GLASNICI

Let us first find an algorithm that, for a given time T (in seconds), checks if all messengers can get the news in the given time.

We will first show that there exists an optimal sequence of moving and shouting in which all shouting is done at the very end (after T seconds). Each sequence of moving and shouting can be modified so that messenger A, after he would shout the news to messenger A+1, continues following his movement exactly. That way, all shouting can be done at the end as well.

The algorithm to check if the news can spread in T seconds now becomes simple.

The first messenger runs T distance units towards the second village.

The second messenger knows exactly which space interval he must be in after T seconds to hear the news, and he always tries to go as far as possible from the first village.

The process continues for all messengers.

If, for any messenger, there is no way for him to be in the range of hearing his predecessor, then T seconds is not enough time to spread the news.

The output is found by binary searching over T.

KOLEKCIJA

Notice first that the order of songs Igor wants to listen to is irrelevant (except for output order) so we can sort the input songs. Obviously, the songs displayed on screen will only change forward.

Suppose that the first A songs in the sorted list have already been played. Let B be the index of the first songs not on screen. This means that the list doesn't need to change while playing songs A+1 to B−1. It can be proven that there exists an optimal sequence in which song B is either on top or on the bottom of the screen while playing.

Using the previous claim we construct a recursive algorithm rec(A, isontop) that calculates how many files still need to be read if the first A songs in the sorted list have been played, and with the variable "isontop" telling us if A was on top or on the bottom of the list while playing.

```
rec( int A, bool isontop )
    If isontop is true:
        B = index of the first song not on screen while A is on top
        last = x[A] + K−1
    else:
        B = A+1
        last = x[A]
    If B > M then return 0
    return min{ K + rec( B, false ), x[B] −last + rec( B, true ) }
```

For the algorithm to be efficient, memoization is needed and we need to calculate in one preprocessing pass for each song A the first song off the screen when A is on top.

TAMNICA

For every room R, let the "parent" room be the room before R such that there could be a wall between R and the room. For example, the parent of 8 is 1, the parent of 16 is 5 etc. Not all rooms have parents, 5 and 26 are examples of this. The first part of any solution is to find a way to determine the parent of a given room, since this is required to parse the input.

There is more than one way to do this, presumably all of which involve some math on the room numbers to achieve complexity better than O(R).

Here is a solution of complexity O(log R). Let corner(n) be the n-th "corner" room (corner room 0 is room 1; the others are 2, 3, 5, 7, 10, 13, 17 etc.). It can be shown that corner(n) = (n/2+1) * (n/2 + n mod 2) + 1, with division rounding down. We can use binary search to find the largest n such that corner(n) is less than R. The parent of R is then R − (corner(n)−corner(n-4)) − 1.

We can also calculate the parent room in O(1). If we group the rooms into groups of sizes 2, 4, 6, 8, 10 etc., then group 2n+1 begins with room n^2+n+1. From this it is rather obvious that room x is in group floor((−1 + sqrt(4x−3))/2), and from this we can calculate the parent of room x, depending on whether the room is in the first or second half of its group.

The graph is now implicitly constructed, but is still too big to manipulate. Notice that almost all rooms are incidental to exactly two edges, connecting them to the rooms right before and right after them. The exceptions are rooms at both sides of a broken wall – these have three or four edges. We can compress the trivial nodes to make a weighted graph with O(K) vertices and O(K) edges, where the vertices are rooms 1, N, and all rooms at both sides of a wall. Running Dijkstra's shortest path algorithm on this graph yields a solution of complexity O(K log K).

UMNOŽAK

The first observation we can make is that, for all positive integers x, the digit-product p(x) is always less than or equal to x. From $x*p(x) \leq B \leq 10^{18}$ we can deduce that $p(x) \leq \sqrt{B} \leq 10^9$.

Because p(x) is a product of digits, its prime factors can only be 2, 3, 5 and 7. The number of different integers with only 2, 3, 5 and 7 in their prime factorization that are less than or equal to 10^9 is quite small (exactly 5194).

Since there are relatively few such values, we can iterate through all of them. Now, it suffices to solve the following problem for each fixed product P:

> Let $P = p(x) = 2^a\ 3^b\ 5^c\ 7^d$. How many numbers between $\lceil A/P \rceil$ and $\lfloor B/P \rfloor$ have product of digits equal to P?

We can count all such numbers by considering them digit by digit from left to right using a recursive algorithm.

Let rec(string prefix, int N, int prod) be such a recursive algorithm. The algorithm returns the total number of integers that start with prefix and have N more digits whose product is prod. Its arguments are:

- prefix is the part of the number generated so far
- N is the number of digits that still needs to generated and
- prod is the target digit-product of the yet to be generated part of the number.

rec(string prefix, int N, int prod)

> Let max be a number we get when we append prefix with N nine digits

Let min be a number we get when we append prefix with N zero digits

If max < $\lceil A/P \rceil$ or min > $\lfloor B/P \rfloor$ then Return 0

If N = 0 then

    If prod = 1 then Return 1

    else Return 0

Count = 0

For each digit D from 1 to 9

    If D divides prod then

        Count = Count+rec(prefix appended by digit D, N-1, prod/D)

Return Count

To speed things up we will add memoization to the algorithm described above. In order to memorize the return values of the function efficiently we consider two cases:

- If min >= $\lceil A/P \rceil$ and max ≤ $\lfloor B/P \rfloor$ hold (i.e. all possible values that can be generated from this moment on are within the bounds), then the result is the same no matter the value of the number we generated so far. Therefore, we can ignore the prefix parameter.
  We are left with only N and prod as state parameters, and since N can take on only 18 different values and prod only 5194 different values, we can store the result in a table and use it whenever we encounter the same parameters again.
- If one of the statements min >= $\lceil A/P \rceil$ and max ≤ $\lfloor B/P \rfloor$ is false then the string prefix is either a prefix of $\lceil A/P \rceil$ or a prefix of $\lfloor B/P \rfloor$, and the number of additional states we have to memorize is linear in N (the number of digits).

Total time complexity of the algorithm is $O(K(\sqrt{B}) * \log_{10} B)$ where K(x) is the number of different integers with only 2, 3, 5 and 7 in their prime factorization that are less than or equal to x. For $B = 10^{18}$ we have $K(\sqrt{B}) = 5194$ and $\log_{10} B = 18$.