Problem Tutorial: "Maxim Buys an Apartment"

Group #1. Go through all possible variants of apartments inhabitance with exactly k apartments are occupied. For each variant calculate number of vacant apartments with inhabited rooms adjacent to them. Output minimum and maximum through this numbers. Expected complexity: $O(n \cdot n!)$ or $O(n \cdot C_n^k)$.

Group #2. Minimum number of good apartments is almost always 1, since rooms with indices from 1 to k could be inhabited. Exceptions are cases in which $k = 0$ and $k = n$, in these cases both minimum and maximum number of good rooms is 0 since there is no inhabitant or vacant apartments.

Maximum number of good apartments could be reached, for example, as follows. Assume apartments with indices 2, 5, 8 and so on are occupied as much as possible. Each of these apartments produces 2 good rooms except from, if it exists, the one with index n (that apartment produces 1 good apartment).

If number of inhabitant apartments is less than k occupy any of rest rooms to reach that number, every such occupation will decrease number of good apartments by one). Simulate this process, than count the number of good rooms to find maximum possible number. Expected complexity: $O(n^2)$ or $O(n)$.

Group #3. Instead of simulating the above process calculate number of apartments with indices 2, 5, 8 and so on excluding the one with index n if it exists. Number of that apartments is equal to $x = \lfloor \frac{n}{3} \rfloor$, and if $k \leq x$ you can occupy some of these apartments to reach maximum number of good rooms equal to $2 \cdot k$. Otherwise, if $k > x$, assume that apartments with indices 2, 5, 8 and so on are occupied, so any room has at least one inhabited room adjacent to it. Therefore number of good apartments is equal to number of vacant apartments and is equal to $n - k$. Implementation of these formulas with keeping in mind cases in which $k = 0$ and $k = n$ will be scored as full solution of problem. Expected complexity: $O(1)$.

Problem Tutorial: "Planning"

Group #1. For the first subtask you can just brute force all possible orders of plane departure (for example you can use next_permutation function if you are using C++) and find order with minimal cost. This solution has complexity $O(n! \cdot n)$ and expected to score 30 points.

Group #2. Next we will show that following greedy solution is correct: let's for each moment of time use a plane, which can depart in this moment of time (and didn't depart earlier, of course) with minimal cost of delay. Proof is quite simple: it's required to minimize $\sum_{i=1}^{n} c_i \cdot (t_i - i) = \sum_{i=1}^{n} c_i \cdot t_i - \sum_{i=1}^{n} c_i \cdot i$. You can notice that $\sum_{i=1}^{n} c_i \cdot i$ is constant, so we just need to minimize $\sum_{i=1}^{n} c_i \cdot t_i$.

Consider the optimal solution when plane i departs at moment b_i and solution by greedy algorithm in which plane i departs at moment a_i. Let x be plane with minimal c_x, such $a_x \neq b_x$. At any moment greedy algorithm takes avaliable plane with lowest c_x, so $a_x < b_x$. Let y be a plane, such that $b_y = a_x$. But $c_y >= b_y$, so $b_x \cdot c_x + b_y \cdot c_y >= b_x \cdot c_y + b_y \cdot c_x$ and it's possible to swap b_x and b_y in optimal solution without losing of optimality. By performing this operation many times it's possible to make $b_i = a_i$ for each i and it means that greedy solution is optimal.

Implementation of greedy solution with $O(n^2)$ complexity will give you 60 points.

To make this solution work faster you need to use some data structures to find optimal plane faster for each moment. This data structure should be able to add number into set, give value of minimal element in set and erase minimal number from set. For this purpose you can use heap (or someting like std::set or std::priority_queue in C++).

Problem Tutorial: "Boredom"

Group #1. $(n, q \leqslant 10)$ For the first subtask we can iterate over all interestring rectangles and for each rectangle iterate over all cells and check if it belongs to both interesting rectangle and rectangle in query. Complexity is $O(q \cdot n^4)$.

Group #2. $(n, q \leqslant 100)$ For the second subtask we can also iterate over all interesting rectangles, but check if two rectangles have non-zero intersection in $O(1)$. Complexity is $O(q \cdot n^2)$.

Group #3. $(n, q \leqslant 5000)$ For the first subtask we can't iterate over all interesting rectangles any more.

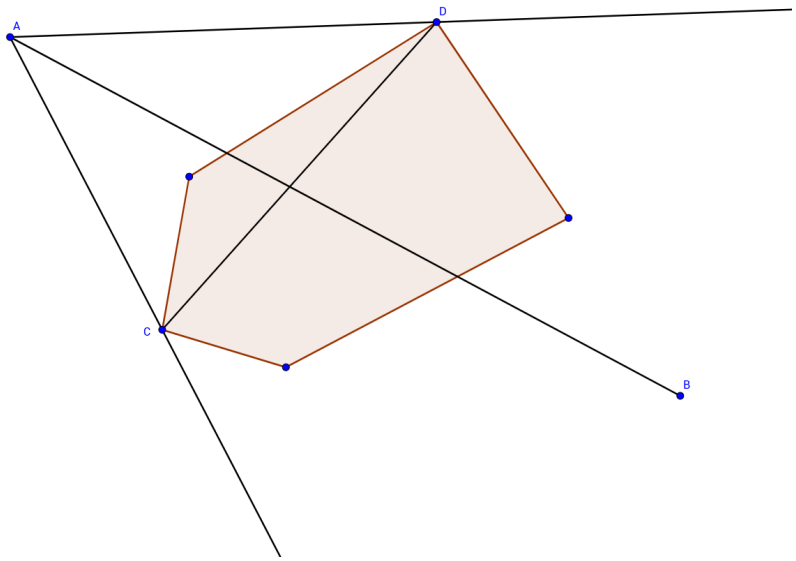
Let's instead count number of rectangles that are not intersecting our rectangle. To do it let's calculate number of rectangles to the left, right, up and down of rectangle in query. It can be easily done in $O(1)$ time: suppose we have rectangle with corners (i, p_i) and (j, p_j). We have $min(i, j) - 1$ points to the left of rectangle, $n - max(i, j)$ to the right, $min(p_i, p_j) - 1$ to the down, etc. If we have x points in some area, there are $\frac{x(x-1)}{2}$ rectangles in that area. But now we calculated twice rectangles that are simultaneously to the left and up of our rectangle, left and down, etc. To find number of such rectangles we can iterate over all points and find points which are in these areas and find number of rectangles in area using formula $\frac{x(x-1)}{2}$. The complexity is $O(q \cdot n)$.

Group #4. $(n, q \leqslant 2 \cdot 10^5)$For the last subtask we need to find number of points in some areas faster. It's quite easy to notice that we just have many queries of finding number of points in some subrectangle. It's classical problem that can be solved with some 2d tree in $O(q \cdot log^2)$ solution. But it can be too slow and can not fit into time limit in case of inaccurate implementation. However, you can notice that all queries are offline and find number of points in subrectangle in $O(q \cdot logn)$ time. It's fast enough to pass all tests.

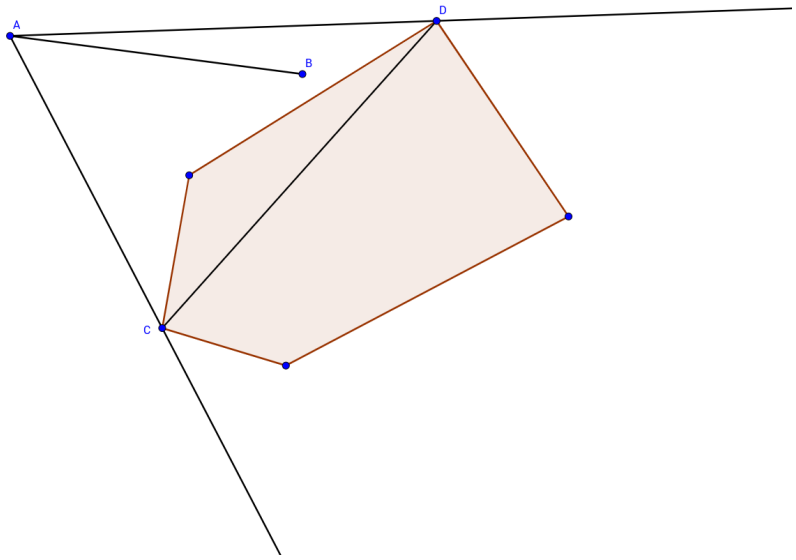
Problem Tutorial: "Offline Pirates"

Group #1. ($n \leq 2000$, $k \leq 10$). For every two points check if their segment intersects with the polygon. $O(n^2 k)$

Group #2. ($n \leq 2000$, $k \leq 1000$). Let's check for intersection in $O(\log k)$ time.

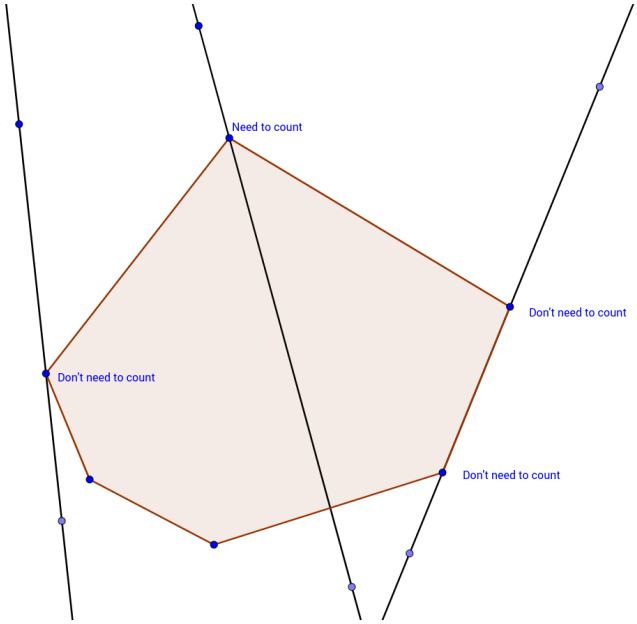


Segment AB intersects with the polygon iff AB intersects with CD. We can find points C and D with binary search. $O(n^2 \log k)$

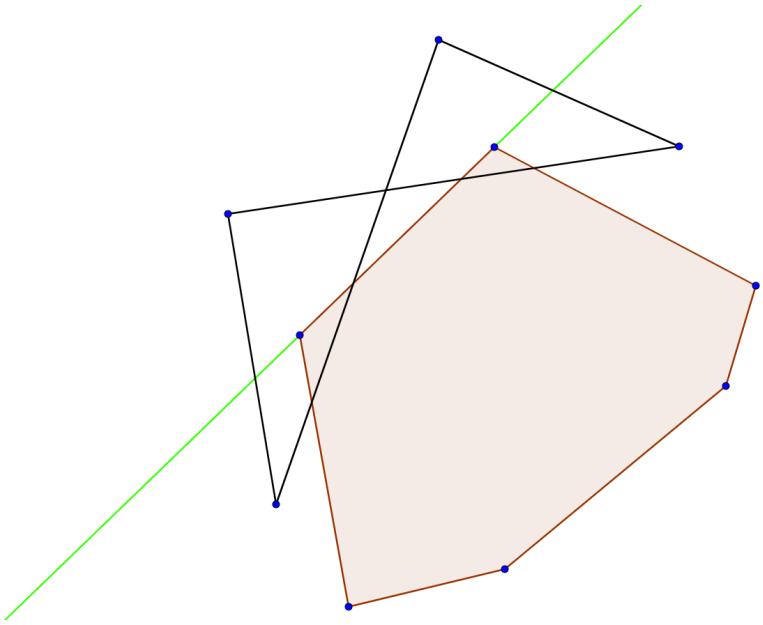


Group #3 ($n \le 7000$, $k \le 200$). We can precalculate tangents of all points and check for intersections in $O(1)$ time. $O(nk + n^2)$

Group #4 ($n \le 100\,000$, $k \le 10$). Let's count the number of segments crossing the polygon. It equals twice the number of intersections since the polygon is convex (though we need not to count some intersections in vertices of the polygon).



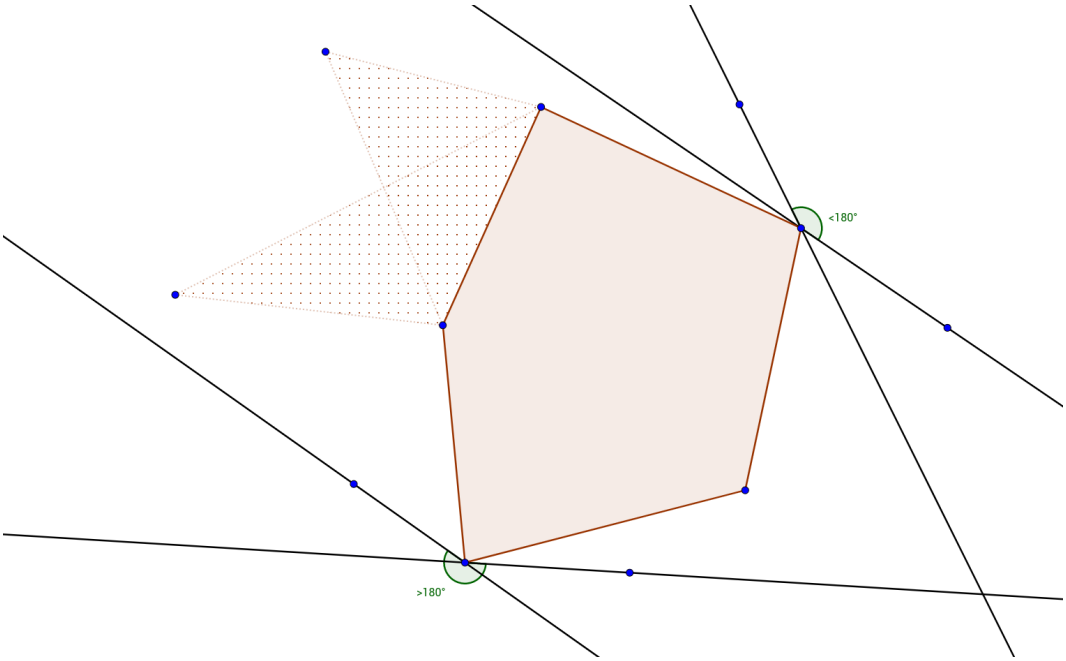
Let's count the number of intersections for each side of the polygon individually. It equals the number of segments crossing the line containing this side minus the number of segments crossing the rays complementing this side. First number is the product of numbers of points to the left and to the right of the line containing the side. Second number can be calculated for each ray separately.



Consider a ray with vertex A. To find the number of segments crossing this ray we can sort all points by angle around point A. Then we can apply two pointers technique. After that, depending on the implementation we have to add or subtract some intersections in vertices (using two pointers technique again). $O(nk \log n)$

Group #5 ($n \leq 100\,000$, $k \leq 100\,000$)

Notice that segment AB doesn't intersect the polygon if and only if either 1) there is a side which can be "seen" from both A and B or 2) there is a vertex which is a tangent point of both A and B and the angle between A and B is less than or equal to 180 degrees. Also both conditions can't be true at the same time which means that the answer is the sum of the number of pairs of points which see the common side and the number of pairs of points which fulfill the second condition.



First part:

Let's enumerate all sides of the polygon with indices from 1 to k. Then for every point, the set of indices of sides which can be seen from this point is either some segment or pair of non-intersecting segments

$[1, l]$, $[r, k]$ $(l < r)$. The answer is the number of pairs of intersecting segments. Let's sort them by left border (we can find these segments by finding tangents to the polygon). If we enumerate them as $1..m$, the number of segments with index greater than i crossing the i-th segment equals $x - i - 1$ where x is the index of the first segment with left border greater than the right border of the i-th segment (or $m + 1$ if such segment does not exist). Here x can be found with binary search.

Second part:

Each point has only two tangent points so for each vertex of the polygon, we can sort by angle all points for which this vertex is a tangent point and count the number of "good" pairs with two pointers method. $O(n \log(n + k))$