2025 ICPC South America - Brazil First Phase problem commentary

Tiago Gonçalves (tfg)

September 15, 2025

This document was written with the goal of being somewhat of an editorial but with the focus of sharing my experiences and opinions about the problem from the First Phase of 2025 ICPC qualifiers in Brazil as a competitor.

Problem A - A healthy menu

The minimum number of students in a class is at least the maximum number of students that like some fruit in that class, since a student isn't counted twice. That's always possible: the i-th student likes fruits with frequency of students that like it at least i. The solution is the sum of the maximum of a column over all columns.

Problem B - Baralho Alho

This problem is composed of 2 parts:

The first part is related to A and B not being permutations. If they were permutations then the period of matching of a cycle in P is exactly the size of that cycle. Since they aren't permutations, there might be a period that's a divisor of the size of the cycle.

Key knowledge: for a cycle, either there isn't a matching cyclic shift or there's a minimum period P such that every period is a multiple of P. To prove that, prove that if there a and b are periods then $gcd(a,b)$ is also a period. You can do that by noticing that there's some i such that $i \cdot a \equiv gcd(a,b) \mod b$. Then $C[0] = C[i \cdot a] = C[i \cdot a - b] = \cdots = C[gcd(a,b)]$. To find the cycle there are multiple options, my favorite being just testing all divisors of the size of the cycle. Using Z function or the Border function from KMP are also good options.

For this problem, find the indices of a cycle. Let $C[i]$ be the i-th index of the cycle. Then use the Z function on:

$A_{C[0]}A_{C[1]} \dots A_{C[size-1]}\{-1\}B_{C[0]}B_{C[1]} \dots B_{C[size-1]}B_{C[0]}B_{C[1]} \dots B_{C[size-1]}$

With that it's possible to find which cyclic shifts are a match. That also gives enough information to figure out the period.

Now for the second part of the problem. We have congruences:

$$x \equiv c_i \mod P_i$$

Each congruency comes from a cycle. c_i comes from the first cyclic shift that's a match. P_i is the minimum period. The problem asks to find if those congruences can be true all at the same time and if they can, the minimum non-negative integer that's a solution.

To find if the congruences can be true at the same time, you can find a congruence with modulo divisor of P_i for the i-th congruence. The congruences with same modulo must not contradict. That's sufficient and very good to code. Another way would be splitting each congruence into congruences for each prime power that divides it but that's way more implementation.

After finding if the congruences can be true at the same time, just use the Chinese Remainder Theorem (CRT). If the modulo goes above 10^9 then you've found a candidate for the answer. Check if that candidate works for every congruence. If it does, then that's the answer, otherwise it's greater than 10^9.

Problem C - Collatz polynomial

Immediately after an operation of the first type, at least one operation of the second type is done. So the space of polynomials that can appear during the process is 2^{N+2}, since the highest power of 2 is at most $N+1$ during the process and the coefficients are in $\{0, 1\}$. The problem guarantees that it doesn't enter a cycle before reaching 1. The solution is to simulate the process. It's convenient to use bitwise operations to have a concise implementation.

Problem D - Dominoes

A set of dominoes is good if and only if:

- Construct an undirected graph G from the dominoes where for each (u, v) domino in the set there's a corresponding (u, v) edge on the graph.
- There must exist an Eulerian path on G. The reason is that you can reorder a complete game that extends from two sides to a complete game that extends from only one side. This breaks into the following conditions:
- There's at most 1 connected component with more than 0 edges.
- There's at most 2 vertices with odd degree.

Now there are at least 2 solutions that I'm aware of:

The first solution looks to be the intended one. Since there aren't repeated dominoes, there are $1 + 2 + \cdots + 6 = 21$ types of dominoes so there are 2^{21} sets of dominoes. For each set, compute if it's good or not then use SOS DP to precompute the number of subsets of each set that are good.

The second solution is the one we used during the contest and it's kinda complicated but maybe better in some other set of constraints. Basically precompute a state machine over all the ways of having connected components[1]. Then a test case is a DP that works in $O(B(d) \cdot N \cdot 2^d)$, with d being the amount of numbers that are allowed in a domino (6 in the problem) and $B(d)$ being the d-th Bell number. This solution also allows repeated dominoes. I won't go into more details because the other solution is way more practical, for more details check the implementation. The TL in codeforces' gym is too strict, this solution is currently getting TLE there.

[1] `https://en.wikipedia.org/wiki/Bell_number`

Problem E - Expansion of the road network

There are at least 2 approaches to this problem.

The first approach is about finding leaves, removing them from the graph and solving the rest starting from that. When I read the problem during the first hour I considered this approach and discarded it because it looked too tricky for me. The day after the contest I realized that it looks to be working but the details are still kinda tricky for me, so I won't go into details on it.

The second approach is the one I used during the contest. First, solve the cases where the tree is a star graph or a tree with diameter with 4 vertices. The problem has been reduced to finding a tree that has a diameter with at least 5 vertices.

Find the shortest paths from a single source using a BFS. Let $h(u, v)$ be the shortest (u, v) path in the tree and $g(u, v)$ be the shortest (u, v) path in the given graph. It's easy to see that $\left\lceil \frac{h(u,v)}{2} \right\rceil = g(u, v)$.

Key observation: if $h(u, v)$ is even, then the shortest path from u to v in the given graph is unique (as in there's only one shortest path). If $h(u, v)$ is 1 then it's also unique, but if it's odd and at least 3, then there will be more than one shortest path. With this observation, it's possible to know the exact distances $h(u, v)$ from u to every vertex v that isn't adjacent to u in the given graph.

To find out if the path is unique you can simply do a DP on the DAG of smallest distances.

There should be many solution from this point onward, but the one that I used is:

- do a BFS starting from a vertex that isn't adjacent to all other vertices in the given graph.
- find the vertex that's most distant to it in the tree, call it u. u should be identifiable (as in the distance in the tree is at least 3) since the diameter is of size at least 5.
- do a BFS from u and find the vertex most distant to it in the tree, call it v.
- the (u, v) path is a diameter and the information from BFS's starting from u and v is enough to find out a 2-coloring of the tree, which can be then used to construct the tree because an edge in the tree connects vertices of different colors.
- after finding the tree, check if the tree is indeed a tree (this algorithm solves for some bipartite graphs that aren't trees as well) and check if the given graph corresponds to the squared graph generated from the tree. To implement that, you can simply do the construction naively and

terminate early in case more edges than were created than there are edges in the given graph.

If at any point of the algorithm above something unexpected happens, then there doesn't exist a tree that generates the given graph.

Problem F - Frangolino ali na mesa

Let $f(i)$ be the expected value of dishes served to X_i with successive orders after the robot goes to X_i. The probability of the i-th operation being an operation for the robot to go to X_i is 2^{-1}. The probability of the j-th operation being an order operation to X_i is $2^{-(j-i+1)}$ because i must be a go to X_i operation and all operations in $(i, j]$ must be order operations. Thus:

$$f(i) = \sum_{j=i+1}^{Q} 2^{-(j-i+1)} \cdot X_j$$

Let $g(i)$ be the expected value of dishes served to the table i.

$$g(i) = \sum_{j|X_j=i} f(i)$$

Except for $g(1)$, which the expected value for prefixes of all order operations must be added to. To compute it fast, define $h(i)$ as:

$$h(Q+1) = 0, h(i) = \sum_{j=i}^{Q} 2^{-(j-i+1)} \cdot X_i = \frac{h(i+1) + X_i}{2}$$

Then:

$$f(i) = \frac{h(i+1)}{2}$$

The case for $g(1)$ you can simply add $h(1)$ to it.

Problem G - Generating patterns

Consider the problem where only operations that all 8 bits from the pattern are inside the text or are “under” the least significant bit. A greedy solution works: find the highest set bit and align it with the highest bit from the pattern in order to decrease the highest set bit.

In linear algebra/system problems like this, it’s common to consider fixing how some operations are done. Fix the set of operations that are done that “spill over” higher than the most significant position. Use the greedy solution to solve the remaining bits. This solution uses $O(2^8 \cdot N)$ bitwise operations, which apparently is sufficient. A way to divide the constant by 2 is noticing that if the highest bit from the pattern isn’t 1 you can multiply the pattern by 2, reducing the space of patterns to 2^7. If you do this, don’t forget to divide the pattern by 2 while you can to find the smallest good pattern.

A tricky case is when the input is already all zeroes.

Problem H - How many teams?

Consider finding the answer for every possible query. The problem is very similar to a classic SOS DP problem: just change "teams of 3" to "teams of any size". The solution to that problem is:

- $A[i] =$ frequency of the mask i among participants
- $A \leftarrow SOSDP(A)$
- $A[i] \leftarrow 2^{A[i]}$
- $A \leftarrow SOSDP^{-1}(A)$

A solution to the current problem is similar:

- $A[i] =$ frequency of the mask i among participants
- $A \leftarrow SOSDP(A)$
- $A[i] \leftarrow \binom{A[i]}{3}$
- $A \leftarrow SOSDP^{-1}(A)$

For more details, google for tutorials on SOS DP. I recommend solving the practice problems from `https://codeforces.com/blog/entry/45223`.

Problem I - Investigating Quadradômeda

Let $D_i = distance(P_i, P_{i+1})$, with P_i being the i-th point.

$$R_i + R_{i+1} = D_i$$

$$R_{i+1} = D_i - R_i$$

It's possible to represent each R_i as $R_i = (-1)^{i-1} \cdot R_1 + C_i$ for constants C_i, in other words, R_i is a linear function of R_1. The problem asks for $(-1)^{i-1} \cdot R_1 + C_i \geq 0$. Find the maximum R_1 that satisfies all the inequalities.

Problem J - João João

The answer is the number of numbers in $\{1, 2, 3, 4\}$ that have frequency 0 in the input.

Problem K - Knockout, swiss and other kinds of tournaments

Let N be the number of participants. The number of participants with i victories and j losses is:

$$P(i,j) = \frac{N\binom{i+j}{i}}{2^{i+j}}$$

The problem asks for the smallest N such that $P(i,j)$ is even for $i \in [0,A), j \in [0,B)$. So the factors of 2 in the prime factorization of N must be at least $f(i,j)+1$, with $f(i,j)$ being the number of factors of 2 in $\frac{\binom{i+j}{i}}{2^{i+j}}$. Let $g(i,j)$ be the number of factors of 2 in $\binom{i+j}{i}$. Using that definition:

$$f(i,j) = g(i,j) - (i+j)$$

The answer to the problem is 2^{-e+1}, with the following definition of e:

$$e = \min_{\substack{0 \le i < A \\ 0 \le j < B}} \{f(i,j)\}$$

It's possible to compute $g(i,j)$ using Legendre's formula[2]. Let $h(N)$ be the number of factors of 2 in the prime factorization of N.

$$h(N) = \sum_{x=1}^{\lfloor \log_2(N) \rfloor} \left\lfloor \frac{N}{2^x} \right\rfloor$$

$$g(i,j) = h(i+j) - h(i) - h(j)$$

A candidate solution is just checking all pairs but it's obviously too slow. Now, it's good to consider trying some pairs but not all pairs. A good intuition but not quite correct is that we simply want $i+j$ to be as high as possible, since we want to minimize $f(i,j)$ and it has $-(i+j)$ added to it. Call "checking a pair (i,j)" computing $f(i,j)$ and doing $e \leftarrow \min(e, f(i,j))$. This intuition leads to only checking the pair $(A-1, B-1)$. It's wrong because $g(i,j)$ might be big enough to make it necessary to check other pairs.

It's sufficient to check only pairs $(A-x, B-y)$ for $x, y \in [1, 100]$. During the competition, you can just guess this and get AC or fake-solve with a wrong proof and get AC (this is what I did).

To prove it, we must dig a bit deeper on the formula for $g(i,j)$.

$$g(i,j) = \sum_{x=1}^{\lfloor \log_2(i+j) \rfloor} \left\lfloor \frac{i+j}{2^x} \right\rfloor - \left\lfloor \frac{i}{2^x} \right\rfloor - \left\lfloor \frac{j}{2^x} \right\rfloor$$

[2]`https://en.wikipedia.org/wiki/Legendre's_formula`

It's well known that for integers a, b it's true that $\frac{a}{b} - 1 < \left\lfloor \frac{a}{b} \right\rfloor \leq \frac{a}{b}$. With that, we can prove that for a triple of positive integers a, b, c it's true that $0 \leq \left\lfloor \frac{a+b}{c} \right\rfloor - \left\lfloor \frac{a}{c} \right\rfloor - \left\lfloor \frac{b}{c} \right\rfloor \leq 2$. That implies that each portion of the sum for $g(i, j)$ is at most 2, so $g(i, j) \leq 2\log_2(i+j)$, thus $f(i, j) \in [-i-j, -i-j+2\log_2(A+B)]$. So in the worst case (as in maximum value) $g(A-1, B-1)$ is $-(A-1+B-1) + 2\log_2(A+B)$ and every pair $(A-x, B-y)$ with $x+y-2 > 2log_2(A+B)$ has $g(A-x, B-y) \geq -(A-1+B-1) + 2\log_2(A+B)$. As we're interested only in pairs that can be the minimum, we can not check pairs $(A-x, B-y)$ with $x+y-2 > 2log_2(x+y)$.

That would give a bound of around 120 for the constraints of the problem. To get a bound closer to $log_2(A+B)$ you can note that for a triple of positive integers a, b, c it's true that $0 \leq \left\lfloor \frac{a+b}{c} \right\rfloor - \left\lfloor \frac{a}{c} \right\rfloor - \left\lfloor \frac{b}{c} \right\rfloor \leq 1$ because the sum of the remainders of the fractions that are subtracted is always less than $2c$.

Problem L - LLMs

The whole problem is just implementing what's described on the statement. The constraints are very low, so many approaches work. A decent option is preprocessing every substring of at most 6 tokens and keeping the information in a map. Refer to the implementation. One optimization is using the fact that $d \cdot a + d \cdot b = d \cdot (a + b)$ to optimize the sum of dot products.

Problem M - Minas Gerais' walls

Statements with keywords like "maximum of minimum" or similar (in this problem it's "largest possible minimum height") should always trigger the question: is it binary searchable?

For this problem, it makes no sense to consider an operation that doesn't increase all values that it can increase for a range [L, R], because otherwise we could take a range [L-1, R] and the resulting array is never worse. Conclusion: the only ranges that make sense to take are [1, R].

Back to the binary search: the question that must be solved is whether $answer \geq mid$. Find the largest index i such that $x_i < mid$. The best operation possible is [1, i]. The range starts at 1 because of the previous observation. The range ends at i because otherwise all numbers from index i and below will be increased not as much as they can be and the indices above i don't need to be increased. After simulating the operation just check if some position is still less than mid after the operation.

Bonus: at first I misread the problem as "You can use at most M operations, find the largest minimum possible". The same solution explained above can be modified slightly to solve this problem.