Problem A. 86

Author: Irfanur Rahman Rafio, Mohidul Haque Mridul
Tester: Fahim Khandakar, Nayeemul Islam Swad
Tags: Probability, Dynamic Programming, Bitmasks, Combinatorics, Sum Over Subset DP
Expected Solve Count: 0
Actual Solve Count: 0

Let $d_i = \frac{x_i}{y_i}$ be the probability of unit i being destroyed in an attack, and $s_i = 1 - d_i$ be the probability of unit i surviving an attack.

Since any random subset of units can be destroyed in a single attack, defining some set notations will make the problem easier to work with.

Let $\mathbb{U} = \{1, 2, \ldots, n\}$ be the universal set of units.
For any set $A \subseteq \mathbb{U}$, let d_A be the probability of all units in A being destroyed in an attack and s_A be the probability of all units in A surviving an attack.
Since the probabilities for each unit is independent, $d_A = \prod_{i \in A} d_i$ and $s_A = \prod_{i \in A} s_i$.
For example, if $A = \{1, 2, 5\}$, then $d_A = d_1 d_2 d_5$ and $s_A = s_1 s_2 s_5$.

Solving a Simpler Version

For now, forget about the different reports. Solve a simpler version of the problem where there is only one report with $m = n$ and $k = 1$ (the war has just begun, all units are currently operational, and the Republic can continue fighting until all n units are destroyed).

There are two very interesting ways to model this problem:

1. Markov Chain
2. Geometric Random Variables

At any point in the war, if you know the current set of operational units, you can determine the probability distribution of the set of operational units after the next attack. This distribution does not depend on what happened in the war till that point. So, you can define a Markov Chain where each state is a set of operational units, and each attack is a transition from the current state to any of its subsets.

Now, the initial state of the Markov Chain is $\mathbb{U}$ and the problem is reduced to finding the expected number of transitions to reach the state defined by the empty set $\varnothing$.

For any set A, the transition probability to any set $B \subseteq A$ is $P_{A \to B} = s_B \times d_{A \setminus B}$. And, $P_{A \to A} = s_A$.

Let E_A be the expected number of transitions to reach from A to $\varnothing$. For any $A \neq \varnothing$, you can get the following Markov Chain equations:

$$E_A = 1 + \sum_{B \subseteq A} (P_{A \to B} \times E_B)$$

$$\implies E_A = 1 + \sum_{B \subset A} (P_{A \to B} \times E_B) + P_{A \to A} \times E_A$$

$$\implies E_A = \frac{1 + \sum_{B \subset A} (P_{A \to B} \times E_B)}{1 - s_A}$$

By definition, $E_\varnothing = 0$.

Now, since your initial state is $\mathbb{U}$ and you want to determine the number of days the Republic can continue fighting, the value you need is $E_{\mathbb{U}}$. This can be computed using subset DP.
You need to calculate E_A for every subset $A \subseteq \mathbb{U}$. To compute a particular E_A, you need the values of all

E_B where $B \subset A$.
This can be done in $\mathcal{O}(3^n)$ time. Unfortunately, that's a bit too slow for the given time limit.

Another interesting observation is that you can define n random variables $D_1, D_2, \ldots, D_n$ where D_i is the number of days until the i-th unit is destroyed. Since each unit and the attacks of each day are independent, for any $i \in \mathbb{U}$, D_i is a geometric random variable with probability d_i.

Let Z be the number of days until all the units are destroyed. Again, all the units are independent and in each day, all operational units are attacked. So, $Z = \max(D_1, D_2, \ldots, D_n)$. Now, the value you need is $E[Z]$, the expected value of Z.

$$
\begin{aligned}
E[Z] &= \sum_{z=1}^{\infty}(zP[Z=z]) \\
&= P[Z=1] + 2P[Z=2] + 3P[Z=3] + \ldots \\
&= P[Z=1] + P[Z=2] + P[Z=3] + \ldots \\
&\quad + P[Z=2] + P[Z=3] + \ldots \\
&\quad\quad + P[Z=3] + \ldots \\
&\quad\quad\quad + \ldots \\
&= \sum_{z=1}^{\infty}(P[Z \geq z]) \\
&= \sum_{z=0}^{\infty}(P[Z > z]) \quad \text{[This is known as the tail sum formula of expectation.]} \\
&= \sum_{z=0}^{\infty} P[\bigcup_{i \in \mathbb{U}} D_i > z] \\
&= \sum_{z=0}^{\infty} \left[\sum_{A \in \mathcal{P}(\mathbb{U}) \setminus \varnothing} \left((-1)^{|A|-1} P\left[\bigcap_{i \in A} D_i > z \right] \right) \right] \quad \text{[Inclusion-Exclusion Principle]} \\
&= \sum_{A \in \mathcal{P}(\mathbb{U}) \setminus \varnothing} \left[\sum_{z=0}^{\infty} \left((-1)^{|A|-1} (s_A)^z \right) \right] \\
&= \sum_{A \in \mathcal{P}(\mathbb{U}) \setminus \varnothing} \left[(-1)^{|A|-1} \frac{1}{1-s_A} \right] \quad \text{[Sum of geometric series]}
\end{aligned}
$$

From here, you can simply loop over all $A \subseteq \mathbb{U}$ and calculate the answer in $\mathcal{O}(2^n)$ time.

Extending the Solution for the Original Problem

Each query (report) replaces $\mathbb{U}$ with a subset $\mathbb{V}$ and introduces a value for k.

For the Markov Chain solution, you now need $E[\mathbb{V}]$ instead of $E[\mathbb{U}]$. If $k > 1$, you just need to modify the base cases. For any A where $|A| < k$, the Republic can't continue fighting from that state. So, $E_A = 0$.

You can precalculate the expectations for all subsets for n different values of k. So, the time complexity for precalculation is $\mathcal{O}(n3^n)$ [again, too slow]. Then you can simply answer each query in $\mathcal{O}(1)$ time.

Now, try to extend the Inclusion-Exclusion solution instead.

Previously, the war ended when all units were destroyed. In other words, if the currently operational set was A, then the war continued as long as at least one unit in A survived.

That is why the earlier event was: $Z > z \iff \bigcup_{i \in \mathbb{U}}(D_i > z)$.

Now, the condition changes. The Republic can continue fighting only if at least k units remain operational.

So, if the currently operational set is $\mathbb{V}$, then after day z, the war continues iff at least k units in $\mathbb{V}$ satisfy $D_i > z$.

At this point, directly applying Inclusion-Exclusion becomes a bit difficult because the event is no longer "at least one event happens". Instead, it becomes: "at least k among several events happen."

Define the events: $E_i = (D_i > z)$.
So, the quantity you need is: $P[\text{at least } k \text{ of the } E_i \text{ occur}]$.

There is a very useful identity for this: $P[\text{at least } k \text{ events occur}] = \sum_{A \subseteq \mathbb{V},\, |A| \geq k} (-1)^{|A|-k} \binom{|A|-1}{k-1} P\left[\bigcap_{i \in A} E_i\right]$.

This is a generalization of the Inclusion-Exclusion Principle. Now,

$$\begin{aligned}
E[Z] &= \sum_{z=0}^{\infty} P[Z > z] \\
&= \sum_{z=0}^{\infty} \left[\sum_{A \subseteq \mathbb{V},\, |A| \geq k} (-1)^{|A|-k} \binom{|A|-1}{k-1} P\left[\bigcap_{i \in A} D_i > z\right] \right] \\
&= \sum_{A \subseteq \mathbb{V},\, |A| \geq k} \left[\sum_{z=0}^{\infty} (-1)^{|A|-k} \binom{|A|-1}{k-1} (s_A)^z \right] \\
&= \sum_{A \subseteq \mathbb{V},\, |A| \geq k} \left[(-1)^{|A|-k} \binom{|A|-1}{k-1} \frac{1}{1-s_A} \right]
\end{aligned}$$

So, for every query $(\mathbb{V}, k)$, the answer is: $\boxed{\sum_{A \subseteq \mathbb{V},\, |A| \geq k} \frac{(-1)^{|A|-k} \binom{|A|-1}{k-1}}{1-s_A}}$

At first glance, this still looks too slow. For one query, iterating over all subsets of $\mathbb{V}$ takes $\mathcal{O}(2^{|\mathbb{V}|})$ time. Since there can be up to 10^5 queries, you cannot do this independently for every query.

So the next idea is that you can preprocess all subset contributions once, so that every query can be answered quickly?

Notice that the term $\frac{1}{1-s_A}$ depends only on the subset A, not on the query.

Define

$$f(A) = \frac{1}{1-s_A}$$

and

$$wt(A,k) = \begin{cases} (-1)^{|A|-k} \binom{|A|-1}{k-1}, & |A| \geq k \\ 0, & |A| < k \end{cases}$$

Then the answer for a query $(\mathbb{V}, k)$ becomes: $\boxed{\sum_{A \subseteq \mathbb{V}} contribution(A,k) = \sum_{A \subseteq \mathbb{V}} f(A) \times wt(A,k)}$.

Time Complexity

For each k, you can precalculate the answer for all $A \subseteq \mathbb{U}$ by building the $contribution(A,k)$ array, and then using Sum Over Subset DP (SOS DP) in $\mathcal{O}(n2^n)$ time. Since there are n possible values of k, you need to run SOS DP n times. So the total complexity for precalculation becomes: $\mathcal{O}(n^2 2^n)$. This is fast enough to pass within the time limit.
After that, every query can be answered in $\mathcal{O}(1)$ time.

Exercise for the Reader

Modify the Markov Chain solution to fit within the time limit. As a hint, observe that the current version of the Markov Chain has 2^n states and 3^n transitions. Try to reduce the number of transitions by slightly increasing the number of states. The idea is somewhat similar to that of SOS DP.

Problem B. All Your Base

Author: Tariq Hasan Rizu
Tester: Fahim Tajwar Saikat, Nayeemul Islam Swad
Tags: Number Theory, Interactive, Modular Arithmetic, Chinese Remainder Theorem
Expected Solve Count: 1
Actual Solve Count: 0

Let the representation of a number X in base n be:

$$X = d_k n^k + d_{k-1} n^{k-1} + \cdots + d_1 n^1 + d_0 n^0 = \sum_{i=0}^{k} d_i n^i$$

where $0 \le d_i < n$ for all i. We can analyze the properties of X under modulo $n-1$ and modulo $n+1$ to derive general divisibility and remainder rules.

Divisibility Rule for $n-1$

In modular arithmetic, $n \equiv 1 \pmod{n-1}$. By the properties of modular exponentiation, raising both sides to any non-negative integer power i yields:

$$n^i \equiv 1^i \equiv 1 \pmod{n-1}$$

Substituting this into the base n expansion of X:

$$X = \sum_{i=0}^{k} d_i n^i \equiv \sum_{i=0}^{k} d_i (1) \pmod{n-1}$$

$$X \equiv d_k + d_{k-1} + \cdots + d_1 + d_0 \pmod{n-1}$$

A number X is congruent to the sum of its digits modulo $n-1$. Consequently, X is divisible by $n-1$ if and only if the sum of its digits is divisible by $n-1$. (This is the generalized version of the "Rule of 9s" in base 10).

If S_e is the sum of digits at even positions and S_o is the sum of digits at odd positions, the total sum of digits is $S_e + S_o$. Thus, our first modular constraint is:

$$X \equiv S_e + S_o \pmod{n-1}$$

Divisibility Rule for $n+1$

Similarly, $n \equiv -1 \pmod{n+1}$. Raising both sides to the power of i yields:

$$n^i \equiv (-1)^i \pmod{n+1}$$

Substituting this back into the base n expansion of X:

$$X = \sum_{i=0}^{k} d_i n^i \equiv \sum_{i=0}^{k} d_i (-1)^i \pmod{n+1}$$

$$X \equiv d_0 - d_1 + d_2 - d_3 + \cdots + (-1)^k d_k \pmod{n+1}$$

A number X is congruent to the alternating sum of its digits modulo $n+1$. Consequently, X is divisible by $n+1$ if and only if the alternating sum of its digits is divisible by $n+1$. (This is the generalized version of the "Rule of 11s" in base 10).

Separating the alternating sum into even positions and odd positions, we get the difference $S_e - S_o$. Thus, our second modular constraint is:

$$X \equiv S_e - S_o \pmod{n+1}$$

Questioning Strategy

For any chosen base n, asking 2 questions ($b = 0$ for S_e and $b = 1$ for S_o) gives us a pair of linear congruences:

1. $X \equiv S_e + S_o \pmod{n-1}$
2. $X \equiv S_e - S_o \pmod{n+1}$

Because $X \leq 10^{15}$, we need to collect enough independent modular constraints so that the LCM of all our moduli strictly exceeds 10^{15}. Once this condition is met, we can uniquely reconstruct X using the Chinese Remainder Theorem (CRT).

To maximize the information gained per question and drastically simplify the CRT implementation, we can strategically choose bases such that both $n-1$ and $n+1$ are prime numbers. When all moduli are distinct primes, they are pairwise coprime, meaning their LCM is simply their product.

A highly effective set of 5 bases is:

$$n \in \{18, 30, 42, 60, 72\}$$

These bases yield the following prime moduli pairs:

- $18 \to (17, 19)$
- $30 \to (29, 31)$
- $42 \to (41, 43)$
- $60 \to (59, 61)$
- $72 \to (71, 73)$

This requires asking 10 questions in total. The product of these 10 distinct primes is roughly 9.5×10^{15}, which securely covers the maximum possible value of X, which is 10^{15}.

However, while picking n such that $n-1$ and $n+1$ are primes makes the math incredibly straightforward, it is not strictly necessary for the moduli to be prime. As long as you select bases resulting in pairwise coprime moduli, you can succeed.

Because the maximum base allowed is 100, you can actually solve this problem by asking just 8 questions (4 bases). For example, 4 carefully chosen moduli pairs around 70–100 will yield an LCM well over 10^{15}.

Time Complexity

$\mathcal{O}(K \log M)$ per test case, where K is the number of moduli equations ($K \leq 10$) and M is the LCM of the moduli (around 10^{15}). Since K and M are bounded by small constants, the time complexity is effectively $\mathcal{O}(1)$ per test case.

Problem C. Chander Gari

Author: Alve Rahman
Tester: Mohidul Haque Mridul, Nayeemul Islam Swad
Tags: Greedy, Dynamic Programming, Data Structure

Expected Solve Count: 30
Actual Solve Count: 17
First Team to Solve: GodFathers Inc.

Define $need[i]$ = minimum liters of fuel needed in the tank before leaving checkpoint i so that the jeep can safely reach checkpoint n. This array can be calculated with the following recurrence relation:

$$\begin{aligned} need[i] &= w[i+1] && \text{when } i = n-1 \\ need[i] &= w[i+1] + \max(0, need[i+1] - r) && \text{when } i < n-1 \end{aligned}$$

This is because we need at least $w[n]$ liters of fuel when leaving checkpoint $n-1$ to reach checkpoint n.

We can refill at checkpoint i, so if we can reach checkpoint n from checkpoint i with minimum fuel $need[i]$, then from checkpoint $i-1$, we need minimum

$$w[i] + \max(0, need[i] - r)$$

liters of fuel to safely reach checkpoint n.

If for any $0 \le i < n$, $need[i] > c$, then it is impossible to reach checkpoint n in any way.

Otherwise, we can construct a solution with a forward simulation like this:

Let's maintain a variable $current\_level$ — the liters of fuel currently in the tank. Initially this is c.

After passing a segment, subtract the corresponding w_i from $current\_level$.

Then decide whether refueling is needed by comparing $current\_level$ with the amount of fuel required from this checkpoint to safely reach checkpoint n using the $need$ array.

If we choose to refill, then the fuel level becomes $\min(c, current\_level + r)$.

Time Complexity

$\mathcal{O}(n)$ per test case.

Problem D. Function Ordering

Author: Bholanath Das Niloy
Tester: Fahim Tajwar Saikat, Nayeemul Islam Swad
Tags: Greedy, Tree
Expected Solve Count: 30
Actual Solve Count: 13
First Team to Solve: BUET_GreyMatter

Every function that appears during the process has the form $Ax + B$, so store it as the pair (A, B). If $g(x) = Ax + B$ and $h(x) = Cx + D$, then

$$g(h(x)) = A(Cx + D) + B = ACx + (AD + B). \tag{1}$$

Thus applying g outside h turns the two pairs into $(AC, AD + B)$. Suppose

$$X(x) = Ax + B, \qquad Y(x) = Cx + D,$$

and X is outside Y. If the part inside them has value z, then

$$\begin{aligned} X(Y(z)) - Y(X(z)) &= (ACz + AD + B) - (ACz + CB + D) \\ &= D(A-1) - B(C-1). \end{aligned} \tag{2}$$

The value z disappears. Thus X should be placed outside Y exactly when

$$B(C-1) \le D(A-1).$$

For two blocks $X = (A_X, B_X)$ and $Y = (A_Y, B_Y)$, write $X \preceq Y$ if X is allowed to appear no later than Y in the outside-to-inside order:

$$X \preceq Y \quad \Longleftrightarrow \quad B_X(A_Y - 1) \le B_Y(A_X - 1). \tag{3}$$

If $A_X > 1$, this is the same as sorting by

$$\rho(X) = \frac{B_X}{A_X - 1},$$

with smaller ρ farther outside. A block with $A_X = 1$ may be treated as $\rho(X) = +\infty$; the cross-multiplied form (3) is easier to think about, since it also covers this case.

Suppose two adjacent blocks are currently in the order $X = (A, B)$ outside $Y = (C, D)$, but $Y \preceq X$. Let the part of the final function outside these two blocks be $P(t) = \alpha t + \beta$, and let the part inside them be $S(t)$. Since all coefficients are positive, $\alpha > 0$. The original final value is

$$P(X(Y(S(0)))),$$

while after swapping X and Y it is

$$P(Y(X(S(0)))).$$

Putting $z = S(0)$, the change in the answer is

$$\begin{aligned} \Delta &= P(Y(X(z))) - P(X(Y(z))) \\ &= \alpha\big(Y(X(z)) - X(Y(z))\big) \\ &= \alpha\big(B(C-1) - D(A-1)\big) \\ &\ge 0, \end{aligned} \tag{4}$$

where the last inequality is exactly $Y \preceq X$. So every adjacent inversion can be swapped without decreasing the answer. Repeating this leaves the blocks sorted by $\preceq$, hence the sorted outside-to-inside order is optimal.

Now consider a subtree $\mathcal{T}_u$. Once all vertices of this subtree except u have been deleted, the whole subtree is a single block at u. Its coefficient is fixed: when $f_v(x) = a_v x + b_v$ is composed outside a current block $H(x) = Ax + B$,

$$f_v(H(x)) = a_v(Ax + B) + b_v = (a_v A)x + (a_v B + b_v),$$

so the coefficient is multiplied by a_v. Every vertex of $\mathcal{T}_u$ appears exactly once, therefore

$$A_u = \prod_{v \in \mathcal{T}_u} a_v. \tag{5}$$

Only the constant term depends on the deletion order. Let $dp[u] = (A_u, B_u)$, where B_u is the maximum constant term obtainable from $\mathcal{T}_u$. This is enough information to keep: if two orders inside $\mathcal{T}_u$ give $A_u x + B_u$ and $A_u x + B'_u$ with $B'_u > B_u$, then

$$(A_u t + B'_u) - (A_u t + B_u) = B'_u - B_u > 0$$

for every input t. Any functions applied later have positive coefficient, so this will only increase the result.

Let the children of u be $c_1, \ldots, c_k$. For a child c, write $G_c(x)$ for the function represented by $dp[c]$. After each child subtree has been reduced to its best block, the only remaining choice at u is the order of these child functions:

$$G_u(x) = G_{c_{\pi_1}}\Big(G_{c_{\pi_2}}\Big(\cdots G_{c_{\pi_k}}(f_u(x))\cdots\Big)\Big), \tag{6}$$

where $c_{\pi_1}, \ldots, c_{\pi_k}$ are written from outside to inside. By the exchange argument above, sort the child blocks by $\preceq$. In the DFS below we sort in the reverse order, so that the children can be applied directly in the loop.

```
void dfs(int u, int p) {
  vector ch;
  for (int v : g[u]) if (v != p) {
    dfs(v, u);
    ch.push_back(v);
  }
  sort(ch.begin(), ch.end(), cmp);
  // cmp(x,y): B[x](A[y] − 1) > B[y](A[x] − 1)
  for (int v : ch) {
    A[u] ← A[u] · A[v];
    B[u] ← B[u] · A[v] + B[v];
  }
}
```

The proof is a short induction on the subtree size. In any optimal order, each child subtree must become one block before it is merged into u. By the induction hypothesis and the fixed-coefficient argument above, replacing this block by $dp[c]$ cannot decrease the result; after that, (4) shows that sorting the child blocks by $\preceq$ is optimal. Thus the transition computes the best block for every u, and the answer is the constant term of $dp[1]$.

Time Complexity

$\sum_{u=1}^{n} \mathcal{O}(\deg(u) \log \deg(u)) \leq \mathcal{O}(n \log n)$ per test case.

Problem E. Helping Knuth

Author: Rudro Debnath
Tester: Alve Rahman, Nayeemul Islam Swad
Tags: Constructive Algorithm
Expected Solve Count: 80
Actual Solve Count: 84
First Team to Solve: IU_NULLVOID

Let us simplify the given expression: $\sum_{i=2}^{n}(a_i - a_{i-1})$

Expanding it: $(a_2 - a_1) + (a_3 - a_2) + \cdots + (a_n - a_{n-1})$

For example, if $n = 5$: $(a_2 - a_1) + (a_3 - a_2) + (a_4 - a_3) + (a_5 - a_4)$

Rearranging the terms: $-a_1 + a_2 - a_2 + a_3 - a_3 + a_4 - a_4 + a_5$

Every middle term cancels with its opposite sign, leaving only: $a_5 - a_1$

Similarly, for any n, the expression becomes: $a_n - a_1$

Therefore, to minimize the result:

- We want a_1 as large as possible.
- We want a_n as small as possible.

Now consider the missing numbers.

- If the first position is missing, place the largest available number there.
- If the last position is missing, place the smallest available number there.
- The remaining missing positions can be filled with the remaining numbers in any order, since middle elements do not affect the final value.

We only need to maintain a valid permutation.

Time Complexity

$\mathcal{O}(n)$ per test case.

Problem F. How Many Valid Collections?

Author: Nazmul Nahian Sagor
Tester: Debojoti Das Soumya, Nayeemul Islam Swad
Tags: Dynamic Programming, Bitmasks, Digit DP
Expected Solve Count: 3
Actual Solve Count: 0

Since the number of queries is very large, we cannot solve each query separately. So for each test case, we **precompute the answer for all possible** k **and** d for the ranges $1..R$ and $1..(L-1)$. Then each query is answered in constant time.

The function `getAnswer(n)` computes `ans`$[k][d]$, where `ans`$[k][d]$ is the total number of ways to choose exactly k positions from all numbers in $1..n$ such that the chosen digits contain exactly d distinct values.

To build this table, the code uses **digit DP**. The DP state is: `dp[pos][started][less][k][mask]`.

Here:

- `pos` = current digit position
- `started` = whether the number has started yet
- `less` = whether the current prefix is already smaller than n
- `k` = number of selected positions so far
- `mask` = bitmask of digits used in the selected positions

At each position, we try every digit from 0 to the allowed limit. If `less = 1`, the limit is 9; otherwise, it is the corresponding digit of n.

If the number has not started yet and we place digit 0, then it remains a leading zero and we simply move to the next position.

Otherwise, once the number has started, there are two transitions:

- **Do not take the current digit:** the value of `k` and `mask` remain unchanged.
- **Take the current digit:** increase `k` by 1 and set the corresponding bit in `mask`.

After processing all positions, the number of distinct selected digits is simply `popcount(mask)`. So every final DP state directly contributes to `ans`$[k][d]$.

In this way, **all values of** k **and** d **are precomputed in one DP run**.

The DP is also memory optimized using only two layers, because each position depends only on the previous one. This keeps the memory usage within the 32 MB limit.

Finally, we run:

- `getAnswer(R)` and store it in `r`
- `getAnswer(L-1)` and store it in `l`

So for every query (k, d), compute, $res = r[k][d] - l[k][d]$

If $res < 0$, add $10^9 + 7$.

Thus, the solution performs the DP only twice per test case, precomputes answers for **all** k **and** d, and then answers every query instantly.

Time Complexity

$\mathcal{O}(2 \cdot |n| \cdot 2 \cdot 2 \cdot |n| \cdot 2^{10} \cdot 10)$ per test case, where:

- the first 2 comes from running the DP for both R and $L - 1$
- $|n|$ = current digit position
- the next two 2's represent the states `started` and `less`
- the second $|n|$ represents the number of selected positions
- 2^{10} = all possible digit masks
- 10 = transitions over all digits

As the sum of $|R|^2$ over all test cases is at most 12000, the total amount of DP transitions remains manageable within the time limit.

Since all queries are precomputed together, each query is answered in $\mathcal{O}(1)$ time.

Problem G. Last Day Harvest

Author: Syeda Raisa Rahman
Tester: Nazmul Nahian Sagor, Nayeemul Islam Swad
Tags: Number Theory
Expected Solve Count: 45
Actual Solve Count: 38
First Team to Solve: RUET_Agdum

The problem simplifies to the equation: $\frac{x}{p} + \frac{y}{q} = 1$

$\implies qx + py = pq$

$\implies pq - qx - py + xy = xy$

$\implies q(p - x) - y(p - x) = xy$

$\implies (p - x)(q - y) = xy$

Now find all the divisor pairs of xy. For each $d_1 d_2 = xy$, $p = d_1 + x$, $q = d_2 + y$

However, a direct $\mathcal{O}(\sqrt{xy})$ divisor generation will exceed the time limit since xy can be as large as 10^{14}. Instead, we can find the divisors of x and y separately and merge them, since if $d_1 \mid x$ and $d_2 \mid y$, then $d_1 d_2 \mid xy$.

Time Complexity

$\mathcal{O}(\sqrt{x} + \sqrt{y} + d_x d_y)$ per test case. Here d_i is the number of divisors of i, which won't exceed 500 for the given limits.

Problem H. Mango Mania

Author: Md Tazim Uddin
Tester: Irfanur Rahman Rafio, Nayeemul Islam Swad
Tags: Data Structure

Expected Solve Count: 15
Actual Solve Count: 9
First Team to Solve: IUT_Hariye_Jawa_Projapoti

For any subarray, let:

- `cnt_ge` = number of elements $\geq X$
- `cnt_lt` = number of elements $< X$
- `cnt_le` = number of elements $\leq X$
- `cnt_gt` = number of elements $> X$

Then the subarray has median exactly X if both of these are satisfied:

- `cnt_ge > cnt_lt` (median is at least X)
- `cnt_le >= cnt_gt` (median is at most X)

So we only need to find a subarray of length at least K satisfying both.

Implementation

Build two prefix sums:

$$A[i] = A[i-1] + \begin{cases} 1 & \text{if } a[i] \geq X \\ -1 & \text{otherwise} \end{cases}$$

$$B[i] = B[i-1] + \begin{cases} 1 & \text{if } a[i] \leq X \\ -1 & \text{otherwise} \end{cases}$$

For a subarray $[l, r]$:

- $A[r] - A[l-1] > 0$ means `cnt_ge > cnt_lt`
- $B[r] - B[l-1] \geq 0$ means `cnt_le >= cnt_gt`

So we need some prefix $j = l - 1$ such that:

- $j \leq r - K$
- $A[j] < A[r]$
- $B[j] \leq B[r]$

Scan r from left to right. Maintain all prefix points $j \leq r - K$ in a segment tree over $A[j]$, storing the minimum $B[j]$ for each prefix sum value. For current r:

- query the minimum $B[j]$ among all inserted prefixes with $A[j] < A[r]$
- if that minimum $\leq B[r]$, answer is YES

The values of $A[i]$ can be negative, but they always lie in the range $[-n, n]$.

So instead of coordinate compression, we use a fixed shift to make all values valid for the segment tree.

The segment tree supports:

- point update: store minimum $B[j]$
- prefix query: minimum $B[j]$ for all $A[j] < A[r]$

Each operation works in $\mathcal{O}(\log n)$ time.

Proof of Correctness

Lemma 1. A subarray has median exactly X if and only if:

- `cnt_ge > cnt_lt`
- `cnt_le >= cnt_gt`

Proof. If more than half elements are $\geq X$, median is at least X. If at least half elements are $\leq X$, median is at most X. Together they force median to be exactly X.

The converse follows directly from the definition of median.

Lemma 2. For subarray $[l, r]$ with $j = l - 1$:

- $A[r] - A[j] > 0 \iff cnt\_ge > cnt\_lt$
- $B[r] - B[j] \geq 0 \iff cnt\_le \geq cnt\_gt$

Proof. Directly follows from the definition of prefix sums.

Lemma 3. If there exists $j \leq r - K$ such that $A[j] < A[r]$ and $B[j] \leq B[r]$, then $[j + 1, r]$ is valid.

Proof. Length condition holds since $r - j \geq K$. Both inequalities imply median conditions via Lemma 2.

Lemma 4. If a valid subarray exists, the algorithm finds it.

Proof. Let valid subarray be $[l, r]$ and $j = l - 1$. Then $j \leq r - K$, and both inequalities hold. Thus j is inserted before processing r, and query will detect it.

Time Complexity

$\mathcal{O}(n \log n)$ per test case.

Problem I. Perils and Patrols

Author: Mohidul Haque Mridul
Tester: Fahim Tajwar Saikat, Nayeemul Islam Swad
Tags: Dynamic Programming
Expected Solve Count: 2
Actual Solve Count: 0

Let $dp[v]$ be the minimum cost to hold exactly v guards immediately before facing the ambush in the current region. Since $D_i, M_i \leq 2000$, we never need to consider a party size greater than $V_{\max} = 2000$.

For each region, transitioning from a previous guard count u to a new guard count v ($|v - u| \leq K$) incurs a cost of:

$$dp[u] + H \cdot \max(0, v - u) + S \cdot v + \text{ambush}(v)$$

After finding the optimal v for the ambush phase, we apply the martial limit: the minimum cost for any $v > M_i$ is simply rolled over to update the cost for exactly M_i guards.

A naive DP takes $O(K)$ per state, yielding an overall complexity of $O(N \cdot V_{\max} \cdot K)$, which is too slow. We can optimize the transition by splitting it into two sliding windows based on whether we are hiring or firing:

- **Hiring ($u \leq v$):** We want to minimize $dp[u] - H \cdot u$ over the window $u \in [\max(0, v - K), v]$.
- **Firing ($u > v$):** We want to minimize $dp[u]$ over the window $u \in [v + 1, v + K]$.

Time Complexity

Since both windows slide forward as v increases, we can maintain the minima using monotonic queues in amortized $\mathcal{O}(1)$ time per state. This reduces the final time complexity to $\mathcal{O}(N \cdot V_{\max})$ per test case, comfortably passing within the time limit.

Problem J. Stable Gears

Author: Nazmul Nahian Sagor
Tester: Rudro Debnath, Nayeemul Islam Swad
Tags: Number Theory
Expected Solve Count: 65
Actual Solve Count: 70
First Team to Solve: BUET_GreyMatter

After k days, the total number of gears is the sum of the first k odd numbers, which equals k^2. Therefore, the gears are numbered from 1 to $n = k^2$.

For any gear numbered x, write it as $x = 2^a \cdot m$ where m is odd. The odd divisors of x are exactly the divisors of m, so their count is $d(m)$.

The total number of divisors of x is $(a + 1)d(m)$ therefore, the number of even divisors is $(a + 1)d(m) - d(m) = ad(m)$

A gear is Mechanically Stable when the number of odd and even divisors are equal. So we need $ad(m) = d(m)$

Since $d(m) > 0$, dividing both sides gives $a = 1$

Thus, x must be divisible by 2 but not by 4.

So among all numbers from 1 to n:

- Multiples of 2 are counted: $\lfloor \frac{N}{2} \rfloor$
- Multiples of 4 must be removed: $\lfloor \frac{N}{4} \rfloor$

Hence, the answer is $\lfloor \frac{n}{2} \rfloor - \lfloor \frac{n}{4} \rfloor$

where $n = k^2$.

This simplifies to $\left\lfloor \frac{k^2}{4} \right\rfloor$

To avoid overflow, compute it safely as $\left(\frac{k}{2}\right)\left(\frac{k+1}{2}\right)$ using integer division.

Time Complexity

$\mathcal{O}(1)$ per test case.