# Universal Cup
# Stage 15: Hangzhou
## Editorial

Zhejiang University

05.07.2023

# A. Turn on the Light

## Description

- There are $n$ $(1 \leq n \leq 10^6)$ lights, all initially turned off, and one hidden light $s$.
- Each time you can choose a light $x$ to turn on, and then you will be told the absolute difference between the number of lights turned on to the left and to the right of $s$.
- Find the hidden light using no more than $40$ queries.

# A. Turn on the Light

## Solution

- Consider binary search. Suppose the current interval is $[l, r]$, let $cl$ be the number of lights turned on in $[1, l-1]$, and let $cr$ be the number of lights turned on in $[r+1, n]$.
- Let $m = \lfloor \frac{l+r}{2} \rfloor$. After turning on light $m$, if $cl \neq cr$, we can determine which side of $m$ the hidden light is on, either $[l, m-1]$ or $[m+1, r]$, and recursively continue the search.
- So we only need to query $r$ when $cl = cr$, and we can determine which side of $r$ the hidden light is on, either $[l, r-1]$ or $r$, and recursively continue the search.
- The number of queries is about $2 \log_2 n$.

# B. Equation Discovering

## Description

- Given no more than 20 pairs of $(x, y)$, find an equation with complexity at most 9 to fit these data.
- The equation can contain: $+$, -, *, $/$, sin, cos, x, (, ).
- Complexity is defined as the number of binary operators multiplied by 2 plus the number of unary operators.

# B. Equation Discovering

## Solution

- Consider writing the final answer in the form of an expression tree. The complexity being no more than 9 means that the number of nodes in the expression tree is no more than 10.
- A brute-force analysis method is to consider the prefix traversal of the expression tree. Each element can be one of 7 symbols, so there are up to $7^{10} \approx 3 \times 10^8$ possible expression trees.
- In reality, there are much less than that. We can use dp to calculate that there are no more than $4 \times 10^5$ valid expression trees, which can be directly enumerated.

# B. Equation Discovering

## Solution

- The method of building the expression tree from bottom to top is to set $f_i$ as the set of expressions with a complexity of $i$, and then transition from $f_i$ to $f_{i+1}$ and from $f_i, f_j$ to $f_{i+j+2}$.
- This method is relatively easy to implement, and one tester provided code that is only 830 Bytes.
- Let $T(x)$ be the number of expressions with a complexity no more than $x$, the time complexity is$\Theta(nT(9))$.

# C. Puzzle: Kusabi

## Description

- Given a tree with three types of nodes, A, B, and C.
- Nodes A and B form pairs where the distance from A to the root is longer than the distance from B to the root.
- Nodes C match with other nodes C where the distances from both nodes to the root are equal.
- Matching two nodes occupies the path between them, and edges cannot be reused.
- Find a successful matching solution for all nodes, or output that it's impossible.

# C. Puzzle: Kusabi

## Solution

- Each node's subtree can only transmit one path towards the root, and whether this path is transmitted depends only on the parity of the number of key nodes in its subtree. Therefore, greedy approach can be applied by processing the nodes in the order of their distance from the leaves to the root.
- For each type of C node, handle each depth separately. If there are more A nodes than B nodes in the subtree, try to keep the deepest A node that can be kept; if there are more B nodes than A nodes, try to keep the shallowest B node that can be kept.
- If it's found that the number of nodes that need to be kept exceeds one or the matching between A and B fails, then the problem is unsolvable.
- It's impossible if the total number of special nodes is odd.
- The time complexity is $\Theta(n \log n)$.

# D. Master of Both III

## Description

- The cost of a set $S$ is defined as the minimum cost to make all elements in $S$ become 0 through the following operations:
- Choose a subset $T$ of $S$, add $x$ to each element in $T$ and take the result modulo $n$, with cost $w_x$.
- Find the total cost of all subsets of $0, 1, 2, \ldots, n-1$ multiplied by their subset number.

# D. Master of Both III

## Solution

- First, we transform the problem into choosing a set $S$ that can generate all numbers in set $T$, and minimize the sum of costs $\sum_{x \in S} w_x$.
- Let $f_S$ be the minimum cost to generate all numbers in set $S$. We can use bit manipulation to efficiently transfer the state by adding $x$ to the knapsack of set $S$.
- Then we find the minimum value of the superset of $S$, which gives the answer when $S$ is used.
- The time complexity is $\Theta(n2^n)$.

# E. Puzzle: Tapa

## Description

- Given a Tapa game, where each clue is a single digit number that is smaller than the number of cells around it by 0 or 1, and the clues are located on odd rows and columns.
- Find a solution or output "no solution".

# E. Puzzle: Tapa

## Solution

- Each clue can only have at most one empty cell adjacent to it. If an empty cell is diagonally adjacent to four clues, it can be shared by two pairs of clues. Thus, we can focus on matching clues in pairs.
- Even clues around the edge can be paired with other even clues. The outer edge clues and the central 7 cannot be matched; otherwise, the black cells around the clues will not form a continuous region. However, two central 7s can be paired.
- Solve the bipartite matching problem using any algorithm like Hungarian or Hopcroft-Karp.
- The time complexity is $\Theta((nm)^2))$ or $\Theta((nm)^{1.5})$ depending on the algorithm used.

# F. Classic: Classical Problem

## Description

- Given a set $S$, multiply all numbers in it by $c$ and take the modulus $p$.
- Find all possible values of $c$ that maximize the mex (minimum excluded value) of all numbers in the resulting set.

# F. Classic: Classical Problem

## Solution

- Special case: $0$ does not exist.
- First, find the primitive root $g$ of the prime number $p$, and convert each number $x$ to $I(x)$, where $x \equiv g^{I(x)} \pmod{p}$.
- Binary search for the answer $w$ and consider how to check it. It is only necessary to check whether there exists an integer $x$ such that $I(1) - x, I(2) - x, ..., I(w-1) - x$ are all in the set. Note that this should be taken modulo $p - 1$.
- Assuming the generating function of the set $S$ is $f(x)$ and the generating function of the set $1, 2, ..., I(w-1)$ is $g(x)$, calculate the cyclic convolution of $g(x)$ and $f(1/x)$, and check if the coefficient at position $w$ or greater is greater than or equal to $w$.
- Use FFT to calculate the convolution with time complexity of $\Theta(p log^2 p)$.

# G. Game: Celeste

## Description

- There are $n$ pillars, each with a number $a_i$ and located at position $x_i$.
- You can jump from one pillar to another, but the distance must be within $L$ and $R$.
- Jump from pillar 1 to n, and output the solution that has the largest lexicographical order after sorting all passed numbers from big to small.

# G. Game: Celeste

## Solution

- Consider dynamic programming. Let $f_i$ be the largest lexicographically sorted sequence of numbers obtained by jumping from 1 to i.
- Then we have transition $f_i = \max_{L \le x_i - x_j \le R} f_j + a_i$, where $+$ means inserting $a_i$ into the sorted sequence.
- Direct transitions have time complexity of $\Theta(n^3)$, which is difficult to pass.

# G. Game: Celeste

## Solution

- First, consider how to handle the constraint of $L \leq x_i - x_j \leq R$. It can be seen that $x_i - R \leq x_j \leq x_i - L$, and each point can be transferred to a continuously moving interval to the right.
- This is a classic sliding window problem, which can be solved by maintaining a monotonically increasing queue.
- At this time, the time complexity is $\Theta(n^2)$, so further optimization is considered.

# G. Game: Celeste

## Solution

- The bottleneck of the current complexity is comparing the sizes of two ordered sequences and inserting a number into a sequence.
- We can use a segment tree to maintain the occurrence frequency sequence of each number. Randomly assign a hash value to each weight value, and the hash value of each node is the sum of the hash values of all numbers in the node, which can be used to find the largest different position between two sequences in $\Theta(\log n)$ time on the segment tree.
- However, rebuilding the segment tree every time is too time-consuming, so we make the segment tree persistent, so that only $\Theta(\log n)$ nodes are changed each time.
- The time complexity is $\Theta(n \log n)$.

# H. Classic: N Real DNA Pots

## Description

- Given $n$ points on a plane, select $k$ points so that the maximum of the minimum slope between any two points is minimized.

# H. Classic: N Real DNA Pots

## Solution

- Use binary search to find the answer $w$. Two points $i < j$ can be selected if and only if $\frac{y_j - y_i}{x_j - x_i} \geq w$, which can be simplified to $y_i - wx_i \leq y_j - wx_j$.
- Treat the elements as $y_i - wx_i$, and the problem becomes finding the longest increasing subsequence of the array. Check if the length of the longest increasing subsequence is greater than or equal to $k$.
- Note that when using binary search, checking $r - l < \epsilon$ may lead to timeout due to precision issues. Instead, check relative error or iterate a fixed number of times.
- The time complexity is $\Theta(n \log n \log x)$.

# I. MEXimum Spanning Tree

## Description

- Given a graph, find the maximum MEX spanning tree.

- Consider any forest that contains no more than one edge of each weight, extra edges can be added to form a tree without decreasing the MEX weight.
- Therefore, intersect the colored matroid (at most one edge of each weight) and the graphic matroid (no cycles), adding edges in ascending order of weight.
- The time complexity is $\Theta(n^{2.5})$.
- Binary search followed by matroid intersection can also pass.

# J. Master of Polygon

## Description

- Given a simple polygon in a plane.
- Each time an inquiry is made about a given line segment, asking whether this line segment has a common point with the boundary of the simple polygon.

# J. Master of Polygon

## Solution

- Divide and conquer according to the abscissa, let $solve(l, r)$ represent the subproblem only considering the graph with abscissa in $[l, r]$ range, where $l < r$.
- Let the minimum and maximum values of the abscissas of the vertices of the polygon be $xl$ and $xr$ respectively. First remove all queries that do not pass through $[xl, xr]$ (obviously cannot intersect with the polygon), and then call $solve(xl, xr)$.
- Note that $l < r$ here, which may miss the case where both the query segment and the polygon edge are perpendicular to the $x$ axis, which needs to be handled separately.

# J. Master of Polygon

## Solution

- Assuming it is currently $solve(l, r)$, take $mid = \lfloor \frac{l+r}{2} \rfloor$, and make two vertical parallel lines $A : x = l, B : x = r$.
- Each edge of the polygon or query segment has one of the following three possibilities:
  - Intersection with both $A$ and $B$, which can be treated as a line and not continue to recursively solve the subproblem.
  - Intersection with the subproblem $solve(l, mid)$, recursive call this subproblem.
  - Intersection with the subproblem $solve(mid, r)$, recursive.

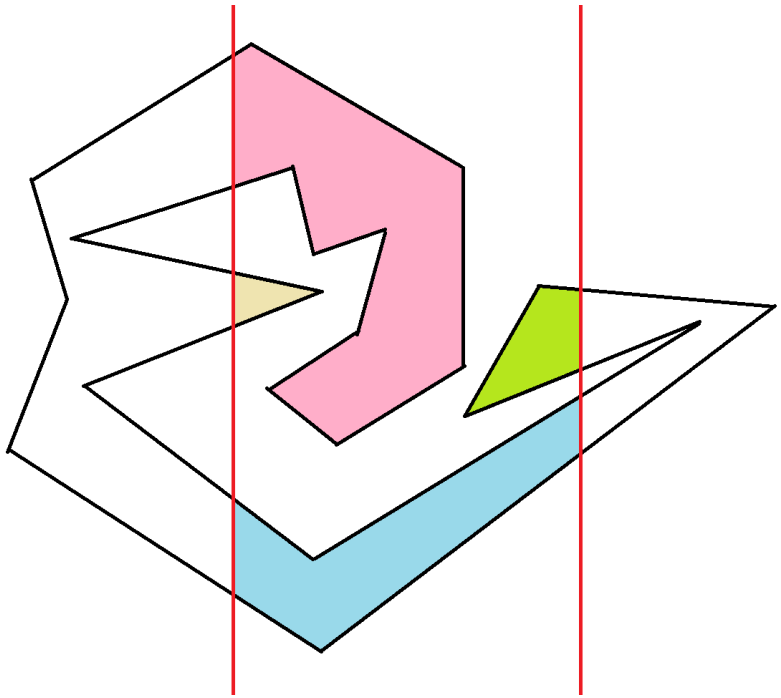
# J. Master of Polygon

## Solution

- Consider how to handle the case where polygon edges can be treated as lines.
- Sort all such edges in ascending order of their intersection point's Y-coordinate with line $A : x = l$. Then, for any $x \in [l, r]$, their relative order with respect to the Y-coordinates of their intersection points with line $x = x'$ remains unchanged.
- Traverse all query line segments and use binary search with complexity $O(\log n)$ to find the intersecting polygon edges.
- Next, consider how to handle the case where query line segments can be treated as lines.

# J. Master of Polygon

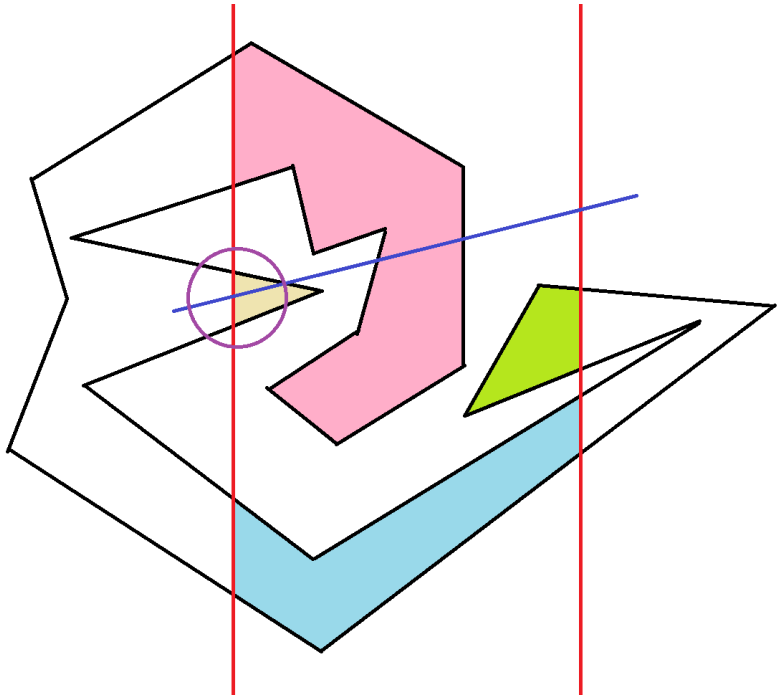
## Solution



- The portion of the polygon within $[l, r]$ is partitioned into several connected line segments, each of which intersects at least one of two parallel lines.

# J. Master of Polygon

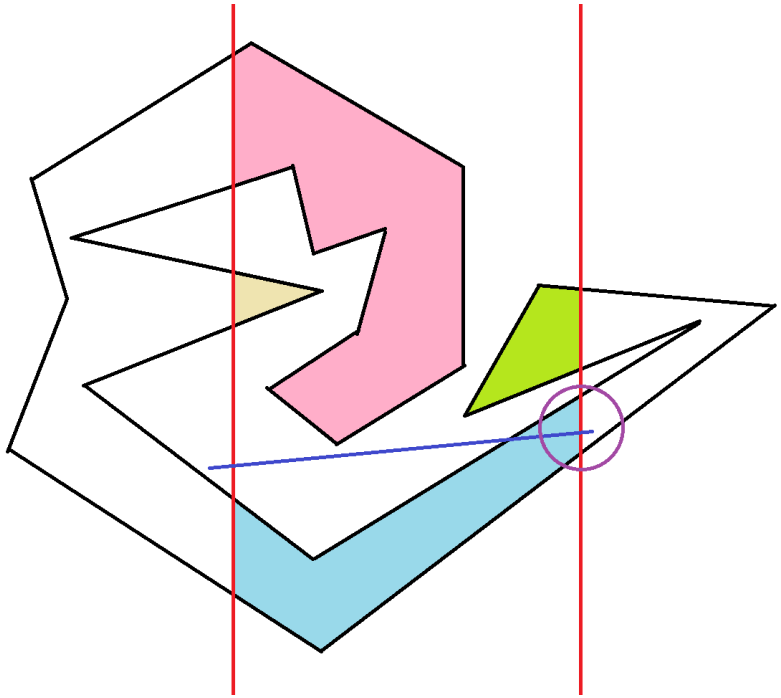
## Solution



- Preprocess the minimum and maximum y-coordinates of intersection points between each line segment connected to the left boundary and the left boundary itself, denoted as $yl$ and $yr$ respectively.
- If the y-coordinate of the intersection point between a query line segment and the left boundary $A : x = l$ is covered by some interval $[yl, yr]$, then there is an intersection.

# J. Master of Polygon

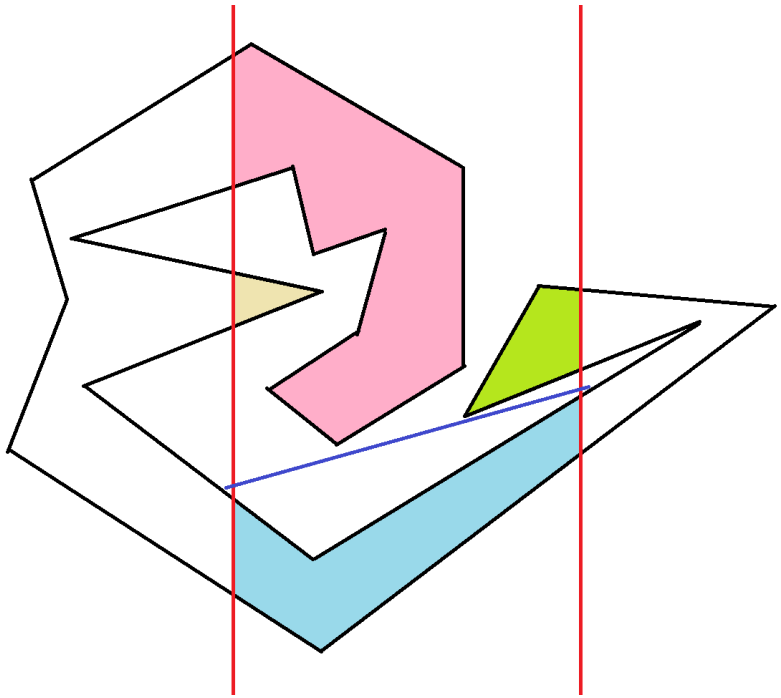
## Solution



- Preprocess the minimum and maximum y-coordinates of intersection points between each line segment connected to the right boundary and the right boundary itself, denoted as $yl$ and $yr$ respectively.
- If the y-coordinate of the intersection point between a query line segment and the right boundary $B : x = r$ is covered by some interval $[yl, yr]$, then there is an intersection.

# J. Master of Polygon

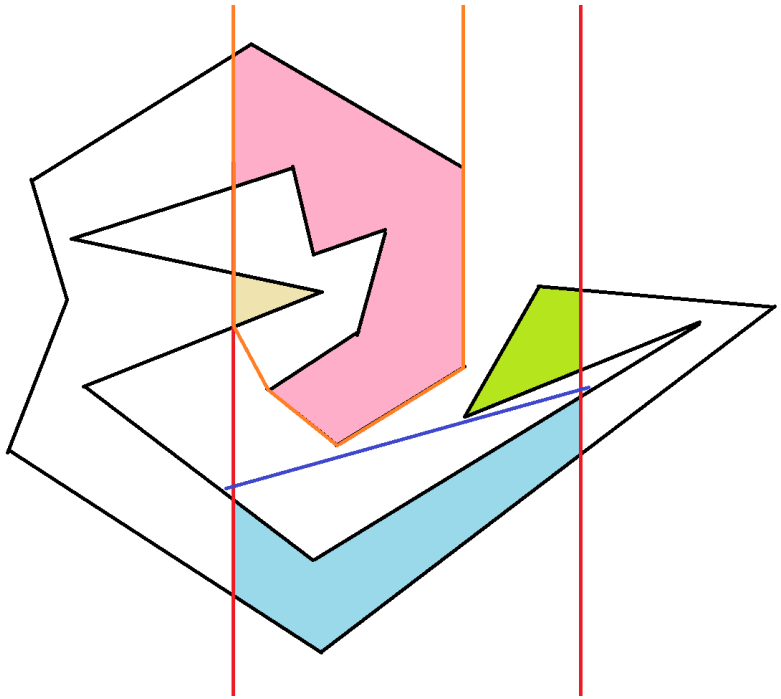
## Solution



- Next, consider whether the line corresponding to the query intersects with some segment of the polyline above or below it.
- Divide all connected polylines into four categories.

# J. Master of Polygon

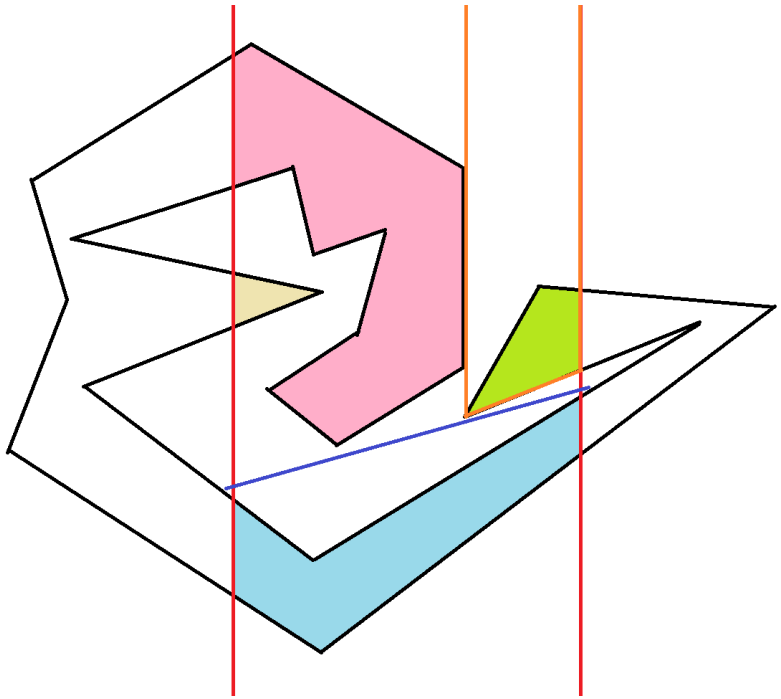
## Solution



- Case 1: The connected polyline is connected to the left boundary and the intersection point with the left boundary is above the intersection point between the query line and the left boundary.
- It is easy to see that the query line intersects with them if and only if it intersects with their convex hull.

# J. Master of Polygon

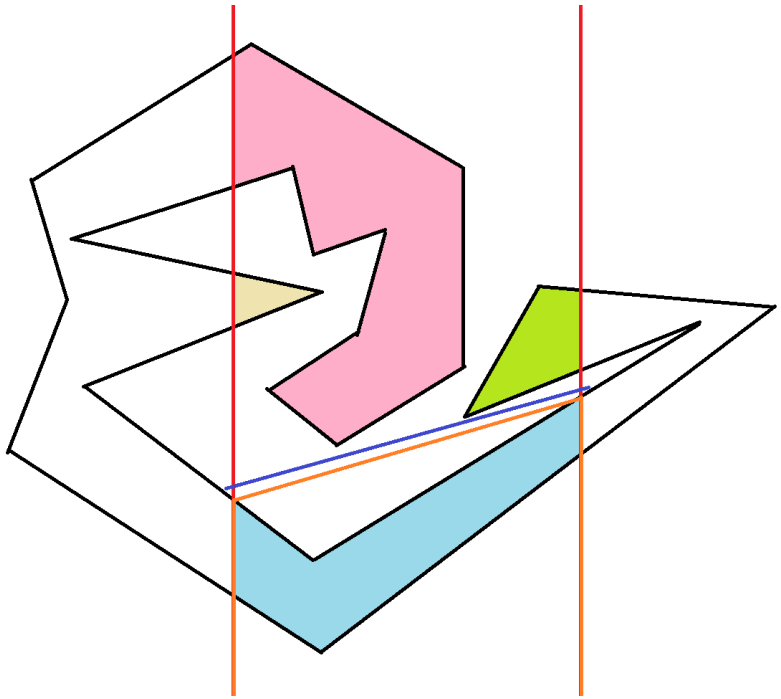
## Solution



- Case 2: The connected broken line is connected to the right boundary, and the intersection point with the right boundary is above the intersection point of the inquiry line and the right boundary.
- Similarly, it is necessary to judge whether the inquiry line intersects with their convex hull.

# J. Master of Polygon

## Solution



- Two cases where the connected broken line is located below the inquiry line are similarly treated.

# J. Master of Polygon

## Solution

- Sort connected polylines and query segments by the y-coordinate of their intersection with the boundary, retaining only the points on their convex hulls.
- Reduce the problem to maintaining a convex hull, supporting merging with a given convex hull and querying whether a given line intersects the convex hull.
- Because the y-coordinates of the new convex hull boundaries are monotonic, and according to the geometric properties of simple polygons, there must exist a dividing point that satisfies the earlier part of the merged convex hull comes from the old convex hull, and the latter part comes from the new one. Traverse the new convex hull to find this dividing point.
- For the classic problem of querying whether a given line intersects the convex hull, use binary search on the convex hull.

# J. Master of Polygon

## Solution

- Let the coordinate range be $v$. Each polygon edge and query line will be recursively processed $O(\log v)$ times, each time paying an $O(\log(n+q))$ cost for sorting or binary search.
- The total time complexity is $O((n+q)\log(n+q)\log v)$.
- Discretizing coordinates can bring it down to $O((n+q)\log^2(n+q))$.

# K. Shuttle Tour

## Description

- Given an unrooted tree with at most $50$ leaves, each node can be either ON or OFF.
- Each operation can modify the state of a node or ask for the shortest path length of all nodes that are ON in the circular tour with indices between $l$ and $r$.

# K. Shuttle Tour

## Solution

- The answer is obviously twice the sum of edge weights of tree edges where both sides have nodes that are ON.
- Starting from root $1$, perform any chain decomposition on the tree to obtain at most $k = 50$ chains.
- Build a segment tree on the indices, each node records the minimum and maximum depth of ON nodes on each chain within the corresponding index range, then the answer can be obtained by recurrence relations.
- The time complexity is $O(nk + qklogn)$.

- Given a sequence of length $n$.
- $q$ queries, each asking for the maximum $gcd$ of the sequence after removing $k$ elements from interval $[l, r]$.
- Guaranteed that $1 \leq k \leq 3$.

## Solution

- Let $g(l, r)$ be the *gcd* of all numbers in the interval $[l, r]$.
- Consider the case of $k = 1$, which requires finding $\max_{l \le x \le r} \gcd(g(l, x-1), g(x+1, r))$.
- Note that if there are consecutive numbers with the same value in $g(l, *)$, say $g(l, x) = g(l, x+1) = g(l, x+2) = \cdots = g(l, t)$, then deleting $[x+1, t+1]$ optimally involves deleting $t+1$.
- There are only $\Theta(\log V)$ distinct values for $g(l, *)$, so we can enumerate the rightmost deletion position efficiently. We can also precompute $g(l, *)$ recursively using $g(l+1, )$.
- For $k = 2, 3$, search for the first deletion position and solve recursively as subproblems.
- Time complexity: $\Theta(n \log^2 V + \sum \log^{k+1} V)$.

# M. Stage Clear

## Description

- Given a DAG with $n$ vertices and $m$ edges, where every vertex except 1 has a monster that can be defeated by spending $a_i$ HP and restoring $b_i$ HP.
- You can only move along directed edges or use magic to fly back to vertex 1.
- Find the minimum amount of HP required at the beginning to defeat all monsters in the optimal way starting from vertex 1.

# M. Stage Clear

## Solution

- If $n \leq 26$, we can use state compression DP to solve this problem.
- Let $f_S$ denote the minimum amount of HP required at the beginning when the set of defeated monsters is $S$.
- Enumerate the previous defeated monster $x$, update $f_{S \cup x}$ with $\max(f_S - b_x, 0) + a_x$, where $x \notin S$ and $S$ does not contain all vertices that have incoming edges from $x$.
- The time complexity is $O(n2^{n-1})$.

# M. Stage Clear

## Solution

- If $n \geq 27$, then we have $m \leq 45$ from the condition $n + m \leq 72$.
- Since vertex $1$ can reach any other vertex, the remaining $n - 1$ vertices must have at least one incoming edge, and after removing these edges, at most $19$ vertices have an in-degree greater than $1$.
- For each vertex, enumerate one of its incoming edges as the parent to obtain a rooted tree. There are a total of $2^{m-n+1}$ possible trees, and the subproblem of rooted trees is a classic greedy algorithm.
- Ignoring the restriction of the parent, we can obtain the optimal attack order through sorting. We take out the first monster to be defeated in the optimal order, and once its parent is defeated, we immediately have to defeat it too. Therefore, we merge it with its parent, reducing the problem size by $1$, and iterate this process for $n - 1$ rounds until the problem size is reduced to $1$.
- The time complexity of each subproblem is $O(n \log n)$, so the total time complexity is $O(2^{m-n+1} n \log n)$

# Thanks!