
**TASK 1**

Let us analyze magical strings that contain only capital letters M and G. Determine the maximum number of magic strings that have no more than **n** letters.

Input
From the first row of standard input, read an integer **n** (1≤n<56).
Output
Print to standard output the number of magic strings that have no more than **n** letters.

Example 1
Input

```
1
```

Output

```
2
```

Clarification
Magical strings: M, G,
Example 2
Input

```
2
```

Output

```
6
```

Clarification
Magical strings: M, G, MG, GM, MM, GG

**SOLUTION**

There are 2 magical strings of the length 1 (M and G).
There are 4 magical strings of the length 2 (GG, GM, MG and MM).
There are 8 magical strings of the length 3 (GGG, GGM, GMG, GMM, MGG, MGM, MMG, MMM).
For each concatenation of one letter, the number of magical strings is doubled. It is easy to prove: to any number of the previous length one can append letter M or G, so one magical string of the prevous length creates two magical strings of the next length.

So for the length n, the total amount of magical strings of the length exactly n is $2^n$. But in our task, we need to determine the total number of magical strings of length not greater than n.
So, let's sum them up,
$2^1 = 2$, $2^1 + 2^2 = 2 + 4 = 6$, $2^1 + 2^2 + 2^3 = 2 + 4 + 8 = 14$,

$2^1 + 2^2 + 2^3 + 2^4 = 2 + 4 + 8 + 16 = 30$.
You may notice that the sum of all previous powers of two is equal to the next power of two minus the first power of two. So the answer to the problem is $2^{n+1} - 2$.

One of the possible implementations of $2^{n+1}$ in a programming language is to shift 1 bitwise to the left for n + 1 binary digits or to shift 2 bitwise to the left for n binary digits. For example, in Java the problem answer formula is (2L << n) - 2, in C++ is (2LL << n) - 2. Suffixes L and LL correspondingly are required so that result of the shift expression have 64-bit integer type (long in Java and long long in C++).

**TASK 02**

Let us notice the two integers x and b $(1 \le x \le 10^{100}, 2 \le b \le 100)$.
It is possible to uniquely define an integer y, such that: $b^y \le x < b^{y+1}$
Write a program that determines the number y.
Input
From the first row of standard input, read an integer x.
From the second row of standard input, read an integer b.
Output
Your program should print one integer – described number y without leading zeros.

Example 1
Input
```
32
2
```
Output
```
5
```
Example 2
Input
```
2
10
```
Output
```
0
```
Example 3
Input
```
123458989131512359209000000000000000000000
90
```
Output
```
21
```

**SOLUTION**

In order to solve the problem, it is necessary to realize a big number arithmetic.

Solution 1: Let us implement two operations - multiplying a big number with a short one and comparing two long numbers.
But, according to time limit, there is no need to focus on brute force solution (raise the number b to the power of 1, 2,. . . , y until the number $b^y$ becomes greater than a. Then it is obvious that y - 1 is the answer to the problem).

But it is also possible to find the solution more quickly.
Let's try to evaluate the result from above. For a given b it is clear that there exists an integer k such that $b^k \le 10 < b^{k+1}$
We notice that x is always strictly less than $10^n$, where n is the length of the number x in the decimal notation. Then, therefore, the required number y is not greater than n * k.

Solution 2:

```
    int b;
    string x;
    cin >> x >> b;
    if(x.size() == 101){
        cout << floor(double(100)/log10(b));
        return(0);
    }
    if(x.size() < 21){
     unsigned long long c = stoull(x);
    cout<< floor(log10(c)/log10(b));
    }
    else
        cout << floor(double(x.size())/log10(b)- 0.000000001);
}
```

**TASK 03**

Let us notice the directed graph with N vertices and N edges (1<N<100001). Write a program that determines how many colors you need to assign to the edges of a graph so that for each node adjacent edges are assigned with a different color. Assume that there is no node in the graph with an input degree greater than 1.

Input
From the first row of standard input, read an integer T - the number of test cases (0 < T < 11). followed by T pairs of rows at standard input.
There are two rows entered for each graph, with the first row being the integer N - the size of the corresponding graph, and the second row - N numbers Ai (0<Ai<N+1,Ai≠i) in the array separated by whitespace character.
Each number Ai, (given at the i-th position) indicate that the graph has an edge directed fromthe vertex Ai to vertex i (i=1,2,..,.N).
Output
In the T line, print an integer - minimum number of colors for each graph..

Example
Input
3

5

3 1 2 1 1

5

4 1 2 3 4

4

4 1 2 3

Output
4

3

2

Clarification
There are three graphs described in test:
The first graph should be colored with 4 colors (following the conditions described in the task), because 4 colors should be assigned when we analyze the total number of different colors in relation to vertex 1.
There are 5 edges in the second graph (4, 1), (1,2), (2,3), (3, 4), (4,5). That graph should be colored with 3 colors (following the conditions described in the task).
There are 4 edges in the third graph (4, 1), (1,2), (2,3), (3, 4). That graph should be colored with 2 colors (following the conditions described in the task).

**SOLUTION**

If there is a cycle of odd length, then for the belonging edges, the answer is 3 colors.
For the other vertices, the answer is the maximum number of edges adjacent to them.

**TASK 4**

Write a program that finds the number of the shortest routes between two cities.
Input
From the first row of the standard input read two integers - the number of cities (N) and the number of roads (M) that connect directly
two different cities (in both directions). The cities are numbered from 1 to N ($0<N<560$).
Each of the following M lines contain a description of one road with the numeric labels of the two cities it connects and the length. Road lengths are positive integers not exceeding 100.
In the next line, read a positive integer Q ($Q \leq N(N-1)/2$). After that, read Q lines that describe Q queries.
There are two numbers entered for each query. Both numbers indicate the pair of cities for which the program must find the number of the shortest routes.

Output
In the only row of standard output (for each of the given Q pairs of cities)
the program must display the number of shortest
routes between the given two cities (answer the result in 10^9+21 modulo). Print a whitespace character between every two numbers.

Example
Input

```
5  10
1 2 3
1 3 3
1 4 2
1 5 2
2 3 3
2 4 3
2 5 3
3 4 3
3 5 1
4 5 2
4
1 5
3 1
2 4
3 4
```

Output

```
1 2 1 2
```

**SOLUTION**

We may solve the problem by modifying Floyd-Warshall algorithm.

In addition to the matrix W (if we choose to graph in the form of an adjacent matrix and in which the Floyd algorithm finds the lengths of the shortest paths from each vertex to every other vertex), we may use another matrix P in which for every two vertices we find the number of the shortest paths from each vertex to any other vertex.

When algorithm establishes the validity of triangle inequality:
if (W [i] [k] + W [k] [j] <W [i] [j])
it indicate that one or more shorter paths have been found between vertices i and j (than the one whose length is remembered so far) passing through vertex k.

Therefore, in addition to replacing the old stored length with the new one:
W [i] [j] = W [i] [k] + W [k] [j];

we must store in P[i] [j] the number of these shorter paths.

Since the number of the shortest paths from i to k is stored in P [i] [k] and the same is valid for P [k] [j], then the number of the shortest paths from i to j will be the product of P [i] [k] and P [k] [j]:
P [i] [j] = P [i] [k] * P [k] [j]

Unlike Floyd-Warshall original algorithm, however, here we need to check another possibility:
if (W [i] [k] + W [k] [j] == W [i] [j])
and if this condition is fulfilled, to the shortest paths already found, passing only through vertices with numbers smaller than k, the length of which is stored in W [i] [j], the new roads with the same must be added. length passing through k:
P [i] [j] + = P [i] [k] * P [k] [j]