# «Titanic»

*Problem author and developer: Alexander Zavarin*

First, let us understand that since the hook can only be fired perpendicular to the rescue boat's movement, there is exactly 1 point on the line passing through points $A$ and $B$. This means that we can compute the segment on that line during which the hook will be blocked if we catch a lifeboat at its initial position. To compute the point where we need to catch the lifeboat, we should draw a perpendicular to line $AB$; the foot of that perpendicular will be the required point. To compute the point where the hook becomes free again, we need to draw from the initial point a vector in the direction of the rescue boat's movement whose length is equal to the length of the perpendicular. Let us call, for a lifeboat, the starting point of this segment the — *capture point*, and the ending point the — *rescue point*.

Consider subgroup 1. In it, there is a restriction on the $y$-coordinate of points $A$ and $B$: for both of them it is equal to 0. W.l.o.g., let the rescue boat sail from left to right (otherwise, we can simply mirror the coordinates with respect to the line $x = 0$). Then, if the $i$-th lifeboat is at point $(x_i, y_i)$, its capture point has coordinates $(x_i, 0)$, and its rescue point is $(x_i + y_i, 0)$, since the distance between the capture point and the lifeboat was exactly $y_i$. We can discard all lifeboats whose capture point has an $x$-coordinate less than $a_x$ or greater than $b_x$, because some of them we will not be able to catch with the hook, and the others we will not have time to pull in before the game ends. Now let us sort all lifeboats by the $x$-coordinate of their capture point in order to use dynamic programming. For each lifeboat $i$, let us compute the value $dp_i$ — the maximum number of lifeboats among those whose number in the sorted order is less than $i$, while still being able to rescue the $i$-th lifeboat. Now we simply iterate once over the lifeboat number in sorted order; let the current one be $i$, and then we iterate over all $j < i$. For the $j$-th lifeboat, we check that the $x$-coordinate of its rescue point is not greater than the $x$-coordinate of the capture point of our $i$-th lifeboat, otherwise we will not be able to rescue it. We compute $dp_i = \max_{j<i}(dp_j) + 1$ over all suitable $j$. This works because for $dp_i$, among the rescued lifeboats with numbers $j < i$, there is a lifeboat with the maximum number $j_{max}$, and for it, if we remove the rescued $i$-th lifeboat, then $dp_{j_{max}} = dp_i - 1$ by the definition of $dp_i$. Then, since the answer also contains some lifeboat $i_{max}$ with the largest number, the answer is $dp_{i_{max}}$, i.e. it can be computed as $\max_i(dp_i)$.

Now let us move to subgroup 2. Sorting the lifeboats so conveniently no longer works, but it can still be done if we now use as the sorting parameter the distance between point $A$ and the capture point of a lifeboat. This can be computed using the formulas for constructing a perpendicular to a line and for computing distances between points, but the author's solution uses vector and dot products to calculate this distance. Let the lifeboat be at point $P$. Then the distance from $A$ to the capture point can be computed as $|AP| \cdot \cos\alpha$, where $\alpha$ is the angle between $AP$ and $AB$. This can be expressed via the dot product of vectors as $|AP| \cdot \cos\alpha = \frac{AB \cdot AP}{|AB|}$. The length of the perpendicular is computed similarly, but via the cross product: $|AP| \cdot \sin\alpha = \frac{|AB \times AP|}{|AB|}$. It is important not to forget to take the absolute value of the cross product: in the first case, the absolute value was not used so that lifeboats outside the rescue boat's reachable area would have a negative distance from point $A$ for convenience. Now, using these two values, we can compute the distance from point $A$ to the rescue point of the lifeboat as their sum. Note that these distances have a common denominator — the length of segment $AB$. So let us simply multiply all values by this number; this will not change the order of the lifeboats or their sorting, but it will allow us to switch to integer computations. Now, having for each lifeboat two values — the distance from point $A$ to the capture point and to the rescue point — and also using the dynamic programming idea from subgroup 1, we can solve this one as well.

Let us move to subgroup 3, and now think about how to improve the computation of $dp_i$. For this, we use a *scanline* idea. We will have 2 types of events: "we arrived at the capture point of some lifeboat" and "we arrived at the rescue point of some lifeboat". We sort all events by their distance from point $A$, and then process them in order while storing the answer — the maximum number of lifeboats that we can rescue after processing the first $i$ events; let us denote it by *score*. If we encounter an event of the 1-st type for the $j$-th lifeboat, then by definition we compute $dp_j = score + 1$. If we encounter an event of the 2-nd type for the $j$-th lifeboat, this means that if we had been rescuing this lifeboat, then the hook would now

become free, and therefore we can use $dp_j$ to update $score$: $score = \max(score, dp_j)$. It is important that if several events are located at the same point, then events of the 2-nd type must be processed first, and only then events of the 1-st type. After processing all events, $score$ will be our answer.

Subgroups 4 and 5 use the combined ideas of subgroups 2 and 3.