18^{th} South African Regional ACM Collegiate Programming Contest

Sponsored by IBM

15 October 2016

Problem A — Orange Balloon
Rice coding

Problem Description

Variable length prefix-free codes are a great way to reduce the storage space required to store integers that have an unequal frequency distribution across a large range, i.e., if we expect small numbers to appear more frequently than larger numbers. Rice coding is a particular variable-length coding scheme that is both simple, and efficient to implement. A Rice code, parameterized by a *Rice parameter* k, comprises a *unary* part, and a *binary* part.

The *unary representation* of the non-negative integer i is just a sequence of i zero (0) bits followed by a one (1) bit, e.g., 5 is encoded as 000001_2 (the subscript "2" denotes the value is represented in base 2). The length of the unary encoding of i is $i + 1$ bits. Note that a unary code can be uniquely decoded by simply counting the number of leading 0 bits; a binary-coded integer cannot be parsed without knowing its length in advance.

To obtain the Rice coding of an integer i, we compute the quotient

$$q(i, k) = \left\lfloor \frac{i}{2^k} \right\rfloor \tag{1}$$

where k is the Rice parameter. The unary part of the Rice code is then the value $q(i, k)$ encoded in a unary representation. Next, we compute the remainder

$$r = i - 2^k q(i, k)$$

which we can then encode as a binary representation in exactly k bits, which forms the binary part of the Rice code.

Let $L_k(i)$ denote the length (in bits) of the Rice encoding of the non-negative integer i using a Rice parameter of k, then

$$L_k(i) = q(i, k) + 1 + k.$$

Negative integers can be accommodated by the mapping function U,

$$U(i) = \begin{cases} -2i - 1 & \text{if} \quad i < 0 \\ 2i & \text{if} \quad i \geq 0, \end{cases}$$

so that the Rice coding length of a signed integer i is denoted $L_k(U(i))$.

If we want to encode 16-bit integers using Rice coding with a parameter $k = 3$, we find that $L_3(32765) = 4099$ bits, as opposed to the 16 bits we started with. To prevent unexpectedly large values (compared to our typical values) from producing such impractically long codes, we limit the value of $q(i, k)$ to an upper limit of 8. Whenever $q(i, k) \geq 8$, we output the value 8 in its unary

representation, followed by the 16-bit binary representation of i, resulting in a length of $8 + 1 + 16$ bits. The length of this modified Rice code is thus:

$$M_k(i) = \begin{cases} L_k(U(i)) & \text{if} \quad q(U(i), k) < 8 \\ 8 + 1 + 16 & \text{if} \quad q(U(i), k) \geq 8 \end{cases} \tag{2}$$

Given a sequence of signed 16-bit integers, your task is to calculate the minimum possible length (in bits) of this sequence encoded using Rice coding where the Rice parameter k may be selected from the range $[0, 14]$. Note that a single value of k must be selected for the entire sequence.

Input

Your input consists of an arbitrary number of records, but no more than 100. Each record starts with a line containing a single integer n, with $1 \leq n \leq 5000$, denoting the number of integers to follow. The next line contains n integers s_i separated by spaces, subject to $-32768 \leq s_i \leq 32767$.

The end of input is indicated by a line containing only the value -1.

Output

For each input record, output the minimum number of bits required to encode the sequence of integers using Rice coding with a Rice parameter k selected from the range $[0, 14]$, with individual Rice code lengths as defined in Equation (2).

Sample input

```
13
0 -1 -13807 -1 0 0 0 -2 0 -1 0 -1 0
16
1 -9 -7 -25 19 -2 -24 3 -60 -19 -27 -8 80 188 -17 -25
-1
```

Sample output

```
44
120
```

Time limit

5 seconds

18^{th} South African Regional ACM Collegiate Programming Contest

Sponsored by IBM

15 October 2016

Problem B — Pink Balloon Scrunched

Problem Description

You are faced with a challenge of epic proportions: a curtain manufactured from a given width of fabric is to be fitted to a window of a known width, subject to the constraint that the curtain must have an exact, specified number of folds. Since this is a curtain manufactured to superior standards, it is guaranteed to naturally fold along a perfect sinusoidal curve.

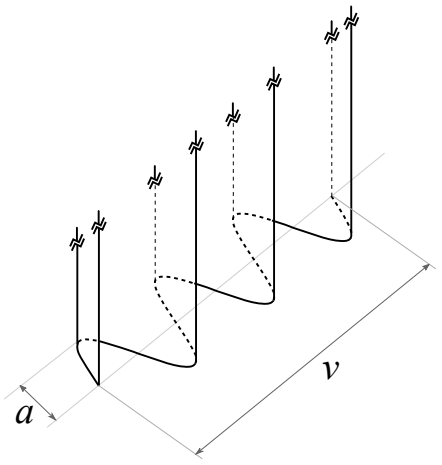


Figure 1: Curtain geometry. Note that this particular example illustrates 3 folds in a window width of v.

What should the amplitude of the folds be in order to produce the required number of folds at the desired scrunched width?

Input

Your input consists of an arbitrary number of records, but no more than 40. Each record comprises three values appearing on a single line: The actual width of the fabric $w \in \mathbb{R}$, with $1 < w \leq 150$; the target width of the scrunched curtain $v \in \mathbb{R}$, with $0.5 < v \leq 10$, subject to $w - v > 0.00004$; the desired integer number of folds $n \in \mathbb{N}$, $2 \leq n \leq 30$.

The end of input is indicated by a line containing only the value -1.

Output

For each input record, output the amplitude of the sinusoidal folds required to produce n folds in a width of v using a curtain made using fabric with a of width w. The amplitude, a, is defined such that the folds form a curve in the shape $a\sin(\frac{2\pi n}{v}x)$, with $0 \leq x \leq v$, and $a > 0$.

Your program must output the value of a, truncated to a multiple of 0.001, and formatted to three digits after the decimal period.

Sample input

```
10.01775695 7.0 11
7.64279256 6.28318530 10
-1
```

Sample output

```
0.152
0.100
```

Time limit

15 seconds

18^{th} South African Regional ACM Collegiate Programming Contest

Sponsored by IBM

15 October 2016

Problem C — Green Balloon
Keeping your balance

Problem Description

A researcher at Impossible Bionic Machines research lab manages a number of particularly nasty underlings. They do not mind working long hours on any particular day (and will continue until the work is done). They do, however, revolt and start demolishing infrastructure should they find out that another underling has done even one task less than themselves. For reasons beyond understanding, the complexity of tasks does not seem to matter, only the number.

It frequently happens that one or more of these underlings are not available, and their work has to be re-assigned to alternative underlings. If this redistribution of work results in an uneven spread of tasks, revolt still happens. To avoid this, the researcher must ensure that the number of available tasks can be evenly distributed even if some underlings are unavailable. Tired of having his laboratory disassembled, he has handed you the problem of calculating the minimum number of tasks he needs to prepare.

You will be given the number N of underlings and the maximum number A that might be absent on any given day. You must find the smallest positive integer M which is an exact multiple of $N - i$, for every i between 0 and A inclusive.

Input

Input consists of an arbitrary number of records (but no more than 20), where each line represents a record and contains two values, N (the number of underlings) and A (the number of underlings which can potentially be absent), with $0 \leq A < N$. It is guaranteed that the inputs will be such that $M < 2^{63}$ (use the `long` type in both C/C++ and Java).

The end of input is indicated by a line containing only the value -1.

Output

For each input record, output a line containing the minimum number of tasks for the researcher to prepare, as described above.

Sample input

```
6 3
4 3
4 2
-1
```

Sample output

```
60
12
12
```

Time limit

1 second

18^{th} South African Regional ACM Collegiate Programming Contest

Sponsored by IBM

15 October 2016

Problem D — Yellow Balloon Folding Paper

Problem Description

Have you ever noticed how difficult it is to fold up the piece of paper with the instructions once you've unfolded it? You've decided to do something about it by writing a program to determine the best way to refold a piece of paper. You have decided to start with the simplest case, a page with regularly spaced vertical creases. Each of these creases forces the page to bend in a particular direction along the crease.

An example of a fold is shown in Figure 1. A crease is selected and the paper is folded over along this crease. Where this causes other creases to overlap, they must be oriented in the same direction. The folded page can then be folded further. As before, every fold must be along an existing crease and where this causes creases to overlap they must have consistent directions.

Your goal is to determine the minimum number of folds required to completely fold up the piece of paper. In the example of Figure 1, two further folds are needed (along the two remaining creases) to completely fold the page.

Input

The input consists of an arbitrary number of records, but no more than 20. Each record starts with a line containing an integer N ($1 \leq N \leq 5\,000$), the number of creases. The second line contains a string with N characters, each of which is either 0 or 1. These correspond to the direction of the creases going left-to-right across the page, with a 0 or 1 indicating a forwards or backwards crease, respectively.

The end of input is indicated by a line containing only the value -1.

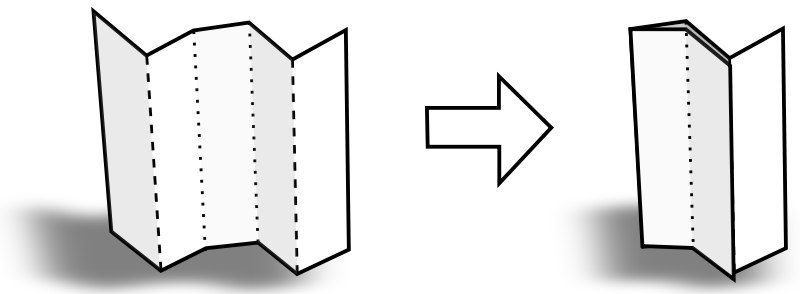


Figure 1: Left: unfolded instructions with creases. Right: The two left panels have been folded to the right.

Output

For each input record, output a line containing the minimum number of folds required for that record.

Sample input

```
4
1001
3
101
-1
```

Sample output

```
3
3
```

Time limit

15 seconds

18^{th} South African Regional ACM Collegiate Programming Contest

Sponsored by IBM

15 October 2016

Problem E — Purple Balloon Powers

Problem Description

Given a number n, solve the equation $n = a^b$ where a and b are positive integers and b is as large as possible.

Input

The input consists of an arbitrary number of records, but no more than 20. Each record is a line containing a positive integer n ($2 \leq n \leq 10^9$).

The end of input is indicated by a line containing only the value -1.

Output

For each input record, output a line of the form `a^b`, where $n = a^b$ and b is as large as possible.

Sample input

```
17
25
6561
-1
```

Sample output

```
17^1
5^2
3^8
```

Time limit

5 seconds

18^{th} South African Regional ACM Collegiate Programming Contest

Sponsored by IBM

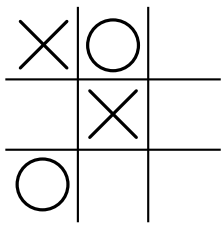
15 October 2016

Problem F — Red Balloon Generic-mint-toe

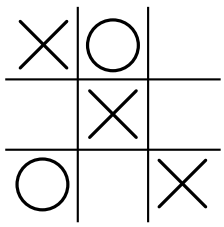
Problem Description

The game of generic-mint-toe is identical to the game tic-tac®-toe, but it manages to avoid trademark infringement through gratuitous punning. The game is played on a 3×3 grid that is initially empty. Play proceeds by each of the players, denoted symbols 'X' and 'O', alternately placing their symbol in an empty grid position.

Players win by forming a row, column or diagonal of length three of their symbol. If all nine of the grid positions are filled without a player winning, the game is said to be tied.



(a) Sample unfinished game state



(b) After X played. X wins.

Figure 1: An example of a valid move played by 'X'.

Assume that you represent the 'X' player. You are given a partially played game board such that the next move must be played by 'X'. Your task is to count the number of grid positions that you can play in this turn such that you can guarantee that the 'O' player does not win at the end of the game, no matter how poorly 'X' plays after this next move. You may assume that the 'O' player will play a perfect game from this point on, i.e., the 'O' player will *always* play a winning sequence of moves if at all possible, and failing that the 'O' player will play for a tie rather than allowing 'X' to win.

Input

Your input consists of an arbitrary number of records, but no more than 30. Each record comprises three lines, which each contains a string of exactly three characters from the set {'X', 'O', '.'}. A blank line follows the third line of each record.

The three lines of three characters represent the three rows of the game board state, in left-to-right, top-to-bottom order. The symbol '.' (a full-stop) represents an empty grid position; the symbols 'X' (uppercase *ex*) and 'O' (uppercase *oh*) represent plays by the indicated player.

The input is guaranteed to represent a valid game state, i.e., if 'X' started the game, then the number of 'X' and 'O' moves will be equal, and if 'O' started, then 'O' will have played one more move than 'X'. The game state is also guaranteed not to reflect a game that is already tied or won by one of the players.

The end of input is indicated by a line containing only the value −1.

Output

For each input record, output the number of grid positions in which 'X' can play in the current turn according to the conditions defined in the Problem Description section above.

Sample input

```
XO.
.X.
O..

X..
OOX
X.O

-1
```

Sample output

```
2
3
```

Time limit

5 seconds

18^{th} South African Regional ACM Collegiate Programming Contest

Sponsored by IBM

15 October 2016

Problem G — Blue Balloon
Talk in numbers

Problem Description

You came across an old manuscript written entirely in numbers and, thankfully, the instructions to decode it were included, but obviously not the means — it is an old manuscript after all and you would rather not decipher the document by hand.

Each word has essentially been sorted first by length then alphabetically such that "a" is the first word followed by "b", etc. Only the lowercase letters a–z are allowed.

The decoding instructions that came with the manuscript included this subset of words as an example:

```
a       1
b       2
c       3
...
z       26
aa      27
ab      28
...
zz      702
...
hello   3752127
...
```

Input

The input consists of an arbitrary number of lines, but no more than 200. Each line contains a single number, which will always be less than 2^{63}.

The end of input is indicated by a line containing only the value -1.

Output

For each input line, output a line containing the word that corresponds to the input number.

Sample input

```
13733
8143861
1241072
4470
4948079
278660
13733
212289
3101
-1
```

Sample output

```
the
quick
brown
fox
jumps
over
the
lazy
dog
```

Time limit

5 seconds

18^{th} South African Regional ACM Collegiate Programming Contest

Sponsored by IBM

15 October 2016

Problem H — White Balloon Juggling Jugs

Problem Description

Dr Evil is a big Die Hard movie fan and is especially keen on enhancing the diabolical problem in which John McClain needed to get a specific quantity of water into a water jug.

A water jug problem works as follows. John is provided with some number of empty jugs, each with a unique, specified capacity. He also has access to a water fountain which he can use to fill the jugs. He must put an exact quantity of water into one of the jugs. He has no other measuring devices, so he must do this using only a sequence of the following types of steps:

1. fill a jug to the brim from the fountain;
2. empty a jug;
3. pour water from one jug to another, until either the first jug is completely empty or the second jug is completely full.

To reduce the chance of John surviving such a riddle again, Dr Evil will provide John with multiple such problems. What's more, at least one third of the problems will be unsolvable, and it is imperative that John avoids wasting time on those so that he can solve the rest before the bomb explodes.

Fortunately for John, Dr Evil has a traitor in his midst who has advised John of his sinister plan. The police department has contracted you to provide John with some assistance. For each problem posed by Dr Evil, you must determine whether it is solvable, and if it is, the least number of steps required to solve it.

Table 1 shows an example of the steps needed to obtain a quantity of 4 litres using two jugs with capacities of 3 and 5 litres. In this case it took six steps to get a quantity of 4 litres in one of the jugs.

Step	Jug 1	Jug 2	
	0	0	
1	0	5	fill jug 2
2	3	2	transfer jug 2 to jug 1
3	0	2	empty jug 1
4	2	0	transfer jug 2 to jug 1
5	2	5	fill jug 2
6	3	4	transfer jug 2 to jug 1

Table 1: Solution to the first sample case

Input

The input consists of an arbitrary number of records, but no more than 30.

Each line of the input describes one problem. It consists of the number of jugs, n ($2 \leq n \leq 5$), the required quantity (in litres), r ($0 \leq r \leq 30$), followed by n numbers indicating the jug capacities (also in litres), $j_1 \ldots j_n$ ($1 \leq j_i \leq 30$). These values are all whole numbers and separated by spaces, and within each problem, the jug capacities are distinct.

The end of input is indicated by a line containing only the value -1.

Output

For each water jug problem, output the least number of steps needed to get to the required quantity. If it is impossible to measure out that amount, output -1 instead.

Sample input

```
2 4 3 5
2 1 2 3
2 3 4 6
3 17 7 11 23
5 23 1 4 5 9 22
-1
```

Sample output

```
6
2
-1
7
-1
```

Time limit

30 seconds