BAPC 2012

The 2012 Benelux Algorithm Programming Contest

The Problem Set

A Another Dice Game

In the game *Pickomino*[1] one has to throw 8 dice to reach at least a certain target score. The rules are as follows:

- The dice contain the values `1`, `2`, `3`, `4`, `5` and `worm`. The dice are fair, so all outcomes are equally likely.
- The game is started by throwing all dice.
- After a throw, the player must pick one of the six possible values and put all dice with this value aside. There must be at least one die with this value.
- After putting some dice aside, the player may choose to either throw the remaining dice again or stop. The player may only stop after at least one `worm` has been put aside.
- Each possible value may only be chosen once during the game.
- When the player stops, his total score is the sum of the values of the dice that were put aside. A `worm` is worth 5 points.
- The player can get stuck by throwing only values that were already put aside, by having put all dice aside but not having a `worm` or by not having reached the target score.
- If the player is stuck he scores 0 points and the game is ended.

Jan is playing Pickomino and wants to score at least n points. When Jan uses an optimal strategy, what is the probability that he reaches this target?

Input

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

- one line with the integer n $(1 \leq n \leq 40)$: the target value.

Output

Per test case:

- one line with a floating point number: the probability that you score at least n points when using an optimal strategy.

This number should be accurate up to 10^{-6} relative or absolute precision.

[1]Dutch people may know this game as *Regenwormen*

Sample in- and output

Input	Output

`3`
`5`
`21`
`40`
	`0.9934260978934218`
	`0.8930267371507457`
	`0.0001461079070016448`

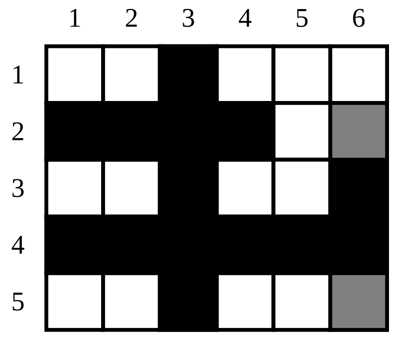
To reach 5 points it is enough to throw at least one `worm`. The optimal strategy in this case is therefore to stop as soon as you have a `worm`. If you did not throw a worm, you should put aside as few dice as possible to maximize the chance of throwing a worm in a later throw.

B Black Out

When you've finished solving all problems in this contest, there's no need to get bored, because we introduce a new and fun game called Black Out! Grab a piece of paper and draw a table of 5 by 6 squares. Next, choose your favorite teammate to play against. Each player in turn selects one or more squares that are adjacent and in the same row or column, and colors them all black. Some of these squares may be black already, but at least one of them must not be. The winner is the player to color the last square(s) black.

Once you got the hang of it, you might want to try and write your own AI player. You can then submit it and let it play a few games against our (= the jury's) player!

After a lengthy analysis, we have managed to prove that the game is theoretically won by the player who starts, so we'll gracefully let your player have the first move every time. If you manage to beat our player in every game, you'll be awarded another point in this contest. But be careful: the jury's player is not to be underestimated and will surely grab any chance it is offered! If you lose, you'll get a Wrong Answer. Also note that we are not that patient, so make sure your player is quick to move.



The move 2 6 5 6 as given in the sample makes all squares from (2,6) to (5,6) black. The squares (3,6) and (4,6) were black already.

Input and output

This is a problem with dynamic input and output, read the following description very carefully.

The first line of the input contains one positive number: the number of games, at most 100. Then, for each game:

- Your program writes a move on a single line to standard output. A move consists of four space-separated integers r_1, c_1, r_2 and c_2 ($1 \leq r_1 \leq r_2 \leq 5$ and $1 \leq c_1 \leq c_2 \leq 6$; also, $r_1 = r_2$ and/or $c_1 = c_2$) on a single line: the row and column number of the two squares between which all squares are colored black (including the two indicated squares). In case only one square is colored black, $r_1 = r_2$ and $c_1 = c_2$.
- After each move, our program writes one of the following two responses on a single line to standard input:

- The word "`MOVE`", followed by a single space, followed by four space-separated integers, representing the response of our program. Your program must now make another move.
- The word "`GAME`", indicating that the game has ended: either your program has won, or our program intends to make a move that ends the game in its favor.

Result

These are the possible causes for **some** of the possible results of your submission:

CORRECT Your program has won all games. Congratulations!

RUN-ERROR There was an error during the execution of your program, or your program's output did not adhere to the specified format and constraints, or all the squares that you wanted colored black were black already (invalid move).

TIMELIMIT Your program was too slow or it did not respond.

WRONG-ANSWER Your program has successfully completed all games, but it has not won all of them.

Notes

- Make sure to print a newline after your move.

C users:

- Do `setlinebuf(stdout);` at the start of your program to make sure output is flushed correctly.
- If you use `scanf` to read the moves (which we recommend), then do not read the newline character at the end. So, to read the input with `scanf`, use `scanf("%s",s)` and `scanf("%d %d %d %d", &r1, &c1, &r2, &c2);`.
  Also, do **not** use `scanf("MOVE")` or `scanf("GAME")`.

C++ users:

- Recommendation: use `std::cin` and `std::cout` to read input and write output. This will ensure output is flushed correctly.

Java/C# users:

- Do not use *buffered* output; either `System.out.println` or `System.Console.WriteLine` will do fine.

Haskell users:

- Do `hSetBuffering stdout LineBuffering` at the beginning of `main`.

Sample in- and output

Notes:

- This example is merely to illustrate the communication between your program and the jury's program; it does **not** demonstrate an example of optimal play for either player.
- The extra newlines in the example below are **for clarity only**, to demonstrate the order of events; you should print exactly one newline character after each move, and the input contains no blank lines either.

Input	**Output**
`1`	
	`2 1 2 4`
`MOVE 3 6 3 6`	
	`1 3 5 3`
`MOVE 4 2 4 6`	
	`4 1 4 6`
`MOVE 2 6 5 6`	
	`1 1 1 6`
`MOVE 3 1 5 1`	
	`3 2 3 5`
`MOVE 5 2 5 5`	
	`2 5 2 5`
`GAME`	

Almost blank page

C Chess Competition

In a chess competition, every player plays one game against every other player. The winner receives one point, the loser none. In case of a draw, both players receive half a point. When all games have been played, whoever scored the most points wins the competition. If multiple competitors are tied for the highest score, they will play a series of tie-break games to determine the winner.

For this problem we consider a competition where the games are played in arbitrary order. Based on the outcome of the games that have been played so far (which could be all, or none, or anything in between) the organizers want to determine which players can still win the competition.

Input

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

- one line with an integer n ($2 \leq n \leq 30$): the number of players.
- n lines with n characters each, describing the intermediate results. The j-th character on the i-th line gives the result of the i-th player against the j-th player:
  - `'1'` for a win,
  - `'0'` for a loss,
  - `'d'` for a draw,
  - `'.'` if this game has not been played yet,
  - `'x'` when $i = j$ (since competitors do not play against themselves).

The intermediate results are guaranteed to be internally consistent. More formally, if the j-th character on the i-th line is a digit (`'0'` or `'1'`) then the i-th character on the j-th line will be the other digit. In all other cases, the j-th character on the i-th line equals the i-th character on the j-th line.

Output

Per test case:

- one line with the 1-based indices of all the players that can possibly win the competition, in increasing order, separated by spaces.

Sample in- and output

Input	Output

`3`
`5`
`x.11d`
`.x1d1`
`00x.0`
`0d.x.`
`d01.x`
`7`
`x00111.`
`1x01d.d`
`11x1.00`
`000x000`
`0d.1xd1`
`0.11dxd`
`.d110dx`
`7`
`x00011.`
`1x00d.d`
`11x0.0.`
`111x111`
`0d.0xd.`
`0.10dx.`
`.d.0..x`
	`1 2`
	`1 2 3 5 6 7`
	`4`

D Digit Sum

For a pair of integers a and b, the digit sum of the interval $[a, b]$ is defined as the sum of all digits occurring in all numbers between (and including) a and b. For example, the digit sum of $[28, 31]$ can be calculated as:

$$2{+}8 \ + \ 2{+}9 \ + \ 3{+}0 \ + \ 3{+}1 = 28$$

Given the numbers a and b, calculate the digit sum of $[a, b]$.

Input

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

- one line with two space-separated integers, a and b ($0 \leq a \leq b \leq 10^{15}$).

Output

Per test case:

- one line with an integer: the digit sum of $[a, b]$.

Sample in- and output

Input	Output

`3`
`0 10`
`28 31`
`1234 56789`
	`46`
	`28`
	`1128600`

Almost blank page

E Encoded Message

Alex wants to send a love poem to his girlfriend Bridget. Unfortunately, she has a nosy friend, Ellen, who might intercept his message and invade their privacy.

To prevent this, Alex has invented a scheme to make his missives indecipherable to Ellen. He arranges the letters into a square, which is rotated a quarter-turn clockwise, and then he puts the resulting letters on a single line again. (For simplicity's sake, Alex doesn't use whitespace or punctuation in his poems.)

For example, the text "`RosesAreRedVioletsAreBlue`" would be encoded as "`eedARBtVrolsiesuAoReerles`" using the following intermediate steps:

R	o	s	e	s
A	r	e	R	e
d	V	i	o	l
e	t	s	A	r
e	B	l	u	e

$\Rightarrow$

e	e	d	A	R
B	t	V	r	o
l	s	i	e	s
u	A	o	R	e
e	r	l	e	s

Ellen has intercepted some of Alex's messages but they make no sense to her. Can you write a program to help her decode them?

Input

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

- one line with an encoded message: a string consisting of upper-case and lower-case letters only. The length of the message is a square between 1 and 10 000 characters.

Output

Per test case:

- one line with the original message.

Sample in- and output

Input	**Output**
`3`	`TOPSECRET`
`RSTEEOTCP`	`RosesAreRedVioletsAreBlue`
`eedARBtVrolsiesuAoReerles`	`SquaresMayBeEven`
`EarSvyeqeBsuneMa`	

Almost blank page

F Fire

You are trapped in a building consisting of open spaces and walls. Some places are on fire and you have to run for the exit. Will you make it?

At each second, the fire will spread to all open spaces directly connected to the North, South, East or West side of it. Fortunately, walls will never catch fire and will keep the fire inside the building, so as soon as you are out of the building you will be safe. To run to any of the four open spaces adjacent to you takes you exactly one second. You cannot run through a wall or into an open space that is on fire or is just catching fire, but you can run out of an open space at the same moment it catches fire.

Given a map of the building, decide how fast you can exit the building.

Input

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

- one line with two space-separated integers w and h ($1 \leq w, h \leq 1\,000$): the width and height of the map of the building, respectively.
- h lines with w characters each: the map of the building, consisting of
  - '`.`': a room,
  - '`#`': a wall,
  - '`@`': your starting location,
  - '`*`': fire.

  There will be exactly one '`@`' in the map.

Output

Per test case:

- one line with a single integer which is the minimal number of seconds that you need to exit the building or the string "`IMPOSSIBLE`" when this is not possible.

Sample in- and output

Input

```
5
4 3
####
#*@.
####
7 6
###.###
#*#.#*#
#.....#
#.....#
#..@..#
#######
7 4
###.###
#....*#
#@....#
.######
5 5
.....
.***.
.*@*.
.***.
.....
3 3
###
#@#
###
```

Output

```
2
5
IMPOSSIBLE
IMPOSSIBLE
IMPOSSIBLE
```

G Good Coalition

The Dutch political system is in turmoil. There have been six coalition governments in the past fourteen years, all of which have fallen before completing their term in office. Recently there have been elections (again), the outcome of which has been described as "impossible" by several political commentators. The only bright spot in this bleak situation is that they have appointed *you* as the "informateur". As the informateur it is your task to find a suitable coalition.

Being the rational person you are, you have decided the first negotiation attempt should be started between the parties forming the *most stable coalition*. A coalition is formed by a set of parties having won a strict majority of seats in the election (i.e. at least 76 seats out of a total of 150). The most stable coalition is one that has the highest chance of completing its term in office. A coalition falls (and new elections must be held) if a single party leaves the coalition. The probability of a coalition completing their term is estimated by the product of the probabilities of each party in the coalition completing their term. This probability is in turn based on historical data.

Find the best coalition and save the Netherlands from becoming a banana republic!

Input

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

- one line with an integer n ($1 \leq n \leq 150$), the number of political parties that have won at least one seat in the election.
- n lines, each with two space-separated integers s_i and p_i ($1 \leq s_i \leq 150$ and $1 \leq p_i \leq 100$): the number of seats won by the i-th party and the probability (expressed as a percentage) that the i-th party will complete its term in office, respectively.

Note that there are exactly 150 seats divided among all parties $\left(\sum_{i=1}^{n} s_i = 150\right)$.

Output

Per test case:

- one line with a floating point number: the probability (expressed as a percentage) of the most stable coalition sitting out their term in office.

This number should be accurate up to 10^{-6} relative or absolute precision.

Sample in- and output

Input	Output

1
4
35 80
25 70
60 60
30 90
	54.0

Almost blank page

H Hot Dogs in Manhattan

The two friends Barack and Mitt have both decided to set up their own hot dog stand in Manhattan. They wish to find the two optimal locations for their stands.

First of all, they both want to put their stand at an intersection, since that gives them maximum exposure. Also, this being Manhattan, there are already quite a few stands in the city, also at intersections. If they put up a stand close to another (or each other's) stand, they might not get that many customers. They would therefore like to put their stands as far from other stands as possible.

We model Manhattan as a finite square grid, consisting of w vertical streets and h horizontal streets. The vertical streets run from $x = 0$ to $x = w - 1$, while horizontal streets run from $y = 0$ to $y = h - 1$. All pairs of consecutive parallel streets are separated by the same distance, which we set as the unit distance. The distance between two intersections (x_1, y_1) and (x_2, y_2) is then given by $|x_1 - x_2| + |y_1 - y_2|$.

We indicate an intersection's suitability by its *privacy*, which is the minimum of all distances from this intersection to all other hot dog stands. Barack and Mitt would like to find two intersections with the maximum amount of privacy, i.e. such that the smallest of the two privacies is as large as possible. Note that the privacy of Barack's location can be determined by the distance to Mitt's location and vice versa.

Input

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

- one line with three space-separated integers n, w and h ($0 \leq n \leq 1\,000$ and $2 \leq w, h \leq 1\,000$): the number of hot dog stands already in place and the number of vertical and horizontal streets, respectively.
- n lines, each with two space-separated integers x_i and y_i ($0 \leq x_i < w$ and $0 \leq y_i < h$): the intersection where the i-th hot dog stand is located.

All hot dog stands are at different intersections. At least two intersections do not contain a stand.

Output

Per test case:

- one line with one integer: the maximum privacy that Barack and Mitt can both obtain.

Sample in- and output

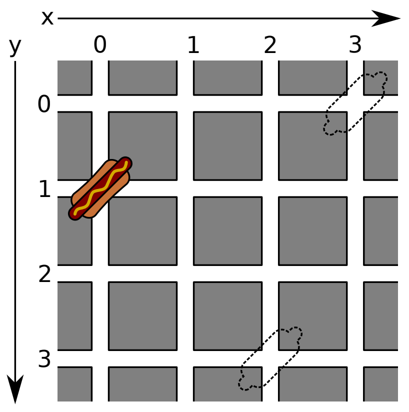
Input

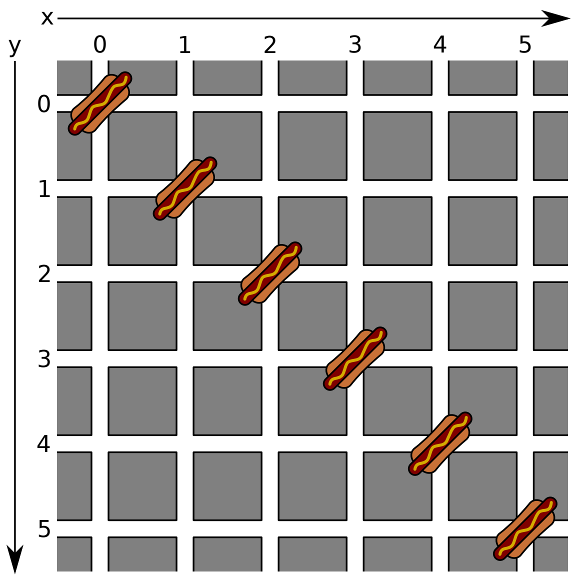
```
3
1 4 4
0 1
6 6 6
0 0
1 1
2 2
3 3
4 4
5 5
2 8 3
0 0
7 0
```

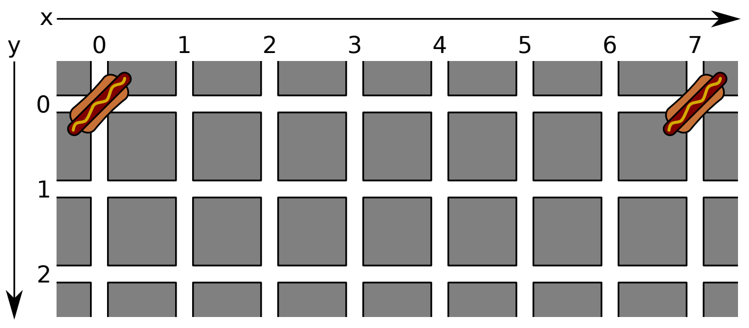
Output

```
4
5
3
```

These sample cases are illustrated below. Only for the first case, the optimal placement of the two new stands is given (indicated by the dashed outlines).







I Integer Lists

The programming language *Better And Portable Code* (BAPC) is a language for working with lists of integers. The language has two built-in functions: 'R' (*reverse*) and 'D' (*drop*).

The function 'R' reverses its input list, and 'D' drops the first element of its input and returns the rest, or gives an error in case its input is an empty list. To get more advanced behavior, functions can be composed: "AB" is the function that first applies 'A' to its input and then 'B' to the resulting list. For example, "RDD" is a function that reverses a list and then drops the first two elements.

Unfortunately, our BAPC interpreter has bit rotted, so we ask you to write a new one. Given a BAPC program and its input, return its output or "error" in case 'D' is applied to an empty list. Lists are represented as the character '[' followed by a comma-separated list of integers followed by the character ']'. Notice that the input and output lists can be quite long.

Input

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

- one line with a string p ($1 \leq \text{length}(p) \leq 100\,000$): a BAPC program, consisting of the characters 'R' and 'D'.
- one line with an integer n ($0 \leq n \leq 100\,000$): the number of elements in the input.
- one line with a list of n integers in the form $[x_1, \ldots, x_n]$ ($1 \leq x_i \leq 100$): the input list.

Output

Per test case:

- one line with the resulting integer list or "error" in case of an error.

Sample in- and output

Input	Output

`4`
`RDD`
`4`
`[1,2,3,4]`
`DD`
`1`
`[42]`
`RRD`
`6`
`[1,1,2,3,5,8]`
`D`
`0`
`[]`
	`[2,1]`
	`error`
	`[1,2,3,5,8]`
	`error`

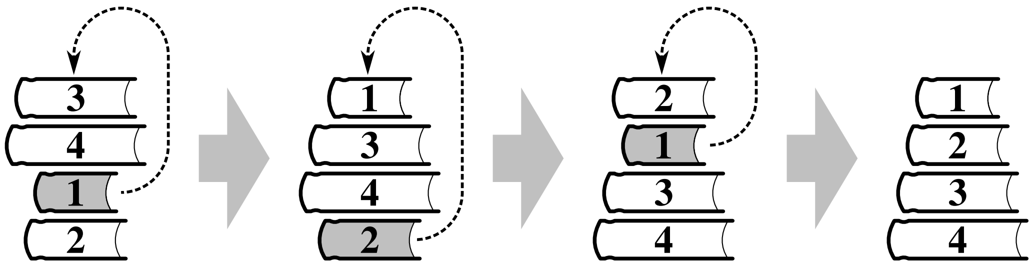
Almost blank page

J John's Book Stack

John has a big stack of books of various sizes. Such a stack is stable if the books have non-decreasing sizes (viewed from top to bottom); otherwise, it is unstable, and likely to fall over.

To prevent this, John wants to sort the books in the stack by size. He does so by pulling out a book from somewhere in the middle (or bottom) of the stack and then putting it back on top. However, he can only pull out a book safely if the books on top of it already form a stable stack.

For example, if John has a stack of four books with sizes 3, 4, 1 and 2 (from top to bottom) then he can sort them as follows:



Your task is to determine how many steps are required to sort a given stack of books. In the example above, which corresponds to the first sample case, the answer is 3.

Input

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

- one line with an integer n ($1 \leq n \leq 50$): the number of books.
- one line containing n space-separated integers s_i ($1 \leq s_i \leq 1\,000$ for $1 \leq i \leq n$): the sizes of the books, as they appear in the initial stack from top to bottom.

Output

Per test case:

- one line with an integer: the minimum number of steps required to sort the stack using the algorithm described above.

Sample in- and output

Input	Output

`4`
`4`
`3 4 1 2`
`8`
`3 1 4 1 5 9 2 6`
`5`
`1 42 42 42 1000`
`22`
`4 1 2 5 6 7 9 10 3 13 17 11 12 14 19 20 22 8 15 16 18 21`
	`3`
	`53`
	`0`
	`1234567`

Almost blank page