A: The One Million Dollar Man

Problem Author: Morteza Muthahhari

- In order for Austin to retire as soon as possible, he must accumulate 1 million dollars as quickly as possible, which can be achieved by taking the largest payment available each day until the accumulated payment is $\geq 1,000,000$.
- Payments can be obtained in two ways, denoted as A^B or B^A. Therefore, we need to determine the values of A and B from a 2-digit code:
  - `B = code[i] % 10`
  - `A = code[i] / 10`
- Next, we perform a greedy operation to select the larger value between A^B and B^A, and store the result in a variable called `sum`: `sum += max(pow(A, B), pow(B, A))`
- To count the days, we maintain a 'count' variable.
- Don't forget, if all codes have been used, the code sequence will reset to the first day on the following day. If the 1 million dollar goal has not been reached yet, the calculation will continue for the next M days until it reaches $\geq 1,000,000$. An infinite loop with a 'break' statement can be employed for this purpose.

A: The One Million Dollar Man

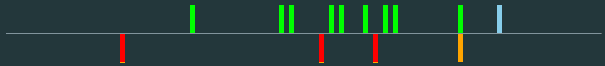
Problem Author: Morteza Muthahhari

- When the 'sum' (total payment) reaches or exceeds 1 million dollars, exit the loop: `if(sum >= 1,000,000) break`
- To display the desired result, show the day as 'count + 1' because Austin will stop working on the following day.
  Time complexity: $O(N + M)$.

Statistics: 55 submissions, 10 accepted, 8 unknown

I: Hoopla Country Heritage

Problem Author: Jonathan Darius

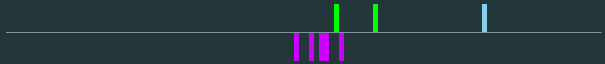


- Given that $N \leq 10000$, we can simply iterate for each digit and store the frequency of each digit in an array.
- We can use either integer division when iterating each digit in an integer, or temporarily treat the integer as a string and count each occurences of the characters.
  Time complexity: $O(N)$.

Statistics: 14 submissions, 9 accepted, 1 unknown

E: WARNING! Deadlock!

Problem Author: Kenny Arlexy



- Given a graph, check if it contains any cycle.
- One idea to do this is to choose every node that is not visited and do DFS. If we encounter a node that has been previously visited before while doing DFS for that node, We have encountered a cycle.
  Time complexity: $O(N + M)$.

Statistics: 8 submissions, 2 accepted, 1 unknown

F: I Want More!

Problem Author: Kenny Arlexy

- Rephrasing the problem more formally: find the longest increasing subsequence of the array containing N integers.
- Let $dp[i]$ be the answer to the problem. Observe that $dp[i]$ is at least 1 (the element itself) for all i.
- Thus, let us set $dp[i]$ to 1 for all i and denote the array A. The recurrence will be

$$dp[i] = \max_{\substack{j<i \\ A[j]<A[i]}} (dp[j] + 1)$$

- The answer is $100 \cdot \max(dp)$.
  Time complexity: $O(N^2)$.

Statistics: 44 submissions, 2 accepted, 8 unknown

D: Perfecting Square

Problem Author: Ardian Arvon

- Problem: Look for the 4th point which is the point that forms the square.
- Given points A, B, and C, check the 3 angle patterns that can be formed:
  - ABC
  - ACB
  - BCA

  Ensure that one of the angle patterns meets the rules of having equal side lengths and an angle of 90 degrees.
- Find the value of the linear equation line ($y = mx + c$) formed by two non-angle points.
- Reflect the angle point against the previous linear equation line.
  Time complexity: $O(1)$.

Statistics: 30 submissions, 1 accepted, 11 unknown

B: Nature

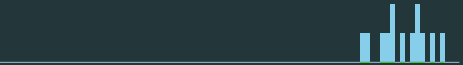
Problem Author: Steven Hans

- Since the span is large, use a map of integers to store the count of vehicles. Let this be $M_x(x)$ and $M_y(y)$. This is enough since the locations won't exceed 5000.
- The tricky part comes when there are multiple vehicles in one place. One way to solve this is to observe that no matter the solution, one of the axis **must** have maximal number of vehicles.
- Let us fix one of the axis such that the total vehicles are maximum. Then, for the other axis, iterate to see whether the coordinate combination contains the most vehicles in total.
- To prevent double counting, subtract with the number of vehicles in (x, y). This can be stored in another map, say M_{xy}. Therefore, the amount of vehicles is $M_x(x) + M_y(y) - M_{xy}(x, y)$
- Thus, the answer is $\max((X_{\max}, Y), (X, Y_{\max}))$.
  Time complexity: $O(n)$

Statistics: 0 submissions, 0 accepted

C: X PrimeNumber After X

Problem Author: Morteza Muthahhari



- If the max perimeter is below 12, output 0 (the smallest one is $(3, 4, 5)$).
- To handle other cases, you can do this online or offline:
- Approach 1: All pythagorean primitives can be generated with Dickson's method. With some optimizations, it is fast enough to pass. Some ideas:
  - No need to iterate through odd r, because there will be no integers s, t such that $r^2 = 2st$.
  - Iterate for r up to 10^4. It's possible to find this bound by outputting the triples and verifying by eye.
  - While finding s, t for $r^2 = 2st$, it's only necessary to iterate up to $\sqrt{\frac{r^2}{2}} = \frac{r\sqrt{2}}{2}$.

  Time complexity: $O(M^2)$.
- Approach 2: Since the max perimeter is just $2 \cdot 10^4$, generate all triples and check if they are coprime. Then, count the frequency of each perimeter. Save this result to an array and use this to build the actual solution that you're going to submit.
  Time complexity: $O(1)$.

  Statistics: 13 submissions, 0 accepted, 13 unknown

G: Aurum Alley Delivery

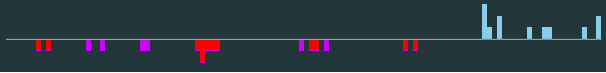
Problem Author: Stanley Baldwin

- Prepare 2 regular arrays for each layer with size x, then input each index with y. Because the items are not rotated, the placement of the items will always point to the right (y-axis) so that the slots available on each index put each type of item into the container provided.
- To get maximum value, because Sushang can place other items if the required items are met, so Sushang provides items with the most expensive ratio (boxes and pots) into containers until there are no slots left in the container.
  Time complexity: $O(y)$.

Statistics: 2 submissions, 0 accepted, 1 unknown

H: Popoi Got Sick Again!!

Problem Author: Meily



- Given popoi who only can eat crystal from size $p_i - m$ to $p_i + m$
- Given all available crystal size t_i
- Each popoi can eat only one it within the size it can eat
- The way to solve:
  - Just sort both of the popoi's desired crystal size and owned crystal size (ascending or descending (sort both the same way))
  - We will explain it in ascending order:
    - While there is still popoi and crystal, we check if the crystal can be eaten by popoi. If yes, then we count the crystall eaten (skip the crystal as the crystall has eat, skip the popoi as each popoi can only eat one crystall, and that popoi eat the crystal)
    - If not, is it too large for popoi? if yes then skip the popoi, that popoi will not eat any crystal (skip that popoi), if not then the crystall is too small for popois (skip that crystall).

Time complexity: $O(N \log N)$.

Statistics: 30 submissions, 0 accepted, 12 unknown

J: Mirage

Problem Author: Steven Hans

- Naively iterating through the excluded indices won't pass the time limit. We need to think.
- Let the initial XOR from L to R be X. Observe that the maximum integer is $2 \cdot 10^5$. This can be stored in 18 bits. Then, build a tree that represents permutation P. The child will be either 0 or 1.
- To determine which number will produce the maximum XOR, greedily traverse the tree to the opposite bit starting from the MSB (most significant bit) if possible.
- This means if the i-th bit of X is 0, traverse to node 1 if possible and vice versa.
- Before traversing to a node, we need to know whether the node contains an element that is **outside** of the $[L, R]$ range. For each node, keep track of min index and max index. If the node has more range than $[L, R]$, then it's valid to be traversed.
- If we reach a leaf, say x, the answer will be $\max(X, X \oplus x)$. Otherwise, output X.
  Time complexity: $O(q \log(n))$.

  Statistics: 0 submissions, 0 accepted

K: The Smallest Difference, The Largest Difference

Problem Author: Morteza Muthahhari

- Add additional constraint, $(1 \leq T \leq 10^6)$.
- Approach 1:
  - Iterate all a, b, and c up to $\sqrt{T}$ with $(a \leq b \leq c)$ to skip permutations of a, b, and c that doesn't produce the smallest a. We then calculate the remaining d^2 and check if that is a perfect square. After that, we can evaluate the equation to get d and find maximum $|d - a|$.

  Time complexity: $O(T\sqrt{T})$.
- Approach 2:
  - Tweak the equation to $a^2 + b^2 = T - c^2 - d^2$.
  - We just need to ensure left side and right side are equal.
  - We just need to *mapping* all sum of two perfect square in $O(T)$. Save this result to an map and use this to iterate c and d then check did $T - c^2 - d^2$ are sum of two perfect square on O(1) each iteration.

  Time complexity: $O(T)$.

  Statistics: 11 submissions, 0 accepted, 6 unknown