Problem Tutorial: "Missing Letters"

There are many ways to solve this problem; the simplest one is the greedy algorithm.

Note that it only makes sense to replace question marks with letters from initials. Let's iterate over the characters of the string from left to right. If we encounter a question mark, then if the previous character is the first letter of the initials, we put the second one, if not, then we put the first one.

Why is this true? Suppose we were mistaken and in the optimal answer, instead of the second letter, we had to put the first one, then we ruined one good substring, but added another, so the total number of good substrings remained the same.

Problem Tutorial: "Julia and Flower Beds"

It turns out the minimum difference is always 0 or 1 (it's 0 if the total number of flowers is even and 1 otherwise). There are multiple greedy strategies that achieve the minimum difference. Let us explain one of them.

We will iterate in reverse order through $i = n,\ n-1,\ \dots,\ 1$, and maintain the current number of flowers in each flower bed (initially, there are 0 flowers in each flower bed). Then, we will plant all flowers of the current type i on the flower bed with fewer flowers. Let's prove that after the i-th iteration, the difference does not exceed i. In the end, this difference will not exceed 1.

We can prove it by induction. Base case: after $i = n$ the difference is $a_n \le n$. If the difference does not exceed i after $i > 1$, then, on the next step, we will add at most than $i-1$ to the smallest of two numbers, so the new difference will not exceed $i-1$.

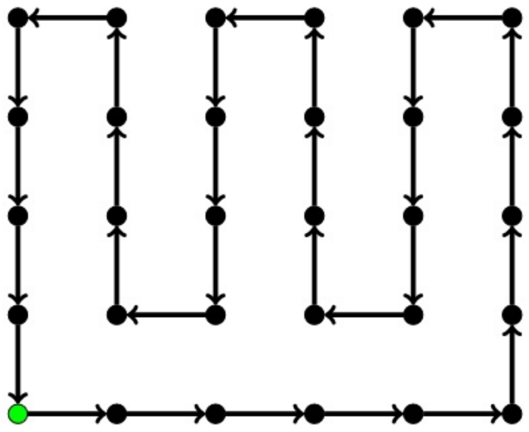
Problem Tutorial: "Divisor Circle"

Let $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$, where p_i are distinct primes. Then any divisor $d = p_1^{\beta_1} p_2^{\beta_2} \dots p_k^{\beta_k}$ can be represented as a point $(\beta_1, \beta_2, \dots, \beta_k)$ in k-dimensional space. So this problem consists of finding a cycle through as many points as possible so that any two adjacent ones are at a distance of 1 from each other.

If $n = p^\alpha$, we have a one-dimensional case, and we cannot put more than two divisors.

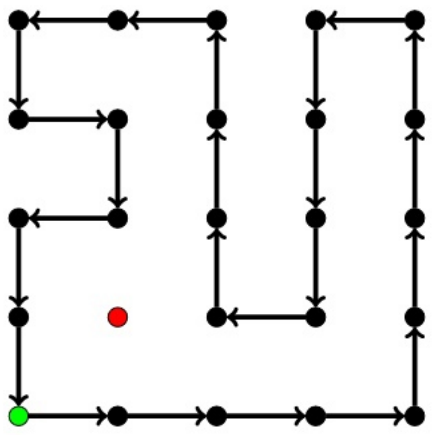
Otherwise, we have at least two dimensions. It turns out, is always possible to construct a cycle from all points or without one. It is clear that if the total number of points is odd, we will not be able to visit all of them: the parity of the sum of all coordinates changes every step.

Let's look at the 2D case first for simplicity: points form a rectangle. A cycle can be done, for example, as follows:



First, we walk through the rectangle along the first dimension, then return in a "snake" shape. Moreover, in the case of an odd number of points, we made another "snake" shape in the end.

In the general case, similar construction works. Let's create a path (not necessarily cyclic) through all



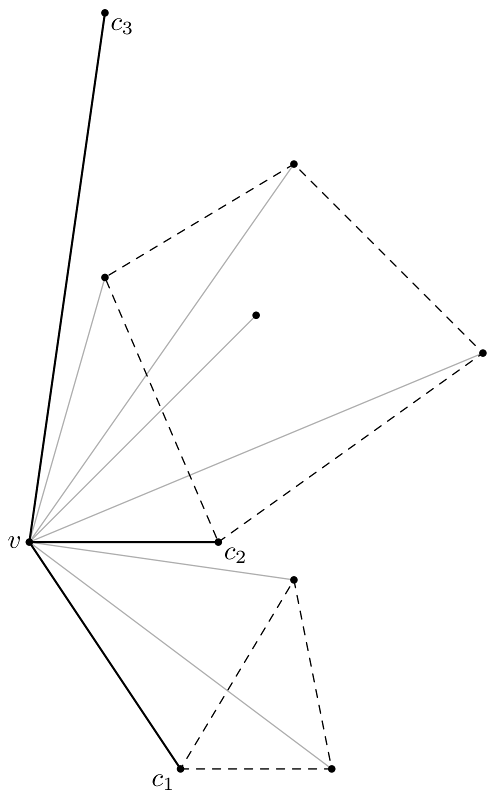
points without the last dimension and make the same walk in reverse order, adding the same "snake" in the last dimension. If the last dimension is even, then we will visit all points, otherwise, we will visit all points except for one.

Problem Tutorial: "Planar Tree"

Let's root the tree at any vertex. The subtree of b is a set of vertices u, such that v belongs to the shortest path from u to root. Let's denote the subtree of v as st_v.

We'll solve the problem recursively. Consider the subtree of v (including v). Our recursive function takes v and the set of points P ($|st_v| = |P|$) as arguments. The goal is to place st_v in points of P, with an additional restriction: v must be placed in a fixed point $p \in P$ (granted that p belongs to the convex hull of P). Of course, the placement must be valid: no two edges can intersect.

Let's remove p from P and sort all remaining points of P by their polar angle using p as the origin. Denote children of v as c_1, c_2, …, c_k. We will place the subtree of c_1 at points $P[1 \ldots |st_{c_1}|]$, while placing c_1 at $P[1]$. Then, we will place the subtree c_2 at points $P[|st_{c_1}| + 1 \ldots |st_{c_1}| + |st_{c_2}|]$, placing c_2 at $P[|st_{c_1}| + 1]$. Repeat for all children of v.



Notice that edges (v, c_1), (v, c_2), $\ldots$, (v, c_k) don't intersect (since they all share a vertex) and don't intersect with edges inside subtrees of c_1, c_2, $\ldots$, c_k. By induction, this procedure builds a correct answer.

To solve the problem, we call this function for the root of the tree and the set of all given points. We can place the root at any vertex on the convex hull; for example, we can choose the lexicographically minimum pair (x, y).

The time complexity is $O(n^2 \cdot \log n)$.

Problem Tutorial: "Redundant Binary Representations"

First, let's figure out how we can calculate $a(n)$ for a given n.

Let n be odd, $n = 2k + 1$. Every representation of n contains exactly one 1 (you can see it for $n = 21$ in the problem statement). If we remove this 1 and divide all remaining terms by 2, we get a representation of k. In conclusion, $a(2k + 1) = a(k)$.

Otherwise, let n be even and $n = 2k + 2$. The number of 1 in redudant binary representation of n is even. If we remove these zero or two 1's and divide all remaining terms by 2, we get a representation of k or $k + 1$. We get another recurrent equation: $a(2k + 2) = a(k) + a(k + 1)$.

Let's look at the pair of consecutive values $x = a(n)$ and $y = a(n + 1)$. The pair $(n, n + 1)$ has exactly one even and one odd number.

Let $n = 2k + 1$. Then $x = a(2k + 1) = a(k)$, and $y = a(2k + 2) = a(k) + a(k + 1)$. Since $a(k) > 0$, then $x < y$. Similarly, if $n = 2k + 2$, then $x = a(2k + 2) = a(k) + a(k + 1)$, and $y = a(2k + 3) = a(k + 1)$. In this case, $x > y$.

Turns out, the maximum of x and y corresponds to an even n. Suppose that $x < y$. We already know that $x = a(2k + 1) = a(k)$, $y = a(2k + 2) = a(k) + a(k + 1)$. We can express $a(k) = x$ and $a(k + 1) = y - x$. Now, if we recursively solve our problem for a $(x, y - x)$ pair, then, to go to (x, y), we just move from k to $2k + 1$.

In other case, if $x > y$, then we apply this solution recursively to $(x - y, y)$, and after that we go from k to $2k + 2$.

This process might be familiar: Euclid's algorithm with subtractions does the same. In the end, if we arrive at $x = 1$, $y = 1$, then the answer is $n = 0$; otherwise, there is no solution (this happens if x and y are not coprime). Euclid's algorithm with subtractions works in $O(x + y)$ time and passed all subtasks except for the last one.

To optimize this solution, we need to use a faster version of Euclid's algorithm with remainders. To do that, we need to know how to transition from $(x, y \bmod x)$ to (x, y). Let $r = \lfloor \frac{y-1}{x} \rfloor$. Performing r subtractions means that we apply $n \to 2n + 1$ operation r times for some initial n. In binary, this operation is a composition of a left shift and adding one. We if do it r times, we get $n \cdot 2^r + (2^r - 1)$. The transition from $(x \bmod y, y)$ to (x, y) is similar. It's easy to prove that this solutions works in $O(\log^2(x + y))$, but it's also possible to prove it works in $O(\log(x + y))$ time.