Problem A. Adding Integers

Define $a_0 = n$ and $a_{q+1} = 0$.

Define $b_i = a_{i-1} - a_i \geq 0$.

Note that $\sum_{i=1}^{q+1} b_i = n$.

Look at $\binom{n}{a_1} \cdot \binom{a_1}{a_2} \cdot \ldots \cdot \binom{a_{q-1}}{a_q} = \binom{b_1+b_2+\ldots+b_{q+1}}{b_2+\ldots+b_{q+1}} \cdot \binom{b_2+b_3+\ldots+b_{q+1}}{b_3+\ldots+b_{q+1}} \cdot \ldots \cdot \binom{b_q+b_{q+1}}{b_{q+1}}$. This is the number of ways to color elements in $q+1$ colors such that there are b_i elements of color i.

Turns out that $f(q)$ is simply the number of ways to color n elements into $q+1$ colors.

So $f(q) = (q+1)^n$.

Problem B. Bottles

We will consider these permutations of all bottles $\frac{(e+p+w)!}{e!w!}$, since the order of poisonous bottles matters. We will count how many of those result in the elf being alive.

Consider we have some sequence of poisonous bottles and bottles with water. For each such sequence, there is the first "poison" event that happens after some time x. It means that if we try all ways to add elixir bottles to that sequence, the ways that result in at least one elixir bottle being among the first x bottles in the sequence keep the elf alive.

So if we count for each x, $f(x)$ — how many sequences of bottles out of $\frac{(p+w)!}{w!}$ consisting of poisonous and water bottles that have the first event happening right after element x. Then the answer is $\frac{(e+p+w)!}{e!w!} - \sum_x f(x) \times \binom{e+p+w-x}{e}$, where $\binom{e+p+w-x}{e}$ — the number of ways to place all elixir bottles in the sequence after position x, hence killing the elf.

How to calculate $f(x)$. Let's calculate $g(x)$ — the number of sequences of poisonous and water bottles, such that the event is **not** happening before x. It means $f(x) = g(x) - g(x+1)$. Consider all poisonous bottles that can poison before x: such i, so that $1 + i + t - 0.5 < x$, so $i \leq x - t - 1$. So there are $m = \min(x - t - 1, p)$ such bottles. Other bottles can be in any order. Bottle 1 has to be at positions from $x - t$ up to the end $p + w$, so there are $p + w - (x - t) + 1$ ways. Bottle 2 has to be at positions from $x - t - 1$ up to the end, but one of the positions is taken by bottle 1, so also $p + w - (x - t) + 1$ ways, and so on. So there are $(p + w - (x - t) + 1)^m$ ways to place these m bottles, and $\frac{(p+w-m)!}{w!}$ to place others. So $g(x) = (p + w - (x - t) + 1)^m \frac{(p+w-m)!}{w!}$

Problem C. Counting Orthogonal Pairs

The angle of a regular polygon in degrees is $\frac{(n-2)\cdot 180}{n}$.

Consider all diagonals; there are $n - 3$ diagonals from the vertex, and it divides into $n - 2$ angles. Each angle is equal and of $\frac{(n-2)\cdot 180}{n(n-2)} = \frac{180}{n}$ degrees. The angle between two line segments coming from the vertex is $180 \cdot \frac{k}{n}$ for some $1 \leq k \leq n - 2$. $180 \cdot \frac{k}{n} = 90$ only when $\frac{k}{n} = \frac{1}{2}$, and for integer k and n, this only happens when n is even, so $k = \frac{n}{2}$, and there are $\frac{n}{2} - 1$ ways to choose a pair of segments.

When n is odd, the answer is 0.

When n is even, the answer is $n \cdot \left(\frac{n}{2} - 1\right)$.

Problem D. Divine Tree

Count the number of G and B coins, let them be g and b, respectively.

Note that because the size is odd, after the type 2 operation, we will definitely know which of the two trees has G coins and which has B coins because their sizes would differ, and their sizes have to be g and b, respectively.

Also note that if we have chosen the edge for the type 2 operation, the number of times for each edge to

be used in the type 1 operation is fixed; we can just count it. For each edge, we can see how many G and B coins are at each side of the edge and how many G and B coins there have to be at each side of the edge. This imbalance is the number of times we need to use the edge. You can prove that you can always make the type 1 operations so that every time you use an edge, the imbalance decreases for it, and finally get 0 imbalance for all edges.

Make a tree rooted at some vertex and get all subtrees of size g and all subtrees of size b. For every edge that you can use in the type 2 operation, either the subtree of size g or the subtree of size b has this edge coming from its root. Store information about subtrees of size b and g independently and in a similar way, and make updates independently. When getting the query answer, get for both and take the minimum.

Now consider only one case: we have subtrees of size g, and we want for each of them to store and update the cost to make type 1 operations, to afterwards make type 2 operations on the edge coming from its root. Note that these subtrees are disjoint.

Let's start with all weights equal to 0 and make an update operation for an edge weight. Consider some edge uv is updated, d is added to its weight, and v is more distant from the root than u. And consider a single subtree A of size g. There are three cases: if an edge is in A, if A is in the subtree of v, and otherwise.

1. If an edge uv is in A, then A's value will change by the number of B vertices in v's subtree times d;
2. If A is in the subtree of v, then A's value will change by the number of G vertices outside v's subtree times d;
3. Otherwise, A's value will change by the number of G vertices in v's subtree times d.

We can update these values for all trees simultaneously by ordering them in the order of DFS visiting them. There is only one subtree of size g for case 1, and cases 1 and 2 can't happen at the same time. Create a data structure that supports range addition and range minimum, and make an update in $\log n$ time.

Problem E. Eve, Adam and Three Integers

If among the two consecutively written numbers there are two numbers of the same parity and two numbers of different parity, it follows that one of the edge numbers has the same parity as the middle number, while the other has a different parity. Thus, the parity of the middle number does not matter, and it is necessary to check that a and c have different parity (for example, by adding them and checking that the sum is not divisible by two).

Problem F. Fibonacci Triangles

Let us consider an isosceles triangle with sides equal to Fibonacci numbers. Let the longest side be f_n. Then the second longest side does not exceed f_{n-1}, and the third side does not exceed f_{n-2}. However, since $f_n = f_{n-1} + f_{n-2}$, such a triangle either does not exist or is degenerate.

Therefore, triangles with sides equal to Fibonacci numbers can only be isosceles. Let us have two sides of length f_k, and we will calculate what value the third side can take. It is obvious that all numbers from f_1 to f_k are suitable. We also note that for $k > 1$, $f_{k+1} < 2 \cdot f_k$. But $f_{k+2} = f_{k+1} + f_k > f_{2k}$, meaning that for $k > 1$ there is another triangle with the third side equal to f_{k+1}; there are no more triangles.

Thus, when moving from n to $n+1$, $n+2$ triangles are added ($n+1$ triangles with sides f_{n+1}, f_{n+1}, f_i for i from 1 to $n+1$, and also a triangle with sides f_n, f_n, f_{n+1}).

Considering that for $n = 1$ the answer is 1 (only the equilateral triangle), and applying the formula for the sum of an arithmetic progression, we find that the answer is $n(n+1)/2 + n - 2$, or $n(n+3)/2 - 2$. Given the specified range for n, the answer fits well within the `long long` type.

A dynamic programming solution does not work for the given range of n due to time and memory constraints.

Problem G. Game of Voleyball

The main part of the solution to this task is the implementation of the function `addPoints`, which adds a point to the team passed as the first parameter. The opposing team is passed as the second parameter. In the function, the number of points for the first team is increased, and then it is checked whether this leads to the end of the set. If the number of points for that team is now greater than 25 (or 15 if the score is 2:2 in sets) and the difference compared to the other team is at least 2, team A has won the set, and the scores of both teams are reset to 0 due to the start of the next set.

Problem H. Heroes and Illusions

Let's say positions of real heroes are $a_1, a_2, ..., a_{m-1}$, also add $a_0 = 0$ and $a_m = n + 1$. Consider values $b_i = a_i - a_{i-1}$, the lengths of segments between consecutive heroes. The segment $[l, r]$ contains odd number of heroes, if its ends belongs to segments with different parity, so the number of such segments is $(a_1+a_3+a_5+...)\cdot(a_2+a_4+a_6+...) = k$. Let's say $x = (a_1+a_3+a_5+...)$, then we have $x\cdot(n+1-x) = k$. From this equation we can find x, and then we have standard problem of counting the number of partitions of x and $n + 1 - x$.

Problem I. Internet Connection Stability

First, let's "deal with" the stations with an odd number of connections by pairing these stations. Since the sum is even, there are an even number of them, and it will be possible to connect them. Thus, only stations with an even number of "free" connections remain.

If the station m with the highest number of connections a_m is not greater than the sum of all other a_i, then it is possible to do without any type 2 connections at all. We will prove this by induction on the number of stations.

For two stations, the statement is obvious: the property holds only when the stations have an equal number of connections; in this case, we assume that all connections were of type 1 and connected different stations.

Assume the statement is true for all $k < n$. Then we take the station with the highest number of connections and start "marking" one connection with the station with the highest number of connections among the remaining ones until the number of "free" connections for this station is no longer among the highest (this we can always do, since a_i is the station with the highest number of connections). After that, we switch to establishing connections with other stations for which the number of "free" connections is maximal, also until this station is no longer among the highest, and so on, until all connections for the considered station are exhausted. Thus, we are left with $n - 1$ stations, and the difference between the highest a_i and the next highest, by construction, does not exceed 1, meaning that the station with the higher number of connections has a value of a_i that is not greater than the sum of the connection counts of the other stations, thus the transition is proven.

So if a_m is not greater than the sum of all other a_i, then the answer is 0.

If a_m for the station with the highest number of connections exceeds the sum of all other values, then we will establish all connections for each of the remaining stations with a_i, leaving $a_m - \sum_{i \neq m} a_i$ connections unused, which will inevitably be type 2 connections. Dividing this difference by 2 gives us the answer.

Problem J. Jumping Game

Consider a more general problem, where we have the undirected graph G and a token initially placed in vertex s. Players take turns moving the token, and the token cannot visit the same vertex twice. If a player cannot move, they lose.

This is a well-known problem, connected to maximal matchings. In particular, it's easy to prove that the first player loses iff there is a maximal matching that doesn't cover vertex s.

Now, in our problem, we have a very specific graph. The intuition suggests that for the large board there

is always some perfect (or almost perfect, if the size is odd) matching. That's actually true. We just need to solve some small cases manually. Namely, cases $1 \times x$, $2 \times x$, 3×3, and 3×5.

Problem K. Kangaroo On Graph

Make dynamic programming, calculate for each edge the value $d[uv]$, the minimal cost of the path ending with edge uv. To calculate this value, let's look at all possible previous edges xu. We need to find the minimum $d[xu]$ over all x, except the ones that form a forbidden triplet. We can store all the forbidden x for each edge uv, and just iterate all edges xu in increasing order by $d[xu]$, and pick the first one that is not forbidden. Time $O(n + m + k \log n)$

Problem L. Lattice Sets

We will show that no figure can have more than 4 axes of symmetry. If a figure has an axis of symmetry, then the side of the unit square must also map to some side. However, the angle between two segments symmetric with respect to a line is equal to twice the angle between the segment and the line, and since the segments can be either horizontal or vertical, the axes of symmetry can either also be horizontal and vertical, or must be at a 45-degree angle to the grid lines. Since there can be no more than one axis of symmetry in one direction, the number of axes cannot exceed 4.

Four axes of symmetry can be obtained when $k = 4n + 1$ or $k = 4n$. In this case, the cells can be chosen in the form of a cross or a cross with a punctured center, respectively. In other cases, the number of axes of symmetry is two (two or three cells cannot be arranged so that there are more than two axes of symmetry).