Mexican Grand Prix 2021 - First Date

August 28th, 2021

solution book

This document contains the expected solutions for the 13 problems used during the competition.

The following people supported developing the problem set either by creating or improving statements, solutions, test cases, input and output checkers:

Lina Rosales

Sara´ÿ Ram´ÿrez

Eddy Ramírez

Leonel Juarez

Alan Enrique Ontiveros

Abraham Macias

Emilio L´opez

Fernando Fori

Moroni Silverio

Juan Pablo Mar´ÿn

v1.0

Alien Crop Triangles

By: Emilio L´opez

To solve the problem, it is first necessary to calculate the total area to cover, this is achieved by adding the areas of all the input triangles. It can be done in a simple way using Heron's formula:

$$\text{area} = \sqrt{s\,(s - a)\,(s - b)\,(s - c)}$$

where s is the semiperimeter:

$$s = \frac{a + b + c}{2}$$

Knowing the area in m2 to cover, we proceed to calculate the kg of seeds needed, rounding up. The statement indicates that 1 kg is needed every 30 m2 , so

$$\text{kg needed} = \text{ceil}\left(\frac{\text{area}}{30}\right)$$

Then we use a backpack (knapsack) to minimize the cost of buying the necessary kg. We must choose the cheapest option that has at least the necessary kg – it may be cheaper to buy extra kilos; it does not have to be exactly the same amount of calculated kg. This cost will be the solution that we will print.

Finally, we must take special care to treat the edge cases of the input correctly, among which stand out the following two:

- If there is no triangle at the entrance, it is not necessary to buy anything (exit 0).
- If we have to buy something and the only bags available are empty (weight 0), the problem cannot be solved. problem (output -1).

Basel Problem

By : Alan Enrique Ontiveros

Let $f(z) = $ a function $\frac{ÿ \cot(ÿz)}{z^n}$, where n ÿ 2. This function will be useful for computing ÿ(n) using the Remainder Theorem, because f(z) has poles at every integer.

Integrating f(z) over a very large square centered at the origin of the complex plane, with vertices at the points: {(N + 1/2)(1 + i),(N + 1/2)(ÿ1 + i),(N + 1/2)(ÿ1 ÿ i),(N + 1/ 2)(1 ÿ i)}, where N ÿ ÿ.

It can be shown that: $H_{ÿ}$ f(z)dz = 0. By the Remainder Theorem, this integral is:

$$I_{ÿ} f(z)dz = 2ÿi\, Xÿ_{k=ÿÿ} \operatorname{Res}_{z=k} f(z)$$

Comparing both expressions, we have:

$$Xÿ_{k=ÿÿ} \operatorname{Res}_{z=k} f(z) = 0$$

We see that f(z) has poles of order 1 at each integer different from 0, that is, if k 6= 0 is an integer, then:

$$\operatorname{Res}_{z=k} f(z) = \lim_{zÿk} f(z)(z ÿ k)$$

$$\lim_{zÿk} \frac{ÿ \cot(ÿz)(z ÿ k)}{z^n} =$$

$$= \lim_{zÿk} \frac{ÿ \cos(ÿz)(z ÿ k)}{\sin(ÿz) z^n} \lim_{zÿk} ÿ$$

$$\cos(ÿz) = \lim_{zÿk} \frac{}{z^n} \lim_{zÿk} \frac{zÿk}{\sin(ÿz)}$$

$$= \frac{ÿ \cos(ÿk)}{nzÿk} \lim k \frac{\text{one}}{ÿ\cos(ÿz)}$$

$$= \frac{ÿ \cos(ÿk)}{k^n} \cdot \frac{1}{ÿ\cos(ÿk)}$$

$$= \frac{\text{one}}{k^n}$$

Since n is a positive integer even number, we have that:

$$2Xÿ_{k=1} \frac{\text{one}}{k^n} + \operatorname{Res}_{z=0} f(z) = 0$$

$$ÿ(n) = ÿ \frac{\text{one}}{2} \operatorname{Res}_{z=0} f(z)$$

So finding $\operatorname{Res}_{z=0} f(z)$ is the same as finding the coefficient of $z^{ÿ1}$ in Laurent's series for f(z), or equivalently, find the constant term in $\frac{ÿz \cot(ÿz)}{z^n}$.

Finally to find all the values of ÿ(n) at the same time, we can observe that ÿ(n) is the z coefficient ^{n} in the Taylor series: ÿ $\frac{1}{\text{two}}$ ÿz cot(ÿz).

We know that $ÿ(n) = \frac{pn}{qn} ÿn$, so:

$$Xÿ_{\substack{n=0\\ \text{not even}}} ÿ(n)z^n = Xÿ_{\substack{n=0\\ \text{not even}}} \frac{pn}{qn} (ÿz)^n \overset{ÿ}{=} \frac{1}{\text{two}} ÿz \cot(ÿz)$$

Substituting x = ÿz, we have that:

$$g(x) = Xÿ_{\substack{n=0\\ \text{not even}}} \frac{pn}{qn} x^n \overset{ÿ}{=} \frac{1}{2} x \cot(x)$$

Note that g(x) = ÿ $\frac{\text{one}}{\text{two}}$xcot(x) = ÿ $\frac{\text{one}}{\text{two}} \cdot \frac{\cos(x)}{\sin(x)/x}$ is the quotient of dividing two Taylor series, and is the function that generates all the answers we are looking for.

We can extract the first n coefficients efficiently using FFT (NTT with the modulus 119 × 2^{23} + 1) taking the convolution of cos(x) = 0! one $\frac{\text{one}}{}$ $\frac{\text{one}}{2!x}$ 2 + $\frac{\text{one}}{4!x}$ 4 $\frac{\text{one}}{6!x}$ 6 + · · · with the multiplicative inverse of sin(x)/x = $\frac{\text{one}}{}$ $\frac{\text{one}}{3!x}$ 2 + $\frac{}{5!x}$ 4 $\frac{\text{one}}{7!x}$ 6 + · · · (the inverse can be calculated using the Newton-Raphson method) one! with a complexity of O(n log n) if we ignore the coefficients after x n of both series.

cypher decypher

By : Eddy Ram´ÿrez

This is a classic problem. Since the value of the numbers will never be greater than 10^6, a sieve can be used to calculate the prime numbers. Once we have the sieve C we can know if a number is prime if C[x] = 1 and is not prime if C[x] = 0. Now, if for each value i, j given in the input, we count how many values C[x] = 1 such that i ÿ x ÿ j we will be finding the correct answer, but, the added iteration will cause the submission to be rejected for exceeding the execution time limit .

We can create a new array S such that S[x] = S[x ÿ 1] + C[x], in this way we can observe that S[x] has the accumulated value of how many prime numbers there are between 1 ei (the sum of how many C[i] = 1 up to i). Using this array, we can find the number of primes between i and j simply by calculating S[j] ÿ S[i ÿ 1].

delivery pizza

By: Moroni Silverio

The best way to solve it is one ingredient at a time ignoring the others starting with the first ingredient, by reading the entry we know which will be the only ones that will pass through there and they must be treated according to the order of entry.

For the other ingredients, likewise focus on the j-th ingredient if the previous ingredients have been resolved, since no more orders will be added to that ingredient later. What must be kept in mind is the moment in which the person serving ingredient J becomes vacant, once he does, he has to choose all the orders that arrived before he became vacant. If no orders arrived before that, you must choose the first order that will arrive after it is vacated. This is done with two priority queues, one to keep track of orders that arrived before it became free, and one to keep track of orders that arrived after it became free.

Escape Room

By: Moroni Silverio

We can see that the answer can be given to any of the questions such that the coordinate is not a '.' on the map just by looking at the map. So, we are interested in finding out how to answer the questions that are asked to these coordinates: The problem of finding the shortest path between two points on a map like the one given in the entry is well known and can be solved by applying a b amplitude search, then, a possible solution to the problem would be for each query to perform the search in amplitude starting from that coordinate to find the shortest path to the exit, in case there is one One Way. Although this solution is correct, it is very slow with the number of queries that could be given as input, and it would be rejected for exceeding the execution time limit.

A feature of breadth search is that it not only finds the shortest path from a source node to a destination, but also finds the shortest path from the source node to any other node that is reachable. on the map starting from the origin, we can see that the shortest path from a position on the map to the exit is the same as the path from the exit to that position, so we can calculate the shortest path short from any position on the map to the output by performing a breadth search originating from the map output. It is important to take into account the priorities with which to respond in case a point has more than one path with the same distance to the exit. Once this precomputation is done, each answer can be computed in O(1).

Fix Subtitles

By: Moroni Silverio

To solve the problem you have to “parse” the input to separate the subtitles from the time they start. Having that information and the time correctly parsed, you just have to add to get the new time. All lines that do not have a time must be printed as received.

Cases where the delay is 0 must be considered.

Game of Baker

By: Lina Rosales

heat wave

By: Fernando Fiori

I first describe a solution for a fixed size w and then it can be extended to any size < 200.

Observation 1) perhaps not all the occurrences of a word can be replaced because then the operation may not be reversible, occurrences that do not overlap can be replaced. Example: T=000, w=00, the correct replacement operation returns T'=20 or T'=02, but never T'=22.

Remark 2) If I know all the occurrences of a word w in T, an optimal way to choose the replacements is to choose the first occurrence of w (traversing T from left to right), skip |w| characters, and repeat the process. There is no way that compresses the text more than this. (Proof, let i be the index of the first occurrence of w in T, and j be the index of the next occurrence, i < j. Suppose I do not choose i as a replacement, but j. Then I can continue choosing occurrences of w in T starting at j + |w|, that is, at S = Tj+|w| ...T|T|ÿ1 But if we had chosen the occurrence of i, then we could continue to choose occurrences in R = Ti+|w| ...T|T|ÿ1 But S is contained in R, and we are never going to find more occurrences in a piece of text than in the whole text, since I can omitting those first ji characters, so choosing j instead of i does not give us a better solution.)

Now, if we look for a |w| of a given size that maximizes the compression of T, we can go through T hashing the window of size w. We can use polynomial hash since the maximum size of the string is greater than 64 bits. We are keeping a hashmap with a string hash as key, and as value a number of occurrences occ, last index where last was seen. So if last < current indexÿ|w|, then I do occ + + and last = current index.

Once I finish traversing T, I review each map entry and keep the word with the highest number of occurrences.

I perform this process for each possible size of w (from 2 to b) and keep the best one.
Then it's just making an account and displaying the result.
The final complexity is O(|T| ÿ b), not counting the cost of accessing hashmap values.

Introducing Teleporting Machine

By: Moroni Silverio

To obtain the time without the teleportation machine, just add city[n] ÿ city[i] for all
i.

Now, to calculate the total time with the machine connecting cities i and j we need:

- The total time without machine T.
- What we save in time D: the number of cities that will use the machine for the distance that is
will save that is the distance between i and j
- The cost of using the machine C: The number of cities up to i times the cost of using the machine

Then, when the machine is placed between cities i and j, the cost will be $T_{i,j}$ = T ÿ D + C, we must find the i,j that minimizes the value of $T_{i,j}$. This can be achieved by iterating over each city i where the machine can be placed and setting j to the furthest city that is less than or equal to the maximum K kilometers that the machine can connect. In the transportation machine, finding city j for each possible value of i can be achieved with a binary search, and then the algorithm to find the values i,j that minimize $T_{i,j}$ is O(NlogN).

Just Send the Email

By : Abraham Mac´ías

Let the employees be nodes and the manager-subordinate relationships be edges, the structure of the company's organization forms a tree rooted at 1. In this problem we are asked to find the expected value of the distance between a random node u from the tree to the nearest leaf. Note that from a node u can be moved to any of its children or to the parent, so it could be considered an undirected tree.

To find the answer, we can find the sum of the shortest distances to a leaf from each node. To find the shortest distance from a node to the nearest leaf, you can do a multi-origin BFS (breadth search) from all leaves to the other nodes (enter all sheets to the queue and then run a BFS). The initial distance of each leaf is 1.

After finding the sum of the distances, multiply it by the modular multiplicative inverse of n. To find the modular inverse, one can use Fermat's little theorem or Euclid's extended algorithm.

Kids at the Party

By : Sara´ÿ Ram´ÿrez

It can be seen that for a number M of children to attend the party, it is necessary that N mod M = 0. This means that M is a divisor of N. Given that Jaime wants his friend Churro to always attend and that there cannot be more than 6 children at the party, then the value of M satisfies that 2 ÿ M ÿ 6. Then, to find the different numbers of children that can attend the party, it will be necessary to find all the numbers M with the given constraints that are divisors of N. One drawback is that the value N can be so large that cannot be stored in an integer, so we must find a strategy that allows us to identify the possible values of M regardless of the size of the number. One way to do this is to apply divisibility criteria:

Divisibility criteria of the numbers 2,3,4,5,6 can be used to identify the possible values of M given N:

- 2 is a divisor of N if the last digit of N is even.
- 3 is a divisor of N if the sum of the digits of N is a multiple of 3
- 4 is a divisor of N if the last two digits of N are a multiple of 4
- 5 is a divisor of N if the last digit of N is 5 or 0.
- 6 is a divisor of N if 2 and 3 are divisors of N.

Leonel and the powers of two

By: Leonel Ju´arez

The notation requested from Leonel is not actually invented by his teacher. As such, it is the recursive function that is used to perform the calculation of an exponent with the algorithm known as binary exponentiation: https://es.wikipedia.org/wiki/Exponenciaci%C3%B3n_binaria.

The problem then boils down to implementing the recursive function that defines the notation that Leonel was asked to do, taking care not to leave blank spaces and to follow the printing according to the 3 rules defined.

Moon Dancers

By: Juan Pablo Mar´ÿn

An important feature to note in the problem is that, since all the dancers will always sit in a position such that the angle is integer, then there are exactly 359 possible positions for a dancer who gets up to rotate towards another dancer, also as all the dancers who get up will rotate the same angle, we can think of a strategy to solve the problem in which we find the maximum number of pairs that can be made if the dancers rotated a fixed amount R, and find the maximum among the 359 values that R can take.

Given R, we can see that dancer i could be a partner with dancer j, if (i + R) mod 360 = j. Let us define a directed graph G, where the vertices are each of the positions where there is a dancer, and there is an edge that goes from the initial position i of a dancer to position j if and only if there is a dancer at position j and the dancer at position i can reach dancer j by rotating R. Given G, we can visualize the problem as coloring as many nodes as possible using two colors A and B so that if two nodes u, and v are colored and there is an edge that joins them, then u is colored with color A and v is with color B. To carry out this task, we observe that , due to the characteristics that have the rotations, each vertex will have at most one output edge and one input edge for each of the possible rotations, then we can verify that there will be no intersecting chains or cycles in G, so ÿ, if we take a connected component of G we could "stretch" it so that it forms a line, due to this character Statistics, we can see that if the connected component has Cv nodes of G, then bc pairs of that component can be made. So, to count how many pairs can be made in one rotation, just add the number of pairs that can be made for each component of the graph. The answer to the problem will be the largest value obtained after obtaining the $\frac{hp}{2}$ answer for each of the 359 possible rotations.

Mexican Grand Prix 2021 - Second Date

September 25th, 2021

solution book

This document contains the expected solutions for the 11 problems used during the competition.

The following people supported developing the problem set either by creating or improving statements, solutions, test cases, input and output checkers:

Eddy Ramírez

Abraham Macias

Alan Enrique Ontiveros

Roberto Solís

Moroni Silverio

Juan Pablo Mar´ÿn

v1.0

alice birthday

By : Abraham Mac´ías

This problem can be solved with DP with bitmasks. Let dp(mask, i) be the number of ways to eliminate edges in the subgraph induced by the nodes in mask, such that there are i connected components in said subgraph induced after eliminating said edges. A node u is in mask if the u-th bit in mask is on.

Let's calculate the DP in increasing order of mask. The value of dp(mask, i) for i ÿ 2 can be found as follows:

$$dp(mask, i) = \sum_{sub} dp(sub, 1) ÿ dp(mask XOR sub, i ÿ 1)$$

such that sub is a subset of mask and sub has the node lsb(mask), where lsb(mask) is the smallest node in mask (least significant bit set in mask).

The reason this works is that, when we add dp(sub, 1) ÿ dp(mask XOR sub, i ÿ 1), we are adding the number of ways to remove edges such that the connected component where lsb is (mask) is sub, and the rest of nodes in mask (mask XOR sub) are distributed in (iÿ1) connected components. So to find dp(mask, i), we consider all possible connected components where lsb(mask) can be.

The value of dp(mask, 1) can be found as follows:

$$dp(mask, 1) = 2E(mask) ÿ \sum_{i=2}^{n} dp(mask, i)$$

where E(mask) is the number of edges in the subgraph induced by mask. The reason for this is that there are 2 E(mask) ways to remove the edges in this subgraph, so the sum of all dp(mask, i) for i ÿ 1 must equal 2E(mask) . We simply isolate dp(mask, 1) to find it.

The answer for a given K is in dp(2n ÿ 1, k). The submasks of each mask can be found as follows: https://cp-algorithms.com/algebra/all-submasks.html.

The complexity is O(n ÿ (3n));

Benford’s Law

By : Abraham Mac´ías

We first note that the answer can be found using the linearity property on the expected value. For each bucket i and each digit d, we can calculate separately how much bucket i contributes to the expected value of cd. It can be seen that the expected value of cd can be represented as follows:

$$E(cd) = X^{n}_{i=1} \; P_{i,d}$$

Where Pi,d is the probability that the number of balls in bucket i have the digit d as the first digit. Let's see a way to calculate the values of E(cd) in O(NM):

For each bucket i from 1 to N, let's iterate for each j from 0 to M, and see what is the probability that during the remaining M steps, Alan inserts j balls into bucket i. There are ways to choose $\binom{m}{j}$ the steps in which Alan inserts the balls into bucket i, and for the remaining M ÿ j steps there are (N ÿ 1)Mÿj ways to choose where to insert the remaining balls. In total there are NM ways to insert the balls in the M steps, so with all of the above, if we define H(j) as the probability of inserting j balls in bucket i, we have:

$$H(j) = \frac{\binom{m}{j} \text{ ÿ } (N \text{ ÿ } 1)^{M\text{ÿ}j}}{SL}$$

It is enough to find out which is the first digit of ai + j to know which is the E(cd) to which H(j) contributes.

To improve the complexity, it should be noted that it is not really necessary to iterate through all the j, since, for example, when we arrive at ai + j = 100, we know that all the numbers between 100 and 199 have 1 as first digit, so we can add to E(c1) all H(j) such that 100 ÿ ai + j ÿ 199 in constant time if we precalculate the prefix sums of H(j), and so on with all ranges of numbers that have the same first digit.

So, if we precalculate all the prefix sums of H(j) and make such jumps for the numbers that have the same first digit, we can obtain a complexity of O(max(ai) + M) for do the precalculation and O(9 ÿ log10(K) ÿ N) to obtain the expected values, since there are O(9 ÿ log10(K)) ranges of numbers that have the same first digit.

Cut the Deck!

By: Juan Pablo Mar´ÿn

Although the game can be tested after making a cut for each card i, this would be O(N2) and with values large of N would exceed the execution time.

Note that if we start a counter at 0, and add 1 for each blue card, and subtract 1 for each red card, Bob will lose the game when this counter turns negative (there have been more ´as red cards than blue). We can see that since there are the same number of red and blue cards, the counter will always start and end at 0 after all the cards are turned over. So, if Bob makes the cut at the point where this counter has the smallest possible value (making that 0), we can see that in that game the counter will never be less than 0 and therefore Bob wins.

Then we just have to find the smallest index im such that the counter at that point is equal to the minimum value that the counter obtains in the initial configuration.

Dislike the Raisins

By: Moroni Silverio

Let us call T the number of units in the cereal box, then T = C + R. Let CS be the total of tablespoons that Jaime can get from the box, this is CS = dT /Se

For Jaime to get the maximum amount of outstanding spoonfuls, he will need the minimum number of spoonfuls that are not outstanding, that is when these spoonfuls have the maximum amount of raisins, let CR be the number of spoonfuls that they can be made by filling the spoonful with the maximum amount of raisins that is possible in each one of them, that is: CR = dR/Se And then the maximum amount of outstanding spoonfuls that Jaime will be able to obtain are CS ÿ CR.

To get the fewest number of overhanging tablespoons, we can distribute the raisins at least one per tablespoon, so the minimum amount of overhanging tablespoons that Jaime can obtain will be max(CS ÿ R, 0)

E-13 Storage Unit

By : Roberto Solís

Given the sizes of each video, precalculate the sums of prefixes of these sizes in order to efficiently calculate the sum of videos from position i to position i + R ÿ 1. Having the sums of prefixes you could try different values of R until you find one that makes the sum exceed the capacity, a more efficient way is to do a binary search on the value of R. The solution would therefore be O (NlogN).

Flipped Factorization

By : Alan Enrique Ontiveros

0.1 Previous concepts • Let f : N

ÿ R be a function that receives a positive integer and returns anything. We call f(n) arithmetic function.

• If f(n) is arithmetic and also satisfies that f(mn) = f(m)f(n) for coprime positive integers m and n, that is, with gcd(m, n) = 1, we also say So f(n) is multiplicative. It seems that there are only two possible cases:

– f(n) = 0 for all n ÿ N. This case is not interesting. – f(1) = 1, we will assume that every multiplicative function f(n) satisfies this.

• A multiplicative function f(n) is fully defined if we know what it is equal to in powers of primes, that is, f(p a) for p prime and a ÿ N. This is true, since we can factor n into primes of the form n = p1 ^{a1} p_2 ^{a2}, and since two different primes are always coprime, then by definition f(n) = f(p1 a1 p_2 ^{a2} · · ·) = f(p1 a1)f(p2 a2) · · · ·

• Let f(n) and g(n) be two arithmetic functions. We define the function h(n) = $\sum_{d|n}$ f(d)g $\left(\frac{n}{d}\right)$ as the Dirichlet convolution of f(n) and g(n) (the sum iterates over all positive divisors d of n) and we write h(n) = (f ÿ g)(n).

• Theorem: if g(n) and h(n) are multiplicative functions, then their Dirichlet convolution is also multiplicative.

• Let f(n) be an arithmetic function, we define F(n) = $\sum_{i=1}^{n}$ f(i) as the function of partial sums of f(n). Usually, we will always use lowercase for the arithmetic function and uppercase for its partial sum function.

0.2 Subproblem 1

Let us first solve this subproblem: let h(n) and u(n) be two arithmetic functions, and let f(n) = (h ÿ u)(n) its Dirichlet convolution. How do we find F(n) efficiently? By definition we have that:

$$F(n) = \sum_{i=1}^{n} f(i) = \sum_{i=1}^{n} \sum_{d|i} h(d)u\left(\frac{i}{d}\right)$$

Let's rearrange the sum as follows: we see that each integer i contributes all its divisors gives the sum with the term h(d)u $\left(\frac{i}{d}\right)$. Now let's iterate through the values of d first, so i is a multiple of dy we can reindex i = dk for k ÿ N. We see that d cannot exceed n and is at least 1, so:

$$F(n) = \sum_{d=1}^{n} \sum_{k=1}^{dk ÿn} h(d)u\left(\frac{dk}{d}\right)$$

$$= \sum_{d=1}^{n} h(d) \sum_{k=1}^{bn/dc} u(k)$$

The inner sum is nothing more than U (bn/dc), where U(n) is the function of partial sums of u(n). For the so much:

$$F(n) = \sum_{d=1}^{n} h(d)U (bn/dc)$$

The advantage of rearranging the sum in this way is that it is often easy to compute U(n), even with constant complexity.

Machine Translated by Google

0.3 Subproblem 2

From the previous subproblem suppose that h(n) is also multiplicative, with the additional restriction that h(p) = 0 for any prime p. How does that restriction help to reduce the complexity to find F(n)?
Let d = p1 a1 · p2 a2 be given in its prime factorization, then h(d) = h(p1 a1)h(p2 a2)· · · , and if some ai is equal to 1, we will have h(d) = 0. This means that for h(d) to be different from zero, all the exponents of each prime of d must be at least 2, we will call these numbers d powerful.

How many powerful positive integers are there less than or equal to n? There is O(ÿ n), and the proof remains as an exercise. Therefore, there are only O(ÿ n) terms in the expansion of F(n), and if both h(n) and U(n) can be found with constant complexity, we have that F(n) can be found with a complexity of O(ÿ n).
To generate all powerful numbers less than or equal to n, we can use a DFS or BFS by precomputing all primes up to b ÿ nc.

0.4 Main problem

Note that the transformation we do to a positive integer m described in the problem can , and what we be described with a multiplicative function that satisfies f(p a) = a p want to find is F(n). Yes we could adequately find h(n) and u(n) such that:

• f(n) = (h ÿ u)(n) •

h(n) and u(n) are multiplicative •

h(p) = 0 • U(n) can be found

easily

then we will have solved the problem with a complexity of O(ÿ n).

Arbitrarily (rather by trial and error) we can choose u(n) = 1 and try to "clear" the function h(n):

$$
\begin{aligned}
f(p) &= u(p) + h(p)) \\
f(p^{2}) &= u(p^{2}) + u(p)h(p) + h(p^{2}) \\
f(p^{3}) &= u(p^{3}) + u(p^{2})h(p) + u(p)h(p^{2}) + h(p^{3}) \\
f(p^{4}) &= u(p^{4}) + u(p^{3})h(p) + u(p^{2})h(p^{2}) + u(p)h(p^{3}) + h(p^{4}) \\
&\vdots
\end{aligned}
$$

$$
\begin{aligned}
&=ÿ h(p) = f(p) ÿ 1 = 1 ÿ 1 = 0 \\
&=ÿ h(p^{2}) = f(p^{2}) ÿ 1 ÿ h(p) = 2p ÿ 1 \\
&=ÿ h(p^{3}) = f(p^{3}) ÿ 1 ÿ h(p) ÿ h(p^{2}) = 3p ÿ 1 ÿ (2p ÿ 1) = 3p ÿ 2p \\
&=ÿ h(p^{4}) = f(p^{4}) ÿ 1 ÿ h(p) ÿ h(p^{2}) ÿ h(p^{3}) = 4p ÿ 1 ÿ (2p ÿ 1) ÿ (3p ÿ 2p) = 4p ÿ 3p \\
&\vdots
\end{aligned}
$$

And by induction, we can show that h(p a) = a p ÿ (a ÿ 1)p for a ÿ 2.

Finally, we have that U(n) = n and that h(p a) practically has constant complexity, so while we calculate the powerful numbers with the DFS/BFS, we also have to simultaneously calculate the value of h(n).

grid of letters

By: Juan Pablo Mar´ÿn

It might be tempting to try to simulate all the paths (as is done in a traditional word search), but with the characteristics of the problem there are so many paths that this solution would exceed the time limit of execution.

Suppose that the position (i, j) has the letter c, the largest path that ends at this position will be 1 + the maximum of the neighbors that have a letter c ÿ 1. Then , we can first find the largest path that can be done in all positions that have the letter 'A', then with 'B', etc. . . and save the largest value found, if the positions in which each letter appears in the matrix are previously stored, the complexity would be O(N*M).

Haunted House

By : Abraham Mac´ías

The problem can be solved by simulating the active periods of each ghost. Let us first notice that we are only interested in the first MX = 105 +N seconds, since in the worst case a person who enters the house in the second 105 will reach the last room in the second 105 +N ÿ1. Let's simulate the first MX seconds.

If we assume that the periods of activity and rest of a ghost are not altered by scaring a person, then a ghost with energy x will have O( ) active periods. Since the energies of the ghosts form a permutation, the total number $\frac{MX}{2x}$ of active ranks will be:

$$\frac{MX}{2\,ÿ\,1} \quad \frac{MX+}{2\,ÿ\,2} + \frac{MX}{23} + \cdots + \frac{MX}{2N} = (\frac{MX}{two})(1 \; \frac{}{one} + \frac{one}{} + \frac{one}{23} + \ldots + N\frac{}{one})$$

O(MX ÿ log(N)) active ranges in total. $\frac{1}{two}$ $\frac{1}{3}$ 1 1 + + 1 + ... + N is known as the harmonic sum, and tends to O(log(N)), so we will have The sum

What about the fact that the ghost has to rest right after scaring someone? Note that this will add an additional range of rest and then one of activity. So the total number of active ranges will be O(MX ÿlog(N)+M) in the worst case. If for each active range we can optimally find if someone is scared away, then the problem can be solved in time.

Let's process the ghosts from i = 0 to i = N ÿ 1 and carry a BST (can be std::set in C++) with the arrival time of all active people (who have not been scared away so far).

For ghost i, let's simulate all active ranks of this ghost starting from second 0. To find out if this ghost i scares someone in the active range that starts at second x, let's search the BST if there is any active person that arrived at any second of the range [x ÿ i, x ÿ i + pi), since a person who visits room i in the second T, would have to have entered the house in the second T ÿ i, and if there are multiple people who arrived in that range, we take the one that came first. This can be done in O(log(M)) with lower bound(x ÿ i) of a std::set in C+ +. If he scares someone off, let's write down his response, remove the person from the BST (since he leaves the house), and simulate the next active rank.

Complexity: O((MX ÿ log(N) + M) ÿ log(M)).

Integer Multiplicative Persistence

By: Juan Pablo Mar´ÿn

For this problem we would have to perform the operation as indicated in the description of the problem. To do the multiplication of each digit, it can be done by obtaining each digit as the remainder of dividing the number by 10 and then removing it by dividing the number by 10, until there are no more as digits to remove.

This process is repeated until the resulting multiplication has only 1 digit and the number of times the operation had to be performed is returned.

Machine Translated by Google

John in the Amusement Park

By : Eddy Ram´ÿrez

This problem can be solved by using DP. At the moment the park opens time = 0 John must make the decision to participate or not participate in any of the attractions. Assuming that John decides to participate in the attraction i starting at time ti,j , then John will have received a total happiness of hi at time ti,j + di, and will find himself having to take the same decision that he had at the beginning, what attraction of those that he can participate that initiates in ti,j + di would help him to obtain the maximum happiness? We can then observe that there is a recurrence relation for the maximum happiness H(time) that John can obtain that depends on the attractions in which he can participate that start at or after time, that is, in the attractions that have some ti,j ÿ time, this relation is then given by: H(time) = max(H(ti,j + di) + hi)

So if this recurrence relation is implemented, we look for the value of H(0), the maximum that John can get since he opens the park. Maintaining memorization at each of the possible times, it will take O(T) of memory, and O(S two) in computation, where S = PN you i=1

K-Binary Repetitive Numbers

By : Juan Pablo Mar´ÿn, Abraham Mac´ías, Alan Enrique Ontiveros

The idea of solution is based on the principle of inclusion and exclusion. The important observation is that a string of size K can only be generated by smaller strings of size d that divide K.
So if we have d divides K the 2d strings of size d can be concatenated K/d times to get a string of size K that is K-binary repetitive. However, if we count all those strings we could be counting too many strings. If a string S is K-binary repetitive and it is formed by a string of size d which in turn is d-binary repetitive, then we will be counting S as many times as divisors d has. Consider then only strings of length d = K/p such that p is a prime number that divides K, so all 2d strings are NOT K-binary repetitive. Then we can count all the strings that are not K-binary repetitive using the inclusion exclusion theorem with all the subsets that can be generated from the prime factors p that divide K.

The number of K-binary repetitive strings that there are will be subtract the non-K-binary repetitive strings from the possible 2K strings, and this is equal to:

(k)
K2 2K / Qk=1 pike
– ÿwX _ (ÿ1)k X
k=0 1ÿi1<···<ikÿw(K)

Which has complexity O(2w(K)) where w(K) is the number of prime factors of K.
In order to answer the questions efficiently, the prime factors of each must be precomputed.
possible N, this can be done with an O(N) sieve.

30 de Octubre
2021
Gran Premio
International Collegiate Programming Contest
icpc
icpc.foundation
Maratona SBC de Programação
Región México
ITESO, Universidad Jesuita de Guadalajara
TAP
Torneo Argentino de Programacion
icpc Centroamérica

You can find the Upsolving session, with the proposal on how to approach the solution of problems, on the Brazilian Programming Marathon channel: https://www.youtube.com/watch?v=wYdb8YWd87Q

To exercise the solutions, you can use the test cases (input / output) that are shared on the following site: http://maratona.sbc.org.br/primfase21.html