Problem A. Archery

Let's connect the targets as follows: for each x, connect target x to target 2^i if and only if the binary representation of x contains bit i. Then, it is enough to shoot at 8 targets with numbers $1, 2, 4, 8, 16, 32, 64, 128$. If the green light never lights up (otherwise the main target has already been hit at this stage), the answer is the bitwise OR of the numbers of the targets for which the yellow light lit up. The ninth shot hits the main target.

Problem B. Backing Up The Password

The problem is mainly an implementation problem for parsing an arithmetic expression.

Since there are only 10^4 possible pin code values, you can run a full search, while previously converting the formula to a polynomial form from variables a, b, c, and d (concatenation is expanded as a linear combination of variables and powers of 10, after which similarities inside brackets (if any) are reduced, brackets are expanded, and similarities are also reduced; after converting concatenation to addition, this part of the problem is reduced to the classic problem of parsing an arithmetic expression).

Problem C. Count The Repetitions

In fact the task can be solved by the Main-Lorentz algorithm: `https://cp-algorithms.com/string/main_lorentz.html`.

Problem D. Distinct Sums

Let's find the biggest prime p that less or equal to $\frac{\sqrt{N}}{2}$.

Because p is prime, its enough to output integers

$2 \cdot p \cdot 1 + 1 \bmod p$, $2 \cdot p \cdot 2 + (4 \bmod p)$, $2 \cdot p \cdot 3 + (9 \bmod p)$, $2 \cdot p \cdot 4 + (16 \bmod p)$, $\ldots$ $2 \cdot p \cdot i + (i^2 \bmod p)$ $\ldots$.

If $a_i + a_j = a_k + a_l$, then

$$a_i - a_k = a_l - a_j$$
$$2 \cdot p \cdot i + (i^2 \bmod p) - 2 \cdot p \cdot k - (k^2 \bmod p) = 2 \cdot p \cdot l + (l^2 \bmod p) - 2 \cdot p \cdot j - (j^2 \bmod p)$$
$$2 \cdot p \cdot (i - k) + (i^2 \bmod p) - (k^2 \bmod p) = 2 \cdot p \cdot (l - j) + (l^2 \bmod p) - (j^2 \bmod p)$$

But $1 - p \le (i^2 \bmod p) - (k^2 \bmod p) \le p - 1$, so if $(i - k) \ne (l - j)$, the equation cannot be held (difference between first terms is atleast $2p$, while the sum of other four terms is between $2 - 2p$ and $2p - 2$). Then $(i - k) = (l - j)$, and $(i^2 - k^2) \bmod p = (l^2 - j^2) \bmod p$. That is impossible if $i \ne k$ and $l \ne j$.

The set with those properties is called the Golomb Ruler (see `https://en.m.wikipedia.org/wiki/Golomb_ruler`).

Problem E. Eligibility Test

Consider the trapezoid $ABCD$, where AB is parallel to CD, $AB \le CD$, and consider the point O on CD. The triangles ABO, BOC and AOD must be equal, considering the equivalence for non-shared sides and that cross-angles that are equal, we have that $AB = CO = DO$, and $CD = AB \cdot 2$. Let's translate B to A, then the trapezoid degenerates into the triangle with sides $CD - AB = AB$, BC and AD.

So if one of integers is exactly twice greater, than some of remaining, and the lesser of those 2 and other 2 integers can be sides of the non-degenrate triangle, the answer is 1, otherwise the answer is 0.

Problem F. Fishing

The obvious quadratic solution: for each k and i counting the number of pairs of type (W, B), (B, W), (W, ?), (?, W), (B, ?), (?, B), (?, ?), filling the pairs with one ?, and only after they are filled, fill the pairs (?,?) with remaining fish. But the solution is too slow.

Let's create three boolean arrays, indicating the presence of the character. Consider those arrays as the polynomial of degree N. Then use FFT to find the pairwise products of the polynomials and then get the answer.

Problem G. Great Upgrade

Note that the costs are very small, namely, there are no more than 300 different costs. Let's try to update the answer in ascending order of costs. That is, we will consider $dp[p][k]$ — the maximum total economic effect, if we have k money and can use items with a cost of no more than p. It is clear that for items of the same weight, it is advantageous to take items with the greatest economic effects.

Suppose we somehow calculated the previous layers in dp and want to calculate the answer for the p-th layer. It is easy to see that for cost k we can use at most k/p items of cost p. If we will update the answer for each k and p based on the previous ones, then for each k we will spend in the worst case $k + k/2 + k/3 + ... + +k/P = O(k \log P)$, and as a result we get complexity $O(K \cdot K \log P)$ which is too slow. Let's speed up the solution.

Let's sort by non-increasing economic effects and calculate the prefix sums B of economic effects for each cost p. Then our dp is rewritten as $dp[p][k] = max(\{p \cdot i \le k | B[i] + dp[p-1][k - p \cdot i]\})$. For convenience, let's iterate over the remainder r and update the answer for all k with a remainder of r when divided by p. Select all such k, and denote $A_i = dp[p-1][r + p * i]$.

Then the formula is $dp[p][r + pq] = max(\{i + j = q | A[i] + B[j]\})$. Let's learn how to quickly make such updates. Consider a set $M = (K - r)/p$ of polylines on a two-dimensional plane, where the i-th polyline contains K/p points, the j-th of those is $(i + j, A[i] + B[j])$. The intuition behind this is the following: for a fixed x, we have all points of different polylines with the first coordinate equal to x have a different representation of the sum $i + j$. That is, if we take the highest point given x and take it y, then we get the answer for $dp[p][r + p \cdot (i + j)]$. Unfortunately, these broken lines cannot be stored explicitly, the asymptotics will worsen to $O(P \cdot K^2)$. Observations are needed for further improvement.

Lemma. Any pair of broken lines intersect at most 1 point, or on a continuous segment.

Proof. Consider two arbitrary broken lines: i-th and j-th ($i < j$). Consider $f(x)$ as the difference at the integer point x of the values of the i-th and j-th polyline, that is, $f(x) = A[i] + B[x - i] - A[j] - B[x - j]$. Consider $f(x)$ and $f(x + 1)$, $f(x + 1) = A[i] + B[x + 1 - i] - A[j] - B[x + 1 - j]$, note that that $f(x)$ is at least $f(x + 1)$, since $B[x - i] - B[x - j] \ge B[x + 1 - i] - B[x + 1 - j]$: we count the difference on a segment of the same length, but the segment $[x - j; x - i]$ is placed to the left from $[x + 1 - j; x + 1 - i]$, the array B is constructed in such a way that the difference between adjacent elements does not increase, that is, B is a convex polyline. Thus we have shown that $f(x)$ does not increase. Q.E.D.

What does it give us? We can use the Li Chao tree to implicitly store polygons, since the necessary condition on the function is met, that is, we add all the polygons to the tree, and then ask for the maximum for each point. The tree can do all this for $\log M$.

Let us estimate the asymptotics. We iterate over p residues for the current p, consider K/p points for each residue, and add the same number of broken lines to Li Chao Tree, as a result, for one p we get $O(K \log(K/p))$, and for all p can be limited to $O(p \cdot K \cdot \log K)$, which is a complete solution to the problem.

Problem H. Hypermagic Squares

1. Let's start with a square board $(2n+1) \times (2n+1)$ filled with numbers from 1 to $(2n+1)^2$ from left to right and in rows from top to bottom (standard row filling); divide the board into $n + 1$ square frames of width 1 (the center «frame» is actually a single square).

   Suppose we manage to rearrange the numbers in each box so that its four sides give the same amount, and two pairs of diagonally opposite corner cells and all pairs of orthogonally opposite side cells also give the same sum. Then these $n+1$ frames form a magic square of the desired kind: if we delete frames sequentially, starting from the furthest, we get a series of successive magic squares, up to the middle unit square.

Consider any of these frames, for example, with a side of $2k+1$. Using the following algorithm, we can get a permutation of exactly the type we need.

2. Mark one corner square and the middle squares on two sides not adjacent to the selected corner square. Let $S = \{2, 4, \dots k-2\}$.
3. If the number of marked squares is equal to the side of the frame (the same — if S is empty) — go to 7) otherwise to point (4).
4. There is exactly one pair of opposite unmarked corner squares. We mark them.
5. Choose a pair of marked corner squares that belong to the same side s. Let's choose a number i from S that was not used. We uncheck the selected pair of corner squares, then mark a pair of X and Y squares on s that are symmetrical about the middle of square s and are at a distance i from each other.
6. Jump to 3).
7. $2k+1$ squares that are marked form some pattern. The four rotations of this template at 0, 90, 180 and 270 degrees are exactly the four sets that should be on the sides of the rearranged frame.

The numbers from the middle squares of the original frame must be transferred to the corner places, the rest is obvious.

Why does this algorithm work? Each of steps (4) by number $k-1$ adds the same number to the sum, and steps (5) do not change the sum. Therefore, each pattern built by this algorithm covers a set of numbers with the same total sum. By choosing a different step distance (e) each time, we ensure that the four turns of the template do not have common squares that are not side averages.

Problem I. Integer Balls

In fact, in this problem, it is required to find the radius of the sphere circumscribed around a tetrahedron, given four integer vertices. The main problems in this problem will be with accuracy - the accuracy of the built-in double type is not enough.

Let A (xa, ya, za), B (xb, yb, zb), C (xc, yc, zc), S (xs, ys, zs) be the vertices of the tetrahedron, O (xo, yo, zo) be the center of the sphere circumscribed around the tetrahedron, and R be its radius.

Consider the system of equations

$$(x_i - x_O)^2 + (y_i - y_O)^2 + (z_i - z_O)^2 = R^2 \quad (i = 1, 2, 3, 4)$$

where x_O, y_O, z_O are the coordinates of the center of the sphere, R is the radius.

Subtracting the equation for $i = 1$ from the system of equations for $i = 2, 3, 4$ and after some transformations, we obtain a system of three linear algebraic equations with three unknown coordinates of the center:

$$(x_i - x_1) \cdot xO + (y_i - y_1) \cdot y_O + (z_i - z_1) \cdot z_O = 1/2(d_i^2 - d_1^2) \quad (i = 2, 3, 4)$$

where d_i^2 is the square of the distance from the origin to the vertex with number i.

As we can see, all coefficients are rational, so by solving the system of linear equations using standard methods, we obtain that the determinant value is a rational number.

The square of the radius will also be a rational number (as the sum of squares of differences between the coordinates of the center of the sphere and any vertex, which are rational numbers).

Thus, it is sufficient to either implement long rational numbers (noting that this is significantly easier to do in Python), or, since in this problem only comparison for equality is required, to use a system of residue classes: represent the numerator and denominator as a set of residues modulo prime numbers.

Problem J. Judging the Mafia game

The general concept is that we can ask whether two vertices belong to the same connected component, and we need to check if there is a component of size greater than $\frac{n}{2}$. To maintain connected components, we will use the Disjoint Set Union data structure, assuming that you are familiar with this data structure.

27-point solution

Let's randomly choose an index i 12 times and ask for the query $i\ i\ \ldots\ i$. If there are more than half ones, then we have found the answer. If we do not find such an index in 12 queries, then the answer is -1.

50-point solution

Our algorithm will be based on the well-known one-pass algorithm for finding the majority element (the number that appears more than half the time) of a set. We can go through the numbers from left to right, maintaining the current candidate for the answer and its current balance. If we encounter an element equal to the candidate, we will increase the balance by one, otherwise decrease it by one. If the balance becomes zero, we will change the candidate.

Similarly, we will store a list of candidates and their balances. At each iteration, we will sort our candidates by their current balance and ask about each pair of neighbors. Now we will start increasing the balance — if the components are the same, we will merge them and add their balances. After that, we will have a sequence of components, and for the neighbors, we know for sure that they represent different components. Then we will subtract from the balances of these components the minimum of their sizes, one of the components will become zero (but will remain in the list, perhaps it will be compared with the next neighbor later), and the balance of the other component will decrease slightly, it has dominated the small component. We will discard all components with zero balance — we will not lose the majority element in this way, because we subtracted the size of this component from the larger one, which means that we removed as many other numbers as we discarded now (and for the majority element, this is impossible, there are not so many numbers). We will continue until there is more than one candidate.

55-85 point solution

Let's make several random queries and one smart query.

- Random queries. Let's shuffle the indices, divide them into blocks of size $\sqrt{n}$. In each block, we will choose one element and ask if it is equal to all other indices in the block (if it is equal, we will merge them into the Disjoint Set Union, if not, we will remember somewhere that these indices are not equal). What can be optimized here is that if you know that the indices are already equal, then ask not with this selected index, but with another random one. Similarly, if you know that the indices are definitely not equal, then ask with a random one.
- Smart query. Let's sort our components by size, from largest to smallest. Now let's consider the i-th largest component, we know that it contains indices $i_1, i_2, \ldots, i_k$. Let's ask i_1 with the root of the $(i+1)$-th component, i_2 with the root of the $(i+2)$-th component, and so on. What can be optimized here is that if we know that these components are already not equal (from our random queries), then we do not need to make this query, it is better to ask with the next component.

If this is all implemented optimally, then you can get 65 points. Tests with small values of n, which can be considered separately, will not work for 85 points.

Full solution

Let's say we want to solve it in Q queries, we will make $Q-1$ good random queries and one smart query (by good, we mean that we will not ask whether x and x are in the same component). Below we will describe how to build a good random query.

For the smart query, let's sort our current components in descending order of size. On the one hand, the larger the component, the more likely it is to be the answer. On the other hand, the smaller the component, the less information we have about it, because the amount of information obtained is directly proportional to the size of the component (for one query, we can ask about one vertex twice (a_i and $a_j = i$)). We will

iterate through the components from smaller to larger and try to attach small components to larger ones. Let the size of the small component be x, then we can first choose x large components (about which we do not know for sure whether we are in the same component), and ask about them using the vertices of the small component. For the remaining components, we will also ask if they are connected to the small component, but using their vertices (one at a time). After that, we will reorder the vertices by the number of free vertices available for the query and continue building the query.

Now about building a good random query. If we already know that two vertices belong to the same component, or that they belong to different components, we will not ask such a pair. If such a pair of components has already been queried in our query, we will not ask such a question either. We will generate a random query until all conditions are met. As a random query, we will generate a 2-regular graph (the degrees of all vertices are 2) and orient each cycle in one direction (the orientation of the graph gives us who is i and who is $a[i]$ for two vertices).

Practice shows that this is enough to determine the answer correctly with two random and one smart query.

Problem K. King of Cabbages

Consider the following matrix A: the main diagonal elements are equal to C, on the diagonal below the main diagonal the integers $(i+1) \cdot K[i+1]$ are placed, other elements of the matris are equal to 0. Then the answer is $F \cdot A^t$, and the fast exponentiaton can be used (note that the elements to be recalculated are only the elements on the main diagonal and on the diagonal below). Time complexity is $O(N \log T)$.

Problem L. Lake Fishless

We are forced to add the fish into some cells (those that have no neighbours point into them)

In test group 2 ($L = 10^9$):

1. Add the fish into cells that are forced. If a solution exists, this will cover all cells except cycles.
2. As long as there is an edge cell left uncovered, add fish into that cell and mark all following cells as covered.
3. If there are non-edge cells left uncovered, answer impossible.

The lake is a directed graph with each node having out degree at most 1

This means that each component is either:

- A tree
- A cycle
- A cycle with trees pointing into it

Trees can be handled with a greedy algorithm

- Add the fish into leaf nodes.
- This creates new leaf nodes, which also need fish.
- The new leaf nodes might not be at the edge of the lake.
- Add the fish into the closest edge-node behind

For the last two subtasks, we need to handle cycles too.

Quadratic solution: Test every start point and greedily fill the rest of the cycle.

Linear solution:

- Number the nodes in the cycle with an arbitrary node as node 0
- Compute $dp[i]$ as a pair of two values:
- Minimum number of cells to add fish into for the nodes $i..N$, given that nodes $0..i-1$ are already covered.
- The amount of fish endurance left over after node N.
- Test every starting node, node i is an OK start node if $dp[i].second$ says that every node in the interval $0..i-1$ is covered.