Editorial - Cuarta Fecha del ICPC Centroamérica

February 4, 2021

Problem A - Average Problem

There are only 21 possible outcomes: $\frac{0}{100}, \frac{5}{100}, \ldots, \frac{95}{100}, \frac{100}{100}$. We can use fractions to represent a grade g and choose the closest outcome o that minimizes $abs(g-o)$, without resorting to floating-point arithmetic. It is possible to solve using float-point arithmetic but it requires very careful rounding manipulation.

Problem B - Breaking Vacations

This is a variation of the classic 0-1 knapsack dynamic programming problem. Each state $dp[day][left]$ represents what is the current day and how many points are left to achieve at least K. The main rules of the recurrence are:

- $dp[N+1][0] = 0$
- $dp[N+1][left] = -\infty$ if $left \neq 0$, where $-\infty$ is represented with a very low negative number.
- $dp[day][left] = \max\left(dp[day+1][\max\left(0, left - s_i\right)]\right), \max\left(v_i, p_i\right) + dp[day+1][left]$ for $1 \leq day \leq N$

The solution $dp[1][K]$ is valid if it is non-negative.

Problem C - Cardinality of Sets

Since each set is represented with a list with unique values, the cardinality of the intersection can be determined with the amount of duplicates when concatenating both A and B. By sorting the resulting list, it is easy to count the duplicate values in a single pass. Time complexity is $O((A+B)log(A+B))$.

Problem D - Determining Fibonaccism

We can first store the results of $fib(i)$ for each i such that $0 \leq i \leq 10000$ in a set. Then, for each query, we can check if $n \bmod 2^{31} - 1$ is contained in the set.

Problem E - Exceeding the Goal

It is possible to binary-search the solution. We have to find the lowest x such that $T = (2S + K \cdot (x-1))/2 = (S \cdot x) + (x \cdot (x-1)/2) \cdot (K) > G$. This can be derived using arithmetic progression formulas. For C/C++ and Java, 64-bit integers should be used and we may need to partially evaluate the expression against G to avoid overflow. One way to partially evaluating is checking if any of the following conditions hold:

- If $S > G$, then $T > G$.
- If $(x \cdot (x-1)/2) > G$, then $T > G$.
- If $K > G$ and $x > 1$, then $T > G$.

Otherwise, since $G \leq 10^9$, it is not possible for T to overflow at this point, so we can evaluate directly if $T = (S \cdot x) + (x \cdot (x-1)/2) \cdot (K) > G$.

Problem F - Farthest Cell in a Maze

This a breadth-first search problem. By visiting the adjacent directions in lexicographic order (D, L, R, U) all the cells will be reached by following the smallest lexicographic route which is also optimal.

Problem G - Geometric Sequence

The original recursive function (very slow):

$$S(n) = \begin{cases} t_1 & \text{if } \; n = 0 \\ t_2 & \text{if } \; n = 1 \\ t_3 & \text{if } \; n = 2 \\ \vdots & \\ t_k & \text{if } \; n = k - 1 \\ S(n \bmod k) \cdot S(n - k) & \text{if } \; n \geq k, \; n \in \mathbb{N} \end{cases}$$

The fast-recursive function:

$$S(n) = \begin{cases} t_1 & \text{if } \; n = 0 \\ t_2 & \text{if } \; n = 1 \\ t_3 & \text{if } \; n = 2 \\ \vdots & \\ t_k & \text{if } \; n = k - 1 \\ S(n \bmod k)^{\lceil \frac{n+1}{k} \rceil} & \text{if } \; n \geq k, \; n \in \mathbb{N} \end{cases}$$

Use **Modular Exponentiation** for calculate $S(n \bmod k)^{\lceil \frac{n+1}{k} \rceil}$.

Problem H - Hidden Professions

The basic way to do it is by using a trie. We can store in each node the set of professions that have passed through it. A profession is valid iff the prefix's node exists and has only one element in its set.

Alternately, you may use a bitmask instead of the set, a prefix is valid if has only one bit set.

Problem I - Imaginary Numbers

You may figure out that i^n has only four possible outcomes:

$i^0 = 1$	$i^n = 1$ if $0 \equiv n \pmod 4$
$i^1 = i$	$i^n = i$ if $1 \equiv n \pmod 4$
$i^2 = -1$	$i^n = -1$ if $2 \equiv n \pmod 4$
$i^3 = -i$	$i^n = -i$ if $3 \equiv n \pmod 4$

So you must just print a with an i or a minus symbol, accordingly.

Problem J - Johann's Function

The value of the input allows to precalculate the results and keep them in an array, with a million positions. That task requires to do the next job:

1. Memoize the Gauss Summation into the array's positions
2. Memoize the Johann's Function just as an accumulative table of Gauss Summations.

n	1	2	3	4	5	6	7	8	9	10	11	12	$\cdots$
$Gauss(n)$	1	3	6	10	15	21	28	36	45	55	66	78	$\cdots$
$Johanns(n)$	1	4	10	20	35	56	84	120	165	220	286	364	$\cdots$

A better solution can be finding the direct formula for every position, calculating the double summation, though it is explained in *Problem M - Mandatory by Summations.*

Problem K - K-Uniform Array

An useful insight is that if we need to create a k-uniform array, it is always optimal to delete numbers in a single subarray, only keeping the number we need repeated at least k times. When we traverse the array from left to right, at the each index i, we will check if we've seen this number at least k times (this can be done with a map that stores a vector v_{a_i} with every index where a value a_i appears). If we've seen the number at least k times, then we are interested in the subarray with indices that starts in $v_{a_i}[len(v_{a_i}) - k]$ and ends in $v_{a_i}[len(v_{a_i}) - 1] = i$. The cost will be the number of distinct numbers in this subarray minus 1, since the values equal to a_i won't be deleted. To answer each one of this distinct-elements queries to determine the number of distinct numbers in a subarray, an useful data structure is the Fenwick Tree/Binary Indexed Tree. The technique is described here: https://www.geeksforgeeks.org/queries-number-distinct-elements-subarray/. Some observations related to the article that apply to this problem:

- The last seen index of a number can be stored with a map instead of an array if the values are greater than 10^6.
- Since we are traversing the array from left to right, there is no need to sort queries by the right index.

The answer will be the minimum of the result of each query, if there were any.

Problem L - Lighting System

The idea is to modify the union-find structure in order to keep the connected components, the main modification is in the union method, to be sure to always join both components into the minimum numbered component.

This can also be solved with a simple DFS to detect connected components in a graph.

Problem M - Mandatory by Summations

$$
\begin{aligned}
JohannsFunction(n) &= \sum_{i=1}^{n}\left(\sum_{j=1}^{i} j\right) \\
&= \left(\sum_{j=1}^{1} j\right) + \left(\sum_{j=1}^{2} j\right) + \left(\sum_{j=1}^{3} j\right) + \cdots + \left(\sum_{j=1}^{n} j\right) \\
&= \frac{1\cdot 2}{2} + \frac{2\cdot 3}{2} + \frac{3\cdot 4}{2} + \cdots + \frac{n\cdot(n+1)}{2} \\
&= \left(\sum_{i=1}^{n} \frac{i\cdot(i+1)}{2}\right) \\
&= \frac{1}{2}\cdot\left(\sum_{i=1}^{n} i\cdot(i+1)\right) \\
&= \frac{1}{2}\cdot\left(\sum_{i=1}^{n} (i^2+i)\right) \\
&= \frac{1}{2}\cdot\left(\sum_{i=1}^{n} i^2 + \sum_{i=1}^{n} i\right) \\
&= \frac{1}{2}\cdot\left(\frac{n\cdot(n+1)\cdot(2n+1)}{6} + \frac{n\cdot(n+1)}{2}\right) \\
&= \frac{n\cdot(n+1)}{4}\cdot\left(\frac{(2n+1)}{3} + 1\right) \\
&= \frac{n\cdot(n+1)}{4}\cdot\left(\frac{2n+4}{3}\right) \\
&= \frac{n\cdot(n+1)}{4}\cdot\left(\frac{2\cdot(n+2)}{3}\right) \\
&= \frac{n\cdot(n+1)\cdot(n+2)}{6}
\end{aligned}
$$

At least one of $a = n$, $b = n+1$, $c = n+2$ will be a multiple of 2, and at least one of them will be a multiple of 3. After taking each one of this factors, we are effectively dividing by 6, and we can perform the multiplication $((((a \,\%\, m)\cdot(b \,\%\, m)) \,\%\, m)\cdot(c \,\%\, m)) \,\%\, m$ where $m = 2^{31}-1$.

Problem N - Neither Divide Nor Use Double

A division is exact, according to the problem, if the result has non-repeating decimals.

A number in base 10 has non-repeating decimals if its reduced fraction $\frac{s}{t}$ has a denominator in the form $t = 2^v \cdot 5^w$ with v, w being non-negative integers. Therefore, if $a = (2^p \cdot 5^q)$, with p, q being non-negative integers, we will obtain that $a^n = (2^p \cdot 5^q)^n = 2^{p\cdot n} \cdot 5^{q\cdot n} = 2^v \cdot 5^w$. Also, if $n = 0$, then $a^n = 2^0 \cdot 5^0 = 2^v \cdot 5^w$. It is easy to see that m must be at least $\max(v, w)$ so that $\frac{10^m}{a^n}$ is an integer. x is then $2^{m-v} \cdot 5^{m-w}$, which must be given with arbitrary precision integers. If a is not of the form $(2^p \cdot 5^q)$ and also $n \neq 0$, then there is a 'Precision Error'.

Problem O - One Piece of Cake

The solution is to check if $P - E \geq 10$.

Problem P - Professor Sabio and the Easy Homework

We can solve this with brute force. The first step is to select a subset of 6 numbers from $0, \ldots, 9$, this can be done with six nested for-loops. Then, we can try all permutations of that selection to assign the values to 'A', 'B', ..., 'F'. In C++, 'next_permutation' is useful for this. For each combination and permutation, we evaluate if the expression holds. We can finally sort the valid solutions and print them.

Problem Q - Queries on the Stack

We can simulate the operations as if we were using a normal stack. The only difference is that when we push a value x into the stack we will store a pair '(x, x)' if the stack is empty or '$(\max(x, \text{stack.top}()), \min(x, \text{stack.top}()))$' when it has at least one elements. The queries can then be answered by subtracting the pair values on the top of the stack.