Problem A. 1024 Stack Edition

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

First consider the case when $N = 0$. Let $f(i)$ be the expected minimal number of coins needed to construct 2^i. Then in case when $p < 100$ we can use the following formulae to calculate $f(i)$:

1. $f(0) = 1/p - 1$;
2. $f(1) = min(f(0) \cdot 2,\ 1/(1-p) - 1,\ 1 - p)$;
3. $f(i) = 2 \cdot f(i-1),\ i > 1$;

These formulae can be easily tuned for the case when $p = 100$. Thus, if $N = 0$ the answer will be $f(10)$.

Now consider $N > 0$.

Let $g(i, j)$ be the expected minimal number of coins needed to construct only one number 2^j from the top i numbers of the stack. To count $g(i, *)$ for fixed i, we will use the following 2-step procedure; here $S(i)$ stands for the binary logarithm of the i-th number from the top of stack.

1. Calculate $g(i, S(i)) = 1 + \min(g(i-1, *))$ for $i > 1$. This represents a situation when we construct some number from the topmost $i-1$ one and then delete it. Also calculate $g(i, S(i)+1) = g(i-1, S(i))$. This formula represents a situation when we construct a number $2^{S(i)}$ from the topmost $i-1$ one and then merge it with the current number, which is also equal to $2^{S(i)}$. For all j other than $S(i)$ and $S(i)+1$, let $g(i, j) = +\inf$.
2. On the previous stage we've processed the number $S(i)$. Before switching to $g(i+1, *)$ and the number $S(i+1)$ we can do something with the i'th number on the stack, constructing some other number from it. So we should recalculate $g(i, *)$ in the following way:
   - $g(i, 0) = \min(g(i, 0), \min(g(i, *)) + 1/p)$, i.e. we either use $g(i, 0)$ or delete the current number and put $1 = 2^0$ instead.
   - $g(i, j) = \min(g(i, j), \min(g(i, *)) + 1 + f(j), g(i, j-1) + f(j-1))$ for $j > 0$. Here we can either delete the i'th number and replace it with 2^j using $f(j)$ coins or use $g(i, j-1)$ steps to create one 2^{j-1}, then $f(j-1)$ steps to create another 2^{j-1} and then merge them.

The answer to our problem will be equal to $g(N, 10)$. Note that during the process of calculation of $g(*, *)$, we need to calculate $g(*, 11)$ as well. The complexity of our solution is $O(NR)$ where R is the highest power of two present in the input data (due to statement limitations $R \leq 10$).

Problem B. String Problem

Input file:	*standard input*
Output file:	*standard output*
Time limit:	4 seconds
Memory limit:	512 mebibytes

Consider pair of strings s and w. String s is *almost less* than string w when and only when string s^{min} (lexicographically minimal string which can be obtained from s by replacing exactly one letter) is strictly less than w.

But it is easy to see that s^{min} can be obtained from s in the following way: replace first character not equal to "a" with character "a"; if s contans only "a"s, then $s = s^{min}$,

Consider string s_i and then lets find C_i: the number of strings s_j (possibly $i = j$) such as $s_i^{min} < s_j$. It adds to answer $C_i - 1$, if $s_i^{min} \neq s_i$, and C_i otherwise.

To calculate values of C_i two ways can be used:

- sort all s_i and then use binary search to find C_i;
- add all strings to a *trie* and then find C_i by descending on it.

Problem C. Dice

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

The key point in any solution is the following: hexomino can be folded in two ways ("inside" and "outside"), and it will generate two dice which can be converted one into another by swapping two numbers fitting to any pair of opposite faces. Taking into account that rotation of the cube by 180 degrees swaps numbers on two pairs of opposite faces, we can say that we do not need to consider more than 2 orientations.

After making the above observation, the problem may be solved in several ways.

One of them is the following.

1. Put the cube on the hexomino such as the bottom face of the cube will stay at the cell labeled by the same number as this face.
2. For any of four possible rotations of the cube, try to roll it to all other elements of hexomino; the cube can be rolled on some cell if and only if this cell is labeled by the same number as the bottom face of the cube after the roll.
3. If at least for one rotation all 6 cells of hexomino are visited, the answer is "`Yes`".
4. Otherwise, we will swap numbers between left and right faces and then repeat from step 2. If the answer "`Yes`" is not reached, then the answer is "`No`".

Another way is to precalculate all 11 unfoldings of the cube and then to rotate the given hexomino; if for some rotation or mirroring, the shapes coincide, check if pairs of numbers on opposite sides fit into appropriate cells in the unfolding.

Problem D. UFO

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

This is easiest problem of this problemset.

It is obvious that the number of dimensions of the parallelepiped is not less than the number D of distinct lenghts of given sticks. Also note that K-dimensional parallelepiped contains 2^{K-1} edges of the same type (i.e. parallel to same coordinate axis), and in the case that we have X sticks of the same length, $[(X-1)/2^{K-1}]+1$ dimensions must be used to have X or more edges.

So we will let $K = D$ and sum these values for all sets of sticks of same size. If the sum is less than K, the answer is found, else increase K and calculate the sum again. Note that for $K = 21$, the number of edges of the same type is equal to $2^{20} > 10^6$, so the number of steps in our algorithm will be very small.

Problem E. Unextendable Matching

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

To solve this problem, we can notice that the number of vertices on the left side is small. We can calculate the recursion $f(i, j)$: the number of ways to assign the edges if we have already taken the first i vertices on the right side; then j is a mask in 3-base system describing the vertices on the left side in the next way:

- 0 means a vertex couldn't be assigned to any vertex on the right side,
- 1 means a vertex could be assigned to some vertex on the right side, but wasn't and still is not assigned to any vertex on the right side,
- 2 means a vertex is connected with edge to some vertex on the right side.

When we take the next vertex from the right side, we can calculate the new 3-based mask for the vertices on the left side. To find the answer, we should take only the masks without 1 in then (there shouldn't exist a vertex that could be assigned to some other vertex). So the answer is the sum of $f(n_2, ValidMasks)$, where $ValidMasks$ is an element of the set of all masks which don't contain 1 in their 3-based representation.

Problem F. Musical World

Input file:	*standard input*
Output file:	*standard output*
Time limit:	1 second
Memory limit:	512 mebibytes

Author's solution is based on idea of the *generating function*. If you are not familiar with this idea, you can read, for example,

`http://en.wikipedia.org/wiki/Generating_function`

Let Y_i be the random variable equal to number of tracks in i-th generation.

Consider generating functions of random variables $\psi_i(x) = \sum_{j=0}^{K_i} p_{ij}x^j$ given in the input and generating functions of random variables Y_i $\varphi_i(x) = \sum_{j=0}^{\infty} \mathbb{P}\{Y_i = j\}x^j$.

We have ψ_i in explicit form, and we know that $\varphi_1(x) = x$. Let us try to express φ_{i+1} using ψ_i and φ_i.

In case when $Y_i = k$, $\varphi_{i+1}(x) = \psi_i(x)^k$. But, because variable Y_i is random, the required generating function is weigted the sum of generating functions for differrent values with weights equal to probabilities of thise values, i.e.

$$\sum_{j=0}^{\infty} \mathbb{P}\{Y_i = j\}\psi_i(x)^j = \varphi_i(\psi_i(x)).$$

Because the required value can be very big, it is impossible to find the generating function for $Y_(n+1)$ explicitly. But let us take into account that

$$\mathbb{E}\frac{Y_i(Y_i-1)}{2} = \sum_{j=0}^{\infty} \mathbb{P}\{Y_i = j\}\frac{j(j-1)}{2} = \frac{1}{2}\varphi_i''(1).$$

Let us express the value and also first and second derivatives of $\varphi_{i+1}(x)$ from (1) using previous functions. We know that $\varphi_i(1) = \psi_i(1) = 1$ and $\varphi_{i+1}(x) = \varphi_i(\psi_i(x))$. Then

$$\varphi_{i+1}'(x) = \varphi_i'(\psi_i(x))\psi_i'(x) \text{ and}$$

$$\varphi_{i+1}''(x) = \varphi_i''(\psi_i(x))\psi_i'(x)^2 + \varphi_i'(\psi_i(x))\psi_i''(x).$$

After substitution of $x = 1$, those equations can be simplified to the following form:

$$\varphi_{i+1}'(1) = \varphi_i'(1)\psi_i'(1) \text{ and}$$

$$\varphi_{i+1}''(1) = \varphi_i''(1)\psi_i'(1)^2 + \varphi_i'(1)\psi_i''(1).$$

Also is easy to see that all calculations can be done using modulo 10^9+7, replacing the given probabilities with appropriate values using this modulo.