III MARATON INTERNA DE PROGRAMACION

UNIVERSIDAD KONRAD LORENZ

16 de Mayo de 2008

PROBLEMAS

Elaborado por: Hector Florez
Basado de www.acis.org.co, www.acm.org

Problem 1

Optimal Parking

Source file name: parking.c, parking.cpp or parking.java
Input: parking.in
Output: standar output

When shopping on Long Street, Michael usually parks his car at some random location, and then walks to the stores he needs. Can you help Michael choose a place to park which minimizes the distance he needs to walk on his shopping round?

Long Street is a straight line, where all positions are integer. You pay for parking in a specific slot, which is an integer position on Long Street. Michael does not want to pay for more than one parking though. He is very strong, and does not mind carrying all the bags around.

Input

The first line of input gives the number of test cases, $1 <= t <= 100$. There are two lines for each test case. The first gives the number of stores Michael wants to visit, $1 <= n <= 20$, and the second gives their n integer positions on Long Street, $0 <= X_i <= 99$.

Output

Output for each test case a line with the minimal distance Michael must walk given optimal parking.

Sample input	Sample Output
2 4 24 13 89 37 6 7 30 41 14 39 42	152 70

Problem 2

He is offside!

Source file name: he.c, he.cpp or he.java
Input: he.in
Output: standar output

Hemisphere Network is the largest television network in Tumbolia, a small country located east of South America (or south of East America). The most popular sport in Tumbolia, unsurprisingly, is soccer; many games are broadcast every week in Tumbolia.

Hemisphere Network receives many requests to replay dubious plays; usually, these happen when a player is deemed to be offside by the referee. An attacking player is offside if he is nearer to his opponents' goal line than the second last opponent. A player is not offside if.

- he is level with the second last opponent or
- he is level with the last two opponents.

Through the use of computer graphics technology, Hemisphere Network can take an image of the field and determine the distances of the players to the defending team's goal line, but they still need a program that, given these distances, decides whether a player is offside.

Input

The input file contains several test cases. The first line of each test case contains two integers A and D separated by a single space indicating, respectively, the number of attacking and defending players involved in the play ($2 <= A, D <= 11$). The next line contains A integers B_i separated by single spaces, indicating the distances of the attacking players to the goal line ($1 <= B_i <= 10^4$). The next line contains D integers C_j separated by single spaces, indicating the distances of the defending players to the goal line ($1 <= C_j <= 10^4$). The end of input is indicated by A = D = 0.

Output

For each test case in the input print a line containing a single character: "Y" (uppercase) if there is an attacking player offside, and "N" (uppercase) otherwise.

Sample input	Sample Output
2 3 500 700 700 500 500 2 2 200 400 200 1000 3 4 530 510 490 480 470 50 310 0 0	N Y N

Problem 3

Ball Toss

Source file name: toss.c, toss.cpp or toss.java
Input: toss.in
Output: standar output

Classmates stand in a circle facing inward, each with the direction left or right in mind. One of the students has a ball and begins by tossing it to another student (It doesn't really matter which one). When one catches the ball and is thinking left, he throws it back across the circle one place to the left (from his perspective) of the person who threw him the ball. Then he switches from thinking left to thinking right. Similarly, if he is thinking right, he throws the ball to the right of the person who threw it to him and then switches from thinking right to thinking left.
There are two exceptions to this rule: If one catches the ball from the classmate to his immediate left and is also thinking left, he passes the ball to the classmate to his immediate right, and then switches to thinking right. Similarly, if he gets the ball from the classmate to his immediate right and is thinking right, he passes the ball to the classmate to his immediate left, and then switches to thinking left (Note that these rules are given to avoid the problem of tossing the ball to oneself).
No matter what the initial pattern of left and right thinking is and who first gets tossed the ball, everyone will get tossed the ball eventually! In this problem, you will figure out how long it takes. You'll be given the initial directions of n classmates (numbered clockwise), and the classmate to whom classmate 1 initially tosses the ball (Classmate 1 will always have the ball initially).

Input

There will be multiple problem instances. Each problem instance will be of the form n k t1 t2 t3 … tn where n (2 <= n <= 30) is the number of classmates, numbered 1 through n clockwise around the circle, k (> 1) is the classmate to whom classmate 1 initially tosses the ball, and ti (i = 1, 2, … , n) are each either L or R, indicating the initial direction thought by classmate i (n = 0 indicates end of input).

Output

For each problem instance, you should generate one line of output of the form:
Classmate m got the ball last after t tosses.
where m and t are for you to determine. You may assume that t will be no larger than 100,000. Note that classmate number 1 initially has the ball and tosses it to classmate k. Thus, number 1 has not yet been tossed the ball and so does not switch the direction he is thinking.

Sample input	Sample Output
4 2 L L L L 4 3 R L L R 10 4 R R L R L L R R L R 0	Classmate 3 got the ball last after 4 tosses. Classmate 2 got the ball last after 4 tosses. Classmate 9 got the ball last after 69 tosses.

Problem 4

Electrical Outlets

Source file name: outlets.c, outlets.cpp or outlets.java
Input: outlets.in
Output: standar output

Roy has just moved into a new apartment. Well, actually the apartment itself is not very new, even dating back to the days before people had electricity in their houses. Because of this, Roy's apartment has only one single wall outlet, so Roy can only power one of his electrical appliances at a time.

Roy likes to watch TV as he works on his computer, and to listen to his HiFi system (on high volume) while he vacuums, so using just the single outlet is not an option. Actually, he wants to have all his appliances connected to a powered outlet, all the time. The answer, of course, is power strips, and Roy has some old ones that he used in his old apartment. However, that apartment had many more wall outlets, so he is not sure whether his power strips will provide him with enough outlets now.

Your task is to help Roy compute how many appliances he can provide with electricity, given a set of power strips. Note that without any power strips, Roy can power one single appliance through the wall outlet. Also, remember that a power strip has to be powered itself to be of any use.

Input

Input will start with a single integer 1 <= N <= 20, indicating the number of test cases to follow. Then follow N lines, each describing a test case. Each test case starts with an integer 1 <= K <= 10, indicating the number of power strips in the test case. Then follow, on the same line, K integers separated by single spaces, O1 O2 ... O_k, where 2 <= O_i <= 10, indicating the number of outlets in each power strip.

Output

Output one line per test case, with the maximum number of appliances that can be powered.

Sample input	Sample Output
3 3 2 3 4 10 4 4 4 4 4 4 4 4 4 4 4 10 10 10 10	7 31 37

Problem 5

The next round number

Source file name: roundn.c, roundn.cpp or roundn.java
Input: roundn.in
Output: standar output

An N-digit *round number* is characterized as follows:

- It is an integer with exactly N digits, each of which is between 1 and 9, inclusively.
- The digits form a sequence with each digit telling where the next digit in the sequence occurs. This is done by giving the number of digits to the right of the digit where the next digit in the sequence occurs. If necessary, counting wraps around from the rightmost digit back to the leftmost.
- The leftmost digit in the number is the first digit in the sequence, and the sequence must return to this digit after all digits in the number have been used exactly once.
- No digit will appear more than once in the number.

For example, consider the number 81362. To verify that this is a round number, we use the steps shown below:

1. Start with the leftmost digit, 8: 8 1 3 6 2
2. Count 8 digits to the right, ending on 6 (note the wraparound): 8 1 3 6 2
3. Count 6 digits to the right, ending on 2: 8 1 3 6 2
4. Count 2 digits to the right, ending on 1. 8 1 3 6 2
5. Count 1 digit to the right, ending on 3. 8 1 3 6 2
6. Count 3 digits to the right, ending on 8, where we began. 8 1 3 6 2

Given a positive integer R, you must determine the smallest round number that is equal to or greater than R.

Input

You will be provided with one or more input lines, each with a single integer R having between 2 and 7 digits followed immediately by the end of line. The last line of the input will contain only the digit 0 in column 1.

Output

For each input number, determine the smallest round number that is equal to or greater than R. There will always be such a number for each of the input numbers. Display the resulting number in the format illustrated below.

Sample input	Sample Output
12 123 1234 81111 82222 83333 911111 7654321 0	Case 1: 13 Case 2: 147 Case 3: 1263 Case 4: 81236 Case 5: 83491 Case 6: 83491 Case 7: 913425 Case 8: 8124956

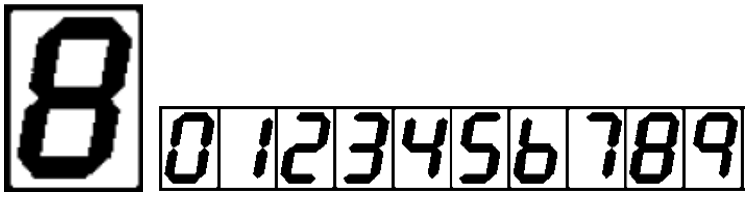
Problem 6

7 segments

Source file name: segments.c, segments.cpp or segments.java
Input: segments.in
Output: standar output

A 7 segments display is a component built whit diodes LED (Light emisor diode) which allow to show decimal numbers.

Next figure shows a 7 segments display and then shows the representation of each digit.



Input

Input consists in several cases. Each one, contain a decimal number and other decimal number between 1 and 5 separated by a space.

Output

For each case, you must print the first number with the number of characters determined by the second number for each segment.

Sample input	Sample Output

3 3
8 2
	` ---`
	`    |`
	`    |`
	`    |`
	` ---`
	`    |`
	`    |`
	`    |`
	` ---`
	` --`
	`|  |`
	`|  |`
	` --`
	`|  |`
	`|  |`
	` --`