first, and can take any positive integer k_1 number of stones from the pile so long as she doesn't take the whole pile; that is, $1 \leq k_1 < N$. Subsequently, the players take turns, on turn t taking an integer k_t number of stones from the pile, where k_t satisfies $1 \leq k_t \leq c \cdot k_{t-1}$. That is, the current player takes up to c times the number of stones just taken by the *other* player in his or her most recent turn. The winner is the player to take the last stone.

Given an integer $2 \leq N \leq 30000$ and rational number $c \geq 1$ with numerator and denominator both at most 1000, find out who wins, assuming perfect play from both Alice and Bob.

Input: The first line contains the number of test cases, $P \leq 40$. Each of P subsequent lines contains three integers, N, A, and B, the number of stones and the numerator and denominator of c, respectively. These numbers satisfy $2 \leq N \leq 30000$ and $1 \leq B \leq A \leq 1000$.

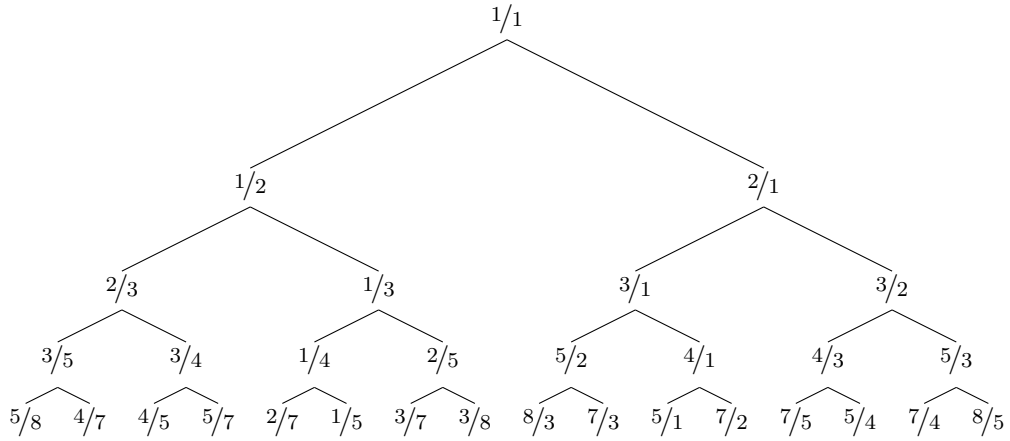
Output: For each test case, a output a single line containing either "Alice Wins" or "Bob Wins", depending on whether Alice or Bob will win with perfect play from both players.

```
Sample Input:
2
10 1 1
8 2 1

Output for Sample Input:
Alice Wins
Bob Wins
```

B Bird tree

The Bird tree[1] is an infinite binary tree, whose first 5 levels look as follows:



It can be defined as follows:

$$bird = \begin{array}{ccc} & 1/1 & \\ \swarrow & & \searrow \\ 1/(bird+1) & & (1/bird)+1 \end{array}$$

This is a *co-recursive* definition in which both occurrences of $bird$ refer to the full (infinite) tree. The expression $bird + 1$ means that 1 is added to every fraction in the tree, and $1/bird$ means that every fraction in the tree is inverted (so a/b becomes b/a).

Surprisingly, the tree contains every positive rational number exactly once, so every reduced fraction is at a unique place in the tree. Hence, we can also describe a rational number by giving directions (`L` for left subtree, `R` for right subtree) in the Bird tree. For example, $2/5$ is represented by `LRR`. Given a reduced fraction, return a string consisting of `L`'s and `R`'s: the directions to locate this fraction from the top of the tree.

Input

On the first line a positive integer: the number of test cases, at most 100. After that per test case:

- one line with two integers a and b ($1 \leq a, b \leq 10^9$), separated by a `'/'`. These represent the numerator and denominator of a reduced fraction. The integers a and b are not both equal to 1, and they satisfy $\gcd(a, b) = 1$.

For every test case the length of the string with directions will be at most $10\,000$.

Output

Per test case:

- one line with the string representation of the location of this fraction in the Bird tree.

[1] Hinze, R. (2009). The Bird tree. *J. Funct. Program.*, 19:491–508.

Sample in- and output

Input	Output
3 1/2 2/5 7/3	L LRR RLLR

Problem B

Money Matters

Our sad tale begins with a tight clique of friends. Together they went on a trip to the picturesque country of Molvania. During their stay, various events which are too horrible to mention occurred. The net result was that the last evening of the trip ended with a momentous exchange of "I never want to see you again!"s. A quick calculation tells you it may have been said almost 50 million times!

Back home in Scandinavia, our group of ex-friends realize that they haven't split the costs incurred during the trip evenly. Some people may be out several thousand crowns. Settling the debts turns out to be a bit more problematic than it ought to be, as many in the group no longer wish to speak to one another, and even less to give each other money.

Naturally, you want to help out, so you ask each person to tell you how much money she owes or is owed, and whom she is still friends with. Given this information, you're sure you can figure out if it's possible for everyone to get even, and with money only being given between persons who are still friends.

Input specifications

The first line contains two integers, n ($2 \leq n \leq 10000$), and m ($0 \leq m \leq 50000$), the number of friends and the number of remaining friendships. Then n lines follow, each containing an integer o ($-10000 \leq o \leq 10000$) indicating how much each person owes (or is owed if $o < 0$). The sum of these values is zero. After this comes m lines giving the remaining friendships, each line containing two integers x, y ($0 \leq x < y \leq n - 1$) indicating that persons x and y are still friends.

Output specifications

Your output should consist of a single line saying "POSSIBLE" or "IMPOSSIBLE".

Sample input 1	**Sample output 1**
5 3 100 -75 -25 -42 42 0 1 1 2 3 4	POSSIBLE

Sample input 2	Sample output 2
4 2 15 20 -10 -25 0 2 1 3	IMPOSSIBLE

Duke of York, Part 1

Oh, the grand old duke of York
He had ten thousand men
He marched them up to the top of the hill
And he marched them down again.

The duke of York has got his men to the top of the hill, and now he needs to get them down again! He would like to know how many ways there are for a man to march down the hill without using up too much energy. He has a map of the hill, but he is not so good at programming, so he has enlisted your help.

The hill can be represented as a rectangular grid of squares with N rows and M columns, $2 \leq N, M \leq 50$. All of the squares in the first row are at the top of the hill, and all of the squares in the Nth row are at the bottom. Every square s on the hill has an associated base energy cost b, which is an integer between 0 and 9 inclusive, or it is impassable. The total energy cost to walk onto the square s is the base energy cost b of s plus the direction cost, which is 0 for going downhill, 1 for going left or right, and 3 for going uphill. One of the duke's men has an total starting energy of E, where $1 \leq E \leq 200$. Making any move subtracts its total energy cost from a man's remaining energy. If a man doesn't have at least as much energy as it takes to move into a square, he can't do it, nor can he move if he has 0 energy left even if his move would cost 0 energy. (The duke is okay with a man having exactly 0 energy on reaching the bottom of the hill, though.)

A path down the hill is a sequence of moves (up, down, left, or right), starting from one of the squares at the top of the hill and ending at one of the squares at the bottom of the hill. Since the duke likes marching his men around, a path is permitted (but not required) to continue on the hill in any direction after reaching the bottom of the hill, provided that it eventually ends at the bottom. A path cannot start on, travel through, or end on an impassable square.

You are required to compute the number of different paths down the hill that can be travelled given a starting energy of E. Since this number may be very large, you should output the answer modulo 1000000007 (that's $10^9 + 7$).

Input: The first line of input will contain an integer $1 \leq C \leq 20$, the number of test cases. The first line of each case will contain three integers: N, M, and E, with $2 \leq N, M \leq 50$, and $1 \leq E \leq 200$. The next N lines will contain a map of the hill, in the following format. Each square on the hill is represented by a single character: either one of the numbers 0-9, if the square is passable and has the corresponding base energy cost, or the letter X, if the square is impassable. There will be no whitespace in the map except for the newlines at the end of each line of M characters.

Output: For each test case, output a single line containing one integer: the number of paths down the hill that can be travelled given a starting energy E, modulo 1000000007.

```
Sample Input:
2
2 2 2
00
00
3 2 2
02
X0
00
```

```
Output for Sample Input:
10
3
```

Explanation of Output: In the first case, the paths are (starting from upper left): D, DR, DRL, RD, RDL. RLD is not possible because after the first two moves the man would have zero energy left and be unable to move. The other five paths are the mirror images of these.

In the second case, the paths are (starting from the upper right) DDD, DDDL, DDDLR. No path starting from the upper left is possible. Beginning on the upper right square does not cost any energy.

Problem C: DuLL

Source file: `dull.{c, cpp, java}`
Input file: `dull.in`

In Windows, a DLL (or dynamic link library) is a file that contains a collection of pre-compiled functions that can be loaded into a program at runtime. The two primary benefits of DLLs are (1) only one copy of a DLL is needed in memory, regardless of how many different programs are using it at the same time, and (2) since they are separate from programs, DLLs can be upgraded independently, without having to recompile the programs that use them. (DLLs have their problems, too, but we'll ignore those for now.) Your job is to calculate the maximum memory usage when running a series of programs together with the DLLs they need.

The DLLs in our system are not very exciting. These dull DLLs (or DuLLs) each require a fixed amount of memory which never changes as long as the DuLL is in memory. Similarly, each program has its own fixed memory requirements which never change as long as the program is executing. Each program also requires certain DuLLs to be in memory the entire time the program is executing. Therefore, the only time the amount of memory required changes is when a new program is executed, or a currently running program exits. When a new program begins execution, all DuLLs required by that program that must be loaded into memory if they are not there already. When a currently running program exits, all DuLLs that are no longer needed by any currently running programs are removed from memory.

Remember, there will never be more than one copy of a specific DuLL in memory at any given time. However, it is possible for multiple instances of the same program to be running at the same time. In this case each instance of the program would require its own memory; however, the instances still share DuLLs in the same way two unrelated programs would.

Input: The input consists of at least one data set, followed by a line containing only 0.

The first line of a data set contains three space separated integers *N P S*, where *N* is the number of DuLLs available, $1 \le N \le 20$, *P* is the number of programs which can be executed, $1 \le P \le 9$, and *S* is the number of state transitions recorded, $1 \le S \le 32$.

The next line contains exactly *N* space separated integers representing the sizes in bytes of each of the DuLLs, $1 \le size \le 1{,}000$. Each DuLL is implicitly labeled with a letter: 'A', 'B', 'C', …, possibly extending to 'T'. Therefore the first integer is the size of 'A', the second integer is the size of 'B', and so on.

The next *P* lines contain information about each of the programs, one program per line. Each line contains a single integer representing the size of the program in bytes, $1 \le size \le 1{,}000$, followed by 1 to *N* characters representing the DuLLs required by that program. There will be a single space between the size of the program and the DuLL labels, but no spaces between the labels themselves. The order of the labels is insignificant and therefore undefined, but they will all be valid DuLL labels, and no label will occur more than once. Each program is implicitly labeled with an integer: 1, 2, 3, … possibly extending to 9.

The final line of the data set will contain *S* space separated integers. Each integer will either be a positive number *q*, $1 \le q \le P$, indicating that a new execution of program *q* has begun, or else it will be a negative number $-q$, $1 \le q \le P$, indicating that a single execution of program *q* has completed. The transitions are given in the order they occurred. Each is a valid program number; if it is a negative number $-q$ then there will always be at least one instance of program *q* running.

Output: There is one line of output for each data set, containing only the maximum amount of memory

required throughout the execution of the data set.

Example input:	**Example output:**
2 2 3 500 600 100 A 200 B 2 1 2 5 4 8 100 400 200 500 300 250 AC 360 ACE 120 AB 40 DE 2 3 4 -3 1 2 -2 1 0	1600 2110

Last modified on October 18, 2009 at 8:36 AM.

Problem G: Line & Circle Maze

Source file: `maze.{c, cpp, java}`
Input file: `maze.in`

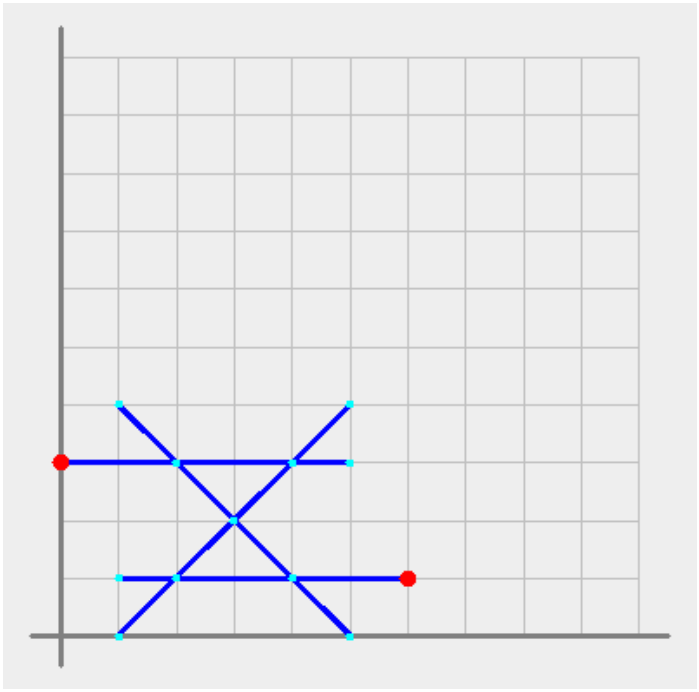
A deranged algorithms professor has devised a terrible final exam: he throws his students into a strange maze formed entirely of linear and circular paths, with line segment endpoints and object intersections forming the junctions of the maze. The professor gives his students a map of the maze and a fixed amount of time to find the exit before he floods the maze with xerobiton particles, causing anyone still in the maze to be immediately inverted at the quantum level. Students who escape pass the course; those who don't are trapped forever in a parallel universe where the grass is blue and the sky is green.

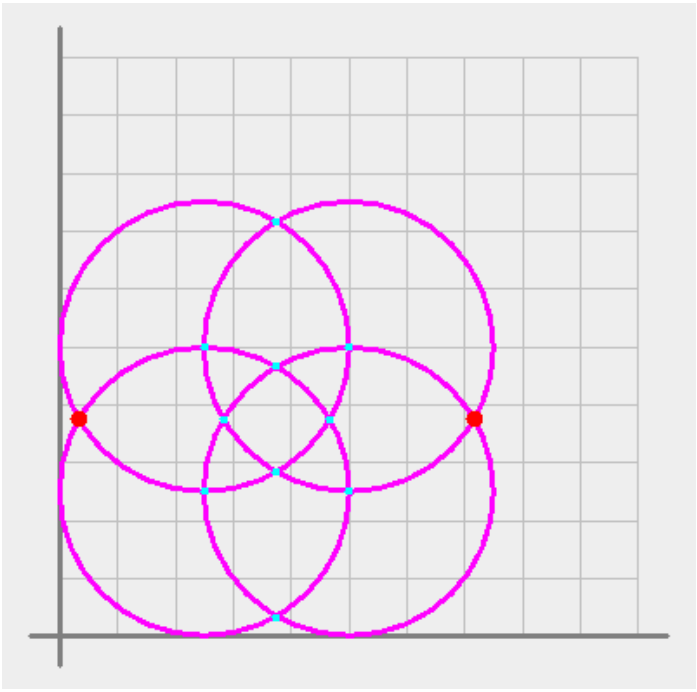
The entrance and the exit are always at a junction as defined above. Knowing that clever ACM programming students will always follow the shortest possible path between two junctions, he chooses the entrance and exit junctions so that the distance that they have to travel is as far as possible. That is, he examines all pairs of junctions that have a path between them, and selects a pair of junctions whose shortest path distance is the longest possible for the maze (which he rebuilds every semester, of course, as the motivation to cheat on this exam is very high).

The joy he derives from quantumly inverting the majority of his students is marred by the tedium of computing the length of the longest of the shortest paths (he needs this to know to decide how much time to put on the clock), so he wants you to write a program to do it for him. He already has a program that generates the mazes, essentially just a random collection of line segments and circles. Your job is to take that collection of line segments and circles, determine the shortest paths between all the distinct pairs of junctions, and report the length of the longest one.

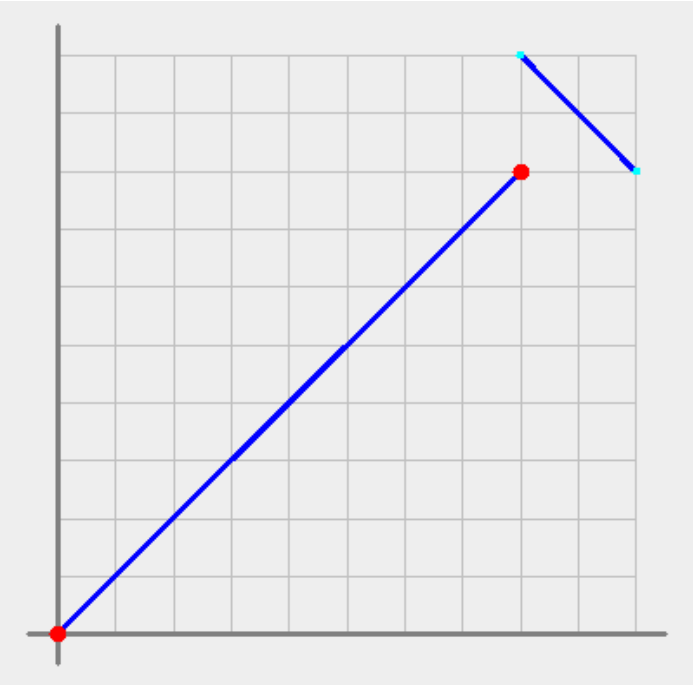
The input to your program is the output of the program that generates his mazes. That program was written by another student, much like yourself, and it meets a few of the professor's specifications: 1) No endpoint of a line segment will lie on a circle; 2) No line segment will intersect a circle at a tangent; 3) If two circles intersect, they intersect at exactly two distinct points; 4) Every maze contains at least two junctions; that is, a minimum maze is either a single line segment, or two circles that intersect. There is, however, one bug in the program. (He would like to have it fixed, but unfortunately the student who wrote the code never gave him the source, and is now forever trapped in a parallel universe.) That bug is that the maze is not always entirely connected. There might be line segments or circles, or both, off by themselves that intersect nothing, or even little "submazes" composed of intersecting line segments and circles that as a whole are not connected to the rest of the maze. The professor insists that your solution account for this! The length that you report must be for a path between connected junctions!

Examples:



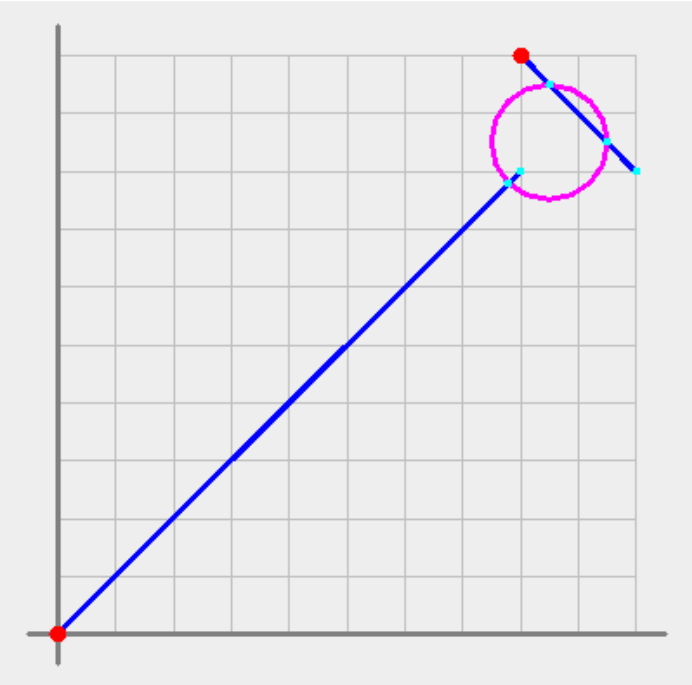


Line segments only. The large dots are the junction pair whose shortest path is the longest possible.



Disconnected components.

An example using circles only. Note that in this case there is also another pair of junctions with the same length longest possible shortest path.



Now the line segments are connected by a circle, allowing for a longer shortest path.

Input: An input test case is a collection of line segments and circles. A line segment is specified as "L X_1 Y_1 X_2 Y_2" where "L" is a literal character, and (X_1,Y_1) and (X_2,Y_2) are the line segment endpoints. A circle is specified by "C X Y R" where "C" is a literal character, (X,Y) is the center of the circle, and R is its radius. All input values are integers, and line segment and circle objects are entirely contained in the first quadrant within the box defined by (0,0) at the lower left and (100,100) at the upper right. Each test case will consist of from 1 to 20 objects, terminated by a line containing only a single asterisk. Following the final test case, a line containing only a single asterisk marks the end of the input.

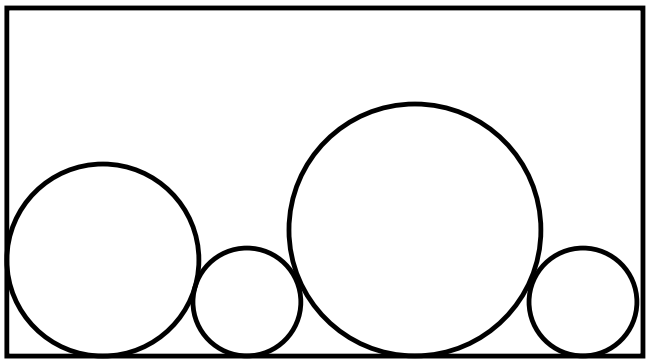
Output: For each input maze, output "Case N: ", where N is the input case number starting at one (1), followed by the length, rounded to one decimal, of the longest possible shortest path between a pair of connected junctions.

Example Input:	**Example Output:**
L 10 0 50 40 L 10 40 50 0 L 10 10 60 10 L 0 30 50 30 * C 25 25 25 C 50 25 25 C 25 50 25 C 50 50 25 * L 0 0 80 80 L 80 100 100 80 * L 0 0 80 80 L 80 100 100 80 C 85 85 10 * *	Case 1: 68.3 Case 2: 78.5 Case 3: 113.1 Case 4: 140.8

Problem C: How Big Is It?

Ian's going to California, and he has to pack his things, including his collection of circles. Given a set of circles, your program must find the smallest rectangular box in which they fit.

All circles must touch the bottom of the box. The figure below shows an acceptable packing for a set of circles (although this may not be the optimal packing for these particular circles). Note that in an ideal packing, each circle should touch at least one other circle (but you probably figured that out).



Input

The first line of input contains a single positive decimal integer n, $n < 50$. This indicates the number of lines which follow. The subsequent n lines each contain a series of numbers separated by spaces. The first number on each of these lines is a positive integer m, $m < 8$, which indicates how many other numbers appear on that line. The next m numbers on the line are the radii of the circles which must be packed in a single box. These numbers need not be integers.

Output

For each data line of input, excluding the first line of input containing n, your program must output the size of the smallest rectangle which can pack the circles. Each case should be output on a separate line by itself, with three places after the decimal point. Do not output leading zeroes unless the number is less than 1, e.g. `0.543`.

Sample Input

```
3
3 2.0 1.0 2.0
4 2.0 2.0 2.0 2.0
3 2.0 1.0 4.0
```

Sample Output

```
9.657
16.000
12.657
```

Problem C: Pizza Pricing

Source file: `pizza.{c, cpp, java}`
Input file: `pizza.in`

Pizza has always been a staple on college campuses. After the downturn in the economy, it is more important than ever to get the best deal, namely the lowest cost per square inch. Consider, for example, the following menu for a store selling circular pizzas of varying diameter and price:

Menu	
Diameter	**Price**
5 inch	$2
10 inch	$6
12 inch	$8

One could actually compute the costs per square inch, which would be approximately 10.2¢, 7.6¢, and 7.1¢ respectively, so the 12-inch pizza is the best value. However, if the 10-inch had been sold for $5, it would have been the best value, at approximately 6.4¢ per square inch.

Your task is to analyze a menu and to report the *diameter* of the pizza that is the best value. Note that no two pizzas on a menu will have the same diameter or the same inherent cost per square inch.

Input: The input contains a series of one or more menus. Each menu starts with the number of options N, $1 \le N \le 10$, followed by N lines, each containing two integers respectively designating a pizza's diameter D (in inches) and price P (in dollars), with $1 \le D \le 36$ and $1 \le P \le 100$. The end of the input will be designated with a line containing the number 0.

Output: For each menu, print a line identifying the menu number and the diameter D of the pizza with the best value, using the format shown below.

Example input:	**Example output:**

`3`
`5 2`
`10 6`
`12 8`
`3`
`5 2`
`10 5`
`12 8`
`4`
`1 1`
`24 33`
`13 11`
`6 11`
`0`
	`Menu 1: 12`
	`Menu 2: 10`
	`Menu 3: 24`

Duke of York, Part 2

Oh, the grand old duke of York
He had ten thousand men
He marched them up to the top of the hill
And he marched them down again.

The duke of York has once again got his men to the top of the hill, and he needs your help again! This time, he wants to march as many of his men as possible down to the bottom of the hill at once. Since his men all march at different speeds, this means that he can't assign any of his men to march on the same square during their descents, for risk of them getting confused and running into each other.

The duke has grown somewhat considerate of his men, however. Thus, among all possible sets of marching orders which maximize the number of men he can march down the hill at once, he wants you to minimize the average energy expenditure of his men.

The setup is as follows: The hill can be represented as a rectangular grid of squares with N rows and M columns, $2 \leq N, M \leq 20$. All of the squares in the first row are at the top of the hill, and all of the squares in the Nth row are at the bottom. Every square s on the hill has an associated base energy cost b, which is an integer between 0 and 9 inclusive, or it is impassable. The total energy cost to walk onto the square s is the base cost b of s plus the direction cost, which is 0 for going downhill, 1 for going left or right, and 3 for going uphill.

A path down the hill is a sequence of moves (up, down, left, or right), starting from one of the squares at the top of the hill and ending at one of the squares at the bottom of the hill. The total energy expenditure of such a path is the sum of all the energy expenditures necessary to enter each square along the path (except for the starting square at the top of the hill). A path cannot start on, travel through, or end on an impassable square.

Input: The first line of input will contain an integer $1 \leq C \leq 20$, the number of test cases. The first line of each case will contain two integers: N and M, with $2 \leq N, M \leq 20$. The next N lines will contain a map of the hill, in the following format. Each square on the hill is represented by a single character: either one of the numbers 0-9, if the square is passable and has the corresponding base energy cost, or the letter X, if the square is impassable. There will be no whitespace in the map except for the newlines at the end of each line of M characters.

Output: For each test case, output a single line containing two space-separated numbers: first, the maximum number of men that the duke can march down the hill at once, and second, a real number giving the minimum average energy cost for those men to march down the hill, with exactly two digits after the decimal point (rounded). You are guaranteed that at least one man can march down the hill.

```
Sample Input:
1
5 3
001
2X7
000
111
X23


Output for Sample Input:
2 8.50
```

Problem B: Antimonotonicity

I have a sequence Fred of length n comprised of integers between 1 and n inclusive. The elements of Fred are pairwise distinct. I want to find a subsequence Mary of Fred that is as long as possible and has the property that:

```
Mary[0] > Mary[1] < Mary[2] > Mary[3] < ...
```

Input

The first line of input will contain a single integer T expressed in decimal with no leading zeroes. T will be at most 50. T test cases will follow.

Each test case is contained on a single line. A line describing a test case is formatted as follows:

```
n Fred[0] Fred[1] Fred[2] ... Fred[n-1].
```

where n and each element of Fred is an integer expressed in decimal with no leading zeroes. No line will have leading or trailing whitespace, and two adjacent integers on the same line will be separated by a single space. n will be at most 30000.

Output

For each test case, output a single integer followed by a newline --- the length of the longest subsequence Mary of Fred with the desired properties.

Sample Input

```
4
5 1 2 3 4 5
5 5 4 3 2 1
5 5 1 4 2 3
5 2 4 1 3 5
```

Sample Output

```
1
2
5
3
```

Tor Myklebust