NAC, Petrozavodsk Summer 2021 Editorial

Ildar Gainullin

August 28, 2021

1 A

In this problem we were given an array of numbers $a_1, a_2, \ldots, a_n$, and the goal was to find a permutation $b_1, b_2, \ldots, b_n$ with $a_i \& b_i = 0$

The property of our array described in the statement means that for any number $x \in \{a_1, a_2, \ldots, a_n\}$, all its submasks are also in a.

We will solve this problem using recursive approach (inductive proof follows from it):

- If all numbers in the array are zeroes, then any permutation works.
- Otherwise, let i be any bit such that $a_j \& 2^i \neq 0$ for some j.

  We will divide our array into two parts: $x_1, x_2, \ldots, x_l$ and $y_1, y_2, \ldots, y_r$, numbers in x have bit i turned on and numbers in y have it turned off.

  Note that for any $X \in x$, there is a number $X - 2^i \in y$.

  Recursively let's find any solution for $\{x_1 - 2^i, x_2 - 2^i, \ldots x_l - 2^i\}$ and $\{y_1, y_2, \ldots y_r\}$, let's denote solutions for them as $X_1, X_2, \ldots, X_l$ and $Y_1, Y_2, \ldots, Y_R$

  To get the required permutation for $a_1, a_2, \ldots, a_n$, we can set X_i as a pair for x_i and $f(Y_i)$ as a pair for y_i, where $f(x) = x + 2^i$ if $x + 2^i \in \{x_1, x_2, \ldots, x_l\}$ and $f(x) = x$, otherwise.

2 B

To solve this problem, we can use **power diagram**.

We can build it in $O(n^2 \log)$ using halfplane intersection (read wiki for more information).

To answer each query, we need to find the sum of areas of intersections of $O(n)$ polygons with circle, it can be done easily with well-written geometric template.

Total complexity is $O(n^2 \log + nq)$

3 C

We can solve this problem with binary search. If we know the size k of a square that we need to check, at first we need to find all possible left corners for our square, check can be done in $O(nm)$ in total with partial sums. After that it is enough to check that all these cells are in one connected component, we can do it with BFS.

4 D

We can sort the array to assume that $a_1 \leq a_2 \ldots \leq a_n$

After that this problem can be solved with naive dynamic programming, $dp_{i,j,k}$ – number of possible sets if two last elements in it are (i, j) and we took k elements.

Total complexity is $O(n^3 k)$.

5 E

Let's transform the string $s_1, s_2, \ldots, s_n \to s_1, s_n, s_2, s_{n-2}, \ldots$ (elements at the even positions are going from the beginning of the string, and elements at the odd positions from the end)

Then it can be shown that Ketek decomposition of the string s is equivalent to the decomposition of some prefix of our newly obtained string into even-length palindromes.

We can use naive dp_n to count the number of ways, to calculate it in $O(n^2)$ we can start expanding our palindromes from all possible middle positions in order.

6 F

Let's divide our array into monotone parts, then we can naively try to find the answer starting from the elements in-between parts. Total complexity is bounded by the sum of lengths of all parts, so it is $O(n)$.

7 G

Let $f(x, s)$ be the length of the array if we start from number x and do s steps.

If $s = 0$, $f(x, s) = 1$, otherwise, $f(x, s) = f(x - 1, s) + f(x, s - 1)$, so we can see that for $s \geq 3$, $f(x, s)$ is $O(x^3)$.

To solve this problem for $s \leq 1$ or $s \leq 2$, we can reduce it to two-dimensional queries, and they can be solved in $O(n \log n)$

In case $s > 3$, we will maintain current pair (x, s), iterate over all elements $\leq x$ in order to see what part goes first into prefix (we will need fast $f(x, s)$ calculation inside, it can be done in $O(1)$ with binomial coefficients) and in at most $a^{\frac{1}{3}}$ we will reduce s by one, eventually reducing the problem to $s \leq 2$.

Total complexity is $O((n + q) \log + sC^{\frac{1}{3}} q)$.

8 H

At first, we can get rid of the cases where we have connected components of special edges which are not path (print the cycle or delete all other components).

After that, we can compress all paths into one edge (deleting all internal vertices)

For each non-interesting edge $u \to v$, let's create one separate copy for u and v, and connect them with an edge.

For each vertex v, create $\deg(v)$ copies of v and connect them to all copies from the edges. If this vertex does not lie in an interesting edge, connect first two newly created copies.

For each interesting edge $u \to v$, connect one newly created copy of u with one newly created copy of v.

9 I

This problem can be solved greedily using some facts about matroids. We can greedily consider edges one by one, and check if this edge can be present in the answer. To check we need to check if there is a bipartite matching between connected components and the guards.

Complexity of this solution is $O(n^3)$, as one can show that it is enough to consider only MST edges.

There is also a second, MCMF solution. See video editorials for more details about it.

10 J

For each pair of letters (a, b), let's find the cost of $ab + ba$, if both runes exist, and add the edge $a \to b$ with this weight to the graph. After that, we can run $dp_{c,k}$ to find the minimum cost of a walk of length k ending at c. Answer from this dp can be found then from the parity of k.

11 K

Note that for each tuple (x_1, x_2, y_1) with $x_1 \neq x_2$, there is only one y_2, such that (x_1, x_2, y_1, y_2) is a losing position. Naively, we can already get $O(n^4)$ solution. We can optimize it with bitset, we just need to find the bitset of all losing y_2's for (x_1, x_2, y_1) and use Findfirst() functon. We can find this bitset by storing some partial sum-like information in bitsets for previous triples.

12 L

Let d_v be the shortest path length from 1 to v.

At first, let's get rid of all the edges (u, v, w) where $d_v \neq d_u + w$.

After that, we can get rid of all the edges (u, v), such that there is a number t in input with $d_u < t < d_v$.

If there is still a path between 1 and n in this graph, we can print it. Otherwise, print 0.

13 M

This problem can be solved with naive implementation. No additional comments are possible.