CETIRI

There are many possible solutions. One is to loop through all values between (say) −200 and 200, and see if it forms an arithmetic progression with the three given numbers. Others calculate the result from the given numbers using arithmetic.

OKTALNI

Follow the instructions in the problem statement. Use strings instead of integers to handle 100-digit numbers.

TAJNA

We calculate R and C from the length of the message.

One way to reconstruct the message is to construct the matrix and write the encrypted message in column-major order, and then read it in row-major order (because this is the opposite of what Mirko does to encrypt the message).

Another way is to see that what we really output are, in order, characters at indices 1, 1+R, …, N−R+1, 2, 2+R, …, N−R+2, …, R, R+R, …, N.

DEJAVU

Suppose we choose the point P in which the angle is 90 degrees. Because the legs of the triangle must be parallel to the coordinate axes, then one of the other two points must have the same x-coordinate as P, and the other must have the same y-coordinate. The exact number of triangles is $[freqx(x_P)-1]\cdot[freqy(y_P)-1]$.

Because the limit is low enough, we can pre-calculate freqx and freqy for each x and y in two arrays. The time complexity of the algorithm is O(N+K), where K is the limit on the coordinates.

CUDAK

We will describe two ways to approach the problem.

The "classic" approach is to develop the function f(N, S) = how many integers less than or equal to N have the digit-sum S. The solution is then f(B, S) − f(A−1, S).

To calculate the value of f, we use an auxiliary function g(len, S) = how many integers of length len have the digit-sum S. The function g is easy to calculate using dynamic programming.

Suppose N is L digits long and its first digit is D. Then f(N, S) = g(L−1, S) + g(L−1, S−1) + … + g(L−1, S−D+1) + f(N−D·10^{L-1}, S−D). This can be calculated iteratively.

An alternative approach is to recursively build the number and take advantage of many subproblems sharing the same solution. Suppose that, in our recursive function, we have decided on a prefix P of the number. Then we know exactly what range of numbers can be built hereafter: if we put K more zeros, we get P·10^K. If we put K nines, we get P·10^{K+1}−1.

If this entire range is outside the interval [A, B], then we return 0. If the entire range is inside the interval, then the solution is equivalent to calculating the function g from above (this is where many

subproblems share the same solution). If the range is partially inside [A, B], then we just continue the recursion (try every digit). The key is to notice that this last case will happen only O(log B) times, so this solution will be as efficient as the first one. The accompanying source code demonstrates the second approach in a single function.

We can find the smallest integer with the digit-sum S while calculating the total number of integers.

An alternative is to use binary search to find the first number X such that the interval [A, X] contains exactly one integer with the digit-sum S (this is equivalent to the first half of the problem).

REDOKS

If we keep the dials in a data structure, then to make the solution efficient enough, each update and query operation will have to run in sub-linear time.

One way to achieve this is to group adjacent dials into buckets containing about $\sqrt{N}$ dials each. For each bucket, we keep track of how many times each digit appears in the bucket. The asymptotic running time is $O(M\sqrt{N})$ and the algorithm can pass in time if carefully implemented.

By using a different data structure, we can reduce the time to O(log N) per query.

Imagine a binary tree in which every leaf represents one dial. Every node in the tree contains 10 numbers – how many times each digit appears in dials contained in the subtree rooted at that node. The root node of this tree contains information about all the dials. There are about 2·N nodes total.

Consider first how to perform a query in this tree. Given an interval [A, B] we want to count how many dials in the interval are displaying each digit. We start from the root node and use the following recursive algorithm:

- If all dials in the subtree of the current node are contained in [A, B], return the 10 numbers contained in this node.
- If the dials [A, B] are entirely contained in the left or right child of the current node, then move to that node (the other node is of no interest).
- Otherwise (some of the dials [A, B] are in the left subtree, and some in the right subtree), recursively query both subtrees and add the results together.

This procedure really splits the query interval into subintervals whose lengths decrease exponentially, using as large subintervals as possible. The time complexity is O(log N).

To keep the entire tree updated, the update operation would need to be linear, because for example, when Luka clicks all the dials, the entire tree would need to be updated.

To remedy this, we can use lazy propagation – when an update operation would update the entire subtree rooted at some node, don't actually do this, but instead increase a "laziness" counter in that node. When another update or query operation would visit one of the node's children, only then propagate the counter to the children.

The official source code demonstrates this second approach. The update and query operations are combined in one function.