UW-Madison ACM ICPC Individual Contest

October 4th, 2015

Setup

Before the contest begins, log in to your workstation and set up and launch the PC2 contest software using the following instructions. You will use this program to submit problem solutions, receive the judges' answers, and communicate clarification requests.

1. Download the custom PC2 package into a directory (different from last week) of your choosing from: `www.cs.wisc.edu/~dieter/ICPC/15-16/pc2_15-16_2.tar.gz`
2. In a terminal window, cd to the directory where you downloaded the package and type `tar -xzvf pc2_15-16_2.tar.gz`
3. Type `cd pc2_15-16_2` followed by the command `bin/pc2team` – this brings up your PC2 terminal that will be your interface to the judges during the contest.
4. Log in using the login ID and password given to you by the judges when you arrive. They will be of the form teamX where X is an integer, and the password will be your UVa online judge username. If you have not yet sent your username to the judges, you will not be given a login and password by default – please see the judges to get one.

The Contest

Begin the contest by solving the problem on the next page **"count"**. This is a warmup problem designed to get you used to submitting problems via PC2. Code your solution to the problem and submit it as follows:

1. Click on the 'submit run' tab in your PC2 window.
2. In the dropdown menu labeled "Problem", choose **"count"**. Choose the programming language you used from the "language" dropdown menu. Then select your source code file by clicking the "select" button in the 'main file' section.
3. Submit your code by clicking "Submit" (note: clicking "Test" doesn't really do anything unless you've created your own test files, so don't expect it to automatically test your program against the sample input).
4. Wait – you will receive a judgment from the judge shortly by way of a pop-up window. If your answer comes back something other than "accepted", try again.

All input comes from standard in, all output should be sent to standard out. You may use any online Java or C++ documentation, but not any other resource. You may use the printer at any time. Collaboration with others or searching the web for solutions to these problems is prohibited. Please turn off your cell phones. You may submit problem clarifications via the PC2 program at any time, but please read the problems thoroughly before doing so.

Warmup Problem: Count

Can you count from one up to any number N? Write a program to prove it!

Input

The input begins with a single number that describes the number of test cases. Each test case follows on its own line, and consists of a single positive integer $N \leq 1,000,000$ that describes how high you should count for that test case.

Output

The output for each test case should be on its own line, and consist of the numbers 1 through N (inclusive), each separated by a space.

Sample Input

```
3
3
5
10
```

Sample Output

```
1 2 3
1 2 3 4 5
1 2 3 4 5 6 7 8 9 10
```

The following pages contain six lettered problems. Please let me know if you do not see all six.

A: Steve's Lotto Tickets

Steve likes to play the lotto. Whenever he does, he buys lots of tickets. Each ticket has 6 unique numbers in the range from 1 to 49, inclusive. Steve likes to "Cover all his bases." By that, he means that he likes for each set of lottery tickets to contain every number from 1 to 49, at least once, on some ticket. Write a program to help Steve see if his tickets "Cover all the bases."

Input

The input file consists of a number of test cases. Each case starts with an integer N ($1 \leq N \leq 100$), indicating the number of tickets Steve has purchased. On the next N lines are the tickets, one per line. Each ticket will have exactly 6 integers, and all of them will be in the range from 1 to 49, inclusive. No ticket will have duplicate numbers, but the numbers on a ticket may appear in any order. The input ends with a line containing only a 0.

Output

Print a list of responses for the input sets, one per line. Print the word `Yes` if every number from 1 to 49 inclusive appears in some lottery ticket in the set, and `No` otherwise. Print these words exactly as they are shown. Do not print any blank lines between outputs.

Sample Input

```
1
1 2 3 4 5 6
9
1 2 3 4 5 6
10 9 8 7 12 11
13 14 15 16 17 18
19 20 21 22 23 24
25 26 27 28 29 30
31 32 33 34 35 36
37 38 39 40 41 42
43 44 45 46 47 48
49 19 34 27 25 13
0
```

Sample Output

```
No
Yes
```

B: River Rapids

You have been hired by a big theme park to design a new attraction: a river rapids ride. You already designed the track; it is a round trip that is described by an inner and an outer polygon. The space in between the two polygons is the track.

You still need to design the rafts, however. It has been decided that they should be circular, so they can spin freely along the track and increase the fun and excitement of the ride. Besides that, they should be as big as possible to fit the maximum number of people, but they can't be too big, for otherwise they would get stuck somewhere on the track.

Your boss knows there might be some geometry involved in deciding the size of the rafts and has taken the time to provide you with some pseudocode to compute the distance from a point to a line segment. The functions are as follows:

```
// returns the distance from point a to point b
Dist(Pt a, Pt b) {
    return sqrt((a.x-b.x) * (a.x-b.x) + (a.y-b.y) * (a.y-b.y));
}

// returns the dot product of a and b
Dot(Pt a, Pt b) {
    return a.x*b.x + a.y*b.y;
}

// returns the distance from point p to line segment ab
DistPtLine(Pt p, Pt a, Pt b) {
    r = Dist(a, b) * Dist(a, b);
    pma = Pt(p.x-a.x, p.y-a.y);
    bma = Pt(b.x-a.x, b.y-a.y);
    r = dot(pma, bma)/r;
    Pt q;
    if (r < 0) q = a;
    else if (r > 1) q = b;
    else q = Pt(a.x + bma.x*r, a.y + bma.y*r);
    return Dist(p, q);
}
```

Using your expert knowledge of geometry and the functions provided, can you determine the maximum radius of the rafts so they can complete the track?

Input

On the first line will be one positive number: the number of testcases, at most 100. After that per testcase:

- One line with an integer n_i ($3 \leq n_i \leq 100$): the number of points of the inner polygon.
- n_i lines with two integers each: the coordinates of the points of the inner polygon in consecutive order.
- One line with an integer n_o ($3 \leq n_o \leq 100$): the number of points of the outer polygon.
- n_o lines with two integers each: the coordinates of the points of the outer polygon in consecutive order.

All coordinates have absolute value no larger than $1,000$. The points of the polygons can be given in either clockwise or counterclockwise order and the two polygons do not intersect or touch themselves or each other. The outer polygon encloses the inner polygon.

Output

Per testcase:

- One line with a floating point number: the maximal radius of the river rapids rafts, rounded to 6 decimal places.

Sample Input

```
2
4
-5 -5
5 -5
5 5
-5 5
4
-10 -10
-10 10
10 10
10 -10
3
0 0
1 0
1 1
5
3 -3
3 3
-4 2
-1 -1
-2 -2
```

Sample Output

```
2.500000
0.707107
```

C: Chicago

As you probably know, the streets and crossroads of Chicago form a perfect grid. What you probably don't know, is that there is exactly one gaming club and exactly one gamer on each of these crossroads. Strangely enough, each gaming club offers exactly one game. Gamers are also a bit strange, but they generally live a simple life, determined by the following rules:

1. A gamer never plays in the club of his own crossroads. Never!!!
2. During a day each gamer has to play each of the games when this is not contradicting Rule 1.
3. Gamers travel from one crossroads to another only by horizontals or verticals.
4. A gamer can not go directly from one club to another. He/she has to come back home and eat something first, and finish the day in his/her own intersection.
5. A gamer has to live optimally, thus always picking the best possible strategy to implement the above rules, so that he/she has enough time for gaming.

Having in mind that all the gamers are the same, the Association of Computer Maniacs decided that the city should be optimized in order to minimize the total non-gaming effort. The *non-gaming effort* is the number of crossroads that a gamer walks through in a day. The *total non-gaming effort* is the sum of the non-gaming efforts for all gamers in the city. In order to perform the optimizations, ACM needs a program that calculates the total non-gaming effort for a given description of the city.

Input

Several city descriptions are given as input. Each of them starts with a line of two integers R and C ($1 \leq R, C \leq 1\,000$), denoting the number of rows and the number of columns of the crossroads in the city. R lines of C characters follow, denoting the kind of game that is offered by the club of each of the crossroads. Each game is coded using a single digit ('0' to '9'). Keep processing input until end of file.

Output

For each of the city descriptions, output a single line containing one integer – the total non-gaming effort for the city.

Sample Input

```
2 2
12
12
3 3
122
313
321
```

Sample Output

```
16
72
```

Explanation: In the first case, each of the four gamers has to play games 1 and 2. Luckily, both are only one intersection away, so the non-gaming effort for each gamer is 4 (from home to club 1, from club 1 to home, from home to club 2 and then from club 2 home). That means a total non-gaming effort of 16.

In the second case, each of the 9 gamers has two of the games 1 intersection away, and one 2 intersections away, meaning a total non-gaming effort of $9 * (1 + 1 + 1 + 1 + 2 + 2) = 72$.

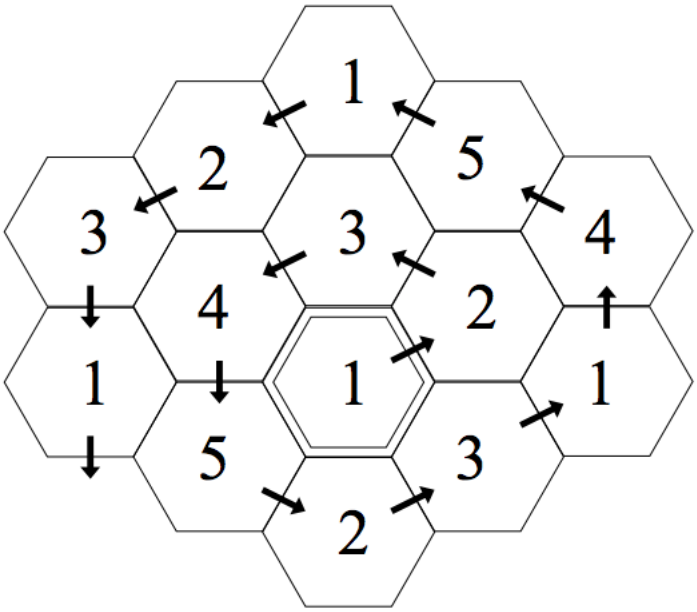
D: Board Game

There is a popular board game consisting of hexagonal resource tiles placed randomly at the start of each game (can you think of what game this might be?). Each of the resources are denoted by the numbers 1 to 5.

Annoying to some players, a random board often has multiple equal resource tiles next to each other. To circumvent this issue, we have invented a new way of creating the playing board. Starting in the middle and spiraling outwards, each time we add a new tile to the board we choose the resource of the tile according to the following rules:

- the new tile must be different from its neighboring tiles on the board so far;
- in case multiple tiles are possible, we choose a resource that occurs the least number of times on the board so far;
- in case multiple tiles are still possible, the new resource must have the lowest number possible.

The figure underneath shows how to spiral outwards and which resource tiles are chosen first. We are curious what the number of the resource is on the nth tile that is added to the board (starting with $n = 1$).



Input

On the first line of the input there is one integer c ($1 \leq c \leq 200$), the number of test cases. Each following test case consists of a single line with one integer n ($1 \leq n \leq 10\,000$), the number of the tile we are curious about.

Output

For each test case, print a single line with one integer, specifying the resource of the nth tile.

Sample Input

```
4
1
4
10
100
```

Sample Output

```
1
4
5
5
```

E: Road Construction

Ryan is repairing roads. The job is concentrated on roads with one lane in each direction. Thus, when Ryan closes down the lane in one direction, all traffic has to go through the other lane. This is done by allowing only one direction of travel at any time. Ryan is often assigned the task of directing the traffic through this lane.

No car drives before given a "go" signal from Ryan, and all the cars drive through the maintained segment at the same speed. Because there is only one lane, cars in one direction must leave the segment before cars in the other direction can enter. For safety reasons, cars driving in the same direction have to keep a distance of at least 3 seconds between each other.

For example, if cars `A` and `B` arrive at the west endpoint at second 10, Ryan can let them go at earliest second 10 and 13 in the order they arrived. If it, in this example, takes 8 seconds to pass and car `C` arrives at the east endpoint at second 17, then car `C` has to wait 4 seconds until Ryan lets it go at second 21.

There is a problem of drivers getting irritated with Ryan; they think they have to stop for too long. Ryan has been logging how long they can bear to wait before they get irritated. One day, to be able to evaluate his work, Ryan noted down when the cars arrived at the two endpoints of the segment. Ryan's question is the following: what is the least number of drivers that can be irritated? We assume that a driver gets irritated if the time between the moment he arrives at the maintained segment and the moment he is actually given the "go" exceeds his irritation time limit.

Input

The input will consist of multiple test cases. The first line of every test case contains two integers t and n ($4 \leq t \leq 180$ and $1 \leq n \leq 250$), where t is the time in seconds needed for a car to pass the segment under maintenance, and n is the total number of cars arriving at the segment. The following n lines describe the cars. The ith line contains the description of the ith car in the following format:

- one character d, being `W` for cars arriving at the west endpoint of the segment, and `E` for the ones that arrive at the east endpoint; and
- two integers a and r ($0 \leq a < 86\,400$ and $0 \leq r \leq 3\,600$), where a denotes the arrival time in seconds after midnight, and r denotes the time in seconds it takes for the driver to get irritated.

The cars arrive in the order specified in the input and they cannot overtake each other. In particular, a car whose driver is already irritated has to stay in the queue until eventually receiving the "go" and passing the maintained segment.

Continue processing input until end of file.

Output

Output one line with the least possible number of irritated drivers.

Sample Input

```
8 3
W 10 0
W 10 3
E 17 4
100 5
W 0 200
W 5 201
E 95 1111
E 95 1
E 95 11
```

Sample Output

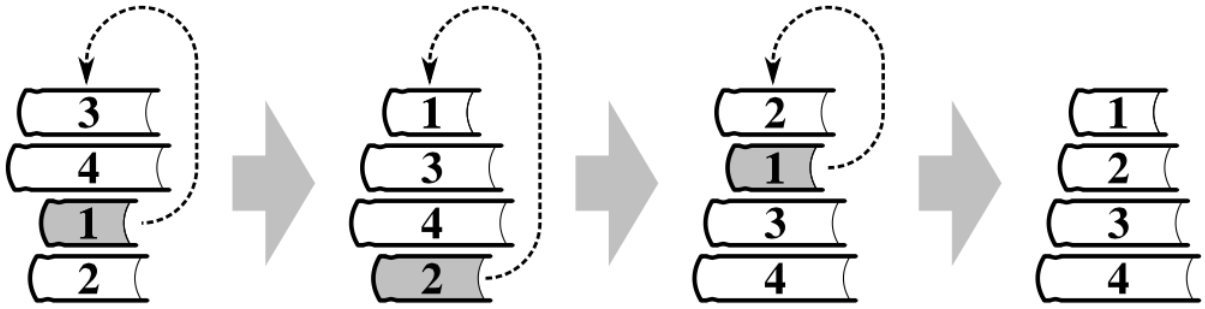
```
0
1
```

F: Kyle's Book Stack

Kyle 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, Kyle 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 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 Kyle 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 will be 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 Input

```
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
```

Sample Output

```
3
53
0
1234567
```