

University of São Paulo

Institute of Mathematics and Statistics

IX USP Freshman Try-outs Editorial

Problem	Author	Prepared by
A - Arc surveillance	Thiago Oliveira	Otávio de Oliveira
B - Battle in space	Rafael Sesoko	Paulo Dias
C - Cosmic candidates	Rafael Sesoko	Rafael Sesoko
D - Dividing the solar pizzas	Enrique Junchaya	Enrique Junchaya
E - Excavating Mercury	Enrique Junchaya	Gabriel Morete
F - Festival of the Moon	Marcelo Lage	Cauê da Costa
G - Genealogy of Aliens	Marcelo Lage	Marcelo Lage
H - Honor to Saturn	Pedro Simões	Pedro Simões
I - Investigating Mars	Marcelo Lage	Nathan Martins
J - Jupiter's Dinner	Thiago Oliveira	Thiago Oliveira

Table 1: Contest problems.

A. Arc surveillance.

Tags : implementation.

Let D be the largest difference between two numbers of adjacent cells in circular order (if c_i are in increasing order, these values are $c_{i+1} - c_i$ for $1 \leq i \leq n - 1$ and $c_1 - c_n + 360$). If the span of cells under surveillance is strictly less than $360 - (D - 1)$, then there exists an interval of at least D unmonitored cells, and among these D cells, some are occupied. Therefore, it is necessary to monitor at least $361 - D$ cells with cameras.

B. Battle in space.

Tags : ad hoc.

Given an integer n , we are only asked to print an integer strictly greater than n . It suffices to print $n + 1$.

C. Cosmic candidates.

Tags : math.

As the Coach must convince the majority of the members in a team, he must convince exactly two members per team. Note that the number of teams is $\frac{n}{3}$. Therefore, the number of members the Coach must convince should be $\frac{2n}{3}$.

D. Dividing the solar pizzas.

Tags : greedy, sortings.

Arrange the prices of the flavors to suppose $p_1 \geq p_2 \geq \dots \geq p_n$. It's easy to see that the i -th most expensive pizza will cost at least p_{2i+1} , because it's not possible to use all $2i + 1$ most expensive flavors in the first i pizzas, which only have a total of $2i$ halves. Therefore, the answer is at least $p_1 + p_3 + \dots + p_{2n-1}$, which can be achieved by placing the flavors p_{2i-1} and p_{2i} on the i -th pizza.

E. Excavating Mercury.

Tags : greedy, implementation.

Since the height of a hill cannot increase, the height of all hills in the end cannot be greater than $m := \min_{1 \leq i \leq n} a_i$. Therefore, we must remove at least $a_i - m$ meters from hill i , and the final answer is $\sum_{i=1}^n (a_i - m)$, which can be calculated in $O(n)$ time.

F. Festival of the Moon.

Tags : linear equations, math.

Let x be the ammount of people who paid full price, and y be the ammount of people who paid half price. Then, we have the following system of equations:

- $x + y = p$
- $vx + \frac{v}{2}y = a$

Substituting the value of x by $p - y$, and solving for y , we have that $y = 2p - \frac{2a}{v}$.

G. Genealogy of aliens.

Tags: math.

For a given n , we want to find the number of solutions for $a + ar + \dots + ar^m = n$, where a, r , and m are integers with $a \geq 1, r > 1$, and $m \geq 0$. Rewriting it as $a = \frac{n}{1+r+\dots+r^m}$, we see that the value we want to find is the number of pairs $(r > 1, m \geq 0)$ of integers with $1 + r + \dots + r^m$ divisor of n . $m = 0$ corresponds to a single solution.

The first idea is to iterate over r from 1 to n and over m from 1 up to some value that makes the sum greater than n and check, but this solution is $O(n \log n)$, which is not efficient enough.

A refined approach is to do this for $m \geq 2$, since $m = O(\log n)$ and $1 + r + r^2 > n$ for $r > \sqrt{n}$, and this part has complexity $O(\sqrt{n} \log n)$.

For $m = 1$, we simply want to find the number of positive divisors of n that are at least 3, and this can be done in $O(\sqrt{n})$ by iterating through all divisors.

The final solution has complexity $O(\sqrt{n} \log n)$.

H. Honor to Saturn.

Tags: strings.

We can represent the given string as a sequence $[c_1, c_2, \dots, c_m]$ of integers corresponding to the size of maximal contiguous monochromatic sequences in the string. For example, the string "abbbaa" corresponds to the sequence $[1, 3, 2]$. To calculate the final answer, we have two options: the asteroid sequence can correspond to only two consecutive elements of the sequence c_i (that is, $\max_{1 \leq i \leq m-1} \min(c_i, c_{i+1})$) or it can be formed by three consecutive elements of the sequence c_i

(that is, $\max_{c_i \leq c_{i-1} + c_{i+1}} c_i$), as in the case of the string "abbbbaa" described in the statement. The solution is in $O(n)$.

I. Investigating Mars.

Tags: ad-hoc, implementation.

To detect when the robot stops its journey, it's not sufficient to detect revisiting a house—it must revisit the house in the same orientation. Here's how we simulate the robot's path and update a three-dimensional boolean array $vis[n][m][4]$ based on its current position and direction.

We maintain the robot's current position and direction, and at each step, we attempt to move in the current direction. If this movement isn't possible (e.g., the next cell is '#' or outside the board), we change direction and continue the loop. Each iteration takes $O(1)$ time, and since we can have at most $4nm$ iterations, the final complexity is $O(nm)$.

J. Jupiter's Dinner.

Tags: sliding window, two pointers.

We will present two solutions.

The first is using binary search on the answer using a sliding window approach. We want to find the largest m such that some interval of size m in the sequence has at most k distinct elements (if m works, any $m' < m$ also works). We start by fixing a window of size m on the initial elements of the sequence, and in each iteration, we "slide" this window to the right, maintaining the count of distinct elements within this window (this can be done using a frequency vector: when a frequency drops to zero, the count of distinct elements decreases by one; when a frequency increases to one, the count increases by one). Thus, in $O(n)$ time, we can determine if m works. The answer is in $O(n \log n)$ using binary search.

The second solution also uses a frequency vector with two pointers (one integer to indicate the start of the current interval and another to indicate the end): we start with an interval of size 1 containing only the first element of the sequence, maintaining the count of distinct elements within it, and we move the right end of the interval while maintaining its size until there are $k + 1$ distinct elements. Then we move the left end of the interval until there are k distinct elements in the interval again. We repeat the process with the current interval until the right end reaches the last position of the sequence. Whenever we have a sequence with at most k distinct elements, we update the size of our maximal interval. This solution runs in $O(n)$ time.