Problem A. Where is the legend?

- $n \le 15$, $S \le 400$. 14 points.

  Any recursive solution.

- $a_i = i$. 13 points.

  Observe that after each remove difference between adjacent elements equals the power of two and any such sequences starting with 1 and ending with n is possible. We can greedily construct such sequences or just take the number of bits in n.

- $a_i \le 3$. 9 points.

  Approach *remove if we can* works.

- $n \le 300, S \le 1000$, 17 points.

  Let's calculate $dp_{l,r}$: if there is a way to remove all elements from $l+1$ to $r-1$. $dp_{i,i+1} = true$ and $dp_{l,r} = true$ if there exist m $(l < m < r)$ such that $dp_{l,m}$, $dp_{m,r}$ and $a_m = \frac{a_l + a_r}{2}$.

  And calculate d_i — minimal length of a prefix after performing operations.

  The answer to the problem will be d_n.

- $n \le 3000, S \le 10000$, 18 points.

  Observe that if $a_l < a_r$ and $dp_{l,r} = true$ than $a_l < a_{l+1} < .. < a_{r-1} < a_r$. Hence there are only one m such that $a_m = \frac{a_l + a_r}{2}$. Find it using two pointers or binary search. $a_l > a_r$ similar.

- No additional constraints. 29 points.

  Another observation it's for fixed l there are only $log_2 10^9$ possible r's cause value $a_r - a_l$ doubled after each r .

Problem B. Zhylan.io

- $n, q \le 500$, 7 points.

  Optimal to eat the smallest snake first.

  For each query, for each snake in query check can it eat all others. Sort other snakes and eat them one by one from the smallest.

- $n, q \le 3000$, 15 points.

  If a snake with length x can eat all others then snakes with length $\ge x$ also can.

  For each query, sort snakes and binary search the smallest snake that can win.

- $a_1 \le a_2 \le \ldots \le a_n$ 24 points.

  i-th snake can eat snake j if:

  - for $j < i$: $a_i + a_1 + a_2 + \ldots + a_{j-1} \ge a_j + k$ hence $a_i - k \ge a_j - pref_{j-1}$
  - for $j > i$: $a_1 + a_2 + \ldots + a_{j-1} \ge a_j + k$ hence $-k \ge a_j - pref_{j-1}$

  where $pref_i$ is sum of first i snakes

  Value of $a_j - pref_{j-1}$ on segment will be simply $a_j - pref_{j-1} + pref_{l-1}$. Store in some data stucture like segment tree minimal $a_j - pref_{j-1}$ on segment.

  For each query, binary search the smallest snake and check it with values from the data structure.

- $n, q \leq 5 \cdot 10^4, a_i \leq 10^6$, 20 points.

  Assume we have a sorted array. Let's find minimal $a_j - pref_{j-1}$.

  $a_j - pref_{j-1} \geq a_{j+1} - pref_j$ hence $2 * a_j \geq a_{j+1}$. There are at most $log_2 10^9$ such positions.

  Answer queries with *mo's algorithm.* Store sorted values of interval in some data structure and keep elements more than previous two times. For each query, binary search answer and obtain the above positions from data structure.

- $n, q \leq 10^5$, 19 points.

  Use data structures that can fast obtain such positions on segment. (e.g. Persistent segment tree, Wavelet tree)

- No additional constraints. 15 points.

  Some optimizations if you need.

Problem C. Mansur vs Tima

- $S \leq 10, k \leq 4$, 6 points.

  Any bruteforce solution.

- $S \leq 50000, k = 1$, 10 points.

  minimal xor ovel all pairs in array = minimal xor over all adjacent pairs in sorted array

- $S \leq 1000, k \leq 2$, 10 points.

  Binary search answer. Build graph with edges between i and j if $a_i\ xor\ a_i \geq x$. Color graph with two color.

- $S \leq 50000, k \leq 2$, 20 points.

  Find Minimal Spaning Tree of graph with edges between i and j with weight $a_i\ xor\ a_i$. And color it with two color

  Proof of correct left for readers

  MST can be obtained by *Boruvka's algorithm.*

- $S \leq 50000, k \leq 4$, 22 points.

  If for the fixed prefix(e.g. 1011???) number of matchings $\leq 2 * k$ elements. Binary search answer. Build graph with edges between i and j if $a_i\ xor\ a_i \geq x$. Color graph with k color. Since k is very small any fast coloring algorithm enough(e.g. O($3^{2*k} * k$).

  If for the fixed prefix(e.g. 1011???) number of matchings $> 2 * k$ elements answer will be less than 2^s where s is the count of undefined bits. Hence we can divide them into two groups and solve them separately(e.g. 10111?? and 10110??).

- No additional constraints. 32 points.

  For the fixed prefix(e.g. 1011???) group matchings by next bit(e.g. 10111?? and 10110??).

  If the size of both groups at most k. Build bipartite graph with the edges from elements of the first group to the second group with weight xor of their values. Binary search answer. Optimal coloring can be obtained from maximal bipartite edge matching.

  Otherwise, solve for groups separately.

  $O(nk^2 log_2 10^9)$

Problem D. Streets in Kaskelen

- $S \leq 10$, $T \leq 10^4$, no queries of second and third type. 12 points. Build graph and check reachibility(e.g. DFS, BFS)

  Any bruteforce solution.

- $S \leq 80$, $T \leq 2 \cdot 10^5$, no queries of second and third type, 15 points.

  Build graph and build reachability matrix(e.g. DFS, BFS) to answer the queries in $O(1)$.

- $a_1 = a_2 = \ldots = a_n$, no queries of second type, 14 points.

  Assume all horizontal lines go right and a vertical line at v go up. All nodes up and right reachable from v. Find first down edge on right side of v. All nodes down v and right this line is also reachable. All other nodes are not reachable. All other cases are similar.

- $S, T \leq 1000$, no queries of second and third type, 16 points.

  TODO

- $S, T \leq 50000$, no queries of second and third type, 22 points.

  Subtask for any other but not optimal ways to calc answer.

- No additional constraints. 21 points.

  The only case when we use the same direction more than once is if it passes through the coordinates of the node in the query. Hence we can divide our grid into 9(3x3) parts and for each direction check if there are edges in this part.

  To make it easy to code take any 0-line and any 1-line in every three parts for x and for y. Build a graph with $(3+3+2)^2 = 64$ (3 0-line and 3 1-line plus v, u line) nodes and check reachability.

Problem E. Challenges of urban planning

- $S \leq 500$, 7 points.

  Fix a, b and calc distances.

- $(u_i, v_i) = (i, i+1)$ for all $1 \leq i \leq n-1$, $p = 3$, 6 points.

  For fixed a b will be on big part and $\frac{2}{3}$ distance to the end.

- $S \leq 4000$, 15 points.

  Consider path from a to b. There is edge such that all nodes from on side add to sum $dist(a, v)$ and another side nodes add $dist(b, v)$. Brute edge and calc sum of $dist(a, v)$ and take minimal sum of $dist(b, v)$.

- $p = 2$, 20 points.

  Distance from cut edge to a equal to the distance to b $+\{-1, 0, 1\}$. For b cut edges is uniquely defined. Just calc answer.

- $p = 1$, 22 points.

  For cut edge optimal a and b is centroids of components. Subtree centroid can be obtained by lifting the biggest subtree centroid. Uppertree centroid can be obtained by Small to Large technique with delete, add node and maintain centroid. But it is too hard to code. So let's reroot the tree on centroid and now uppertree centroid can be only in the biggest or in the second biggest subtree of root. Instead of lifting centroid in every node, precalc centroid by size of removed subtree. $O(n)$

- $S \leq 30000$ 21 points.

  For fixed cut edge and b, answer for a depends only on distance from cut edge to a. There are lot of ways to do it.(centroid + cht, lichao) $O(nlog^2n)$

- No additional constraints, 21 points

  Distance from cut edge to a equal to the distance to b $+\{-1, 0, 1\}$. So it became a standard centroid decomposition problem. $O(nlogn)$

Problem F. Green Line

- $n \le 100$, $a_i \le 100$ for all $1 \le i \le n$, 6 points.

  Just simulate the process.

- $n \le 100$, 21 points.

  Store event "when sign between a_i and a_{i+1} changes". There are $O(n^2)$ such events and every next event can be obtained in $O(n)$

  $O(n^3)$

- $n \le 2000$, 20 points.

  Actually if you store events in heap and write transitions like in Dijkstra it will be $O(n^2 logn)$

- No additional constraints. 53 points.

  Magic part: build another array b. Initially $b = a$. In next step b_i grows like a but only if $b_{i-1} > b_i$. And similarly build c by $c_{i+1} > c_i$. At every step for every i true $a_i = max(b_i, c_i)$. poof..