Problem A. The Lawyer

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 mebibytes

n segments on a straight line are given. The goal is to find two segments that do not cross (or determine such a pair does not exist)

- Find the segment with minimum right endpoint and the segment with maximum left endpoint. If those segments do not cross, that is the answer.

Runtime $O(n)$.

Problem B. Petrol

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1.8 seconds
Memory limit:	256 mebibytes

We are given an undirected weighted graph G with n vertices, m edges and a subset of the vertices (stations). We have to answer q queries of the form "is it possible to go between two stations so that the maximum distance between any two subsequent stations on the way is at most δ".

- For each vertex x in G find the closest station s_x and distance to it $d(s_x, x)$. Use Dijkstra's algorithm in $O(m \log n)$ time.
- For a fixed edge $xy \in G$, a cistern car of tank capacity δ can use it if $w*(xy) = d(s_x, x) + w(xy) + d(y, s_y) \leq \delta$.
- Sort the edges by $w*$ and the queries by δ.
- Add edges to the graph, maintaining the connected components.

Time complexity: $O(m \log m + q \log q)$.

Problem C. The Prices

Input file:	*standard input*
Output file:	*standard output*
Time limit:	2.5 seconds
Memory limit:	256 mebibytes

We want to buy $m \le 16$ products, and there are $n \le 100$ warehouses. We know the price of each product in each warehouse, and the cost of reaching the warehouse. What is the minimum total cost of our purchases?

- For each subset of the products find the cheapest store that sells that subset. To do this, simply consider each subset in each warehouse. Runtime $O(nm2^m)$ or $O(n2^m)$.
- For each subset of products find the minimum cost to buy these products. To do this, consider each subset of this subset and assume it was bought in one warehouse. This gives a standard dynamic programming algorithm over subsets that runs in $O(3^m)$ time. This yields runtime $O(n2^m + 3^m)$.
- For each subset S of products find $c[i][S]$ — the minimum cost to buy these products in warehouses $1 \dots i$.
- Adding the warehouse $i + 1$: for each subset of products S determine whether it is profitable to use that warehouse and buy some products for a lower price. We use the fact that the current minimum cost to buy the j-th product is $c[i][S] - c[i][S|\{j\}]$

Problem D. Divisors

Input file:	`standard input`
Output file:	`standard output`
Time limit:	3.5 seconds
Memory limit:	256 mebibytes

For a given sequence $a_1, \ldots, a_n (a_i < M)$ we want to find the number of pairs (i, j) such that a_i divides a_j.

- Let $c[i]$ be the count of occurences of i in the sequence; we can compute these in $O(n + M)$ time.
- For each $i \leq M$ consider all of its multiples $j \leq M$ and add $c[i] \cdot c[j]$ to the answer. There are $O(M/i)$ multiples of i, so this step works in $O(M + M/2 + M/3 + \ldots + M/M) = O(MlogM)$ time.

The total runtime is $O(n + M \log M)$.

Problem E. Euclidean Nim

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 mebibytes

There is a stack of n stones on the table. Players p and q are playing a game, taking alternating moves. The move of player x is taking away a number of stones that is a multiple of x or (if it is not possible), adding x stones. The player that empties the stack wins; determine the winner.

- If n is not divisible by $d = GCD(p, q)$, the game goes on forever. Otherwise, we divide all the numbers by d and we can assume that p and q are coprime.
- Assume $p < q$ (but we consider both cases where p or q move first)

  (A) If q moves first and $n < q$, they lose. (Since p can always enforce $n < p$ after their move, and q has to add q. Because p and q are coprime, eventually adding q causes n to be divisible by p.

  (B) Hence if p makes the first move and $n \geq p$, p wins, since they can lead to case (A).

  (C) If p moves first and $n < p$, they have to add p, and either win because of forcing (A) or $n+p \geq q$ and q has to remove q, which summarily decreases n by $q - p$. Hence p loses iff n is divisible by $q - p$.

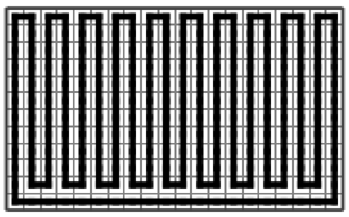
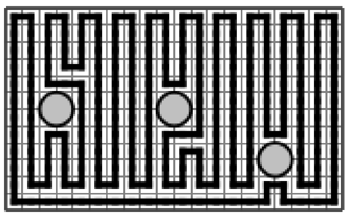
  (D) If q moves first and $n \geq q$ then for $z = n \mod q$, z divisible by $q - p$ and $4z < p$, they win by forcing (C). Otherwise they lose since any move leads to (B).

Problem F. Pillars

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 mebibytes

f 2×2 squares were removed from a rectangle of even dimensions $n \times m$ so that the centers of every two squares are at least 6 apart and the center of every square is at least 3 away from the rectangle's edge. Find a cycle going through all the remaining cells.

- First fill the empty rectangle with a cycle.
- Then remove subsequent squares, locally fixing the cycle (we consider three cases).

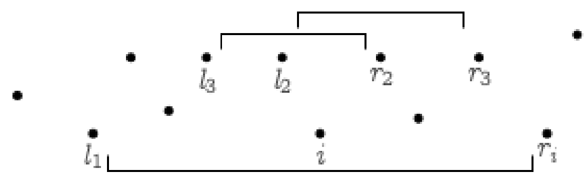
Runtime: $O(nm + f)$.

Problem G. Global Warming

Input file:	`standard input`
Output file:	`standard output`
Time limit:	1 second
Memory limit:	256 mebibytes

For a given sequence $a_1, \ldots, a_n$ we want to find the longest interval containing one minimum and one maximum.

- For each element i we want to compute in constant time the longest interval containing that element as the unique minimum. The borders of this interval are given by l_1, r_1 that are the closest elements not greater than a_i.



- Let l_3, l_2, r_2, r_3 be the maximum elements in (l_1, r_1) closest to i. The candidates for the sought for interval are (l_3, r_2) and (l_2, r_3).
- Cases where one of the indices l_3, l_2, r_2, r_3 doex not exist, need to be considered separately.
- Finding the indices can be done on a segtree in total time $O(n \log n)$, or using a stack in time $O(n)$.

Problem H. Hit of the Season

Input file:	`standard input`
Output file:	`standard output`
Time limit:	5.9 seconds
Memory limit:	256 mebibytes

We are given a text of length $n \leq 3000$ over the alphabet "`RGB*`", containing $k \leq 19$ * symbols. We need to cover it using the shortest possible pattern over "`RGB`", assuming that '`*`' matches everything.

- All of the patterns longer than $n/2$ can be checked in total time $O(n^2)$.
- When considering a shorter pattern we will guess the value of the `*` (there are at most $k/2$ of them in either the first or the second half of the text).
- We do this recursively, maintaining the set of occurrences of the pattern in the text and the maximum distance between two subsequent occurrences. If the distance does not exceed the length of the pattern, it is admissible. Updating the set for '`*`' can be done in $O(n)$ time.

Runtime: $O(n^2 + 3^(k/2)n)$.

To efficiently maintain the set of occurences of the pattern, first compute in $O(n^2)$ time for the first half of the text P:

- the longest match of P in every possible location.
- For each letter and each '`*`' in P, all alignments of P that become inadmissible after replacing '`*`' by that letter.

Problem I. The Staging

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2.9 seconds
Memory limit:	256 mebibytes

There are n gangsters in a shootout; the i-th of them will shoot at the p_i-th. How many gangsters will remain alive for a given order of shooting, and after each of the q changes in this ordering?

- Decompose the permutation $p_1, \ldots, p_n$ into cycles and consider each of them separately.
- Renumber the gangsters in the fixed cycle and unroll the cycle twice. If i is the first to shoot in the cycle, then $i+1$ definitely dies and we only have to compute how many of the gangsters in $[i+1, i+n]$ survive.
- We construct a segtree in which we can perform a change in $O(\log n)$ time, so that the solution will work in $O((n+q)\log n)$ time.
- For an interval of gangsters $[x, y]$ of length 2^k we store how many gangsters from that interval survive and what happens to gangster y depending on whether or not x shoots.
- For the interval $[x, y] = [x, y_s] \cup [x_s, y]$ we update that in constant time. The key observation is that gangster x_s shoots if $t_{x_s} < t_{y_s}$ or gangster y_s does not survive:

$$c[x, y, i] = c[x, y_s, i] + c[x_s, y, t_{x_s} < t_{y_s} \vee p[x, y_s, i] = 0]$$
$$p[x, y, i] = p[x_s, y, t_{x_s} < t_{y_s} \vee p[x, y_s, i] = 0]$$

We can simplify:

- Gangster i is a surefire if $t_i < t_j$ and $p_j = i$.
- If the distance to the next surefire is d, then $\lfloor d/2 \rfloor$ gangsters survive there.
- It is enough to maintain the set of surefires and after every change t_i update the gangsters i and p_i.

Runtime: $O((n+q)\log n)$.

Problem J. The Cave

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2.6 seconds
Memory limit:	256 mebibytes

Given are a tree with n vertices and m of its subtrees: S_i consists of the vertices whose sum of distances to a_i and b_i is at most d_i. Find a vertex in the intersection $S = \bigcap S_i$, or determine it is empty.

- For a given vertex v in time $O(n + m)$ we can compute the distances to it and determine whether it belongs to S.
- Find the vertex v whose all subtrees have size at most $2/3n$. Determine that either v belongs to S or a subtree of v containing S exists, or S is empty.

We perform $O(\log n)$ recursive steps, hence the total complexity is $O(n + m \log n)$.

There is a faster solution:

- The distance from v to S_i is:
$$max(0, \lceil (d(v, a_i) + d(v, b_i) - d_i)/2 \rceil).$$
The distances from v to all S_i can be computed in $O(n + m)$ time.
- We find the distances from vertex 1, then the S_i farthest from that vertex.
- Let x be a vertex from S_i that is closest to 1. If S is nonempty, it contains x.

Runtime: $O(n + m)$.

Problem K. The Captain

Input file:	`standard input`
Output file:	`standard output`
Time limit:	2.9 seconds
Memory limit:	256 mebibytes

n points on the plane are given. The cost to go directly between points (x_A, y_A) and (x_B, y_B) is $min(|x_A - x_B|, |y_A - y_B|)$. Find the minimum cost of travel between two given points.

- If we have three points (x_A, y_A), (x_B, y_B), (x_C, y_C), $x_A \le x_B \le x_C$ and the minimum cost to go from A to C is $x_C - x_A$, the cost of travel will not increase if we visit B on the way.
- This implies that for any given point it is enough to consider going to the closest points in each of the four cardinal directions. To find these closest points it is enough to sort the points twice — once by the x coordinate and once by the y coordinate.

The minimum cost can be found using Dijkstra's algorithm in a graph with n vertices and at most $4n$ edges in time $O(n \log n)$.