Problem A. Olympiad Participants

Author problem: Nikolay Vedernikov, developer: Konstantin Bats

To minimize the number of non-participants, it is necessary to maximize the number of students who participated in at least one Olympiad. This is achieved if the sets of participants in the three Olympiads do not intersect, meaning each participant belongs to only one set. In this case, the total number of participants is $a+b+c$. However, if the sum exceeds n, it can be arranged so that all n students participate in at least one Olympiad. Therefore, the minimum number of non-participants is $\max(0,\ n-(a+b+c))$.

To maximize the number of non-participants, it is necessary to minimize the number of students participating in at least one Olympiad. This is possible if the sets of participants overlap as much as possible— for example, when the same students belong to all three sets. In this case, the number of participants is $\max(a, b, c)$. Thus, the maximum number of non-participants is $n - \max(a, b, c)$.

Problem B. Suitable Envelope

Problem author: Pavel Skobelin, developer: Margarita Sablin

Note that the width and height of the envelope must match the height and width of one of the postcards (not necessarily the same one) — they cannot be less than the height and width of any postcard; if we make the sides larger than the sides of the postcards, the result will not be optimal. Let us assume that if the postcard i does not have equal height and width, the height h_i will be considered the smaller of the two sides, and the width w_i — the larger of the two sides. We will choose an envelope whose height H_{max} will be the largest of all postcard heights, and whose width W_{max} will be equal to the maximum width of the postcards. Note that it is not possible to take an envelope with smaller side values: we cannot take a smaller width because we have chosen the largest from all width values, and any height is less than or equal to any width. $\forall i$: $w_i \le W_{\max}$, $\forall i$: $h_i \le w_i$. It is not possible to take an envelope with a smaller height because it will not fit the postcard with the largest height. Thus, this solution minimizes H and W, and therefore minimizes $H \cdot W$.

Problem C. Merging Amulets

Author and problem developer: Egor Yulin

To solve this problem, we will use the fact that the `gcd` changes at most $\log A$ times, where A is the maximum value of a_i. We will then find the prefix and suffix `gcd`, denoted as p_i and s_i, respectively.

We will divide the calculation of the final sum of `gcd` into 3 stages:

1. The segment $[1, n]$. To find the answer, we need to calculate the `lcm` of the entire array. This can be done using the Sieve of Eratosthenes and the prime factorization of each element;
2. Segments $[i, j]$ such that $p_i = p_{i+1}$ and $s_{j-1} = s_j$. The answer for such segments will be $\texttt{gcd}(p_i, s_j)$, since the `lcm` will definitely be divisible by $\texttt{gcd}(p_i, s_j)$, and the final `gcd` cannot exceed this value. Such segments can be counted quickly if we know in advance the positions where p_i and s_j change;
3. The remaining segments, specifically those $[i, j]$ where $p_i \neq p_{i+1}$ or $s_{j-1} \neq s_j$. Since the `gcd` changes at most $\log A$ times, there will be at most $O(n \log n)$ such segments. Therefore, they can be counted with a straightforward pass through the array.

The final answer will consist of the sum of all three types of segments.

Problem D. Building a Stage

Problem author: Demid Kucherenko, developer: Egor Yulin

To find the maximum triangle, note that one of the vertices will always lie on the boundaries of some arc. This is true because if all vertices do not lie on the boundaries of the arcs, we can "rotate"the triangle until one of the vertices lies on the boundary.

Let's fix one vertex; the maximum triangle that can be inscribed in the circle is equilateral. If it can be inscribed under the current constraints, it will be the answer.

Now suppose the answer gives a triangle with exactly two points lying on the boundary; to maximize the area, the third point should lie at the midpoint of the arc formed by these two points. If such a point does not exist, then the answer will be the triangle where all three points lie on the boundaries.

We will show that if a triangle has only one vertex on the boundary, it must necessarily be an equilateral triangle. If not, we can definitely move one of the points to increase the area of our triangle (this cannot be done only if the triangle is equilateral).

Thus, the answer is found by checking three cases: when one vertex lies on the boundaries and the triangle is equilateral, when exactly two vertices lie on the boundaries, and when all three vertices lie on the boundaries.

Problem E. Cubic Bushes

Problem author: Nikolay Vedernikov, developer: Egor Yulin

If we represent the initial array as a difference array, that is, $b_i = a_{i+1} - a_i$, then in the array b we want to find the longest subarray where the first half consists of 1s and the second half consists of -1s.

To find such a maximum subarray, we will use a segment tree, where we will store the current answer, the maximum prefix for the answer, and the maximum suffix.

Then, to recalculate, we update the answer and check if we can extend the prefix and suffix.

Height changes are just regular point updates in the segment tree.

Problem F. A Mess Nobody Needs

Author and problem developer: Pavel Skovelin

According to the Sprague-Grundy theorem, if we calculate the Grundy function for each thread and then take the `xor` of all the obtained values, we will get a value of $= 0$ if Petya's position is losing, and a value > 0 otherwise. The only problem is that the constraints on the lengths of the threads in the problem are quite large, so it is not possible to calculate the Grundy function by definition.

To solve this, let's learn to calculate the Grundy function for a thread of length x. First, let's write down the value of the Grundy function by definition:

$$g(x) = \operatorname{mex}\big\{\, g(a) \oplus g(b) \;\big|\; a + b = x,\ \gcd(a, b) > 1 \big\}.$$

It is claimed that:

$$g(x) = \begin{cases} 0, & \text{if } x = 2 \text{ or } x \text{ is odd}, \\ 1, & \text{if } x \equiv 0 \pmod 4, \\ 2, & \text{if } x \equiv 2 \pmod 4. \end{cases}$$

During the contest, it was not difficult to notice this pattern by calculating the Grundy function for small lengths. Now we will prove it by induction:

The base case for $1 \le x \le 3$ is obvious.

Assume the statement is true for all $s < x$, then we will prove it for x.

Consider 3 cases:

1. x is odd. Then all partitions $x = a + b$ have the form: (even number > 2) + odd. The even number cannot be equal to two, as in that case the second condition will not be satisfied: there will necessarily be $\gcd(2, x-2) = 1$. Thus, by the induction hypothesis, $g(\text{odd}) = 0$, $g(\text{even}) > 0$, so their $\oplus$ will be > 0, meaning that in the set $\left\{ g(a) \oplus g(b) \;\middle|\; a + b = x,\ \gcd(a, b) > 1 \right\}$ all numbers will be > 0, hence its mex will be zero. Therefore, the Grundy function for an odd number is zero.

2. $x \equiv 0 \pmod 4$. Take the partition $x = \frac{x}{2} + \frac{x}{2}$, we get $g(\frac{x}{2}) \oplus g(\frac{x}{2}) = 0$. At the same time, $\texttt{xor} = 1$ cannot be obtained, as cutting into 2 odd numbers will give 0, while cutting into 2 even numbers will give 0 or 3. Hence, the mex will be equal to 1.

3. $x \equiv 2 \pmod 4$. Considering the partitions $n = \frac{n}{2} + \frac{n}{2}$ and $n = 2 + (n-2)$, we get `xor` equal to 0 and 1, respectively. From the parity of x, it is clear that `xor` equal to 2 is impossible. Thus, $g(x) = 2$.

Next, we needed to calculate the Grundy function for the lengths of all threads and take the `xor` of the obtained values.

In total, the solution works in $\mathcal{O}(n)$.

Problem G. The Magic Suitcase

Problem author: Mikhail Ivanov, developer: Margarita Sablin

The simplest solution to this problem is to generate a square randomly. The square is a random permutation of the numbers from 1 to n^2. It is necessary for the $2 \cdot n + 2$ sums to be distinct. One can estimate the probability that all such sums will be different. In total, there are $n^3 - n^2 + 1$ distinct sums of n numbers from 1 to n^2. The probability that $2 \cdot n + 2$ randomly chosen numbers from them will be distinct approaches 1 for large n. This estimate will not be exact, as the sums in the square depend on each other, but it is possible to assess the order of magnitude.

Problem H. Submask Parity

Author and developer problem: Ildar Gainullin

Let us consider two cases:

- Every number appears an even number of times in the array.

  In this case, no matter which number x we choose, the array will contain an even number of its submasks. Therefore, the answer is *-1*.

- There exists a number that appears an odd number of times in the array.

  Let x be the smallest such number. Note that if y is a submask of x, then $y \leq x$. All numbers smaller than x appear an even number of times, and the number x itself appears an odd number of times. Thus, x is a valid solution.

Problem I. Playlist

Author and developer problem: Ekaterina Vedernikova

To solve the problem, it is necessary to simulate the process of playing the player with support for the song queue.

It is necessary to keep track of the current moment in time *curTime* and the queue of songs. To store the queue of songs, it is convenient to use a deque structure, as this structure allows for quick addition of elements to both the front and the back.

To correctly handle additions at the moment in time $t = 0$, before starting the simulation process, all events with $t_j = 0$ should be placed at the front of the queue.

At each step, one of two events occurs:

- if a song is currently playing, all events with $t_j \le curTime + curLen$ should be processed, adding new songs to the front of the queue, then increase $curTime$ by the duration of the current song;
- if the player is not playing music (the queue is empty), $curTime$ should be moved to the moment of the next song's addition, and all songs starting at that same time should be added to the queue.

After that, the next song from the front of the queue should be selected and started playing. At this point, the start time of this song and its name should be recorded in the answer.

Each song will be added to the queue once and removed once, and each operation is performed in $O(1)$. The overall time complexity is $O(n + m)$.

Problem J. Quantum Cats

Author problem: Alexey Zabashta, developers: Alexey Zabashta, Andrey Omelchenko

If N is an odd number, simply write in cell i, j the number $((i + j - 2) \cdot \text{inv2modN}) \mod N + 1$, where $\text{inv2modN} = (N + 1)/2$ — the number that is the multiplicative inverse of 2 modulo N, meaning $(2 \cdot \text{inv2modN}) \mod N = 1$.

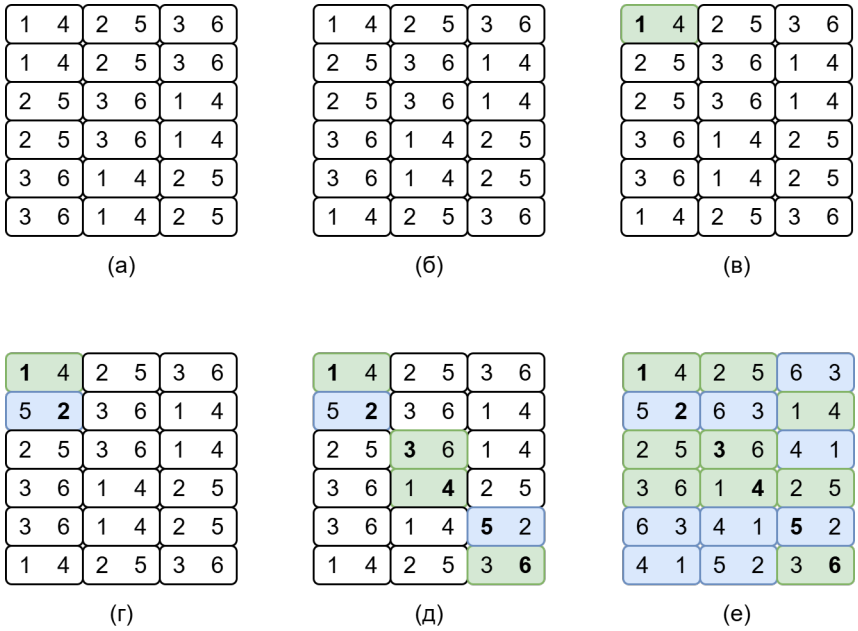
Filling $(i+j-2) \mod N+1$ will give a correct arrangement without considering the diagonal constraint, as addition modulo N forms a group. Multiplying by inv2modN will correct the diagonal, as without moduli $(i + i)/2 = i$.

Let's solve the problem for even N. If $N = 2$, then the arrangement does not exist; for other even N, it does exist.

Imagine we have $N/2$ variants of blocks of two numbers of size 1 by 2, which can be mirrored. We number each variant from 1 to $N/2$ and place the numbers i and $i + N/2$ in the i-th variant of the block.

These blocks need to tile an N by N square. There will be N blocks in height and $N/2$ in width. Consider a template of blocks with consecutive numbers from 1 to $N/2$. We fill the first and second rows with this template. Then we perform a cyclic shift of the template to the left and fill the next two rows. And so on until the end of the table. An example of the resulting table for $N = 6$ can be seen in the figure (a). Next, we will perform a cyclic shift of the entire table upwards (see figure (b)).

Then we will go through the cells of the table on the diagonal, see which block they belong to, and reflect the corresponding block so that the desired number appears on the diagonal. In the example, the first block remains unchanged (see figure (c)), while the second is reflected (see figure (d)). The result of the diagonal traversal can be seen in figure (e), with unreflected blocks colored green and reflected ones colored blue. After this, we will go through all the other blocks and reflect those blocks whose variants have already appeared above with the same reflection. The final result can be seen in figure (f).



The initial arrangement of blocks guarantees that in each row, each number from 1 to N appears exactly once, and each variant of the block appears exactly twice in each column. Therefore, through reflections, we can always ensure that in each column, each number from 1 to N appears exactly once. The initial choice of pairs of numbers for the blocks and the arrangement guarantees that there will always be a block with the desired number on the diagonal.

Problem K. Restoring Weights

Author and developer problem: Ildar Gainullin

Consider any edge (u, v) with weight w, and let $d_u > d_v$.

- $d_u + w \geq d_v$, because otherwise there would exist a shorter path to 1 through v with weight $d_u + w$.
- $d_v + w \geq d_u$, because otherwise there would exist a shorter path to 1 through u with weight $d_v + w$.

Then the weight of each edge must belong to the interval $[\max(1, d_v - d_u), l]$.

Additionally, for each vertex $v \neq 1$, there must exist at least one edge such that $d_u + w = d_v$.

Note that if all these conditions are satisfied, the weight assignment is valid.

For edges connecting two vertices u, v with equal $d_u = d_v$, any weight can be assigned.

All other edges are oriented from smaller d_u to larger d_v.

For each vertex, the assignment of weights to edges incident to it is independent of the assignment of weights to edges incident to other vertices.

Thus, if $f(v)$ is the number of ways to assign weights to edges incident to v correctly, the answer is $f(2) \cdot f(3) \cdot \ldots \cdot f(n)$.

For a fixed vertex, we first count the number of ways to assign weights to edges incident to this vertex. This value is equal to the product of the lengths of the intervals of possible edge weights over all edges.

However, in some situations, the actual distance to v will exceed d_v, specifically when there does not exist an edge (u, v) with weight w such that $d_u + w = d_v$. In these cases, the weight of each edge $u \to v$ must be strictly greater than $d_v - d_u$. For such "*bad*"cases, the number of assignments is computed similarly, and then subtracted from the current value.

For edges with pre-determined weights, it is sufficient to check that their weight lies within the required interval; otherwise, the current answer is replaced with 0.

Problem L. Long Jump Home

Author and problem developer: Pavel Skovelin

First, let's understand the criterion for being able to reach point L from point 0 using a set of jump lengths $x_1, x_2, \ldots, x_n$. From the extended Euclidean algorithm, it is not difficult to understand that from point 0, we can reach point $g = \gcd(x_1, x_2, \ldots, x_n)$ using a linear combination of jumps. Obviously, after that, we can visit all coordinates of the form $g \cdot k$, where $k \in \mathbb{Z}$.

Next, we will show that the points described above are all the points that can be visited with such a set of x values. Indeed, each of $x_1, x_2, \ldots, x_n$ is divisible by g, thus the coordinate after a jump will always be a multiple of g.

After that, we want point L to be equal to $g \cdot k$ for some integer k. Thus, the necessary and sufficient condition for the set of x values is: $L \vdots \gcd(x_1, x_2, \ldots, x_n)$.

Then the problem reduces to finding a set of x values with the minimum cost such that its gcd divides L. Therefore, from the current set, we are only interested in its current gcd and cost. Let's set up a dynamic programming approach:

$dp_{i,z}$ — the minimum cost of a set formed only from the first i abilities that has gcd $= z$. We will understand how to recalculate such a DP: if the current ability has parameters x_i, c_i, then the recalculation from the previous layer will be as follows:

$$dp_{i,\gcd(w,x_i)} = \min(dp_{i,\gcd(w,x_i)}, dp_{i-1,w} + c_i), \qquad 0 \le w \le L$$

Thus, the DP works in $\mathcal{O}(n \cdot L \cdot \log C)$. How to find the answer? We need to look at all DP values of the form $dp_{n,d}$, where d divides $|L|$.

Problem M. Remainder of the Sum of Remainders

Problem author: Pavel Skobelin, developer: Maria Zhogova

Note that directly iterating over all pairs (i, j) in $O(n^2)$ is impossible with such constraints on n.

The goal of the tutorial is to derive a formula that allows us to compute the sum in $O(\sqrt{n})$.

Part 1. Fixing j

Let $T(j) = \sum_{i=1}^{n} (i \bmod j)$. Then $S(n) = \sum_{j=1}^{n} T(j)$.

Let's compute $T(j)$.

Consider the numbers $1, 2, \ldots, n$ and their remainders when divided by j. They are divided into blocks of length j:

$$[1, \ldots, j], \quad [j+1, \ldots, 2j], \quad \ldots$$

and possibly the last incomplete block.

Let

$$q = \left\lfloor \frac{n}{j} \right\rfloor \quad \text{(number of complete blocks)}, \qquad r = n \bmod j = n - qj \quad \text{(length of the tail)}.$$

Then:

- Each complete block gives a set of remainders $0, 1, 2, \ldots, j-1$, their sum equals $0+1+\cdots+(j-1) = \frac{j(j-1)}{2}$. There are q such blocks, so the contribution of complete blocks is: $q \cdot \frac{j(j-1)}{2}$.
- The last (incomplete) block gives remainders $0, 1, 2, \ldots, r$, their sum equals $0+1+\cdots+r = \frac{r(r+1)}{2}$.

Thus,

$$T(j) = q \cdot \frac{j(j-1)}{2} + \frac{r(r+1)}{2}, \qquad \text{where } q = \left\lfloor \frac{n}{j} \right\rfloor, \quad r = n - qj.$$

Substituting this into the sum:

$$S(n) = \sum_{j=1}^{n} \left(q \cdot \frac{j(j-1)}{2} + \frac{(n-qj)(n-qj+1)}{2} \right).$$

For convenience, let's denote the two parts:

$$A = \sum_{j=1}^{n} q \cdot \frac{j(j-1)}{2}, \qquad B = \sum_{j=1}^{n} \frac{(n-qj)(n-qj+1)}{2}.$$

Then $S(n) = A + B$.

Part 2. Grouping by values of $q = \left\lfloor \frac{n}{j} \right\rfloor$

Key observation: the value $q = \left\lfloor \frac{n}{j} \right\rfloor$ does not change for every j, but is piecewise constant over entire intervals.

For a fixed q, the set of j that gives this value q forms an interval:

$$j \in [L, R], \text{where } L = \left\lfloor \frac{n}{q+1} \right\rfloor + 1, \qquad R = \left\lfloor \frac{n}{q} \right\rfloor.$$

An equivalent way to construct intervals (convenient for implementation): we will iterate j from left to right. Let's say we are currently at some $j = i$. Then

$$q = \left\lfloor \frac{n}{i} \right\rfloor, \quad R = \left\lfloor \frac{n}{q} \right\rfloor.$$

Clearly, for all j in the interval $[i, R]$, the value $\left\lfloor \frac{n}{j} \right\rfloor$ is the same and equals q. After processing this interval, we move to $i = R + 1$.

It is important that the number of such intervals is on the order of $O(\sqrt{n})$, since after the point $j > \sqrt{n}$, the values of q are small and take only about $\sqrt{n}$ different values, and for j up to $\sqrt{n}$, there are also about $\sqrt{n}$ such j.

Next, we need to be able to quickly compute sums over the interval $[L, R]$:

$$\sum_{j=L}^{R} 1, \qquad \sum_{j=L}^{R} j, \qquad \sum_{j=L}^{R} j^2.$$

Let's denote:

$$\text{cnt} = R - L + 1,$$

$$S_1(L, R) = \sum_{j=L}^{R} j,$$

$$S_2(L,R) = \sum_{j=L}^{R} j^2.$$

These sums can be conveniently expressed using prefix formulas:

$$\sum_{j=1}^{x} j = \frac{x(x+1)}{2}, \qquad \sum_{j=1}^{x} j^2 = \frac{x(x+1)(2x+1)}{6}.$$

Then

$$S_1(L,R) = \frac{R(R+1)}{2} - \frac{(L-1)L}{2},$$
$$S_2(L,R) = \frac{R(R+1)(2R+1)}{6} - \frac{(L-1)L(2L-1)}{6}.$$

All these formulas will be computed modulo M, using the inverse elements of 2 and 6 (since M is prime).

Part 3. Contribution of the interval $[L, R]$ to A

Recall:

$$A = \sum_{j=1}^{n} q \cdot \frac{j(j-1)}{2}.$$

Over the entire interval $[L, R]$, the value of q is constant, that is

$$A_{[L,R]} = \sum_{j=L}^{R} q \cdot \frac{j(j-1)}{2} = q \cdot \frac{1}{2} \sum_{j=L}^{R} (j(j-1)).$$

Note that

$$j(j-1) = j^2 - j.$$

Then

$$\sum_{j=L}^{R} (j(j-1)) = \sum_{j=L}^{R} j^2 - \sum_{j=L}^{R} j = S_2(L,R) - S_1(L,R).$$

From this, we have

$$A_{[L,R]} = q \cdot \frac{1}{2}\big(S_2(L,R) - S_1(L,R)\big).$$

Part 4. Contribution of the interval $[L, R]$ to B

Now consider

$$B = \sum_{j=1}^{n} \frac{(n-qj)(n-qj+1)}{2}.$$

Let $r = n - qj$. Then

$$\frac{r(r+1)}{2} = \frac{(n-qj)(n-qj+1)}{2}.$$

Expanding the brackets:

$$r(r+1) = (n-qj)(n-qj+1) = (n-qj)(n+1-qj).$$

Multiplying:

$$(n-qj)(n+1-qj) = n(n+1) - qj(2n+1) + q^2 j^2.$$

Thus,

$$\frac{(n-qj)(n-qj+1)}{2} = \frac{1}{2}\left(n(n+1) - q(2n+1)\cdot j + q^2\cdot j^2\right).$$

Then the contribution of the interval $[L, R]$:

$$B_{[L,R]} = \sum_{j=L}^{R} \frac{1}{2}\left(n(n+1) - q(2n+1)\cdot j + q^2\cdot j^2\right) = \frac{1}{2}\Big[n(n+1)\cdot \mathrm{cnt} - q(2n+1)\cdot S_1(L,R) + q^2\cdot S_2(L,R)\Big],$$

where

$$\mathrm{cnt} = R - L + 1.$$

Part 5. Final Formula

By summing the contributions $A_{[L,R]}$ and $B_{[L,R]}$, we obtain the contribution of the interval $[L, R]$ to the overall answer:

$$S_{[L,R]} = A_{[L,R]} + B_{[L,R]}.$$

It remains to sum this over all intervals corresponding to different values of $q = \left\lfloor \frac{n}{j} \right\rfloor$.

Since the number of such intervals is $O(\sqrt{n})$, the overall asymptotic complexity of the algorithm will be $O(\sqrt{n})$, which is feasible for $n \le 10^{12}$.

Part 6. Implementation Modulo

Calculations are performed modulo $M = 998244353$.

You need to be able to:

- quickly multiply large numbers with modulo. In C++, for intermediate products, you can use the type `__int128`;
- compute inverse elements $\frac{1}{2}$, $\frac{1}{3}$ (and thus $\frac{1}{6}$) modulo M using exponentiation to $M-2$ by Fermat's little theorem;
- carefully bring all formulas S_1, S_2, cnt, $A_{[L,R]}$, $B_{[L,R]}$ to arithmetic modulo.

Code structure:

1. Read n.
2. Precompute $\mathrm{inv2} = 2^{M-2} \bmod M$, $\mathrm{inv3} = 3^{M-2} \bmod M$.
3. Define functions for the sum of the first x numbers and the sum of the first x squares modulo:

$$\sum_{k=1}^{x} k = \frac{x(x+1)}{2}, \qquad \sum_{k=1}^{x} k^2 = \frac{x(x+1)(2x+1)}{6}.$$

4. Start a loop over intervals:

```
i = 1
while i <= n:
    q = n // i
    R = n // q
    L = i
```

```
compute contribution from [L, R]
i = R + 1
```

5. Output the accumulated sum modulo M.

Part 7. Verification with Example

Let $n = 5$. Then manually:

$i \bmod j$	$j=1$	2	3	4	5
$i=1$	0	1	1	1	1
2	0	0	2	2	2
3	0	1	0	3	3
4	0	0	1	0	4
5	0	1	2	1	0

The sum of all elements in the table equals 26. The algorithm described above also yields 26.

Conclusion

Main ideas of the solution:

1. Renumber the sum over j and express $\sum(i \bmod j)$ for a fixed j in terms of the number of complete blocks $q = \lfloor n/j \rfloor$ and the tail.
2. Notice that $q = \lfloor n/j \rfloor$ takes few different values, and process entire intervals $j \in [L, R]$ with the same q.
3. For each interval, express the contribution using elementary sums 1, j, j^2, which are computed using formulas.
4. Compute everything modulo 998244353 and use modular inverses.

The asymptotic complexity of the solution is $O(\sqrt{n})$, which is sufficient for $n \le 10^{12}$.