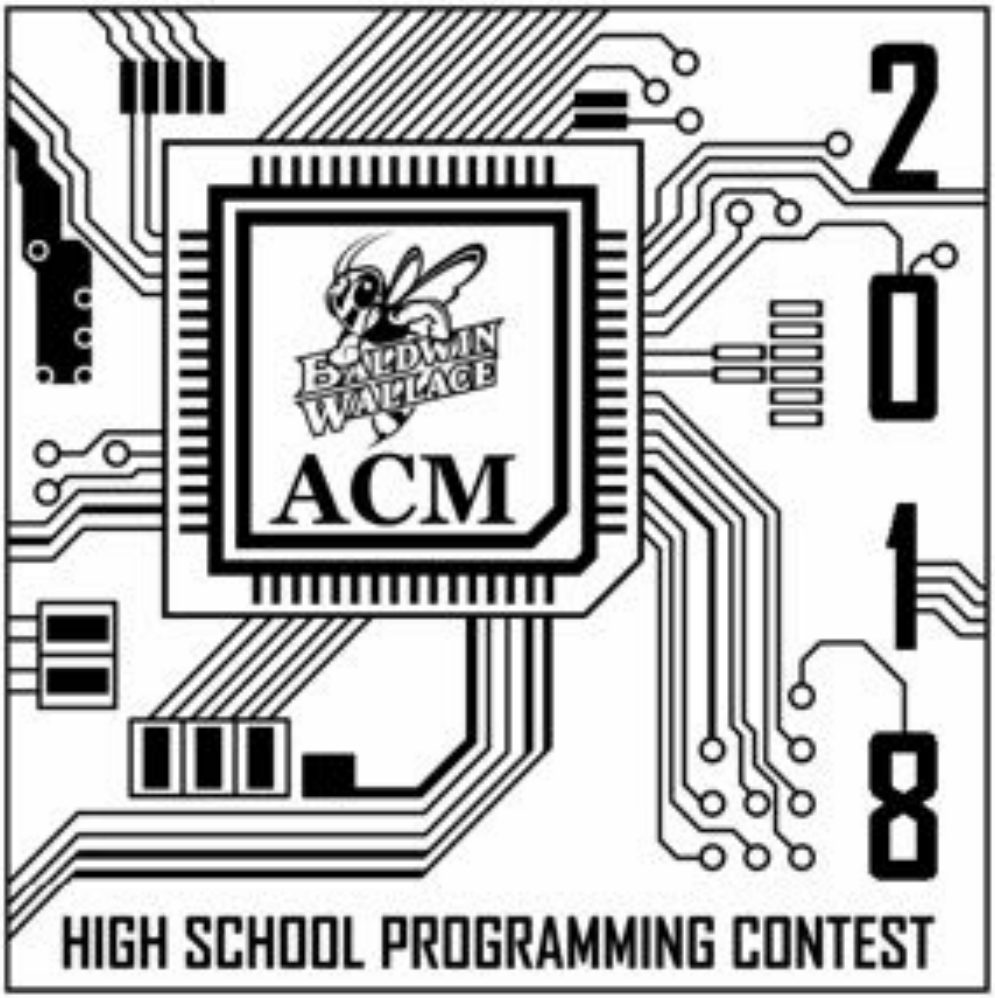
BALDWIN WALLACE UNIVERSITY

2018 HIGH SCHOOL PROGRAMMING CONTEST



DO NOT OPEN UNTIL INSTRUCTED TO DO SO

Twin Binary Engine

Han Solo is in the process of upgrading the Millennium Falcon's engines. The new model he is designing is based on observations he has made of binary star systems. However, he is not sure how the new engines will perform. Unlike warp engines, Han's new design works by creating a stasis field around the Millennium Falcon. When this occurs, time is bent around the Millennium Falcon, while the Falcon stays in place. To mathematically prove the functionality of this new design, Han requires the help of his assistant, you. Han has given you a list of numbers ranging from 0 to 1,000,000. He wants you to calculate the average length of the binary representation of these numbers from 0 up to the given number. Because these numbers will be used to help prove the new engine design, Han requires that each average be rounded down to the nearest integer.

For example, consider the value 2. Then…

numbers up to 2	0	1	2
binary representation	0	1	10

The average length of the binary representation of these numbers is 1.

Details of the input
The first line of input consists of a single positive integer, *c*, representing the number of cases to be processed. For each case, there is one line of input. Each line contains a positive integer, *l*, such that $0 <= l <= 1{,}000{,}000$.

Details of the output
For each case, one line of output is generated. The line begins with a label of the form `Case i:` (i=1…*c*) and then prints the average you have calculated.

Sample input
```
5
257
10058
492847
22547
10
```

Sample output
```
Case 1: 7
Case 2: 12
Case 3: 17
Case 4: 13
Case 5: 2
```

Number Thieves

The Empire has discovered your location due to your incompetent droid co-pilot who turned on your locator beacon. Thankfully, you were able to turn it off before the Empire could get a fix on your location. Sadly, they were able to detect some transmissions you were waiting for, which you knew were each to contain some number of consecutive integers in a random order (e.g., 6 2 8 4 7 5 3). The Empire removed one number from each transmission. Before you can leave the sector, you must write a program to determine the missing number in each transmission.

Details of the input
The first line of input consists of a single positive integer, *c*, which is the number of transmission cases. For each case, there will be a single line containing multiple numbers separated by spaces. The first number, *n*, is a positive integer (2<=*n*<=8) representing the number of values sent in the original transmission. The second number, *s,* is a positive integer s (1<=*s*<=7) representing the starting value of the intended consecutive sequence. Following will be *n-1* integers representing the values actually received.

Details of the output
For each test case a single line of output is printed with the following form:

```
Case#: #
```

Where the first # is the transmission number and the second # is the missing number.

Sample input

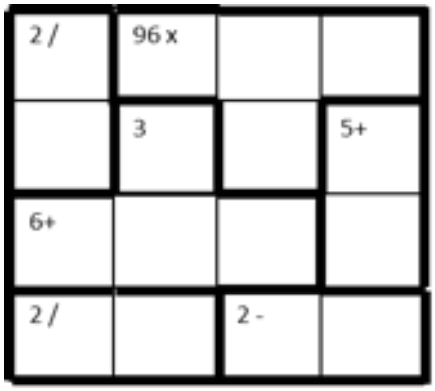
```
4
6 1 1 2 3 5 6
8 1 6 3 4 1 7 5 8
3 1 1 3
3 4 4 6
```

Sample output

```
Case 1: 4
Case 2: 2
Case 3: 2
Case 4: 5
```

Safe Hackers

DJ, one of the galaxy's most skilled slicers, is attempting to be the first to hack into the Rotating Bio-quadrocrypt Puzzle Safe. He is almost able to crack the safe by himself but he has come across an issue. On a few of his practice runs he notices that some of his solutions were incorrect. Since he only has one shot until the safe's anti-theft deterrent system kicks in, he has recruited you to create a program which checks his work and ensure a successful entry.



The rules of the puzzle safe are fairly simple:

- Each row must include one instance of the numbers 1..*n*, where *n* is the size of the dimensions of the square grid.
- Each column must include one instance of the numbers 1..*n*, where *n* is the size of the dimensions of the square grid.
- The digits in each of the heavily outlined groupings must combine to give the result indicated within the grouping when the operation indicated is applied to the operands. Note that "+" and "*" may have any number of operands while "-" and "/" will only have two.
- The order of the numbers within the heavily outlined grouping is determined based on meeting the first two rules. It is assumed that when "-" and "/" are involved, the first operand is always the larger of the numbers in the group in order that an integer result is obtained.

Details of the input

The first line of the input is a single positive integer, *c,* which indicates the number of puzzle safes to be checked. For each case, the first line of input is a single positive integer, *s,* that specifies the size of the grid involved. The next *s* lines will contain the integers 1..*s* in the order that they are to be entered into the solution grid. The next line is a single positive integer, g, that indicates the number of groupings the grid contains. The next *g* lines, specify the coordinates of grouped cells, the operation to be applied to the

values in the cells and the expected result in that order. The values on a line of input will be separated from each other by a single blank space.

Details of the output

For each case you will output one line with one of the following forms:

```
Case #: Solution is correct.
Case #: Solution is incorrect.
```

Sample input

```
1
4
1 4 3 2
2 3 4 1
3 1 2 4
4 2 1 3
6
0 0 1 0 / 2
0 1 0 2 0 3 1 2 * 96
1 3 2 3 + 5
2 0 2 1 2 2 + 6
3 0 3 1 / 2
3 2 3 3 - 2
```

Sample output

```
Case 1: Solution is correct.
```

Intercepted Frequencies

The Resistance has been trying to intercept communications between Kylo Ren and several other key members of the First Order. The Resistance has finally succeeded in capturing this information however the messages have arrived in a random order. The Resistance has determined that each message is split into several lines where each line contains a message number, followed by a line number, and then finally a part of the message. However, Leia and the Resistance lack a means to unscramble these messages and have thus entrusted this vital task to you. Based on Resistance Analysis reports, the messages appear as follows.

Details of the input

The first line of input consists of a single positive integer, *c*, specifying the number of message cases to be processed. For each case, the first line of input is a single positive integer, *t*, indicating the total number of lines within the case. Each line consists of three components. The first component is a positive integer, *m*, representing the message number this line belongs to. The second component is a positive integer, *l*, indicating the line number within the message. The final component is a series of strings representing the message component itself.

Details of the output

The output for each case should begin with the label "`Case #:`". The messages of the case follow with each message having the form:

```
Message #:
[The lines of the message go in order here, one per line of output]
```

Sample input

```
1
5
0 1 the shadows. This ultimately led to Anakin's fall as a Sith
1 0 After seeing her in action, Han offered Rey a permanent
0 0 Unbeknownst to Anakin, the Emperor was manipulating him from
1 1 position as a member of his crew.
0 2 lord.
```

Sample output

```
Case 1:
Message 0:
Unbeknownst to Anakin, the Emperor was manipulating him from
the shadows. This ultimately led to Anakin's fall as a Sith
lord.
Message 1:
After seeing her in action, Han offered Rey a permanent
position as a member of his crew.
```

Circuit Breaker

BB-8 has recently undergone an upgrade that has increased his circuit breaking abilities tenfold. Unfortunately, the upgrade left him with damage to his own internal circuitry. You are given the task of repairing BB-8s circuits. Fortunately for you, BB-8s internal circuitry patterns always follow the same layout, a 10x10 matrix. However, you only have a limited number of repairs you can make on the matrix.

Your goal is to travel through the matrix so that each move goes to an adjacent cell that will contribute the best value toward a running total that is computed along the way (which will either be getting larger or smaller, depending on the needed repairs). If you move correctly, then you will successfully repair the circuitry. This repair process continues until you are either out of repairs (moves) or you run out of valid adjacent cells (see the following notes). Note that the coordinates of the matrix are defined such that the upper left hand cell is (0,0) and the bottom right cell is (9,9).

Some notes on valid moves:

- You are not allowed to wrap around the matrix (that is, jump from the end of a row back to its beginning, for example).
- A valid adjacent cell is any cell that is both within the matrix **and** has not already been visited.

For instance, say we are currently at the location in the matrix indicated by the star and our total is to be getting larger. We would evaluate adjacent cells in the order indicated in the upper left of each cell. After evaluating our cells, we would then pick cell three (3) as our next cell and add its value to our total (because it is the first best cell we came across during evaluation). Cell three would then become the cell we evaluate from for the next iteration:

1 100	2 -27	3 223
4 154	★	5 -247
6 223	7 -50	8 -117

Details of the input

The first line of input consists of a single integer, *c*, representing the number of cases to be processed. Each case begins with a line containing four positive integers. The first integer, *row*, represents the starting row in the matrix. The second integer, *col*, represents the starting column in the matrix. The third integer, *r*, represents the number of repairs you can make in this matrix (*r* >= 1); and the fourth integer, *m*, indicates whether you want your total to be increasing or decreasing as you travel the matrix (0=decreasing, 1=increasing). The following 10 lines of input each contain 10 integer values (separated by spaces) that represent the values in the rows of the matrix.

Details of the output

For each case, one line of output is generated. The line begins with the label of the form Case *i*: (*i*=1…c) and then prints the value of the circuit you found for the case.

Sample input

```
2
5 3 2 1
155 -67 -204 -152 204 -5 -185 134 -172 248
52 157 -198 -89 130 -109 -78 -4 114 81
74 211 48 73 62 -166 141 98 114 143
-139 79 79 -81 -12 51 3 149 101 43
44 -173 -146 107 -9 66 -102 17 -74 182
85 -50 -144 160 -94 159 41 187 -56 -40
-211 105 -175 236 -9 -174 39 79 -103 -200
-130 241 168 65 6 141 136 243 -10 -26
10 29 12 -144 -116 20 -247 -57 -58 158
-68 -199 122 236 244 66 104 -101 -183 -89
2 4 19 1
242 -182 184 2 -143 -198 -138 -119 -206 228
-91 -213 148 -186 66 104 -6 250 242 -140
-193 230 18 -151 182 197 166 117 190 -147
163 -103 -110 169 -158 14 87 100 -100 14
-100 1 37 131 60 -41 -35 63 -153 240
76 -30 -96 -56 172 -239 51 85 71 -90
-114 -5 -58 192 -185 -238 148 229 222 -199
-234 144 99 -23 211 57 -57 138 39 -27
-19 -148 20 201 27 -58 244 196 247 -170
214 -248 -221 241 -122 155 -192 96 65 48
```

Sample output

```
Case 1: 564
Case 2: 3198
```

Hop-along Porgs

Porgs are simple creatures with simple desires. To pass the time, porgs created a simple game which they could play almost anywhere. The porgs would go up to a flat cliff and draw a grid. They would then toss a random number of rocks and twigs onto the grid. The porgs would then start at the left most position, and count up the number of rocks and twigs on that position and move to a new square relative to the number of rocks vs twigs. However the porgs who stowed away on the Millennium Falcon no longer have rocks and twigs to play their game, so the porgs have started to bug Chewbacca. In order for Chewbacca to fly the Millennium Falcon in peace, Chewbacca created little flash cards with positive and negative numbers on them as well as a grid on the floor of the Falcon. This way the Porgs can entertain themselves and not bother Chewbacca.

The rules of the revised game are as follows:

- The starting position of the game is the first position of the grid.
- On each turn, the porg looks at the number at the current position of the sequence. If it is negative, the porg moves that many positions to the left, otherwise the porg moves that many positions to the right.
- Repeat the last step until moving will takes the porg beyond the left or right end of the sequence or until the porg realizes it would be stuck forever.

For example, if the starting sequence is

2 -2 -1

the porg will go from the beginning of the sequence (2) to the end of the sequence (-1) to the middle of the sequence (-2) and then off the left end.

If the sequence is

1 2 3 -1

the porg will go from the beginning of the sequence (1), to the second position (2), to the last position (-1), to the third position (3) and then off the right end.

Finally, if the sequence is

3 2 -1 -1

the porg will go from the beginning of the sequence (3) to the end of the sequence (-1), to the third position (-1), to the second positon (2), back to the last position (-1), to the third position (-1), to the second position (2), etc.

Details of the input

The first line of the input will contain a single positive integer, *c*, which specifies the number of cases of the game to follow. For each case, there are two lines of input. The first line contains a single positive integer, *n* ($1 \leq n \leq 100$) indicating the length of the sequence. The second line is the sequence itself; that is, *n* integers that will be in the range $-n$ to *n*.

Details of the output

For each case, you will output one line with the one of the forms:

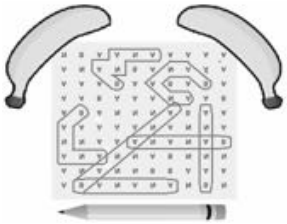
```
Case #: Left
Case #: Right
Case #: Neither
```

Sample input

```
2
3
2 -2 -1
4
1 2 3 -1
```

Sample output

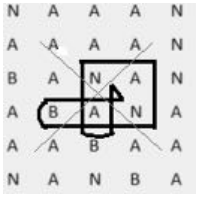
```
Case 1: Left
Case 2: Right
```



Chewy's Super Banana Bonanza

There has been some suspicion that Chewbacca was particularly fond of bananas. Hints of this surfaced periodically. One such instance occurred when Chewy created a word search puzzle, Banana Bonanza, in which the only words to search for were BANANA … and BANANA … and BANANA …

Banana Bonanza is a unique word search puzzle that is only filled with the 3 letters: B, N, and A. The goal of the puzzle is to figure out the total number of times BANANA is spelled throughout the word search. In Chewy's version, one is allowed to spell the word BANANA forwards, backwards, diagonally, as well as to change direction at any point of spelling the word. The only restriction is that the same instance of a letter cannot be used more than once in a single spelling of BANANA. Thus, the only reason the outlined BANANA at right cannot be counted is the fact that the same 'A' is used twice. The goal is to count the total number of unique times BANANA appears according to these rules.



Details of the input

The first line of input consists of a single positive integer, c, representing the number of cases to be processed. For each case, its first line of input contains a single integer, n ($3 < n < 11$). For the next n lines, each input line will be filled with n letters, which are only B,N, or A, representing the letter filled in that position on the n x n puzzle starting from the top left, to the bottom right.

Details of the output

For each case, one line of output is generated. The line begins with a label of the form Case *i*: (*i*=1…*c*) and then prints the number of times BANANA appeared in the puzzle.

Sample input

```
2
4
BANA
NNAN
NNNA
NNNN
7
BBBBBBB
NNNNNNN
NNNNNNN
BANANAB
NNNNNNN
NNNNNNN
BBBBBBB
```

Sample output

```
Case 1: 9
Case 2: 18
```