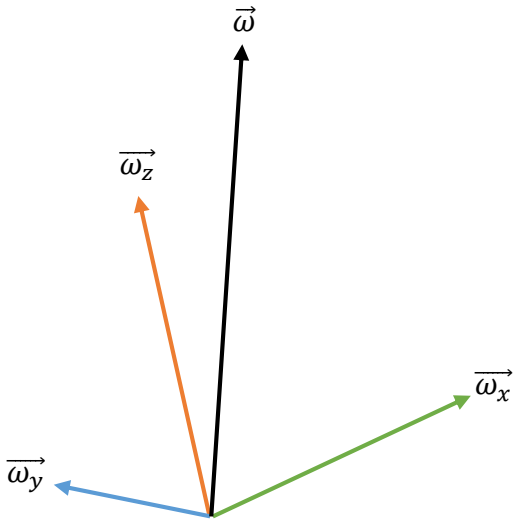
1001. Avian Darts

Difficulty: Medium Author: Rim Gwang Song

This problem has the most complex and polite description in this problemset.

Once the pilot makes a turn, his aircraft will fly along a "spring" on the surface of a cylinder.

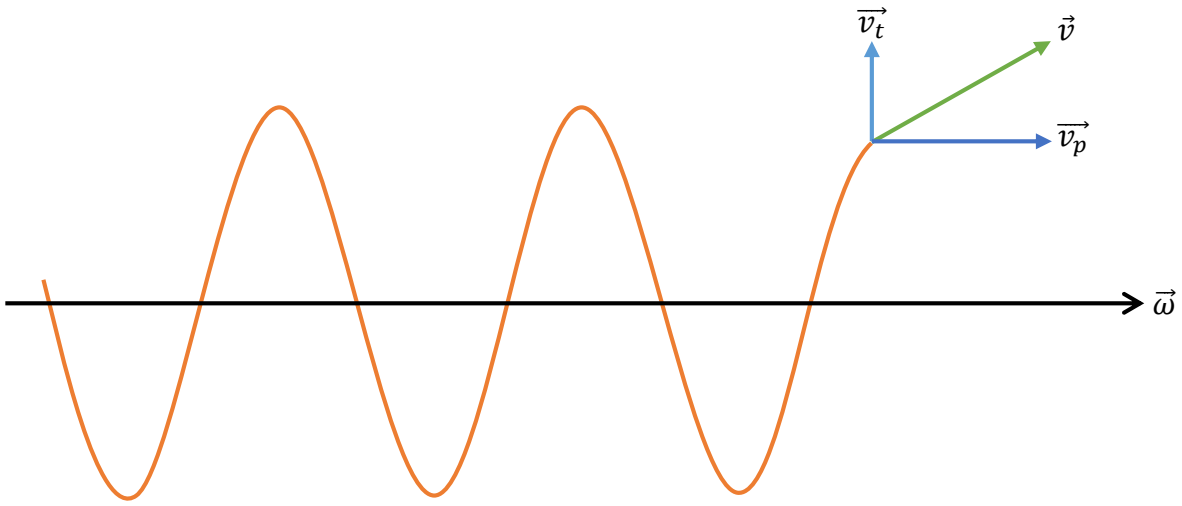
Let $\vec{i}, \vec{j}, \vec{k}$ be unit vectors towards front, left and up of the aircraft. Then $\overrightarrow{\omega_x} = \omega_x\vec{i}$, $\overrightarrow{\omega_y} = \omega_y\vec{j}$, $\overrightarrow{\omega_z} = \omega_z\vec{k}$ and $\vec{v} = v\vec{i}$. Finally, $\vec{\omega} = \overrightarrow{\omega_x} + \overrightarrow{\omega_y} + \overrightarrow{\omega_z}$.



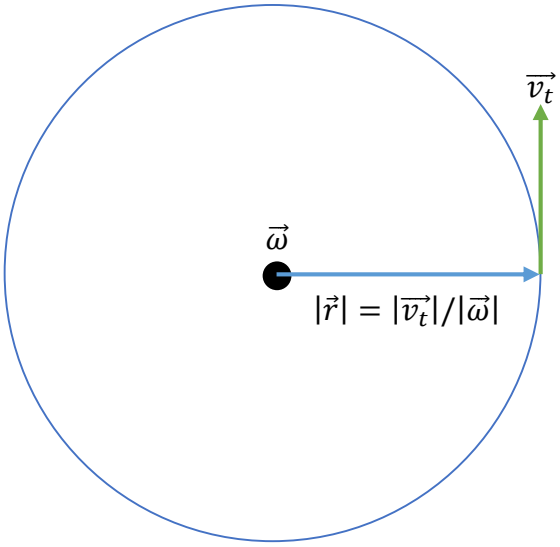
After performing some simple experiments, you can easily recognize that $\vec{\omega}$ is the angular velocity, which means that $\vec{\omega}$ is parallel to the axis of revolution and length of $\vec{\omega}$ equals speed of revolution.

Therefore, $\vec{i}$ which has an identical direction as the aircraft's velocity turns around $\vec{\omega}$.

We can represent $\vec{v}$ as follows.



$$\vec{v} = \overrightarrow{v_p} + \overrightarrow{v_t} \ (\overrightarrow{v_p} \parallel \vec{\omega}, \overrightarrow{v_t} \perp \vec{\omega})$$



Here, $\overrightarrow{v_p}$ is constant and $\overrightarrow{v_t}$ rolls around $\vec{\omega}$. Consequently, projection of the aircraft to a plane vertical to $\vec{\omega}$ always draw a circle with radius of $|\overrightarrow{v_t}|/|\vec{\omega}|$.

As a result, trajectory of the aircraft can be described as a "spring".

Thus, you can solve this task in linear time.

1002. Boring Task

Difficulty: Hard Author: Mun So Min

A few observations might be needed. Sort all the tasks in ascending order of T and renumber the tasks according to this order. Let the last processed task be the i^{th} task. For $j > i$ and $k < j$, if the k^{th} task is the next task scheduled after the j^{th} task has finished, then we can swap the order of these two tasks. Thus we may assume that the order of the last $N - i + 1$ tasks should be $i + 1, i + 2, \cdots, N$ and then i.

One more observation should be made. For the i^{th} task (i.e. the last processed task in above schedule) and for all $j > i$, we may assume that $T_i + D_i > T_j + D_j$. If not, for some $j > i$, $T_i + D_i \leq T_j + D_j$ should be held. We can choose the largest such j and then change the schedule of the last $N - i + 1$ tasks into $i + 1, i + 2, \cdots, j - 1, i, j + 1, j + 2, \ldots, N, j$, maintaining *gaps* as the original schedule. Here, a *gap* is a time interval the machine becomes free between two adjacent task. For each $k > j$, since $T_i + D_i > T_k + D_k$, start time for the k^{th} task won't be less than T_k. So new schedule will also be valid. The finish time of new schedule will remain the same as the original one. From the inequality $T_i + D_i \leq T_j + D_j$, waiting time of the j^{th} task will become no longer than waiting time of i^{th} task of original schedule. On the other hand, waiting time of other tasks won't increase for the same reason.

We may use binary search approach to find the answer. Then we should solve the following decisive problem. **For any integer M, is it possible to adjust start time of the tasks so that waiting time of any task never exceeds M?**

In order to solve the decisive problem described above, let's observe the latest start time when we deal with the last $N - i + 1$ tasks, and denote it by $DP[i]$. From above analysis, one

can sufficiently derive $O(N^2)$ algorithm to determine all these values, but it's too slow. The intended solution will calculate all those values in $O(N)$ time.

For each $i(1 \le i \le N)$, we can find the least integer $j > i$ satisfying $T_i + D_i \le T_j + D_j$. Let's define the value, for each i, as $next[i]$. Then iterate all integers k between $i+1$ and $next[i]$, and obtain the maximum value of $min(DP[k], T_i + D_i + M) - (Dsum[k-1] - Dsum[i-1])$. Here $Dsum[k]$ is a prefix sum of array D. But we must be careful with start time constraints. The constraint is that when we put the i^{th} task at time $min(DP[k], T_i + D_i + M) - D_i$ and then put $k-1, k-2, \cdots, (i+1)^{\text{th}}$ tasks before it in order, start time of those tasks must be not less than T value of them.

When we are iterating the k^{th} task, there are two possible situations:

- Some tasks of $\{i+1, i+2 \cdots, k-1\}$ do not satisfy their start time constraints.

  The value $DP[k]$ is not necessary any more for newly added tasks. If it is accessed again after $DP[i]$ has been determined, this implies that the current task has a larger $T + D$ value than $T_i + D_i$. T values are non-increasing when tasks are newly added, so D value is greater than D_i. At that time, the situation for the k^{th} task will become worse so it will never satisfy the constraint.

- All start time constraints are fulfilled.

  First, consider the case $DP[k] \le T_i + D_i + M$, then further updates from $DP[k]$ will be included in $DP[i]$. Therefore, in this case, $DP[k]$ will also be unnecessary.

  Now, observe the case $DP[k] > T_i + D_i + M$. To satisfy this inequality, k should be equal to $next[i]$. So we will face such case at most once for each i.

After that, the answer for the decisive problem will be "yes" if and only if $DP[i] \ge T_i$ stands for all i.

From all the above observations, we can solve the decisive problem in linear time. Since we've adapted binary search, the running time of full solution will be $O(N \cdot \log \sum D_i)$.

1003. Cookies

Difficulty: Medium Author: Kang Chol Ryong, Mun So Min

Initial index of k^{th} cookie can be obtained using binary search approach. Let's denote it by m. Then, you should find the value of f_m. It's difficult to get the value directly in reasonable time.

You may need to know some values of array f. Let's define an array g of length 4000, as $g_i = f_{2500000i}$. You can calculate all g values by using a sieve like an Eratosthenes's sieve 4000 times. Here, you may need the fact that $divmed(x)$ is the largest factor of x, not exceeding $\sqrt{x}$.

Preparation of all g values takes 10 minutes or so. You can store all $g_i - g_{i-1}$ values as decimal representation in your source code. It sufficiently fits the source code limit.

After that, f_m can be calculated by iterating between m and its nearest multiple of 2500000, by the same approach as pre-calculation. Then calculation amount of obtaining f_m is $1250000 \ln \sqrt{10^{10}} \approx 1.4 \cdot 10^7$ for the worst case.

1004. Distinct Sub-palindromes

Difficulty: Easy Author: Hwang Baek I

Here is the easiest problem.

If $n \leq 3$, any string is available since they have n distinct sub-palindromes.

If $n > 3$, the minimum possible number of distinct sub-palindromes is 3 when the string is represented as a form of "abcabcabcabc⋯".

1005. Fibonacci Sum

Difficulty: Easy Author: Hwang Baek I

As you know n^{th} element of Fibonacci sequence can be presented as follows:

$$F_n = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right]$$

Here $383008016^2 \equiv 616991993^2 \equiv 5 \pmod{10^9+9}$ stands. This is the key of the task.

By introducing some constants, the formula can be represented by $F_n \equiv d(a^n - b^n)\ (mod\ 10^9+9)$. Then

$$\sum_{i=0}^{N}(F_{Ci})^K \equiv \sum_{i=0}^{N}\{d(a^{Ci}-b^{Ci})\}^K\ \ (mod\ 10^9+9)$$

Once this expression is expanded each term is a summation of a geometrical series, which can be calculated quite easily. Time complexity will be $O(K \cdot \log N)$.

1006. Finding a MEX

Difficulty: Medium Author: Kang Chol Ryong

The only data structure problem.

First, for each vertex, *MEX* number can be at most its degree. So, we need to keep values which less than or equal to its degree.

Next, there are at most 10^5 edges, so the number of vertices with degree greater than or equal to 350 is less than 350. Let's call those vertices big vertices, and call the others small vertices.

And then, for each vertex u, the MEX number of S_u can be easily calculated using binary indexed tree(BIT) data structure. Also, for each vertex, as described above, MEX number can be at most its degree, so we don't need to create BIT with size of more than its degree.

For each query of type 1, there are two cases. In case of given vertex is a small vertex, we traverse all vertices which is directly connected from that vertex and add given value to them. In the other case, we just change the value of given vertex.

For each query of type 2, there are two types of vertices.

For all small vertices which are directly connected from given vertex, we have already gathered their values on BIT. The other big vertices still remain, but the number of such vertices is less than 350, so we can traverse all of them and get the answer.

Time complexity of this algorithm is $O(n\sqrt{n} \cdot \log n)$.

1007. Hunting Monsters

Difficulty: Hard Author: Mun So Min

For a monster, if his A value is not greater than his B value, then let's call him a $good\ monster$ and otherwise, let's call him a $bad\ monster$.

First of all, for every two monsters, we can determine the eliminating order if we eliminate both monsters as follows:

- If we eliminate a good monster and a bad monster it is better to eliminate good monster first.
- For every two good monsters, the monster who has a smaller value of A precedes the other one.
- For every two bad monsters, the one with a larger value of B should be put before the other one.

Sort all monsters according to the order observed above. After that, let's name the resulting sequence $killing\ sequence$.

One can derive quadratic dynamic programming solution, iterating killing sequence from back to front. But this is too slow. The intended solution has a time complexity of $O(N \cdot \log N)$.

The solution has two parts. First part of them is to deal with bad monsters. The other part is to process good monsters and combine these two results.

Let's define $dp[i][j]$ as the least energy to kill j bad monsters among the last i bad monsters based on the killing order. The transition of dynamic programming is as follows:

$$dp[i][j] = min(max(dp[i-1][j-1] - B[i], 0) + A[i], dp[i-1][j])$$

Here, $A[i]$ and $B[i]$ are parameters of the ith bad monster counted from the last.

There are 3 cases for $dp[i-1][j]$:

- If $dp[i-1][j] \le B[i]$, then the equality $dp[k][j] = dp[i-1][j]$ stands for all $k > i-1$, since $A[i] > B[i]$ and $B[i]$ increases when i increases.
- Else if $dp[i-1][j] > B[i]$ and $dp[i-1][j] \le A[i]$, then $dp[i][j] = dp[i-1][j]$.
- Else if $dp[i-1][j] - dp[i-1][j-1] \ge A[i] - B[i]$, then $dp[i][j] = dp[i-1][j-1] + A[i] - B[i]$.
- Else $dp[i][j] = dp[i-1][j]$.

The minimum j that fits the second/third/fourth case will be non-decreasing, so we can hold it. These j values will form an interval $[j_0, i-1]$, because of a trivial inequality $dp[i-1][j] \ge dp[i-1][j-1]$.

And also, we can notice that for those j values, $dp[i][j]$ will form a $\boldsymbol{convex}$ function on variable j. That is $dp[i][j+1] - dp[i][j] \ge dp[i][j] - dp[i][j-1]$, for $j \ge j_0$. Here, we define

$dp[i][j_0 - 1] = B[i]$, for conveniently. This convexity can be proved by mathematical induction on i. Try yourself and have fun.

Therefore, there exists an integer k, for all values l not smaller than k, $dp[i-1][l] - dp[i-1][l-1] \geq A[i] - B[i]$. Then we can easily find that if $l > k$, $dp[i][l] = dp[i-1][l-1] + A[i] - B[i]$, while $dp[i][l] = dp[i-1][l]$ for $l \leq k$.

To efficiently implement the first part, you may need a binary tree data structure (for example, `priority_queue`, `set` or `multiset` in STL of C++) for storing difference values, i.e. $dp[i][j] - dp[i][j-1]$. Each time, we should increase j_0 until it fits the second/third/fourth case. After that a new difference $A[i] - B[i]$ should be added to the data structure. The first part has just ended.

The second part is relatively easy than the first part. This part has two phases.

- First, one can show that if we kill K good monsters, the optimal way is to choose the very first K good monsters from the killing sequence. So in linear time, we can calculate the least energy values needed to kill K good monsters for all K.
- Merging phase can be done by solving the reverse problem. Given a certain amount of energy, calculate how many monsters we can eliminate. The amount of energy can be divided into intervals according to killing good monsters. For each interval, the maximum number of good monsters we can eliminate, is determined and also the final energy after killing good monsters. This energy is linear to the initial energy, so we can determine how much energy will be needed to kill at least a certain number of bad monsters. This process can be easily done in linear time.

So the total time complexity will be $O(N \cdot \log N)$.

1008. Integral Calculus

Difficulty: Medium Author: Mun So Min

This is the only FFT task of this problemset.

In order to solve the task, you should evaluate the following:

$$f(k) = \int_0^{+\infty} \frac{\tau^{-2k-1}}{e^{\frac{1}{\tau}} - 1} d\tau$$

After replacing argument as $t = \frac{1}{\tau}$, the above integral will be changed into the following:

$$f(k) = \int_0^{+\infty} \frac{t^{2k-1}}{e^t - 1} dt = \int_0^{+\infty} \left(\sum_{j=1}^{\infty} t^{2k-1} e^{-jt} \right) dt = \sum_{j=1}^{\infty} \left(\int_0^{+\infty} t^{2k-1} e^{-jt} dt \right) = \sum_{j=1}^{\infty} \frac{1}{j^{2k}} = \zeta(2k)$$

The following equation may help you to get $\zeta(2k)$, for $k = 1, 2, \cdots N$.

$$\sum_{k=1}^{\infty} \frac{x^k}{(2k)!} = \left(\sum_{k=0}^{\infty} \frac{x^k}{(2k+1)!} \right) \cdot \left(- \sum_{k=1}^{\infty} \frac{\zeta(2k)}{\pi^{2k}} \left(1 - \frac{1}{2^{2k}}\right) (-x)^k \right)$$

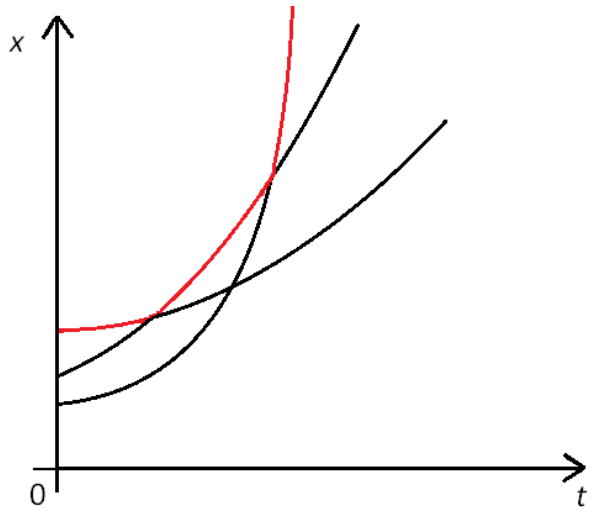
From the fact, the task can be done in $O(N \cdot \log N)$ by FFT.

1009. Leading Robots

Difficulty: Easy Author: Kim Song San

First of all, sort the robots in an ascending order of initial position.

Next, you have to maintain a queue of robots who can be the leader. You should iterate all the robots one by one, and update queue.



From the figure, you can easily notice that a robot can be pushed and popped at most once. You should be careful when some robots have the same initial position and acceleration speed.

Total running time of solution will be $O(N \cdot \log N)$.

1010. Math is Simple

Difficulty: Medium Author: Ryang Guk Hyok, Mun So Min

Let's define f_n and g_n, for all $n > 1$, as below:

$$f_n = \sum_{\substack{1 \le a < b \le n \\ \gcd(a,b)=1 \\ a+b \ge n}} \frac{1}{ab}$$

$$g_n = \sum_{\substack{1 \le a < b \le n \\ \gcd(a,b)=1 \\ a+b=n}} \frac{1}{ab} = \frac{1}{n} \sum_{\substack{1 \le a \le n \\ \gcd(a,n)=1}} \frac{1}{a}$$

Then the following fact stands for all $n > 1$:

$$f_n = f_{n-1} + \sum_{\substack{1 \le a \le n \\ \gcd(a,n)=1}} \frac{1}{an} - \sum_{\substack{1 \le a < b \le n \\ \gcd(a,b)=1 \\ a+b=n-1}} \frac{1}{ab} = f_{n-1} + g_n - g_{n-1} =$$

$$= f_{n-2} + g_{n-1} - g_{n-2} + g_n - g_{n-1} = f_{n-2} - g_{n-2} + g_n = \cdots = f_2 - g_2 + g_n = \frac{1}{2} + g_n$$

And also $g_n = \sum_{d|n} \mu(d)\, h_{n/d} \frac{1}{d}$ will be held, where $h_n = \sum_1^n \frac{1}{i}$ is a *harmonic sequence*.

You can pre-calculate all $h_{0\cdots10^8}$ values in a second, or so.

Before obtaining f_n, you may factorize n and then should calculate g_n by the above formula. Thus each query can be solved in $O(\sqrt{n} + |\{d|d|n, |\mu(d)| = 1\}|)$ time.

1011. Minimum Index

Difficulty: Easy Author: Mun So Min

There are many linear approaches to solve this task. The solution introduced here used Lyndon factorization. When we adapt this factorization, the last string will be the lexicographically minimum suffix.

We can incrementally add characters one by one, while using Lyndon factorization algorithm. During this process, we can hold the lexicographically minimum suffix and its smallest period, if we consider the current string appended by 'z'+1. When the period p is less than its length, we can find the answer from the stored value for $i - p$, otherwise it should be $i - p + 1$.

1012. Mow

Difficulty: Easy Author: Rim Gwang Song

This problem can be easily solved using half-plane intersection(HPI) algorithm.

You can mow the entire lawn by hand. You can also mow the lawn by the machine and mow remaining grass near the corners by hand.

To mow with a machine, you have to carefully check whether it can be positioned inside the convex polygon, or not. If so, you are to obtain an intersection of half-planes which is also a convex polygon. Imagine that this intersection has area of a and perimeter of p.

Then the maximum area machine can mow is:

$$v = a + p \cdot r + \pi r^2$$

The "fans" whose centers are located on the corner of HPI form an entire circle together.

Since all above processes can be done in linear time, the solution will be linear.