Problem A. Magician

Let's make a mapping of each color c with some other integer M_c. For the mapping, we are going to use only the numbers with an even number of *set bits* in their binary representation. For example, 3, 5, 6, etc. *Note that there are more than* 500 000 *of such numbers between* 0 *and* $2^{20} - 1$.

Gnome i will do the following in phase 1:

- Let X be the XOR of all the mapped colors of other gnomes. More formally, $X = \bigoplus_{j=1}^{N} M_{C_j}$.
- Then, gnome i will write the i-th bit of X in his card.

Gnome i will do the following in phase 2:

- Let's find the mapped value corresponding to our color. Then we can use that to find the actual color.
- The j ($j \neq i$)-th bit of the mapped value is the j-th bit of $(A_j \cdot 2^j) \oplus M_{C_j} \oplus (M_{C_1} \oplus M_{C_2} \oplus \ldots \oplus M_{C_N})$.
- Since all mapped values have an even number of set bits, we can restore the i-th bit (the last one).

Problem B. Word Generator

Let's first solve the easier version of the problem. Each slot i has the same number of cards, i.e., $|S_1| = |S_2| = \ldots = |S_M| = K$ for some $K > 0$. Let's form an alphabet consisting only of the first M characters, and every character has occurred an equal number of times (exactly K times). Let's prove by induction that we can always create exactly K words.

- Base case, $K = 1$: as every character is different, it is a valid word.
- $K > 1$: assuming that if we can create one word, then we can create all K. Let's create a bipartite graph, with slots on the left side, and characters on the right side. One needs to find a perfect matching in this graph. Using Hall's theorem, we can prove that it is always possible.

Now, let's solve the problem only for the version $|S_1| = |S_2| = \ldots = |S_M| = K$ (i.e., all characters are allowed, and they do not have to occur the same number of times).

Observation 1. Let c_x ($\texttt{a} \leq x \leq \texttt{z}$) be the number of occurrences of letter x inside all the slots. Then, there exists a way to create exactly K words if and only if $c_x \leq K$ holds for all x.

The statement can be proven by induction, and the construction is made by Kuhn's algorithm.

Now, let's try to find the maximum number of beautiful words that can be created. Let's find the set of cards F_i, which will be used for each slot i.

Let's use binary search on the value K. Construct a flow-graph in the following manner:

- Add an edge from the source to each slot i ($1 \leq i \leq M$) with a capacity of K.
- Add an edge from each character j ($\texttt{a} \leq j \leq \texttt{z}$) to the sink with a capacity of K.
- Add an edge from each slot i to each character j with a capacity of $c_{i,j}$, where $c_{i,j}$ is the number of cards in the slot i with character j written on them.

It is possible to construct exactly K words if and only if the maximum flow in this graph is equal to $K \cdot M$.

Problem C. Glitch

Let's enumerate odd indices from 1 to $\frac{N+1}{2}$.

Observation 1. There is always an answer when N is odd, and it can be proven by induction.

Now, let's do binary search. Let $l = 1, r = \frac{N+1}{2}$. Do the following until $l + 1 < r$:

- Let $m = \lfloor \frac{l+r}{2} \rfloor$
- If $b_{2 \cdot m-1} = a_{2 \cdot m}$, then $l = m$; otherwise, $r = m$.

At the end, the answer will be $l \cdot 2 - 1, l \cdot 2 + 1$.

Problem D. Electi Lamps

For each lamp i, let L_i be the leftmost lamp to which one can reach if all the lamps from the right side are destroyed and all the lamps from the left side are turned off.

Similarly, let R_i be the rightmost lamp to which one can reach if all lamps from the left side are destroyed while all the lamps on the right side are in their initial states.

Observation 1. Let's assume that all the lamps before i are destroyed. You can walk from lamp i to lamp j while turning on every lamp on the way; then you can come back to lamp i. After, you can walk again to lamp j, but this time, turn off every lamp on the way. This 3-way journey can be done if and only if $L_j \leq i \quad \& \quad j \leq R_i$.

Arrays L and R can be computed using a stack in $O(N)$.

Now, we can use DP. For now, let's work only with indices where $F(i) \neq 0$, i.e. $R_i = N, L_N \leq i$. $F(i)$ $(L_N \leq i < N)$ can be computed as the minimum of the following:

- $F(i+1) + |X_{i+1} - X_i|$: If segment $[X_{i+1}, X_i]$ is covered by some lamp, which is turned on from the right side.
- $F(j) + 3 \cdot |X_j - X_i|$: For every index j, where the following $i < j \leq N, L_j \leq i$ holds.

There are $O(N^2)$ possible transitions of the second type, which is too slow.

Observation 2. It is always optimal to use only the minimum index j, where the following $i < j \leq N, L_j \leq i, R_j = N$ holds.

Using this observation, our DP can be optimized to $O(N)$, making the complexity $O(N)$ overall.

Problem E. XOR Again?

Let's reformulate the problem. Instead of making blocks, we are going to cut the array. There are exactly $N-1$ places $[i, i+1]$ $(1 \leq i < N)$, where we can cut the array.

For each bit b, let's find the indices of the elements where the b-th bit is set. Let the indices be $v_1, v_2, \ldots, v_k$. If k is odd, then no matter how we make the division, the b-th bit of the answer will always be set. For even k, the b-th bit of the answer will be set if there's a cut in one of the following segments $[v_1, v_2), [v_3, v_4), \ldots, [v_{k-1}, v_k)$. Thus, let's create another array B of size $N-1$, where B_i indicates which bits on the answer would be set if there is a cut in $[i, i+1]$.

Now, we need to find for each k $(0 \leq k \leq N-1)$: What is the minimum bitwise OR of the elements if we choose exactly k elements from B? This can be done using SOS DP.

Problem F. Binary Permutation

```
int solve(vector<int> A) {
    int N = A.size();
    long long res = 1;
    for (int i = 0; i < N; i++) {
        if (A[i] == 1) {
            cnt += 1;
        } else {
            res = res * cnt % mod;
```

```
            cnt -= 1;
        }
    }
    reverse(A, A + N);
    if (N % 2) {
        N -= 1;
    }
    cnt = 0;
    for (int i = 0; i < N; i++) {
        if (A[i] == 0) {
            cnt += 1;
        } else {
            res = res * cnt % mod;
            cnt -= 1;
        }
    }
    return res;
}
```

Problem G. Amoeba Tree

Let's choose some vertex to be the root. For each amoeba i, we will introduce its head h_i to be the highest vertex which it occupies (the vertex with minimum depth).

Now, let's solve the problem without any queries.

Observation 1. If we sort all amoebas by non-increasing order of their heads' depth, then shooting the head of the first amoeba that's still alive in this order, until they're all dead, results in the minimum number of shots.

When queries are added, either the answer doesn't change, or the number of shots is increased by one. Let's see how the original simulation would be affected when a new amoeba is added.

- The new amoeba's body is covered by a vertex that was already shot. Then the answer would not change.
- The new amoeba's body is not covered by the vertices that were shot. This means that in our greedy algorithm, we would shoot its head h_i, and we are interested in what would happen after. Now, let's find the first vertex from its parent P, which we shot in our initial greedy algorithm. Now, one needs to find all amoebas which were killed in vertex P and are not killed by shooting at vertex h_i. Let the head of the amoeba with the maximum depth among them be G. If $G = P$ or if G or P don't exist, then it means the answer is increased by one for this query. Otherwise, we are going to solve recursively for G (i.e., what would happen if we would shoot at vertex G).

Note that we don't have to recalculate everything for each query. It can be pre-calculated, and all the queries can be answered in $O(l_i)$.

Problem H. Magical Puzzles

Let's first solve the problem for $k_i = 2$. For each puzzle, we can generate a random integer. Now, let's define the value of each edge. The value of an edge is the XOR of all random integers generated for the items of the puzzles that are located on this edge.

A path is beautiful if and only if the XOR of all edge values along the path is equal to 0. If f_x is the XOR of all values from the root to vertex x, then the path (x, y) is beautiful if and only if $f_x = f_y$. Therefore, we group all vertices that have the same f value and find the distance between each of them. This can be done using virtual trees.

This idea can be generalized for $k_i > 2$, and it is left as an exercise for the reader.

Problem I. Bobs Rating

Sort all edges in non-decreasing order of W_i, and create a DSU-tree (`https://codeforces.com/blog/entry/85714`) from it. Note that while merging two different components X and Y, we can introduce a new variable, R_X, which represents "*What is the minimum rating Bob should have in order to pass the edge from component X to Y for free.*"More formally, $R_X = \max(0, W_i - A_X)$, where A_X is the sum of all the free ratings in component X.

Now, let's solve one query for reaching vertex T from vertex S with an initial rating of R_0. Let $P = \text{LCA}(S, T)$ in the DSU-Tree. Then, one can notice that we should be moving from vertex S to vertex P in the DSU-Tree by adjusting the initial rating to satisfy the values of R_X. If $R_0 \leq R_S$, then we can move to the parent of S; otherwise, we have to buy $R_S - R_0$ rating for C_S coins and only then move to its parent. Note that we should adjust the value of R_0.

This solution works in $O(N)$ for one query in the worst case, but it can be optimized to $O(\log(N))$ using binary lifting. For each vertex X in the DSU-Tree, we can find the first vertex where we will have to adjust (update) the initial rating after updating it in the vertex X. More formally, we have to find the first vertex P, which is the ancestor of X and $R_P > R_X$. Using this information, we can optimize the above solution to $O(\log(N))$.

Problem J. AND Components

We can use a segment tree, where in each node representing some range $[L, R]$, we are going to store the following information:

- For each bit b, the first (minimum) index $i \in [L, R]$ where the b-th bit of A_i is set.
- For each bit b, the last (maximum) index $i \in [L, R]$ where the b-th bit of A_i is set.
- Information about their connectivity (which indices belong to which components).
- Information on whether each pair of indices i and j has already been found (i.e., whether an edge has already been added or not).
- The number of connected components.

This information is sufficient to recalculate everything with a total complexity of $O(N \cdot \log(N) \cdot \log(10^6))$.

Problem K. Hidden Digits

For $N > 2$, the answer is always `1`. For $N \leq 2$, we can query the entire hidden string and find the answer naively.

Problem L. Robots

One possible construction that works:

Let's root the tree and solve each subtree recursively. Every tree consisting of more than one vertex will be described as follows:

- We are going to use only two types of robots: Green (Type B) and Blue (Type D).
- The types of vertices alternate based on their depth. That is, all vertices located at even depths have Type B, while those at odd depths have Type A.
- The grid will be covered by blocked cells, with exactly one empty cell on its perimeter.

- Only the root can exit (i.e., can reach the empty cell on the perimeter), while all other vertices cannot exit from within.

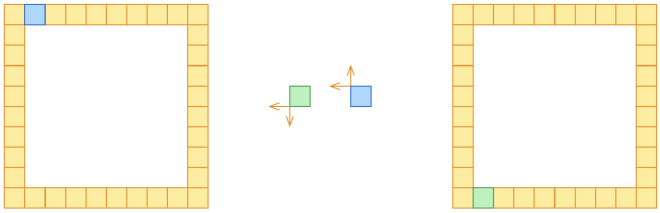

Figure 1. Structure of the trees.

Now, let's look at how we construct trees. The base cases are trees where the root has only leaves beneath it.

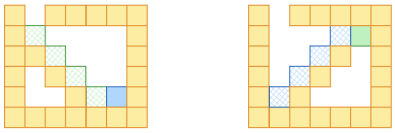

Figure 2. Trees where every vertex is a leaf except for the root.

For different cases, let's divide the vertices connected to the root into two sets: A — vertices that are leaves, and B — vertices that are not leaves.

We can recursively solve each subtree of B and then place them in the following way.

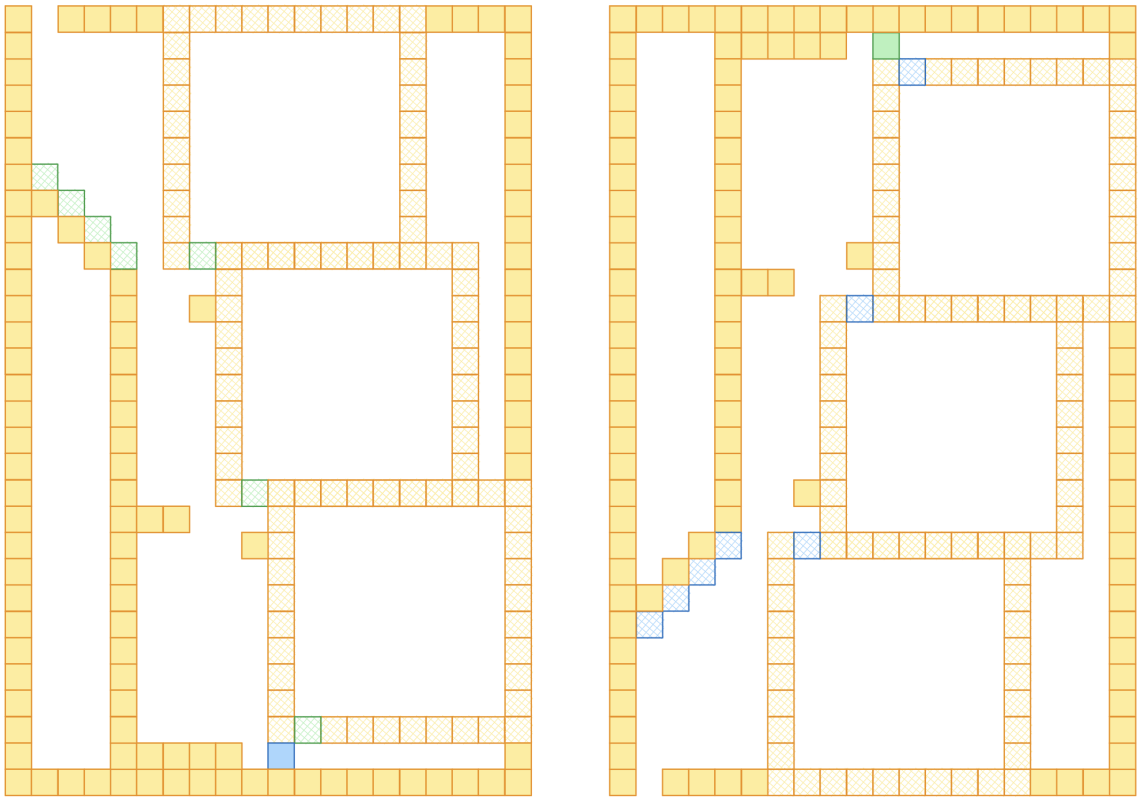

Figure 3. Complex trees.

Notes:

- When we are merging non-complex trees, we must sort them by non-increasing order of their grid sizes.
- Only some cells can be chosen as root to allow it to fit in $(N+1) \cdot (N+1)$.