TAMa 2024 Editorials

Cisco Ortega

Contents

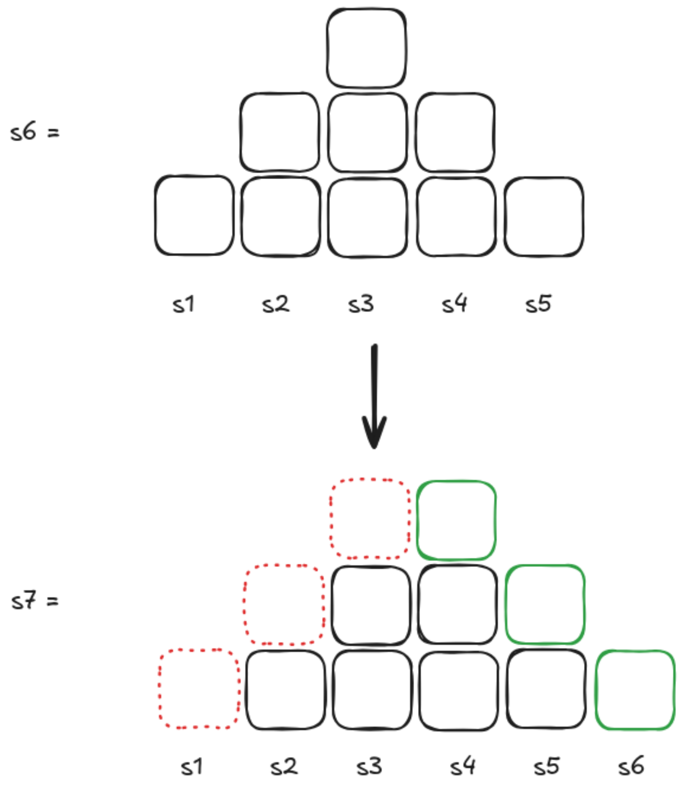
A) Nostalgia

Subtasks 1 & 2

Just code it.

Subtask 3

You can create a "sliding window" of sorts. Let $w_t = s_t + s_{t-1} + \cdots + s_{t-(k-1)}$. Then, you can compute the value of s_n from the value of s_{n-1} in just $O(1)$ by using two windows. See the following diagram.



You can compute the leftmost window in $O(k)$ time, then each subsequent window can be computed *from the previous one* in $O(1)$ time by subtracting out the old leftmost element and adding in the new rightmost element (of the new window).

Thus, the ultimate solution is $O(n + k)$.

Subtask 4

Google `quickly evaluate linear recurrence relations comp prog` or something similar and you'll discover the standard technique of how to do so using *fast matrix exponentiation.* You can ask around for good tutorials on this technique.

This solves the problem in $O(k^3 \lg n)$.

For an example of what to expect, consider the following matrix-vector equation for $k = 3$. Let $a_n = s_1 + s_2 + \cdots + s_n$ be the value we want to compute.

$$\begin{bmatrix} 1 & 2 & 3 & 2 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_{t+4} \\ s_{t+3} \\ s_{t+2} \\ s_{t+1} \\ s_t \\ a_t \end{bmatrix} = \begin{bmatrix} s_{(t+4)+1} \\ s_{(t+3)+1} \\ s_{(t+2)+1} \\ s_{(t+1)+1} \\ s_{t+1} \\ a_{t+1} \end{bmatrix}.$$

Repeated application of the above matrix-vector equation then (inductively) leads us to the following relation:

$$\begin{bmatrix} 1 & 2 & 3 & 2 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}^n \begin{bmatrix} s_4 \\ s_3 \\ s_2 \\ s_1 \\ s_0 \\ a_0 \end{bmatrix} = \begin{bmatrix} s_{4+n} \\ s_{3+n} \\ s_{2+n} \\ s_{1+n} \\ s_{0+n} \\ a_{0+n} \end{bmatrix}$$

which will give us the value of a_n that we want (if we speed up repeated matrix multiplication using fast exponentiation).

Your solution should smell like this, or at least something close to this, though you may choose to have slightly different "implementation details" with the matrix you choose to construct.

B) Holy Molé

Subtasks 1 & 2

For small n, write a brute forcer that directly simulates the problem (or do that on pen and paper).

(The brute forcer program can be directly used to solve Subtask 1)

Bowl	Touched by Angel 0	Touched by Angel 1	Touched by Angel 2
1	Y	Y	Y
2	N	Y	Y
3	Y	N	Y
4	N	N	Y
5	Y	Y	N
6	N	Y	N
7	Y	N	N
8	N	N	N

Then, make the following observation:

- There is a one-to-one correspondence between the 2^n bowls, and the 2^n possible subsets of "Which angels touched this bowl?"

You can prove this by induction.

So, to count how many bowls have been touched by r angels, we can equivalently count how many ways there are to choose r out of the n angels (to be the ones to touch those bowls). Thus, the answer is

$$\sum_{r=m}^{n} \binom{n}{r}.$$

Because n is small, you can compute the binomial coefficients by just about any method and it'll probably work out.

Subtask 3

Recall that what we want to compute is each of the following:

$$\binom{n}{r} \pmod p$$
$$= \frac{n!}{r!(n-r)!} \pmod p.$$

But we can precompute all values of $k!$ for k from 1 to n in just $O(n)$ time—use the fact that $k! = k \cdot (k-1)!$

We can also precompute the **modular multiplicative inverses** of each of the factorials using Fermat's Little Theorem or the Extended Euclidean Algorithm.

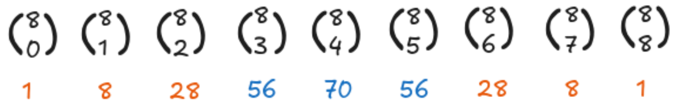
Thus, after these precomputations, each binomial coefficient can be computed in $O(1)$ time by direct use of the factorial formula, leading to an $O(n)$ solution overall.

Subtask 4

Do the digits of n and m look peculiar? Note that m is *really close* to $n/2$. Specifically, $m - n/2 = 10^6$ only.

By leveraging the symmetry of the binomial coefficients, you only need to compute this middle bit that ranges from $n - m$ to m (**exclusive** of those endpoints), from which you can derive the actual answer we want.

For example, suppose $n = 8$ and $m = 6$. In the diagram below, we break the sequence of binomial coefficients into three parts. Due to symmetry, we know that the two orange parts are equal (and the value we desire is *one* of these orange parts).


$\binom{8}{0}$ $\binom{8}{1}$ $\binom{8}{2}$ $\binom{8}{3}$ $\binom{8}{4}$ $\binom{8}{5}$ $\binom{8}{6}$ $\binom{8}{7}$ $\binom{8}{8}$
1 8 28 56 70 56 28 8 1

We know the sum of *all* the binomial coefficients is 2^8. We can subtract out $\binom{8}{3} + \binom{8}{4} + \binom{8}{5}$ (the central blue part), then divide the result by two (to get the total of *just one* orange part).

We only have to evaluate 2×10^6 binomial coefficients for the "central blue part" of our actual problem, which sounds reasonable. But we still need to evaluate gigantic binomial coefficients very quickly.

The key is that $p = 10^7 + 19$ is also quite small, and also it is prime. So, leverage this small prime modulus. Here are some theorems you could use:

- Lucas' Theorem (relates binomial coefficients with modulo)
- Wilson's Theorem (relates factorials with modulo)

C) The Amazons in their Prime

Subtasks 1 & 2

In one session, Maxi completes $\lceil \frac{n}{k} \rceil$ reps and Mini completes $\lfloor \frac{n}{k} \rfloor$ reps.

So, just directly compute

$$M = \sum_{\ell_n \le n \le r_n} \sum_{\ell_k \le k \le r_k} \left\lceil \frac{n}{k} \right\rceil$$

and

$$m = \sum_{\ell_n \le n \le r_n} \sum_{\ell_k \le k \le r_k} \left\lfloor \frac{n}{k} \right\rfloor$$

then compute $M - m$.

Subtask 3

Combine both summations into one.

$$M - m = \sum_{\ell_n \le n \le r_n} \sum_{\ell_k \le k \le r_k} \left(\left\lceil \frac{n}{k} \right\rceil - \left\lfloor \frac{n}{k} \right\rfloor \right).$$

However, $\lceil \frac{n}{k} \rceil$ and $\lfloor \frac{n}{k} \rfloor$ are almost exactly the same, except one rounds up and the other rounds down. In fact, they are only exactly the same when k divides n exactly and there is no remainder. So,

$$\left\lceil \frac{n}{k} \right\rceil - \left\lfloor \frac{n}{k} \right\rfloor = \begin{cases} 0, & k \text{ divides } n \\ 1, & \text{otherwise.} \end{cases}$$

So, rewrite the above summation as

$$\sum_{\ell_n \le n \le r_n} \sum_{\ell_k \le k \le r_k} (0 \text{ if } k \text{ divides } n, \text{ else } 1)$$

But, reasoning with divisors is easier than reasoning about *non-divisors*, so rewrite the above as:

$$\begin{aligned} & \sum_{\ell_n \le n \le r_n} \sum_{\ell_k \le k \le r_k} (1 - (1 \text{ if } k \text{ divides } n, \text{ else } 0)) \\ = (r_n - \ell_n + 1)(r_k - \ell_k + 1) \; - & \sum_{\ell_n \le n \le r_n} \sum_{\ell_k \le k \le r_k} (1 \text{ if } k \text{ divides } n, \text{ else } 0). \\ = (r_n - \ell_n + 1)(r_k - \ell_k + 1) \; - & \sum_{\ell_n \le n \le r_n} (\text{count of divisors of } n \text{ that are in } [\ell_k, r_k]). \end{aligned}$$

Thus, for each n from ℓ_n to r_n, we want to count how many of its divisors lie in the range $[\ell_k, r_k]$, which can be computed in $O(r_n \lg r_n)$ by a slight modification of a standard divisor sieve.

Subtask 4

It can be shown that if we can solve the following problem:

$$\sum_{1\le n\le N}\ \sum_{1\le k\le K}(1 \text{ if } k \text{ divides } n, \text{ else } 0)$$

(that is, both: n ranges from 1 *to N, and k from* 1 *to K)* then we can solve the original problem (kind of similar to 2D prefix sums' flavor).

Let's unpack how the divisor sieve works, mathematically. It's just an interchange of summations:

$$\begin{aligned}\sum_{1\le n\le N}\ \sum_{1\le k\le K}(1 \text{ if } k \text{ divides } n, \text{ else } 0) &= \sum_{1\le k\le K}\ \sum_{1\le n\le N}(1 \text{ if } k \text{ divides } n, \text{ else } 0)\\ &= \sum_{1\le k\le K}(\text{count of } \textbf{multiples} \text{ of } k \text{ that are in } [1,N]).\end{aligned}$$

It turns out this is nicer, because enumerating multiples is *much easier* than finding divisors. The multiples of k go: k, $2k$, $3k$, $\dots$, therefore:

$$\sum_{1\le k\le K}(\text{count of } \textbf{multiples} \text{ of } k \text{ that are in } [1,N]) = \sum_{1\le k\le K}\left\lfloor \frac{N}{k}\right\rfloor.$$

We can evaluate this sum in $O(\sqrt{N})$ using the following "standard" fact:

- Let N be fixed and let k vary. Then, the expression $\lfloor \frac{N}{k}\rfloor$ takes on only $\approx 2\sqrt{N}$ different values:
  - Each of $k = 1, 2, 3, \dots, \sqrt{N}$ yields a different unique value of $\lfloor \frac{N}{k}\rfloor$
  - For each of $v = 1, 2, 3, \dots, \sqrt{N}$, consider the equation $\lfloor \frac{N}{k}\rfloor = v$; the solutions in k consist of some contiguous range of integers.

Brute force the contributions of the terms with $k = 1, 2, 3, \dots, \sqrt{N}$. Then, for each $v = 1, 2, 3, \dots, \sqrt{N}$, if you can **count** how many k there are such that $\lfloor \frac{N}{k}\rfloor = v$ (and $k \le K$), then you can determine how many times each of these v should be added to the sum.

D) SRS

Subtasks 1 & 2

Just directly generate the first n terms of the SRS.

Note that the entries in the spreadsheet consist of the terms in

$$(s_1 + s_2 + s_3 + \cdots + s_n)(s_1 + s_2 + s_3 + \cdots + s_n).$$

So from now on, we'll only care about computing the sum of the first n entries in the SRS. Just square that result to get the true final answer.

Fun fact: In the OEIS, you'll find that the SRS' "proper" name is the *Golomb sequence*!

Subtask 3

Naively compute the first few terms of the sequence, just like you would for Subtasks 1 and 2. These first few terms tell us more about an even *longer* prefix of the sequence.

More precisely, the first t terms of the sequence allow us to then infer properties about the first $\sum_{i=1}^{t} s_i$ terms of the sequence, because we know the SRS should look like this:

- First, s_1 occurences of 1 (so we can add $s_1 \times 1$ to the total sum)
- Then, s_2 occurences of 2 (so we can add $s_2 \times 2$ to the total sum)
- Then, s_3 occurences of 3 (so we can add $s_3 \times 3$ to the total sum)
- Then, s_4 occurences of 4 (so we can add $s_4 \times 4$ to the total sum)
- Then, s_5 occurences of 5 (so we can add $s_5 \times 5$ to the total sum)
- $\vdots$

Concretely, for example, knowing that the first **five** terms are $[1, 2, 2, 3, 3]$ allows us to deduce that the first $1 + 2 + 2 + 3 + 3 = 11$ terms are $[1, 2, 2, 3, 3, 4, 4, 4, 5, 5, 5]$ (the appearances of the values up to 5).

Let t be the first index such that $\sum_{i=1}^{t} s_i \geq n$. Then, the sum of the first n terms of the SRS consists of

$$\sum_{i=1}^{t-1} s_i \times i + (\text{some “incomplete” portion of the subsegment of } t\text{'s})$$

It turns out that $t \approx 5.4 \times 10^8$ is enough to reach $n = 10^{14}$.

Subtask 4

Take this reasoning and apply it one level deeper. As a concrete example:

- We compute that $s_4 = 3$. This means that that the number 4 appears 3 times in the sequence. In particular:
  - Note that $s_1 + s_2 + s_3 = 5$, which means we have already appended 5 terms into the sequence so far (that is, it currently looks like $[1, 2, 2, 3, 3]$)
  - That means the subsegment of $[4, 4, 4]$ we append will run from index 6 until index 8
- **That means** we know numbers 6 through 8 *each appear* 4 *times*, which means we can predict that later on in the sequence, we will see a part that looks like $[...6, 6, 6, 6, 7, 7, 7, 7, 8, 8, 8, 8...]$—these are all the subsegments that each consist of repeating a number **four** times, and we know those repeated numbers are 6 and 7 and 8.

In general:

- Suppose we have directly computed s_i, for some i.
- From here, we can compute the ℓ such that $s_j = i$ for all $j \in [\ell, \ell + s_i)$.
- *From here* we know then that there is a part of the sequence that goes,

$$\underbrace{\ell, \ell, \ldots, \ell,}_{i \text{ times}} \underbrace{(\ell+1), (\ell+1), \ldots, (\ell+1),}_{i \text{ times}} \ldots \underbrace{(\ell+s_i-1), (\ell+s_i-1), \ldots, (\ell+s_i-1),}_{i \text{ times}}$$

That is, knowing the first t terms of the sequence then lets us compute the first $\sum_{i=1}^{t} i \times s_i$ terms of the sequence. Just as before, take the first t such that $\sum_{i=1}^{t} i \times s_i \geq n$. Then, the sum of the first n terms in the SRS is again "sum of contiguous runs + some incomplete subsegment at the end".

It turns out that t that is just a little bit bigger than 10^7 is enough to reach $n = 10^{18}$ using this method.

E) Sussy Spanning Trees

Subtasks 1 & 2

Googling `counting spanning trees` leads us to the Wikipedia page for spanning trees, which contains Cayley's formula: The number of spanning trees of the complete graph with n vertices is n^{n-2}.

So, just directly evaluate the sum

$$\sum_{n=1}^{N} n^{n-2} \pmod{m}.$$

Remember to use fast exponentiation so that each term can be computed in $O(\lg n)$ time only.

Subtask 3

The modulus is not prime. So, let's do a standard technique in Number Theory: prime factorize m, compute the value modulo each prime power, and then combine the results together using the Chinese Remainder Theorem.

We prime factorize,

$$693789912465408 = 2^{19} \times 3^3 \times 11 \times 47^2 \times 2017$$

and note that **all the prime powers are small**.

Note these two things:

$$r^{r-2} \equiv (r \bmod m)^{r-2} \pmod m$$

and, if $\gcd(r, m) = 1$, then

$$r^{r-2} \equiv r^{(r-2) \bmod \varphi(m)} \pmod m$$

where φ is Euler's totient function (this is Euler's Theorem). If $\gcd(r, m) \neq 1$, recall that our plan is to take moduli that are prime powers—for large enough r that are divisible by p, we have that $r^{r-2} \bmod p^k$ all become 0.

The sequence of bases is periodic modulo m, and the sequence of exponents is (eventually, mostly) periodic modulo $\varphi(m)$. Combining these insights, the overall sequence is eventually periodic, with period at most $\operatorname{lcm}(m, \varphi(m))$.

We can apply standard period-bashing techniques to compute each sum modulo each prime power, in linear time of the period (so overall, the running time is quadratic for each prime power).

Subtask 4

We're still going to use the Chinese Remainder Theorem. We prime factorize,

$$693789912465408 = 2^{22} \times 3 \times 5 \times 11 \times (10^6 + 3).$$

The largest prime factor is now too large for quadratic time solutions.

Let's hone in on this fact:

$$r^{r-2} \equiv (r \bmod m)^{r-2} \mod m.$$

Now, what if we focus on all terms in the sequence that have the same remainder r modulo m? Concretely, for example, suppose $m = 9$, and we look at adding up *only* the terms that are congruent to $r = 4$, modulo 9:

$$\begin{aligned} 4^2 + 13^{11} + 22^{20} + 31^{29} + \dots &\equiv 4^2 + 4^{11} + 4^{20} + 4^{29} + \dots && \mod 9 \\ &\equiv 4^2(4^0 + 4^9 + 4^{18} + 4^{27} + \dots) && \mod 9 \\ &\equiv 4^2(4^0 + (4^9)^1 + (4^9)^2 + (4^9)^3 + \dots) && \mod 9. \end{aligned}$$

This is a geometric series with common ratio 4^9, and so it can be evaluated in logarithmic time! Be careful about the cases where $\gcd(r-1, m) \neq 1$, because then $r - 1$ would not be invertible.

Sum up, for each possible remainder $r = 1, 2, \dots, m-1$, the geometric series consisting of the terms congruent to r (modulo m). The common ratio is r^m, and the initial term is r^{r-2}. The number of terms in the geometric series of some r is not hard to find. Sum all the results together.

$$\sum_{r=1}^{m-1} r^{r-2}(1 + r^m + (r^m)^2 + (r^m)^3 + \dots + (r^m)^{\text{something}}).$$

This now gives a solution that runs in linearithmic time for each prime factor.

F) Digivisible

Subtask 1

You can solve this ad hoc with pen and paper.

- If the number has a 2, then it must be the last digit.
- If the number has a 5, then it must be the last digit.

This makes it tractable as a classic "How many n-digit numbers satisfy this property...?" introductory combinatorics problem.

Subtask 2

There are only $3^3 + 3^4 + \cdots + 3^{16}$ possible numbers to consider, which is not that many. Just write a backtracking program that enumerates all of them, and test each one for digivisibility.

Subtasks 3 & 4

You can maybe come up with some ad hoc casework solution for Subtask 3 which uses the choose function (considering there are divisibility tests for 3 and 6 and 9). But I'll just jump to talking about the general method now.

Consider this simpler task: How many n-digit numbers using only the digits from some set S are divisible by m?

This has a pretty classical solution using DP. Construct the number digit-by-digit, at each step maintaining only the value modulo m of the prefix you've constructed so far. Your state should look something like `dp(i, r)`, where i is the index of the digit we're currently at, and $r \in [0, m)$ is the value of the prefix so far (modulo m).

For the transition, consider each possible next digit to append. To append a new digit, recall that concatenating `d` to the end of some number n is equivalently to performing $10n + d$.

For digivisibility, we just need to modify this solution slightly. This is enough to solve Subtask 3.

- Keep track of the value of the prefix so far, modulo $m := 2520$, which is $\operatorname{lcm}(1, 2, \ldots, 9)$.
    - It can be shown that if $d \mid m$, then: $n \bmod d = (n \bmod m) \bmod d$.
- Also include in our state a bitmask `mask` which keeps track of which digits we've used so far, since this tells us the digits whose divisibility we need to check when we're done constructing the number.

To extend it to Subtask 4, where ℓ and r are not powers of 10 any more, we just need to include an extra boolean parameter in order to do standard "digit DP" things (you can ask around for a good digit DP tutorial).

The total number of states comes out to be $\operatorname{len}(r) \times \operatorname{lcm}(1, 2, \ldots, 10) \times 2^9 \times 2$, which comfortably passes.

G) Supersetter

Subtasks 1 & 2

Just code it.

For subtask 2, you can use either backtracking or bitmasking to enumerate all possible partitions of the digits.

Subtask 3

Suppose we have a subsegment that ranges from indices ℓ to r inclusive. Regardless of how we choose to partition the rest of the digits, this subsegment always contributes the same value to the total sum.

For example, suppose we have `1234567890` and focus on what if `3456` formed a subsegment. Then, the number would look something like

```
????(34563456)????????
```

That is, whenever `3456` is a subsegment, it always contributes 3456345600000000 to the sum. There are eight trailing zeros because there are four digits that come after `3456` in the original number—regardless of how those are partitioned, it will always result in eight digits coming after the `34563456` part.

Next, we need to *find out* how many partitions there are that have e.g. `3456` as one contiguous block. This is not too hard either—it consists of, independently, determining how to partition the digits to its left, and determining how to partition the digits to its right.

For example, there are $2^{2-1} \times 2^{4-1}$ partitions of the digits of `1234567890` that have `3456` as a contiguous block, therefore we must add 3456345600000000 to our sum a total of that many times.

Putting that all together, we have a formula for the contribution of some $[\ell, r]$ subsegment, so we can just do a summation that ranges over all $O(s^2)$ such subsegments:

$$\sum_{l=1}^{s}\sum_{r=\ell}^{s}\text{duplicate}(\texttt{n[l..r]}) \times 10^{\text{appropriate no. of trailing zeros}} \times$$
$$2^{\max(0,\ \text{no. of digits to its left}-1)} \times 2^{\max(0,\ \text{no. of digits to its right}-1)}$$

Even if you implement these functions somewhat naively, cubic time solutions should also be acceptable.

Subtask 4

Take the idea from Subtask 3 one step further. For each *individual index i*, what is the contribution of the digit at this index i to the overall sum?

We once again consider all subsegments $[\ell, r]$ that contain i. For example, suppose we have `1234567890` and focus on the digit `4`, and suppose we consider all cases where `3456` form a subsegment. Then, the contribution of *specifically* `4` in this case is:

```
????(?4???4??)????????
```

In this case, similar to before: the values of ℓ and r determine the number of trailing zeros in the contribution of the digit and of the duplicated digit. So, we get a similar summation as the one in Subtask 3:

$$\sum_{i=1}^{s}\sum_{l=1}^{i}\sum_{r=i}^{s}\text{duplicate}(\texttt{n[i]}, \ell, \text{r}) \times 10^{\text{appropriate no. of trailing zeros}} \times \\ 2^{\max(0,\text{ no. of digits to its left}-1)} \times 2^{\max(0,\text{ no. of digits to its right}-1)}$$

Fully write out all of the terms shown above, and then just do a lot of pen-and-paper algebra on summations: interchange their order, perform index shifts, factor out terms independent of the dummy variable, etc. etc. After a lot of work, you should eventually up with something like this:

$$\sum_{i=1}^{s}\texttt{n[i]} \cdot (\text{some geometric series}) + \sum_{i=1}^{s}\texttt{n[i]} \cdot (\text{some other geometric series})$$

(or possibly more, depending on how messy you are with handling edge cases) where each geometric series can be evaluate in logarithmic time, using its summation formula (and modular multiplicative inverses).

This yields a solution which runs in linearithmic time (with respect to the number of digits).

H) Expected Value of Civil War

Subtasks 1 & 2

Just code it.

For subtask 2, you can use either backtracking or bitmasking to enumerate all possible ways to choose $n/2$ mercenaries.

Subtasks 3 & 4

First things first, you can generate primes using the Sieve of Eratosthenes.

Now, onto the actual problem. Let s be the sum of the first n prime numbers. Then, express sum(T') as $s -$ sum(T).

Let H be the set of all $n/2$ sized subsets of the first n prime numbers. Now, we just do summation algebra with the given expression.

$$\begin{aligned}
\sum_{T\in H}(\mathrm{sum}(T) - \mathrm{sum}(T'))^2 &= \sum_{T\in H}(2\ \mathrm{sum}(T) - s)^2 \\
&= \sum_{T\in H}(4\ \mathrm{sum}(T)^2 - 4s\ \mathrm{sum}(T) + s^2) \\
&= 4\sum_{T\in H}\mathrm{sum}(T)^2 - 4s\sum_{T\in H}\mathrm{sum}(T) + s^2\sum_{T\in H}1.
\end{aligned}$$

So, the problem is reduced to solving each of those summations independently.

Part I: $\sum_{T\in H} 1$

This is just *counting* the number of elements of H, i.e. the number of subsets with $n/2$ elements out of a larger set of size n. But we know that's just $\binom{n}{n/2}$.

Part II: $\sum_{T \in H} \operatorname{sum}(T)$

Expand out sum(T) and interchange the order of summations. Let P be the set of the first n primes.

$$\begin{aligned}\sum_{T \in H} \sum_{\substack{p \in P \\ p \in T}} p &= \sum_{p \in P} \sum_{\substack{T \in H \\ T \ni P}} p \\ &= \sum_{p \in P} p \sum_{\substack{T \in H \\ T \ni p}} 1.\end{aligned}$$

Instead of iterating over all T, then enumerating all primes p contained in this T: we iterate over all primes p, then *count* the number of sets T which contain this p. But note that "the number of T which contain some prime p" is always the same, regardless of the actual value of p.

The answer is always $\binom{n-1}{n/2-1}$. We include p in the subset; then, of the remaining $n-1$ primes, we need to choose the other $n/2-1$ elements of the subset.

So, the above summation is just

$$\sum_{p \in P} p \binom{n-1}{n/2-1}$$

which can be computed in linear time.

Part III: $\sum_{T\in H} \operatorname{sum}(T)^2$

Again, expand out the inner summation and interchange the order of summations again.

$$\begin{aligned}\sum_{T\in H}\left(\sum_{\substack{p\in P\\ p\in T}} p\right)^2 &= \sum_{T\in H}\sum_{\substack{p\in P\\ p\in T}}\sum_{\substack{q\in P\\ q\in T}} pq\\ &= \sum_{p\in P}\sum_{q\in P}\sum_{\substack{T\in H\\ T\ni p,q}} pq\\ &= \sum_{p\in P}\sum_{q\in P} pq \sum_{\substack{T\in H\\ T\ni p,q}} 1.\end{aligned}$$

That is to say, consider all pairs of primes (p,q). We wish to *count* the number of sets T which contain *both* p and q. But that's again (mostly) independent of p and q:

- If $p = q$, then the count is $\binom{n-1}{n/2-1}$, as earlier.
- If $p \neq q$, then the count is $\binom{n-2}{n/2-2}$ for basically the same reason.

All in all, we are now at this summation:

$$\sum_{p\in P} p^2 \binom{n-1}{n/2-1} + \sum_{\substack{p,q\in P\\ p\neq q}} pq \binom{n-2}{n/2-2}.$$

Directly evaluating the summation on the right in $O(n^2)$ gets you Subtask 3.

Computing it in linear time (which isn't too hard) gets you Subtask 4.

Hint: Consider again the expression $\left(\sum_{p\in P} p\right)^2$.

I) Stratosphere

Subtask 1

You can enumerate all paths using a computer program (or by hand). Or, you can do the Subtask 2 solution by hand.

Subtask 2

This is a classic DP problem. Let $\mathrm{dp}(i, j)$ be the sum of the thrills of all paths *that start at* (i, j) and end on the bottom right corner.

Construct the path step by step. From some position (i, j), ask: Do I go down, or do I go right? Either way, the path has already passed through cell (i, j). Then, after the move, we need to consider the sum of the thrills of all paths from our new position.

You should get something like:

$$\mathrm{dp}(i, j) = v_{i,j} \cdot (\mathrm{dp}(i + 1, j) + \mathrm{dp}(i, j + 1))$$

where $v_{i,j} = r - (i - 1)$ (assuming 1-indexing) is the value of the number at that cell.

Subtasks 3 & 4

Note that each path can be uniquely identified by *how many squares of our path are in each row.* For example, the path in the sample is the unique one that passes through 1 thrice, through 2 once, and through 3 twice.

So, essentially, we want to evaluate this sum of products:

$$\sum 1^{i_1}\ 2^{i_2}\ 3^{i_3} \ldots r^{i_r}$$

across all tuples $(i_1, i_2, i_3, \ldots, i_r)$ such that:

- $1 \leq i_t \leq c$ for each t;
- $\sum i_t = r + c - 1$.

This is some kind of convolution, and smells like a perfect place to apply generating functions! Specifically, what we want is the coefficient of x^{r+c-1} in the polynomial:

$$(1x + (1x)^2 + \cdots + (1x)^c)(2x + (2x)^2 + \cdots + (2x)^c) \ldots (rx + (rx)^2 + \cdots + (rx)^c).$$

First, apply the following simplification. The above polynomial is equal to (using product notation now):

$$r!x^r \prod_{t=1}^{r} ((tx)^0 + (tx)^1 + \cdots + (tx)^{c-1})$$

by factoring out tx from each term for $t = 1$ to r. But, the coefficient of x^{r+c-1} in this polynomimal is equivalent to the coefficient of x^{c-1} in

$$r! \prod_{t=1}^{r} ((tx)^0 + (tx)^1 + \cdots + (tx)^{c-1})$$

(that is, we divided everything by x^r).

We want the coefficient of x^{c-1} in this polynomial. That means even if we introduce *extra terms* in the polynomial, the answer won't change, as long as all terms have degree c or higher. So, we can equivalently consider:

$$r!\prod_{t=1}^{r}((tx)^0 + (tx)^1 + \cdots + (tx)^{c-1} + (tx)^c + (tx)^{c+1} + \dots)$$

Our finite geometric series becomes an *infinite* one, which has a nicer formula!

$$r!\prod_{t=1}^{r}\frac{1}{1-tx}.$$

At this point, you *may* recognize this as one of the generating functions of the Stirling numbers of the second kind! Then, you can go to Wikipedia and find an explicit formula that can be computed in $O(r)$. But if not, we can proceed—these next steps essentially provide a *proof* of that explicit formula.

From here, we can do familiar partial fraction decomposition. There exist coefficients $k_1, k_2, \dots, k_r$ such that

$$r!\prod_{t=1}^{r}\frac{1}{1-tx} = r!\sum_{t=1}^{r}\frac{k_t}{1-tx}.$$

From standard partial fractions techniques, we know that we can compute each k_t using the following relation:

$$k_t\prod_{q\neq t}(1-q/t) = 1.$$

Then, the coefficient of x^{c-1} overall can be computed as the sum of the coefficient of x^{c-1} in each individual summand. But each individual summand is an infinite geometric series, so it contributes $k_t \cdot t^{c-1}$ to the sum.

If you compute the k_t directly in quadratic time, you get Subtask 3.

To get Subtask 4, you have to compute them more smartly, which can be done by noticing shared computations between each k_t (instead of starting over from scratch every time).

J) Roll the Dice

Subtasks 1 & 2

Just code it.

Note that for Subtask 2, cubic time solutions won't pass. You can afford to iterate over all subarrays, but you cannot afford to literally compute each subarray's value from scratch.

Fortunately, it's not too hard to fix, and rather intuitively too—the result of applying commands ℓ through r can be found by: taking the results from applying commands ℓ through $r-1$, *and then* applying command r.

So, each of the $O(n^2)$ subarrays can be processed in $O(1)$ time only.

Subtask 3

We're pretty much going to do DP.

Let $\mathrm{dp}_r(x)$ count the number of subarrays whose right endpoint is r, and which result in face x facing upwards. Let y be the face such that: if face y is on top, then performing a right hand twist around face a_r results in face x now being on top.

We see that if we want actions ℓ through r to land on face x, then it must be the case that actions ℓ through $r-1$ land on face y (and then action r makes it land on face x).

$$\mathrm{dp}_r(x) = \mathrm{dp}_{r-1}(y) + (1 \text{ if } y = 1, \text{ else } 0),$$

Here, the (1 if $y = 1$, else 0) term corresponds to the fact that we can start a *new* subarray here whose left endpoint is r (but as per the problem statement, a fresh die always start with 1 facing up).

We can evaluate all of $\mathrm{dp}_r(x)$ from $r = 1$ to n and $x = 1$ to 6 in linear time.

Subtask 4

Note that the sequence b *(and thus also the sequence a)* is periodic, with period 2^{18}. To solve the problem for $n = 10^{18}$, we just abuse this periodicity with standard period-bashing techniques, since the same 2^{18} actions will be repeated over and over again.

We have six linear recurrence relations that evolve in lockstep with each other. Thus, we can encode each possible transition as one of six possible *matrices* (one for each possible value of a_i, i.e. which face we're twisting around).

Let A be the matrix that we get by multiplying the corresponding matrices of commands 1 through 2^{18}, in order. Then, applying the transition encoded by A over and over and over again can be represent by fast matrix exponentiation—we raise A to the power of $10^{18}/2^{18}$.

In order to get the *sum* of these values for all r from 1 to n, we have a few options:

- Augment the matrices so that we also keep track of the running sum for each face.
- Use the geometric series formula
- Use divide and conquer

Any of these should work.