Problem A. Alternating Function
Category: AA1
Author: Rodrigo Chavez
Solved by 24 teams

Given the function $h(n) = (-1)^n \cdot n \cdot p + k$, we are asked to find the smallest positive integer x such that $h(x) \geq m$. There a two cases for n:

- When n is odd, the function is a decreasing linear function, since the factor $(-1)^n$ is equal to -1. The case $x = 1$ should always be tested, since it is the only odd x that can be a solution.
- When n is even, the function is an increasing linear function, since the factor $(-1)^n$ is equal to 1. For $n = 2s$, with $s \geq 1$, we have that $h(2s) = 2s \cdot p + k$. From $h(2s) \geq m$ we can deduce that $s \geq \left\lceil \frac{m-k}{2 \cdot p} \right\rceil$. If $x = 1$ is not an answer, then the solution is

$$max(2, 2 \cdot \left\lceil \frac{m-k}{2 \cdot p} \right\rceil).$$

  We take the maximum with 2 to avoid 0 as a solution.

Problem B. Board Game
Category: GT3
Author: Alberto González Rosales
Solved by 8 teams

The main observation that we have to make in order to solve this problem is that every game that Alice and Bob play can be interpreted as a classic Nim game, where a token placed in column y represents a stone pile of size $y - 1$.

To determine the winner of a Nim game, we have to calculate the nim-sum of the game, which is $a_1 \oplus a_2 \oplus a_3 \cdots \oplus a_n$, where $\oplus$ stands for the XOR logical operator and $a_1, a_2, \cdots, a_n$ are the sizes of the stone piles of the game. Alice wins if nim-sum is different from 0, otherwise Bob wins.

To solve the problem efficiently, we have to process games in non decreasing order of its value r. We maintain a variable $nimsum$, which holds the nim-sum value of games up to the current column r we are analizing. Every time we find a token whose column y lies in the range $[1, r]$ we make $nimsum = nimsum \oplus (y - 1)$. After that we just have to determine if $nimsum$ equals 0 or not in order to find out the winner.

We have to carefully print the answers of every game in the order given in the input.

The time complexity of this approach is $O(c \log c)$.

Problem C. Counting Products
Category: DP3
Author: Daniel Enrique Cordovés Borroto
Solved by 16 teams

First of, we can change the condition of obtaining a number x to $\sum x_i \leq k$ and $x_i \geq 2$, this is because we get the same product x and we can complete the remaining x_i's with ones.

Now for an x, we need to obtain the smallest sum of x_i such that its product is x. The key observation here is that every x_i must be a prime number, because if one of them were composite then there exists integers $a, b \geq 2$ such that $x_i = a \cdot b$, but $a + b \leq a \cdot b$ holds for $a, b \geq 2$, which means that we can obtain a smaller sum if we put a and b instead of x_i. We can apply this reasoning until all x_i are primes.

To proof the previous equation, suppose WLOG that $a \leq b$, then $a + b \leq 2 \cdot b$ and as $a, b \geq 2$ it follows that $2 \cdot b \leq a \cdot b$, and hence $a + b \leq 2 \cdot b \leq a \cdot b$.

Finally the solution is, iterate every x from 1 to n, get their prime factors (with multiplicity), add them and check that their sum is less than or equal to k, if it is, add 1 to the answer.

The time complexity is $O(n \cdot F)$, where F is the complexity of factoring a number, which in the simplest case is $O(\sqrt{n})$.

Problem D. Decorating Trees
Category: GT4, DS4
Author: Rubén Alcolea Núñez
Solved by 4 teams

In this problem you have a tree with N vertices numbered from 1 to N and rooted at vertex 1. Initially, each vertex has a color c_i. The problem ask you to perform Q queries of the following types:

1. Update the colors of all vertices in the subtree of vertex v. For each vertex of the subtree replace its color by the formula: $c_i = (c_i + 1)\ mod\ 64$. Where $x\ mod\ y$ stands for the remainder that results of dividing x by y.
2. Count the number of vertices in the subtree of vertex v with color c.

Due to the input sizes for N and Q, it is not possible the simulation of queries with a naive method. It would take $\mathcal{O}(N \cdot Q)$ which is not enough to pass the time limit for this problem.

In this kind of problems where we need to apply an operation to a subtree, we can transform the rooted tree into an array by running a depth-first search and storing the discovery and finish times of each vertex. It works because during a depth-first search traversal of a subtree x, the descendant vertices of subtree rooted at x will be visited as a contiguous sequence and stored at the range $[discovery[x]..finish[x]]$ of the array.

For example, suppose you have a tree with the following edges: 1-2, 1-3, 1-4, 3-7, 3-5 and 3-6. After running a depth-first search starting at vertex 1, we obtain the following sequence of vertices, colors, discovery and finish times:
vertices: {1, 2, 3, 7, 5, 6, 4}
colors: {1, 2, 3, 1, 1, 1, 1}
discovery: {1, 2, 3, 7, 5, 6, 4}
finish: {7, 2, 6, 7, 5, 6, 4}

Once we have this information, if we need to check colors for a subtree rooted at x, we check the range $[discovery[x]..finish[x]]$ of array *colors*. For example, the info of subtree rooted at 3 is in the range $[discovery[3]..finish[3]]$ of the arrays vertices $(3, 7, 5, 6)$ and colors $(3, 1, 1, 1)$ and these are the correct values for this subtree.

At this point, the original problem was reduced to the following problem: Given an array A of integer numbers, you can perform 2 operations:

1. Replace all the elements of the range $[x, y]$ by the formula: $new\_value = (current\_value + 1)\ mod\ 64$. Where $x\ mod\ y$ stands for the remainder that results of dividing x by y.
2. Count how many numbers of the range $[x, y]$ have value equals c.

The new problem can be solved with a segment tree and lazy propagation to answer both kinds of queries in $\mathcal{O}(\log N)$. Each node of the segment tree stores the frequency of each color in the range ($int\ colors[MAX\_COLORS]$), the starting color of the range if we handle each update operation as a left rotation of the array *colors* ($int\ start$) and the number of pending updates ($int\ lazy$) to apply lazy propagation.

The solution has three steps. The first one is the preprocessing stage with the depth-first search algorithm ($\mathcal{O}(N)$) in order to compute discovery and finish times. The second step is the construction of the segment tree ($\mathcal{O}(N)$). The final step is the processing of queries ($\mathcal{O}(Q \log N)$).

$$\begin{aligned} T(n) &= \mathcal{O}(N) + \mathcal{O}(N) + \mathcal{O}(Q \log N) \\ &= \mathcal{O}(Q \log N) \end{aligned}$$

Problem E. Encoding Matrices
Category: BF1
Author: Rubén Alcolea Núñez
Solved by 115 teams

Easy problem. Ad Hoc 1

Compute the frequency of each bit and encode the matrix according to the rules given in the problem.

Check carefully the case in which there is a tie at the frequency of bits and encode with the symbol '*' the bit located at the top left corner of the matrix.

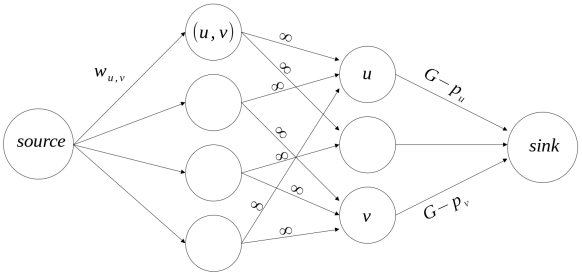
Time Complexity: $O(n^2)$

Problem H. Heavy Graph
Category: GT5, Flow
Author: Alfonso Peterssen
Solved by 0 teams

We have to find the **densest** subgraph, but taking node weights into account.

Let's assume we have guess G (real number) of the maximum score. If we can determine if a subgraph $S \subseteq V$ exists with a score $> G$, we can binary search the answer.

We can model this comparison as a max-flow problem using the following graph: every edge and vertex of the original graph are represented with nodes as follows:



We can re-arrange the score function compared to our guess G as follows, akin to the flow network:

$$\frac{\sum_{u \in S} p_u + \sum_{\substack{(u,v,w) \in E \\ u,v \in S}} w_{u,v}}{|S|} \quad ? \quad G$$

$$\sum_{\substack{(u,v,w) \in E \\ u,v \in S}} w_{u,v} \quad ? \quad G \cdot |S| - \sum_{u \in S} p_u$$

$$\sum_{\substack{(u,v,w) \in E \\ u,v \in S}} w_{u,v} \quad ? \quad \sum_{u \in S} G - p_u$$

The left side of the equation represent the edges within S, leaving the *source*. The right side represent the vertices of S e.g. edges reaching the *sink*.

By running a max-flow algorithm in the described network for a guess G:

- If the left side is $\leq$ the right side of the equation, then all edges from the *source* will be saturated, so the source-side of the cut will be just $\{source\}$.

- If the left side is $>$ the right side the equation, some edges won't be saturated; moreorver, the source-side of the cut will contain the nodes that represent S in the riginal graph s.t. $score(S) > G$.

We can do a binary search for the maximum score, using any max-flow algorithm, keeping the source-side of the cut as the final answer.

Implemented with Dinic's max-flow algorithm, the worst case complexity is $\mathcal{O}((V+E)^3 \cdot \log X)$, but runs much faster in practice due to the shape of the graph.
There's also an alternative (smaller) construction for the flow graph that can achieve $\mathcal{O}(V^2 \cdot E \cdot \log X)$ when implementedd with Dinic's algorithm, but this is not required to get AC.

Problem K. Koa the Koala
Category: AH3, GT3
Author: Dennis Gómez Cruz
Solved by 24 teams

The first step to solve this problem is to determine when a solution exists. For this, it's helpful to analyse the problem as a graph, where cats create directed edges to dogs and viceversa.

In particular the following happens:

- each cat will add a edges, and each dog will add b edges, so in any solution there will be $n \cdot a + n \cdot b$ edges.
- the maximum amount of edges that can be created in any solution, such that there is not a cat and a dog looking each other simultaneously, is $n \cdot n$.

So, $n \cdot a + n \cdot b \leq n \cdot n$ holds. Simplifying we get that $a + b \leq n$. This is a necessary condition for the existence of a solution.

With this in mind, let's create a construction to place the edges on the graph. The idea is: for every cat create a edges to the following a dogs cyclically (starting from 0 if we are on the last dog), more formally: the i-th cat will add edges to dogs $i \cdot a \mod n, (i \cdot a + 1) \mod n, \ldots, (i \cdot a + a - 1) \mod n$ (all indexes start from 0).

To complete the construction, each dog j will add exactly b edges to cats i such that the edge (i, j) was not added on the step explained below. It turns out that each dog will be able to find the needed b cats.

To see this, note that the maximum in-degree of a dog needs to be at most a, if not, there were more than $n \cdot a$ edges added from cats. So, it follows that the amount of available cats for each dog is at least $n - a$, and by the inequality shown above $n - a \geq b$ (each dog will have enough cats to add edges to them).

The time complexity is $O(n^2)$ per test.

Problem L. LCS Recovery
Category: STR3
Author: Daniel Enrique Cordovés Borroto
Solved by 5 teams

To solve this problem we need to obtain some information from the LCS matrix. The idea is to detect the pairs (i, j) such that $A[i] = B[j]$ or $A[i] \neq B[j]$.

But first, let's note the following property of any LCS matrix:

- Let (a, b) and (c, d) be two adjacent positions (by row, column or diagonal) on a LCS matrix M, then $|M[a][b] - M[c][d]| \leq 1$.

The proof of this is simple, if (a, b) and (c, d) are on the same row or column, then in the DP transition only one character was added, so the LCS could not increase; and similarly in the case where they are adjacent diagonally, in the DP transition one character was added to each string, so the LCS could increase in at most one.

So, with this and noting that the LCS increases (or stays the same) on rows (from left to right) and columns (from top to bottom); there are only five cases we need to analyse on a LCS matrix:

1.
$x-1$	x
x	x

2.
$x-1$	$x-1$
x	x

3.
$x-1$	x
$x-1$	x

4.
x	x
x	x

5.
$x-1$	$x-1$
$x-1$	x

Where $x = M[i][j]$.

In the first, second and third case we don't get any information. In the fourth case, we know that $A[i] \neq B[j]$ because if not $M[i][j]$ would be $x + 1$; and in the fifth case we know that $A[i] = B[j]$.

Now with this information, we create a graph G connecting, with undirected edges, the node i with the node $j + n$ if $A[i] = B[j]$ and compress that graph. Then we add undirected edges in a new graph G' (this graph has number of connected components of G nodes) from the component where node i is to the component where node $j + n$ is if $A[i] \neq B[j]$ (all indexes start from 1).

In this stage, the problem reduces to find a bipartition on this graph G' (note that this graph is bipartite, because we always connected indexes of A to indexes of B), but we need to do this processing each node from 1 to $n + m$ coloring with 0 first, to obtain the lexicographically smallest solution.

The time complexity is $O(n \cdot m)$.

Problem M. Matrix Parity
Category: AH2
Author: Carlos Joa Fong
Solved by 25 teams

First, we observe that changing a cell more than once is not optimal:

- if we change it an even number of times, the parity of a cell value remains the same as its original value
- if we change it an odd number of times, the parity of the cell flips from odd to even or even to odd

Second, if the number of rows with odd parity equals the number of columns with odd parity, we can save operations by *pairing up* those rows and columns with odd parity. For each pair, change the cells at the intersection of the row and column numbers in the pair. For instance, if the sets of rows with odd parity are $\{1, 4\}$ and the set of colums are $\{2, 4\}$, we can change cells at $(1, 2)$ and $(4, 4)$.

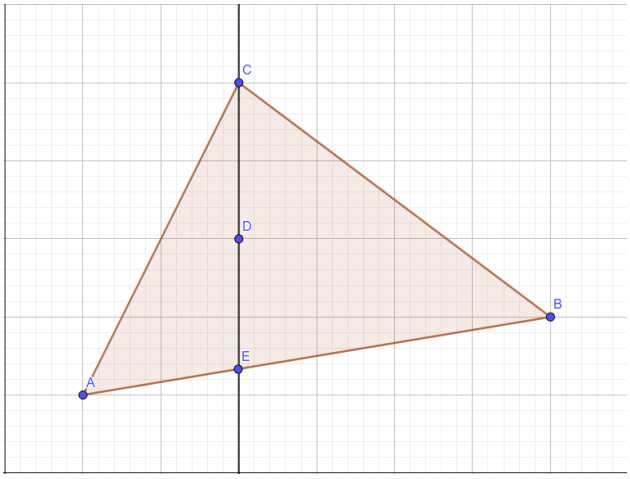
If the numbers of rows and columns with odd parity are not the same, we greedily match as many pairs as possible. For the remaining unmatched rows (or columns), we match them with an arbitrary (but fixed) column.

Problem N. New Combination
Category: GEO4, AA4
Author: Ernesto David Peña Herrera
Solved by 0 teams

In the problem we are tasked to find a **convex combination** of the input's points which gives as a result a given query point, for several queries. As the name suggest, it is related to *convex sets theory*, furthermore, it is possible possible to do so **if and only if** the point is contained inside the convex hull of the given set of points (proof to this fact is the base of convex sets theory and can be found in any related bibliography).

We already know when it's possible or not to find such combinations, just remains to find the actual combination in a positive case. Note that we just need to find a triangle from the convex hull containing the point, since a triangle is a convex polygon and if the point is inside the convex hull then it is inside at least one of its triangles.

Let's suppose we found such triangle:



where D is the query point. We can extend line $\overline{CD}$ and mark point E.

From the vector equation of the line it follows that $\exists t \in [0,1] : D = (1-t) \cdot C + t \cdot E$. The same way $\exists k \in [0,1] : E = k \cdot A + (1-k) \cdot B$. Combining both results we get

$$D = (1-t) \cdot C + t \cdot k \cdot A + t \cdot (1-k) \cdot B$$

which is a convex combination of points A, B and C $\big($you may check $(1-t)+t \cdot k+t \cdot (1-k) = 1\big)$.

In order to fit time constrains we need to find the triangle in a clever way, an idea is with binary search over the diagonals with one of the points fixed. With this approach we get a time complexity of $\mathcal{O}((n+q) \cdot \log n)$, which is enough to get AC. We never get to use more than 3 elements in our convex combination.

Problem O. Or Exclusive Sum
Category: BF3
Author: Ernesto David Peña Herrera
Solved by 6 teams

Let's say we have values a, b, c consecutive in the given array and we want to change b to maximize the value of $(a \oplus b) + (b \oplus c)$ (note that b wont be involved in any other term of the summation for the whole array). Since $\oplus$ is a bitwise operator we can proceed independently for each bit. Let's say we have for some bit i the values 0 and 0 in a and c respectively, then it's more convenient to set the $i-th$ bit of b to 1. The same way we can check all four cases and construct the best value of b for a given pair a, c.

From the above statement we can get the best solution with only one changed element by just trying all possible triplets of consecutive elements.

Let's suppose now we want to change two elements, there are two cases:

1. The two elements are consecutive:

   Consider the values a, b, c, d consecutive in our array, this operation is similar to the previous described, with the difference that now we want to get b and c, given a pair a, d in order to maximize $(a \oplus b) + (b \oplus c) + (c \oplus d)$. It can be solved again by checking all four cases on a's and d's bits, and then trying all possible $4-$tuples of consecutive elements.

2. The two elements are not consecutive: (we can change its values independently of each other)

   Let's say we calculate $pref(i)$ as the best result we can get by changing only one element in the positions from 1 to i (eg. the $i-th$ prefix). We can easily do this by using some simple *Dynamic Programming* technique.

   Then suppose we want to change element j as the second element; we just need to check the triplet $A[j-1], A[j], A[j+1]$ (where A denotes the array) as previously described and consider the value of $pref(j-2)$ (since they must not be consecutive) as our base value for the summation.

Overall complexity is $\mathcal{O}(n \cdot \log M)$ where M is the maximum value among the array's elements. The log factor is included because we check all bits for each element.

Note: Special care need to be taken when changing elements in the first/last positions of the array.