XII NEERC Western Subregional Contest

Problem	Supposed difficulty	Solved	Tried
Arrays	impossible	0	14
“Bulls and Cows”	trivial	52	52
Courier	medium	14	18
Dales and Hills	easy	36	44
Extremal Permutations	medium	6	12
Figure and Spots	easy	29	37
Game	medium	5	17
Hotel in Ves Lagos	medium	16	29
Illumination of Buildings	hard	0	4
Journal	hard	1	1
Kids and Prizes	easy	21	22
L-Shapes	easy	46	49

Problem A. Arrays

Idea: Yuri Stepin, Pavel Irzhavski

Solutions: Pavel Irzhavski

Tests: Pavel Irzhavski

Analysis: Pavel Irzhavski

Statement: Serge Kashkevich, Pavel Irzhavski, Ivan Metelsky

Problem A. Arrays

Let's sort numbers X_i in non-decreasing order:

$$0 \leq X_1 \leq \cdots \leq X_{3 \cdot N}.$$

Problem A. Arrays

Let's sort numbers X_i in non-decreasing order:

$$0 \leq X_1 \leq \cdots \leq X_{3 \cdot N}.$$

Also we can assume that B_i will be sorted in increasing order (otherwise we can rearrange all three sequences to satisfy this condition).

Problem A. Arrays

Let's sort numbers X_i in non-decreasing order:

$$0 \leq X_1 \leq \cdots \leq X_{3 \cdot N}.$$

Also we can assume that B_i will be sorted in increasing order (otherwise we can rearrange all three sequences to satisfy this condition).
Consider $N = 1$.

Problem A. Arrays

Let's sort numbers X_i in non-decreasing order:

$$0 \leq X_1 \leq \cdots \leq X_{3 \cdot N}.$$

Also we can assume that B_i will be sorted in increasing order (otherwise we can rearrange all three sequences to satisfy this condition).

Consider $N = 1$. We have $3! = 6$ ways to get A_1, B_1 and C_1, but taking $A_1 = 3$, $B_1 = 1$ and $C_1 = 2$ we always get value of S that is not less than in any other case.

Problem A. Arrays

To prove this fact we have to prove five inequalities

Problem A. Arrays

To prove this fact we have to prove five inequalities:

$$(X_1 - X_2) \cdot X_3 \le (X_3 - X_1) \cdot X_2, \tag{1}$$

$$(X_1 - X_3) \cdot X_2 \le (X_3 - X_1) \cdot X_2, \tag{2}$$

$$(X_2 - X_1) \cdot X_3 \le (X_3 - X_1) \cdot X_2, \tag{3}$$

$$(X_2 - X_3) \cdot X_1 \le (X_3 - X_1) \cdot X_2, \tag{4}$$

$$(X_3 - X_2) \cdot X_1 \le (X_3 - X_1) \cdot X_2. \tag{5}$$

Problem A. Arrays

To prove this fact we have to prove five inequalities:

$$(X_1 - X_2) \cdot X_3 \le (X_3 - X_1) \cdot X_2, \tag{1}$$
$$(X_1 - X_3) \cdot X_2 \le (X_3 - X_1) \cdot X_2, \tag{2}$$
$$(X_2 - X_1) \cdot X_3 \le (X_3 - X_1) \cdot X_2, \tag{3}$$
$$(X_2 - X_3) \cdot X_1 \le (X_3 - X_1) \cdot X_2, \tag{4}$$
$$(X_3 - X_2) \cdot X_1 \le (X_3 - X_1) \cdot X_2. \tag{5}$$

(1), (2) and (4) are right since their left parts are non-positive and right parts are non-negative.

Problem A. Arrays

To prove this fact we have to prove five inequalities:

$$(X_1 - X_2) \cdot X_3 \le (X_3 - X_1) \cdot X_2, \quad (1)$$
$$(X_1 - X_3) \cdot X_2 \le (X_3 - X_1) \cdot X_2, \quad (2)$$
$$(X_2 - X_1) \cdot X_3 \le (X_3 - X_1) \cdot X_2, \quad (3)$$
$$(X_2 - X_3) \cdot X_1 \le (X_3 - X_1) \cdot X_2, \quad (4)$$
$$(X_3 - X_2) \cdot X_1 \le (X_3 - X_1) \cdot X_2. \quad (5)$$

(1), (2) and (4) are right since their left parts are non-positive and right parts are non-negative.

$$(3) \Leftrightarrow -X_1 \cdot X_3 \le -X_1 \cdot X_2 \Leftrightarrow 0 \le (X_3 - X_2) \cdot X_1.$$

Problem A. Arrays

To prove this fact we have to prove five inequalities:

$$(X_1 - X_2) \cdot X_3 \le (X_3 - X_1) \cdot X_2, \tag{1}$$
$$(X_1 - X_3) \cdot X_2 \le (X_3 - X_1) \cdot X_2, \tag{2}$$
$$(X_2 - X_1) \cdot X_3 \le (X_3 - X_1) \cdot X_2, \tag{3}$$
$$(X_2 - X_3) \cdot X_1 \le (X_3 - X_1) \cdot X_2, \tag{4}$$
$$(X_3 - X_2) \cdot X_1 \le (X_3 - X_1) \cdot X_2. \tag{5}$$

(1), (2) and (4) are right since their left parts are non-positive and right parts are non-negative.

$$(3) \Leftrightarrow -X_1 \cdot X_3 \le -X_1 \cdot X_2 \Leftrightarrow 0 \le (X_3 - X_2) \cdot X_1.$$

$$(5) \Leftrightarrow X_3 \cdot X_1 \le X_3 \cdot X_2 \Leftrightarrow 0 \le X_3 \cdot (X_2 - X_1).$$

Problem A. Arrays

Now consider $N = 2$.

Problem A. Arrays

Now consider $N = 2$. It would be good to get again unique best values for our sequences.

Problem A. Arrays

Now consider $N = 2$. It would be good to get again unique best values for our sequences. But it is rather easy to get contrary instance.

Problem A. Arrays

Now consider $N = 2$. It would be good to get again unique best values for our sequences. But it is rather easy to get contrary instance.

In truth we have $\frac{6!}{3! \cdot 3!} = 20$ ways to divide set $\{1, \cdots, 6\}$ into two subsets $\{A_1, B_1, C_1\}$ and $\{A_2, B_2, C_2\}$ (for each of this triples we know that we should consider only $A_i > C_i > B_i$).

Problem A. Arrays

Now consider $N = 2$. It would be good to get again unique best values for our sequences. But it is rather easy to get contrary instance.

In truth we have $\frac{6!}{3! \cdot 3!} = 20$ ways to divide set $\{1, \cdots, 6\}$ into two subsets $\{A_1, B_1, C_1\}$ and $\{A_2, B_2, C_2\}$ (for each of this triples we know that we should consider only $A_i > C_i > B_i$). But we assumed $B_1 < B_2$, so we'll consider only 10 of them.

Problem A. Arrays

Now consider $N = 2$. It would be good to get again unique best values for our sequences. But it is rather easy to get contrary instance.

In truth we have $\frac{6!}{3!\cdot 3!} = 20$ ways to divide set $\{1, \cdots, 6\}$ into two subsets $\{A_1, B_1, C_1\}$ and $\{A_2, B_2, C_2\}$ (for each of this triples we know that we should consider only $A_i > C_i > B_i$). But we assumed $B_1 < B_2$, so we'll consider only 10 of them.

It may be noticed that always at least one of two values $(X_6 - X_1) \cdot X_5 + (X_4 - X_1) \cdot X_3$ and $(X_6 - X_1) \cdot X_4 + (X_5 - X_1) \cdot X_3$ is not less than value of S in any other case.

Problem A. Arrays

Now consider $N = 2$. It would be good to get again unique best values for our sequences. But it is rather easy to get contrary instance.

In truth we have $\frac{6!}{3! \cdot 3!} = 20$ ways to divide set $\{1, \cdots, 6\}$ into two subsets $\{A_1, B_1, C_1\}$ and $\{A_2, B_2, C_2\}$ (for each of this triples we know that we should consider only $A_i > C_i > B_i$). But we assumed $B_1 < B_2$, so we'll consider only 10 of them.

It may be noticed that always at least one of two values $(X_6 - X_1) \cdot X_5 + (X_4 - X_1) \cdot X_3$ and $(X_6 - X_1) \cdot X_4 + (X_5 - X_1) \cdot X_3$ is not less than value of S in any other case.

We have to prove for each of other eight ways to divide $\{1, \cdots, 6\}$ into two triples that corresponding value of S is not greater than at least one of expressions above.

Problem A. Arrays

So we have to prove eight inequalities

Problem A. Arrays

So we have to prove eight inequalities:

$$
\begin{aligned}
(X_3 - X_1)X_2 + (X_6 - X_4)X_5 &\le (X_6 - X_1)X_5 + (X_4 - X_2)X_3,\\
(X_4 - X_1)X_2 + (X_6 - X_3)X_5 &\le (X_6 - X_1)X_5 + (X_4 - X_2)X_3,\\
(X_4 - X_1)X_3 + (X_6 - X_2)X_5 &\le (X_6 - X_1)X_5 + (X_4 - X_2)X_3,\\
(X_5 - X_1)X_2 + (X_6 - X_3)X_4 &\le (X_6 - X_1)X_4 + (X_5 - X_2)X_3,\\
(X_5 - X_1)X_3 + (X_6 - X_2)X_4 &\le (X_6 - X_1)X_4 + (X_5 - X_2)X_3,\\
(X_5 - X_1)X_4 + (X_6 - X_2)X_3 &\le (X_6 - X_1)X_4 + (X_5 - X_2)X_3,\\
(X_6 - X_1)X_2 + (X_5 - X_3)X_4 &\le (X_6 - X_1)X_4 + (X_5 - X_2)X_3,\\
(X_6 - X_1)X_3 + (X_5 - X_2)X_4 &\le (X_6 - X_1)X_4 + (X_5 - X_2)X_3.
\end{aligned}
$$

Skip proof

Problem A. Arrays

We will use several times following statement: if $a \leq b$ and $c \leq d$ then $ad + bc \leq ac + bd$.

Problem A. Arrays

We will use several times following statement: if $a \leq b$ and $c \leq d$ then $ad + bc \leq ac + bd$.

In truth $ad + bc \leq ab + cd \Leftrightarrow 0 \leq (d - c)(b - a)$.

Problem A. Arrays

We will use several times following statement: if $a \leq b$ and $c \leq d$ then $ad + bc \leq ac + bd$.

In truth $ad + bc \leq ab + cd \Leftrightarrow 0 \leq (d - c)(b - a)$.

$$(X_3 - X_1)X_2 + (X_6 - X_4)X_5 \leq (X_6 - X_1)X_5 + (X_4 - X_2)X_3 \Leftrightarrow$$
$$\Leftrightarrow X_3X_2 - X_1X_2 - X_4X_5 \leq -X_1X_5 + X_4X_3 - X_2X_3 \Leftrightarrow$$
$$\Leftrightarrow 2X_3X_2 + X_1X_5 \leq X_4X3 + X_1X_2 + X_4X_5$$

Problem A. Arrays

We will use several times following statement: if $a \leq b$ and $c \leq d$ then $ad + bc \leq ac + bd$.

In truth $ad + bc \leq ab + cd \Leftrightarrow 0 \leq (d - c)(b - a)$.

$$(X_3 - X_1)X_2 + (X_6 - X_4)X_5 \leq (X_6 - X_1)X_5 + (X_4 - X_2)X_3 \Leftrightarrow$$
$$\Leftrightarrow X_3X_2 - X_1X_2 - X_4X_5 \leq -X_1X_5 + X_4X_3 - X_2X_3 \Leftrightarrow$$
$$\Leftrightarrow 2X_3X_2 + X_1X_5 \leq X_4X3 + X_1X_2 + X_4X_5$$

This follows from:

$$X_3X_2 \leq X_3X_4$$
$$X_3X_2 \leq X_4X_2$$
$$X_4X_2 + X_1X_5 \leq X_1X_2 + X_4X_5 \text{ (since } X_1 \leq X_4 \text{ and } X_2 \leq X_5\text{)}$$

Problem A. Arrays

$$(X_4 - X_1)X_2 + (X_6 - X_3)X_5 \le (X_6 - X_1)X_5 + (X_4 - X_2)X_3 \Leftrightarrow$$
$$\Leftrightarrow X_4X_2 - X_1X_2 - X_3X_5 \le -X_1X_5 + X_4X_3 - X_2X_3 \Leftrightarrow$$
$$\Leftrightarrow X_4X_2 + X_3X_2 + X_1X_5 \le X_4X_3 + X_1X_2 + X_3X_5$$

Problem A. Arrays

$$(X_4 - X_1)X_2 + (X_6 - X_3)X_5 \le (X_6 - X_1)X_5 + (X_4 - X_2)X_3 \Leftrightarrow$$
$$\Leftrightarrow X_4X_2 - X_1X_2 - X_3X_5 \le -X_1X_5 + X_4X_3 - X_2X_3 \Leftrightarrow$$
$$\Leftrightarrow X_4X_2 + X_3X_2 + X_1X_5 \le X_4X_3 + X_1X_2 + X_3X_5$$

Follows from:

$$X_4X_2 \le X_4X_3$$
$$X_3X_2 + X_1X_5 \le X_1X_2 + X_3X_5$$

Problem A. Arrays

$$(X_4 - X_1)X_2 + (X_6 - X_3)X_5 \le (X_6 - X_1)X_5 + (X_4 - X_2)X_3 \Leftrightarrow$$
$$\Leftrightarrow X_4X_2 - X_1X_2 - X_3X_5 \le -X_1X_5 + X_4X_3 - X_2X_3 \Leftrightarrow$$
$$\Leftrightarrow X_4X_2 + X_3X_2 + X_1X_5 \le X_4X_3 + X_1X_2 + X_3X_5$$

Follows from:

$$X_4X_2 \le X_4X_3$$
$$X_3X_2 + X_1X_5 \le X_1X_2 + X_3X_5$$

$$(X_4 - X_1)X_3 + (X_6 - X_2)X_5 \le (X_6 - X_1)X_5 + (X_4 - X_2)X_3 \Leftrightarrow$$
$$\Leftrightarrow -X_1X_3 - X_2X_5 \le -X_1X_5 - X_2X_3 \Leftrightarrow$$
$$\Leftrightarrow X_1X_5 + X_2X_3 \le X_1X_3 + X_2X_5, \text{ which is true.}$$

Problem A. Arrays

$$
\begin{aligned}
&(X_5 - X_1)X_2 + (X_6 - X_3)X_4 \leq (X_6 - X_1)X_4 + (X_5 - X_2)X_3 \Leftrightarrow \\
&\quad \Leftrightarrow X_5X_2 - X_1X_2 - X_3X_4 \leq -X_1X_4 + X_5X_3 - X_2X_3 \Leftrightarrow \\
&\quad \Leftrightarrow X_5X_2 + X_1X_4 + X_2X_3 \leq X_3X_4 + X_1X_2 + X_5X_3
\end{aligned}
$$

Problem A. Arrays

$$(X_5 - X_1)X_2 + (X_6 - X_3)X_4 \le (X_6 - X_1)X_4 + (X_5 - X_2)X_3 \Leftrightarrow$$
$$\Leftrightarrow X_5X_2 - X_1X_2 - X_3X_4 \le -X_1X_4 + X_5X_3 - X_2X_3 \Leftrightarrow$$
$$\Leftrightarrow X_5X_2 + X_1X_4 + X_2X_3 \le X_3X_4 + X_1X_2 + X_5X_3$$

This is a sum of three inequalities:

$$X_5X_2 + X_1X_3 \le X_1X_2 + X_5X_3$$
$$X_1X_4 + X_2X_3 \le X_1X_3 + X_2X_4$$
$$X_2X_4 \le X_3X_4$$

Problem A. Arrays

$$(X_5 - X_1)X_2 + (X_6 - X_3)X_4 \le (X_6 - X_1)X_4 + (X_5 - X_2)X_3 \Leftrightarrow$$
$$\Leftrightarrow X_5X_2 - X_1X_2 - X_3X_4 \le -X_1X_4 + X_5X_3 - X_2X_3 \Leftrightarrow$$
$$\Leftrightarrow X_5X_2 + X_1X_4 + X_2X_3 \le X_3X_4 + X_1X_2 + X_5X_3$$

This is a sum of three inequalities:

$$X_5X_2 + X_1X_3 \le X_1X_2 + X_5X_3$$
$$X_1X_4 + X_2X_3 \le X_1X_3 + X_2X_4$$
$$X_2X_4 \le X_3X_4$$

$$(X_5 - X_1)X_3 + (X_6 - X_2)X_4 \le (X_6 - X_1)X_4 + (X_5 - X_2)X_3 \Leftrightarrow$$
$$\Leftrightarrow -X_1X_3 - X_2X_4 \le -X_1X_4 - X_2X_3 \Leftrightarrow$$
$$\Leftrightarrow X_2X_3 + X_1X_4 \le X_1X_3 + X_2X_4, \text{ which is true.}$$

Problem A. Arrays

$$(X_5 - X_1)X_4 + (X_6 - X_2)X_3 \le (X_6 - X_1)X_4 + (X_5 - X_2)X_3 \Leftrightarrow$$
$$\Leftrightarrow X_5X_4 + X_6X_3 \le X_5X_3 + X_6X_4,$$ which is true.

Problem A. Arrays

$$(X_5 - X_1)X_4 + (X_6 - X_2)X_3 \le (X_6 - X_1)X_4 + (X_5 - X_2)X_3 \Leftrightarrow$$
$$\Leftrightarrow X_5X_4 + X_6X_3 \le X_5X_3 + X_6X_4, \text{ which is true.}$$

$$(X_6-X_1)X_2+(X_5-X_3)X_4 \le (X_6-X_1)X_4+(X_5-X_2)X_3 \Leftrightarrow$$
$$\Leftrightarrow X_6X_2-X_1X_2+X_5X_4-X_3X_4 \le X_6X_4-X_1X_4+X_5X_3-X_2X_3 \Leftrightarrow$$
$$\Leftrightarrow X_6X_2+X_1X_4+X_5X_4+X_2X_3 \le X_6X_4+X_1X_2+X_5X_3+X_3X_4$$

Problem A. Arrays

$$(X_5 - X_1)X_4 + (X_6 - X_2)X_3 \le (X_6 - X_1)X_4 + (X_5 - X_2)X_3 \Leftrightarrow$$
$$\Leftrightarrow X_5X_4 + X_6X_3 \le X_5X_3 + X_6X_4, \text{ which is true.}$$

$$(X_6-X_1)X_2+(X_5-X_3)X_4 \le (X_6-X_1)X_4+(X_5-X_2)X_3 \Leftrightarrow$$
$$\Leftrightarrow X_6X_2-X_1X_2+X_5X_4-X_3X_4 \le X_6X_4-X_1X_4+X_5X_3-X_2X_3 \Leftrightarrow$$
$$\Leftrightarrow X_6X_2+X_1X_4+X_5X_4+X_2X_3 \le X_6X_4+X_1X_2+X_5X_3+X_3X_4$$

Follows from:

$$X_6X_3 + X_5X_4 \le X_5X_3 + X_6X_4$$
$$X_6X_2 \le X_6X_3$$
$$X_1X_4 + X_3X_2 \le X_1X_2 + X_3X_4$$

Problem A. Arrays

And finally:

$$(X_6-X_1)X_3+(X_5-X_2)X_4 \leq (X_6-X_1)X_4+(X_5-X_2)X_3 \Leftrightarrow$$
$$\Leftrightarrow X_6X_3-X_1X_3+X_5X_2-X_2X_4 \leq X_6X_4-X_1X_4+X_5X_3-X_2X_3 \Leftrightarrow$$
$$\Leftrightarrow X_6X_3+X_5X_2+X_1X_4+X_2X_3 \leq X_6X_4+X_5X_3+X_1X_3+X_2X_4$$

Problem A. Arrays

And finally:

$$(X_6-X_1)X_3+(X_5-X_2)X_4 \leq (X_6-X_1)X_4+(X_5-X_2)X_3 \Leftrightarrow$$
$$\Leftrightarrow X_6X_3-X_1X_3+X_5X_2-X_2X_4 \leq X_6X_4-X_1X_4+X_5X_3-X_2X_3 \Leftrightarrow$$
$$\Leftrightarrow X_6X_3+X_5X_2+X_1X_4+X_2X_3 \leq X_6X_4+X_5X_3+X_1X_3+X_2X_4$$

Follows from:

$$X_6X_3 \leq X_6X_4$$
$$X_5X_2 \leq X_5X_3$$
$$X_1X_4 + X_2X_3 \leq X_1X_3 + X_2X_4$$

Problem A. Arrays

So there are two ways to form sequences A, B, C that we are interested in:

Problem A. Arrays

So there are two ways to form sequences A, B, C that we are interested in:

$$A_1 = 6, B_1 = 1, C_1 = 5 \;\; A_2 = 4, B_2 = 2, C_2 = 3$$
$$A_1 = 6, B_1 = 1, C_1 = 4 \;\; A_2 = 5, B_2 = 2, C_2 = 3$$

Problem A. Arrays

So there are two ways to form sequences A, B, C that we are interested in:

$$A_1 = 6, B_1 = 1, C_1 = 5 \;\; A_2 = 4, B_2 = 2, C_2 = 3$$
$$A_1 = 6, B_1 = 1, C_1 = 4 \;\; A_2 = 5, B_2 = 2, C_2 = 3$$

Consider arbitrary N.

Problem A. Arrays

So there are two ways to form sequences A, B, C that we are interested in:

$$A_1 = 6, B_1 = 1, C_1 = 5 \;\; A_2 = 4, B_2 = 2, C_2 = 3$$
$$A_1 = 6, B_1 = 1, C_1 = 4 \;\; A_2 = 5, B_2 = 2, C_2 = 3$$

Consider arbitrary N. Now we have some information about our sequences.

Problem A. Arrays

So there are two ways to form sequences A, B, C that we are interested in:

$$A_1 = 6, B_1 = 1, C_1 = 5 \;\; A_2 = 4, B_2 = 2, C_2 = 3$$
$$A_1 = 6, B_1 = 1, C_1 = 4 \;\; A_2 = 5, B_2 = 2, C_2 = 3$$

Consider arbitrary N. Now we have some information about our sequences.

Let $1 \leq i < j \leq N$.

Problem A. Arrays

So there are two ways to form sequences A, B, C that we are interested in:

$$A_1 = 6, B_1 = 1, C_1 = 5 \;\; A_2 = 4, B_2 = 2, C_2 = 3$$
$$A_1 = 6, B_1 = 1, C_1 = 4 \;\; A_2 = 5, B_2 = 2, C_2 = 3$$

Consider arbitrary N. Now we have some information about our sequences.

Let $1 \le i < j \le N$. Consider $N' = 2$ and $\{X'_1, \cdots, X'_6\} =$
$= \{X_{A_i}, X_{B_i}, X_{C_i}, X_{A_j}, X_{B_j}, X_{C_j}\}$.

Problem A. Arrays

So there are two ways to form sequences A, B, C that we are interested in:

$$A_1 = 6, B_1 = 1, C_1 = 5 \;\; A_2 = 4, B_2 = 2, C_2 = 3$$
$$A_1 = 6, B_1 = 1, C_1 = 4 \;\; A_2 = 5, B_2 = 2, C_2 = 3$$

Consider arbitrary N. Now we have some information about our sequences.

Let $1 \leq i < j \leq N$. Consider $N' = 2$ and $\{X'_1, \cdots, X'_6\} =$ $= \{X_{A_i}, X_{B_i}, X_{C_i}, X_{A_j}, X_{B_j}, X_{C_j}\}$.

Then $S' = (X_{A_i} - X_{B_i})X_{C_i} + (X_{A_j} - X_{B_j})X_{C_j}$, 'cause otherwise we could rearrange $A_i, B_i, C_i, A_j, B_j, C_j$ and get greater value of S.

Problem A. Arrays

So we can rearrange $A_i, B_i, C_i, A_j, B_j, C_j$ without decreasing S in such way that $B_i < B_j < C_j < C_i < A_j < A_i$ or
$B_i < B_j < C_j < A_j < C_i < A_i$.

Problem A. Arrays

So we can rearrange $A_i, B_i, C_i, A_j, B_j, C_j$ without decreasing S in such way that $B_i < B_j < C_j < C_i < A_j < A_i$ or $B_i < B_j < C_j < A_j < C_i < A_i$.

Anyway we get:

Problem A. Arrays

So we can rearrange $A_i, B_i, C_i, A_j, B_j, C_j$ without decreasing S in such way that $B_i < B_j < C_j < C_i < A_j < A_i$ or
$B_i < B_j < C_j < A_j < C_i < A_i$.

Anyway we get:

$$B_k < C_l \text{ for any } 1 \leq k, l \leq N, \tag{6}$$

Problem A. Arrays

So we can rearrange $A_i, B_i, C_i, A_j, B_j, C_j$ without decreasing S in such way that $B_i < B_j < C_j < C_i < A_j < A_i$ or
$B_i < B_j < C_j < A_j < C_i < A_i$.

Anyway we get:

$$B_k < C_l \text{ for any } 1 \leq k, l \leq N, \tag{6}$$

$$B_k < A_l \text{ for any } 1 \leq k, l \leq N, \tag{7}$$

Problem A. Arrays

So we can rearrange $A_i, B_i, C_i, A_j, B_j, C_j$ without decreasing S in such way that $B_i < B_j < C_j < C_i < A_j < A_i$ or $B_i < B_j < C_j < A_j < C_i < A_i$.

Anyway we get:

$$B_k < C_l \text{ for any } 1 \leq k, l \leq N, \tag{6}$$

$$B_k < A_l \text{ for any } 1 \leq k, l \leq N, \tag{7}$$

$$C_k < A_l \text{ for any } 1 \leq l \leq k \leq N, \tag{8}$$

Problem A. Arrays

So we can rearrange $A_i, B_i, C_i, A_j, B_j, C_j$ without decreasing S in such way that $B_i < B_j < C_j < C_i < A_j < A_i$ or $B_i < B_j < C_j < A_j < C_i < A_i$.

Anyway we get:

$$B_k < C_l \text{ for any } 1 \leq k, l \leq N, \tag{6}$$
$$B_k < A_l \text{ for any } 1 \leq k, l \leq N, \tag{7}$$
$$C_k < A_l \text{ for any } 1 \leq l \leq k \leq N, \tag{8}$$
$$A_k < A_l \text{ for any } 1 \leq l < k \leq N, \tag{9}$$

Problem A. Arrays

So we can rearrange $A_i, B_i, C_i, A_j, B_j, C_j$ without decreasing S in such way that $B_i < B_j < C_j < C_i < A_j < A_i$ or $B_i < B_j < C_j < A_j < C_i < A_i$.

Anyway we get:

$$B_k < C_l \text{ for any } 1 \le k, l \le N, \tag{6}$$

$$B_k < A_l \text{ for any } 1 \le k, l \le N, \tag{7}$$

$$C_k < A_l \text{ for any } 1 \le l \le k \le N, \tag{8}$$

$$A_k < A_l \text{ for any } 1 \le l < k \le N, \tag{9}$$

$$C_k < C_l \text{ for any } 1 \le l < k \le N.$$

Problem A. Arrays

So we can rearrange $A_i, B_i, C_i, A_j, B_j, C_j$ without decreasing S in such way that $B_i < B_j < C_j < C_i < A_j < A_i$ or
$B_i < B_j < C_j < A_j < C_i < A_i$.

Anyway we get:

$$B_k < C_l \text{ for any } 1 \le k, l \le N, \tag{6}$$
$$B_k < A_l \text{ for any } 1 \le k, l \le N, \tag{7}$$
$$C_k < A_l \text{ for any } 1 \le l \le k \le N, \tag{8}$$
$$A_k < A_l \text{ for any } 1 \le l < k \le N, \tag{9}$$
$$C_k < C_l \text{ for any } 1 \le l < k \le N. \tag{10}$$

This implies that $B_i = i$, $A_1 = 3N$, $C_1 \ge 2N$.

Problem A. Arrays

Now we can find S using following recursive algorithm:

Problem A. Arrays

Now we can find S using following recursive algorithm:

- If $N = 0$ than $S = 0$.

Problem A. Arrays

Now we can find S using following recursive algorithm:

- If $N = 0$ than $S = 0$.
- Otherwise for $C_1 \in [2N, 3N-1]$ let $N' = N - 1$ and $\{X_i'\} = \{X_i\} \backslash \{X_{A_1}, X_{B_1}, X_{C_1}\}$ and find corresponding sum $S'(C_1)$.

Problem A. Arrays

Now we can find S using following recursive algorithm:

- If $N = 0$ than $S = 0$.
- Otherwise for $C_1 \in [2N, 3N-1]$ let $N' = N - 1$ and $\{X'_i\} = \{X_i\} \backslash \{X_{A_1}, X_{B_1}, X_{C_1}\}$ and find corresponding sum $S'(C_1)$.
- $S = \max\limits_{C_1} \left\{ S'(C_1) + (X_{A_1} - X_{B_1}) X_{C_1} \right\}$.

Problem A. Arrays

Now we can find S using following recursive algorithm:

- If $N = 0$ than $S = 0$.
- Otherwise for $C_1 \in [2N, 3N-1]$ let $N' = N - 1$ and $\{X'_i\} = \{X_i\} \backslash \{X_{A_1}, X_{B_1}, X_{C_1}\}$ and find corresponding sum $S'(C_1)$.
- $S = \max\limits_{C_1} \left\{ S'(C_1) + (X_{A_1} - X_{B_1}) X_{C_1} \right\}$.

This algorithm has $O(N!)$ complexity, that is much better than trivial one with complexity $O\big((3N)!\big)$, but it is still rather slow to be used as solution.

Problem A. Arrays

We can optimize this algorithm using bit masks. Let's describe recursive function taking values i, that is index of our sequences we consider at the moment, and M, that is current bit mask of used numbers from $[N+1, 3N]$:

Problem A. Arrays

We can optimize this algorithm using bit masks. Let's describe recursive function taking values i, that is index of our sequences we consider at the moment, and M, that is current bit mask of used numbers from $[N+1, 3N]$:

- If $i = N+1$ return zero.

Problem A. Arrays

We can optimize this algorithm using bit masks. Let's describe recursive function taking values i, that is index of our sequences we consider at the moment, and M, that is current bit mask of used numbers from $[N+1, 3N]$:

- If $i = N + 1$ return zero.
- If result was calculated for current mask M return this result.

Problem A. Arrays

We can optimize this algorithm using bit masks. Let's describe recursive function taking values i, that is index of our sequences we consider at the moment, and M, that is current bit mask of used numbers from $[N+1, 3N]$:

- If $i = N+1$ return zero.
- If result was calculated for current mask M return this result.
- Otherwise let A_i be the largest number such that corresponding bit in current mask M is clear (since (8) and (9)).

Problem A. Arrays

We can optimize this algorithm using bit masks. Let's describe recursive function taking values i, that is index of our sequences we consider at the moment, and M, that is current bit mask of used numbers from $[N+1, 3N]$:

- If $i = N + 1$ return zero.
- If result was calculated for current mask M return this result.
- Otherwise let A_i be the largest number such that corresponding bit in current mask M is clear (since (8) and (9)).
- Let C_i be any number such that corresponding bit in current mask M is clear and there are at least $N - i$ numbers less than C_i with corresponding bit clear (since (10)).

Problem A. Arrays

We can optimize this algorithm using bit masks. Let's describe recursive function taking values i, that is index of our sequences we consider at the moment, and M, that is current bit mask of used numbers from $[N+1, 3N]$:

- If $i = N + 1$ return zero.
- If result was calculated for current mask M return this result.
- Otherwise let A_i be the largest number such that corresponding bit in current mask M is clear (since (8) and (9)).
- Let C_i be any number such that corresponding bit in current mask M is clear and there are at least $N - i$ numbers less than C_i with corresponding bit clear (since (10)).
- For each possible C_i set bits corresponding to A_i and C_i in M, get recursively result $S'(C_i)$ for $i' = i + 1$, $M' = M$, then clear bits corresponding to A_i and C_i.

Problem A. Arrays

We can optimize this algorithm using bit masks. Let's describe recursive function taking values i, that is index of our sequences we consider at the moment, and M, that is current bit mask of used numbers from $[N+1, 3N]$:

- If $i = N+1$ return zero.
- If result was calculated for current mask M return this result.
- Otherwise let A_i be the largest number such that corresponding bit in current mask M is clear (since (8) and (9)).
- Let C_i be any number such that corresponding bit in current mask M is clear and there are at least $N-i$ numbers less than C_i with corresponding bit clear (since (10)).
- For each possible C_i set bits corresponding to A_i and C_i in M, get recursively result $S'(C_i)$ for $i' = i+1$, $M' = M$, then clear bits corresponding to A_i and C_i.
- Return $\max\limits_{C_1} \left\{ S'(C_1) + (X_{A_i} - X_{B_i}) X_{C_i} \right\}$.

Problem A. Arrays

We can optimize this algorithm using bit masks. Let's describe recursive function taking values i, that is index of our sequences we consider at the moment, and M, that is current bit mask of used numbers from $[N+1, 3N]$:

- If $i = N+1$ return zero.
- If result was calculated for current mask M return this result.
- Otherwise let A_i be the largest number such that corresponding bit in current mask M is clear (since (8) and (9)).
- Let C_i be any number such that corresponding bit in current mask M is clear and there are at least $N-i$ numbers less than C_i with corresponding bit clear (since (10)).
- For each possible C_i set bits corresponding to A_i and C_i in M, get recursively result $S'(C_i)$ for $i' = i+1$, $M' = M$, then clear bits corresponding to A_i and C_i.
- Return $\max\limits_{C_1}\left\{S'(C_1) + (X_{A_i} - X_{B_i})X_{C_i}\right\}$.

This algorithm has $O\left(N \cdot 2^{2N}\right)$ complexity, that is better, but still not enough to be accepted.

Problem A. Arrays

Running ahead we can say that this algorithm is much better than it seems now.

Problem A. Arrays

Running ahead we can say that this algorithm is much better than it seems now. Really it has complexity $O\left(N \cdot 2^N\right)$ which seems to be enough for solving this task.

Problem A. Arrays

Running ahead we can say that this algorithm is much better than it seems now. Really it has complexity $O\left(N \cdot 2^N\right)$ which seems to be enough for solving this task. However, memorizing results requires some complicated structures (like `map<long long, int>` or `Map<Long, Integer>`) since mask can be up to $2^{50} - 1$ and this structures are far too slow to keep within given time.

Problem A. Arrays

Let's consider more carefully bit masks we deal with.

Problem A. Arrays

Let's consider more carefully bit masks we deal with.

If we choosed $A_1, \cdots, A_i$ and $C_1, \cdots, C_i$, then:

Problem A. Arrays

Let's consider more carefully bit masks we deal with.

If we choosed $A_1, \cdots, A_i$ and $C_1, \cdots, C_i$, then:

- Bits corresponding to numbers $\{3N - i + 1, \cdots, 3N\}$ are set since every time we choose the number corresponding to highest clear bit as A_j.

Problem A. Arrays

Let's consider more carefully bit masks we deal with.

If we choosed $A_1, \cdots, A_i$ and $C_1, \cdots, C_i$, then:

- Bits corresponding to numbers $\{3N - i + 1, \cdots, 3N\}$ are set since every time we choose the number corresponding to highest clear bit as A_j.
- Bits corresponding to numbers $\{N + 1, \cdots, 2N - i\}$ are clear since we left this numbers intact every time we choose C_j.

Problem A. Arrays

Let's consider more carefully bit masks we deal with.

If we choosed $A_1, \cdots, A_i$ and $C_1, \cdots, C_i$, then:

- Bits corresponding to numbers $\{3N - i + 1, \cdots, 3N\}$ are set since every time we choose the number corresponding to highest clear bit as A_j.
- Bits corresponding to numbers $\{N + 1, \cdots, 2N - i\}$ are clear since we left this numbers intact every time we choose C_j.
- Finally there are exactly $2i$ bits set.

Problem A. Arrays

Let's consider more carefully bit masks we deal with.

If we choosed $A_1, \cdots, A_i$ and $C_1, \cdots, C_i$, then:

- Bits corresponding to numbers $\{3N - i + 1, \cdots, 3N\}$ are set since every time we choose the number corresponding to highest clear bit as A_j.
- Bits corresponding to numbers $\{N + 1, \cdots, 2N - i\}$ are clear since we left this numbers intact every time we choose C_j.
- Finally there are exactly $2i$ bits set.

So we can resore i and mask using only N bits from mask:

$$\underbrace{1\ldots1}_{i}\underbrace{\ldots\ldots}_{N}\underbrace{0\ldots0}_{N-i}$$

Problem A. Arrays

Let's consider more carefully bit masks we deal with.

If we choosed $A_1, \cdots, A_i$ and $C_1, \cdots, C_i$, then:

- Bits corresponding to numbers $\{3N - i + 1, \cdots, 3N\}$ are set since every time we choose the number corresponding to highest clear bit as A_j.
- Bits corresponding to numbers $\{N + 1, \cdots, 2N - i\}$ are clear since we left this numbers intact every time we choose C_j.
- Finally there are exactly $2i$ bits set.

So we can resore i and mask using only N bits from mask:

$$\underbrace{1\ldots1}_{i}\underbrace{\ldots\ldots}_{N}\underbrace{0\ldots0}_{N-i}$$

- To restore i we can calculate number of set bits among this N.

Problem A. Arrays

Let's consider more carefully bit masks we deal with.

If we choosed $A_1, \cdots, A_i$ and $C_1, \cdots, C_i$, then:

- Bits corresponding to numbers $\{3N - i + 1, \cdots, 3N\}$ are set since every time we choose the number corresponding to highest clear bit as A_j.
- Bits corresponding to numbers $\{N + 1, \cdots, 2N - i\}$ are clear since we left this numbers intact every time we choose C_j.
- Finally there are exactly $2i$ bits set.

So we can resore i and mask using only N bits from mask:

$$\underbrace{1 \ldots 1}_{i}\underbrace{\ldots\ldots}_{N}\underbrace{0 \ldots 0}_{N-i}$$

- To restore i we can calculate number of set bits among this N.
- To restore mask we can pad this N bits with i ones from the left side and with $N - i$ zeros from the right side.

Problem A. Arrays

Thus we proved that our algorithm operates with at most 2^N different masks (therefore it has complexity $O\left(N \cdot 2^N\right)$) and showed efficient way to store result for each mask.

Problem A. Arrays

Thus we proved that our algorithm operates with at most 2^N different masks (therefore it has complexity $O\left(N \cdot 2^N\right)$) and showed efficient way to store result for each mask.

There are some more optimizations (avoiding recursion and 64bit numbers and some others), but described algorithm was good enough.

Problem B. “Bulls and Cows”

Idea:	Serge Kashkevich
Solutions:	Pavel Irzhavski, Serge Kashkevich, Vladimir Kerus, Ivan Metelsky
Tests:	Ivan Metelsky
Analysis:	Pavel Irzhavski
Statement:	Serge Kashkevich

Problem B. “Bulls and Cows”

It is more convinient to work with strings than with numbers.

Problem B. “Bulls and Cows”

It is more convinient to work with strings than with numbers. All you need is comparison of every symbol in first string with every symbol in second one.

Problem B. “Bulls and Cows”

It is more convinient to work with strings than with numbers. All you need is comparison of every symbol in first string with every symbol in second one. It may be done without any loop.

Problem B. “Bulls and Cows”

It is more convinient to work with strings than with numbers. All you need is comparison of every symbol in first string with every symbol in second one. It may be done without any loop. And even without any conditional construction.

Problem C. Courier

Idea:	Yuri Orlovich
Solutions:	Pavel Irzhavski, Serge Kashkevich, Ivan Metelsky
Tests:	Ivan Metelsky
Analysis:	Pavel Irzhavski
Statement:	Serge Kashkevich
Checker:	Serge Kashkevich

acm International Collegiate Programming Contest

Problem C. Courier

If there is a city that can't be reached from capital, then the answer is obviously "`No`".

Problem C. Courier

If there is a city that can't be reached from capital, then the answer is obviously "`No`". Otherwise let's construct desired path.

Problem C. Courier

If there is a city that can't be reached from capital, then the answer is obviously "`No`". Otherwise let's construct desired path.

Consider graph with cities being vertices and roads between them being edges (edges may be multiple).

Problem C. Courier

If there is a city that can't be reached from capital, then the answer is obviously "`No`". Otherwise let's construct desired path.

Consider graph with cities being vertices and roads between them being edges (edges may be multiple). This graph is connected and each vertex has even degree (to be more precise, degree of each vertex is 4).

Problem C. Courier

If there is a city that can't be reached from capital, then the answer is obviously "No". Otherwise let's construct desired path.

Consider graph with cities being vertices and roads between them being edges (edges may be multiple). This graph is connected and each vertex has even degree (to be more precise, degree of each vertex is 4).

So we can construct an Euler cycle in this graph.

Problem C. Courier

If there is a city that can't be reached from capital, then the answer is obviously "`No`". Otherwise let's construct desired path.

Consider graph with cities being vertices and roads between them being edges (edges may be multiple). This graph is connected and each vertex has even degree (to be more precise, degree of each vertex is 4).

So we can construct an Euler cycle in this graph. Then we can go round this cycle from capital and for each visited edge put in two gate numbers in the answer (the first one is the gate we left previous city through and the second one is the gate we entered next city through).

Problem C. Courier

If there is a city that can't be reached from capital, then the answer is obviously "`No`". Otherwise let's construct desired path.

Consider graph with cities being vertices and roads between them being edges (edges may be multiple). This graph is connected and each vertex has even degree (to be more precise, degree of each vertex is 4).

So we can construct an Euler cycle in this graph. Then we can go round this cycle from capital and for each visited edge put in two gate numbers in the answer (the first one is the gate we left previous city through and the second one is the gate we entered next city through).

This solution has $O(N)$ complexity, but less efficient approaches (like $O\left(N^2\right)$) were allowed.

Problem D. Dales and Hills

Idea:	Serge Kashkevich
Solutions:	Serge Kashkevich, Ivan Metelsky, Vladimir Kerus, Pavel Irzhavski
Tests:	Ivan Metelsky
Analysis:	Pavel Irzhavski
Statement:	Serge Kashkevich

Problem D. Dales and Hills

Let b_j be the least possible $i \geq 1$ such that $a_t < a_{t+1}$ for any $i \leq t < j$. We can calculate all b_j in following way:

Problem D. Dales and Hills

Let b_j be the least possible $i \geq 1$ such that $a_t < a_{t+1}$ for any $i \leq t < j$. We can calculate all b_j in following way:

- $b_1 = 1$.

Problem D. Dales and Hills

Let b_j be the least possible $i \geq 1$ such that $a_t < a_{t+1}$ for any $i \leq t < j$. We can calculate all b_j in following way:

- $b_1 = 1$.
- For j moving from 2 to N let $b_j = b_{j-1}$ if $a_j > a_{j-1}$ and $b_j = j$ otherwise.

Problem D. Dales and Hills

Let b_j be the least possible $i \geq 1$ such that $a_t < a_{t+1}$ for any $i \leq t < j$. We can calculate all b_j in following way:

- $b_1 = 1$.
- For j moving from 2 to N let $b_j = b_{j-1}$ if $a_j > a_{j-1}$ and $b_j = j$ otherwise.

Let c_j be the most possible $k \leq N$ such that $a_t > a_{t+1}$ for any $j \leq t < k$. We can calculate all c_j in following way:

Problem D. Dales and Hills

Let b_j be the least possible $i \geq 1$ such that $a_t < a_{t+1}$ for any $i \leq t < j$. We can calculate all b_j in following way:

- $b_1 = 1$.
- For j moving from 2 to N let $b_j = b_{j-1}$ if $a_j > a_{j-1}$ and $b_j = j$ otherwise.

Let c_j be the most possible $k \leq N$ such that $a_t > a_{t+1}$ for any $j \leq t < k$. We can calculate all c_j in following way:

- $c_N =$ N.

Problem D. Dales and Hills

Let b_j be the least possible $i \geq 1$ such that $a_t < a_{t+1}$ for any $i \leq t < j$. We can calculate all b_j in following way:

- $b_1 = 1$.
- For j moving from 2 to N let $b_j = b_{j-1}$ if $a_j > a_{j-1}$ and $b_j = j$ otherwise.

Let c_j be the most possible $k \leq N$ such that $a_t > a_{t+1}$ for any $j \leq t < k$. We can calculate all c_j in following way:

- $c_N =$ N.
- For j moving from $N-1$ to 1 let $c_j = c_{j+1}$ if $a_j > a_{j+1}$ and $c_j = j$ otherwise.

Problem D. Dales and Hills

Let b_j be the least possible $i \geq 1$ such that $a_t < a_{t+1}$ for any $i \leq t < j$. We can calculate all b_j in following way:

- $b_1 = 1$.
- For j moving from 2 to N let $b_j = b_{j-1}$ if $a_j > a_{j-1}$ and $b_j = j$ otherwise.

Let c_j be the most possible $k \leq N$ such that $a_t > a_{t+1}$ for any $j \leq t < k$. We can calculate all c_j in following way:

- $c_N = \mathrm{N}$.
- For j moving from $N - 1$ to 1 let $c_j = c_{j+1}$ if $a_j > a_{j+1}$ and $c_j = j$ otherwise.

Let's call a_j a *peak* of a hill $a_i, \cdots, a_j, \cdots, a_k$ if $a_t < a_{t+1}$ for any $i \leq t < j$ and $a_t > a_{t+1}$ for any $j \leq t < k$.

Problem D. Dales and Hills

Let b_j be the least possible $i \geq 1$ such that $a_t < a_{t+1}$ for any $i \leq t < j$. We can calculate all b_j in following way:

- $b_1 = 1$.
- For j moving from 2 to N let $b_j = b_{j-1}$ if $a_j > a_{j-1}$ and $b_j = j$ otherwise.

Let c_j be the most possible $k \leq N$ such that $a_t > a_{t+1}$ for any $j \leq t < k$. We can calculate all c_j in following way:

- $c_N =$ N.
- For j moving from $N-1$ to 1 let $c_j = c_{j+1}$ if $a_j > a_{j+1}$ and $c_j = j$ otherwise.

Let's call a_j a *peak* of a hill $a_i, \cdots, a_j, \cdots, a_k$ if $a_t < a_{t+1}$ for any $i \leq t < j$ and $a_t > a_{t+1}$ for any $j \leq t < k$.

Now for each a_j we can fastly determine, whether it is a peak of some hill and if so also the height of the highest possible hill with peak a_j.

Problem D. Dales and Hills

Let b_j be the least possible $i \geq 1$ such that $a_t < a_{t+1}$ for any $i \leq t < j$. We can calculate all b_j in following way:

- $b_1 = 1$.
- For j moving from 2 to N let $b_j = b_{j-1}$ if $a_j > a_{j-1}$ and $b_j = j$ otherwise.

Let c_j be the most possible $k \leq N$ such that $a_t > a_{t+1}$ for any $j \leq t < k$. We can calculate all c_j in following way:

- $c_N = \mathrm{N}$.
- For j moving from $N-1$ to 1 let $c_j = c_{j+1}$ if $a_j > a_{j+1}$ and $c_j = j$ otherwise.

Let's call a_j a *peak* of a hill $a_i, \cdots, a_j, \cdots, a_k$ if $a_t < a_{t+1}$ for any $i \leq t < j$ and $a_t > a_{t+1}$ for any $j \leq t < k$.

Now for each a_j we can fastly determine, whether it is a peak of some hill and if so also the height of the highest possible hill with peak a_j. More precisely a_j is peak of some hill if $b_j < a_j < c_j$ and value $\max\{a_j - b_j, c_j - a_j\}$ is the height of the highest hill with peak a_j.

Problem D. Dales and Hills

Let b_j be the least possible $i \geq 1$ such that $a_t < a_{t+1}$ for any $i \leq t < j$. We can calculate all b_j in following way:

- $b_1 = 1$.
- For j moving from 2 to N let $b_j = b_{j-1}$ if $a_j > a_{j-1}$ and $b_j = j$ otherwise.

Let c_j be the most possible $k \leq N$ such that $a_t > a_{t+1}$ for any $j \leq t < k$. We can calculate all c_j in following way:

- $c_N = \text{N}$.
- For j moving from $N-1$ to 1 let $c_j = c_{j+1}$ if $a_j > a_{j+1}$ and $c_j = j$ otherwise.

Let's call a_j a *peak* of a hill $a_i, \cdots, a_j, \cdots, a_k$ if $a_t < a_{t+1}$ for any $i \leq t < j$ and $a_t > a_{t+1}$ for any $j \leq t < k$.

Now for each a_j we can fastly determine, whether it is a peak of some hill and if so also the height of the highest possible hill with peak a_j. More precisely a_j is peak of some hill if $b_j < a_j < c_j$ and value $\max\{a_j - b_j, c_j - a_j\}$ is the height of the highest hill with peak a_j. Thus we can calculate the height of the highest hill in linear time.

Problem D. Dales and Hills

The depth of the deepest dale can be calculated similarly.

Problem D. Dales and Hills

The depth of the deepest dale can be calculated similarly. To reduce code size we may let $a'_i = -a_i$. Then the height of the highest hill for sequence $a'_1, \cdots, a'_N$ is equal to the depth of the deepest dale for $a_1, \cdots, a_N$.

Problem E. Extremal Permutations

Idea: Pavel Irzhavski

Solutions: Pavel Irzhavski, Ivan Metelsky, Vladimir Kerus

Tests: Pavel Irzhavski

Analysis: Pavel Irzhavski

Statement: Pavel Irzhavski

Problem E. Extremal Permutations

Number of such permutations that $p_1 < p_2 > p_3 < \cdots$ is equal to number of permutations with $p_1 > p_2 < p_3 > \cdots$.

Problem E. Extremal Permutations

Number of such permutations that $p_1 < p_2 > p_3 < \cdots$ is equal to number of permutations with $p_1 > p_2 < p_3 > \cdots$. Permutations of both types are extremal and for $n > 1$ any extremal permutation is permutation of exactly one of this types.

Problem E. Extremal Permutations

Number of such permutations that $p_1 < p_2 > p_3 < \cdots$ is equal to number of permutations with $p_1 > p_2 < p_3 > \cdots$. Permutations of both types are extremal and for $n > 1$ any extremal permutation is permutation of exactly one of this types. Let's call permatations of first type *good*.

Problem E. Extremal Permutations

Number of such permutations that $p_1 < p_2 > p_3 < \cdots$ is equal to number of permutations with $p_1 > p_2 < p_3 > \cdots$. Permutations of both types are extremal and for $n > 1$ any extremal permutation is permutation of exactly one of this types. Let's call permatations of first type *good*.

We can find number of good permutations and multiply this number by 2 to get number of extremal permutations (this doesn't hold only when $n = 1$).

Problem E. Extremal Permutations

Number of such permutations that $p_1 < p_2 > p_3 < \cdots$ is equal to number of permutations with $p_1 > p_2 < p_3 > \cdots$. Permutations of both types are extremal and for $n > 1$ any extremal permutation is permutation of exactly one of this types. Let's call permatations of first type *good*.

We can find number of good permutations and multiply this number by 2 to get number of extremal permutations (this doesn't hold only when $n = 1$).

Let $F(n, k)$ be number of good permutations with $p_1 = k$.

Problem E. Extremal Permutations

Number of such permutations that $p_1 < p_2 > p_3 < \cdots$ is equal to number of permutations with $p_1 > p_2 < p_3 > \cdots$. Permutations of both types are extremal and for $n > 1$ any extremal permutation is permutation of exactly one of this types. Let's call permatations of first type *good*.

We can find number of good permutations and multiply this number by 2 to get number of extremal permutations (this doesn't hold only when $n = 1$).

Let $F(n, k)$ be number of good permutations with $p_1 = k$.

Let $p'_{i-1} = p_i - [p_i > p_1]$ for $2 \le i \le n$, that is $p'_{i-1} = p_i$ if $p_i < p_1$ and $p'_{i-1} = p_i - 1$ if $p_i > p_1$.

Problem E. Extremal Permutations

Number of such permutations that $p_1 < p_2 > p_3 < \cdots$ is equal to number of permutations with $p_1 > p_2 < p_3 > \cdots$. Permutations of both types are extremal and for $n > 1$ any extremal permutation is permutation of exactly one of this types. Let's call permatations of first type *good*.

We can find number of good permutations and multiply this number by 2 to get number of extremal permutations (this doesn't hold only when $n = 1$).

Let $F(n, k)$ be number of good permutations with $p_1 = k$.

Let $p'_{i-1} = p_i - [p_i > p_1]$ for $2 \le i \le n$, that is $p'_{i-1} = p_i$ if $p_i < p_1$ and $p'_{i-1} = p_i - 1$ if $p_i > p_1$. Then $p'_1 > p'_2 < p'_3 > \cdots$, all $p'_1, p'_2, \cdots, p'_{n-1}$ are distinct and are in $\{1, \cdots, n-1\}$.

Problem E. Extremal Permutations

Number of such permutations that $p_1 < p_2 > p_3 < \cdots$ is equal to number of permutations with $p_1 > p_2 < p_3 > \cdots$. Permutations of both types are extremal and for $n > 1$ any extremal permutation is permutation of exactly one of this types. Let's call permatations of first type *good*.

We can find number of good permutations and multiply this number by 2 to get number of extremal permutations (this doesn't hold only when $n = 1$).

Let $F(n, k)$ be number of good permutations with $p_1 = k$.

Let $p'_{i-1} = p_i - [p_i > p_1]$ for $2 \le i \le n$, that is $p'_{i-1} = p_i$ if $p_i < p_1$ and $p'_{i-1} = p_i - 1$ if $p_i > p_1$. Then $p'_1 > p'_2 < p'_3 > \cdots$, all $p'_1, p'_2, \cdots, p'_{n-1}$ are distinct and are in $\{1, \cdots, n-1\}$. Let $p''_i = n - p'_i$ for $1 \le i \le n$. Then $p''_1 < p''_2 > \cdots$, all $p''_1, \cdots, p''_{n-1}$ are distinct and are in $\{1, \cdots, n-1\}$, thus this is good permutation.

Problem E. Extremal Permutations

Number of such permutations that $p_1 < p_2 > p_3 < \cdots$ is equal to number of permutations with $p_1 > p_2 < p_3 > \cdots$. Permutations of both types are extremal and for $n > 1$ any extremal permutation is permutation of exactly one of this types. Let's call permatations of first type *good*.

We can find number of good permutations and multiply this number by 2 to get number of extremal permutations (this doesn't hold only when $n = 1$).

Let $F(n, k)$ be number of good permutations with $p_1 = k$.

Let $p'_{i-1} = p_i - [p_i > p_1]$ for $2 \le i \le n$, that is $p'_{i-1} = p_i$ if $p_i < p_1$ and $p'_{i-1} = p_i - 1$ if $p_i > p_1$. Then $p'_1 > p'_2 < p'_3 > \cdots$, all $p'_1, p'_2, \cdots, p'_{n-1}$ are distinct and are in $\{1, \cdots, n-1\}$. Let $p''_i = n - p'_i$ for $1 \le i \le n$. Then $p''_1 < p''_2 > \cdots$, all $p''_1, \cdots, p''_{n-1}$ are distinct and are in $\{1, \cdots, n-1\}$, thus this is good permutation. If $p_1 = k$ then $p''_1 \le n - k$. Number of such permutation with $p''_1 = l$ is $F(n-1, l)$.

Problem E. Extremal Permutations

For fixed k there is one-to-one correspondence between good permutations of $1, \cdots, n$ with $p_1 = k$ and good permutations of $1, \cdots, n-1$ with $p_1 \geq n-k$.

Problem E. Extremal Permutations

For fixed k there is one-to-one correspondence between good permutations of $1, \cdots, n$ with $p_1 = k$ and good permutations of $1, \cdots, n-1$ with $p_1 \geq n-k$. Thus we have:

$$F(n,k) = \sum_{l=1}^{n-k} F(n-1,l) \quad \text{for any } 2 \leq n \text{ and } 1 \leq k \leq n.$$

Problem E. Extremal Permutations

For fixed k there is one-to-one correspondence between good permutations of $1, \cdots, n$ with $p_1 = k$ and good permutations of $1, \cdots, n-1$ with $p_1 \geq n-k$. Thus we have:

$$F(n,k) = \sum_{l=1}^{n-k} F(n-1,l) \quad \text{for any } 2 \leq n \text{ and } 1 \leq k \leq n.$$

To make solutioin more efficient we can notice that:

$$F(n,k) = \sum_{l=1}^{n-k} F(n-1,l) = F(n-1,n-k) + \sum_{l=1}^{n-k-1} F(n-1,l) =$$
$$= F(n-1,n-k) + F(n,k+1) \quad \text{for } 2 \leq n \text{ and } 1 \leq k < n.$$

Problem E. Extremal Permutations

For fixed k there is one-to-one correspondence between good permutations of $1, \cdots, n$ with $p_1 = k$ and good permutations of $1, \cdots, n-1$ with $p_1 \geq n-k$. Thus we have:

$$F(n,k) = \sum_{l=1}^{n-k} F(n-1,l) \quad \text{for any } 2 \leq n \text{ and } 1 \leq k \leq n.$$

To make solutioin more efficient we can notice that:

$$F(n,k) = \sum_{l=1}^{n-k} F(n-1,l) = F(n-1,n-k) + \sum_{l=1}^{n-k-1} F(n-1,l) =$$
$$= F(n-1,n-k) + F(n,k+1) \quad \text{for } 2 \leq n \text{ and } 1 \leq k < n.$$

Taking into account that $F(n,n) = 0$ if $n > 1$ and $F(1,1) = 1$ we get full solution with $O\left(N^2\right)$ complexity.

Problem E. Extremal Permutations

For fixed k there is one-to-one correspondence between good permutations of $1, \cdots, n$ with $p_1 = k$ and good permutations of $1, \cdots, n-1$ with $p_1 \geq n-k$. Thus we have:

$$F(n,k) = \sum_{l=1}^{n-k} F(n-1,l) \quad \text{for any } 2 \leq n \text{ and } 1 \leq k \leq n.$$

To make solutioin more efficient we can notice that:

$$F(n,k) = \sum_{l=1}^{n-k} F(n-1,l) = F(n-1,n-k) + \sum_{l=1}^{n-k-1} F(n-1,l) =$$
$$= F(n-1,n-k) + F(n,k+1) \quad \text{for } 2 \leq n \text{ and } 1 \leq k < n.$$

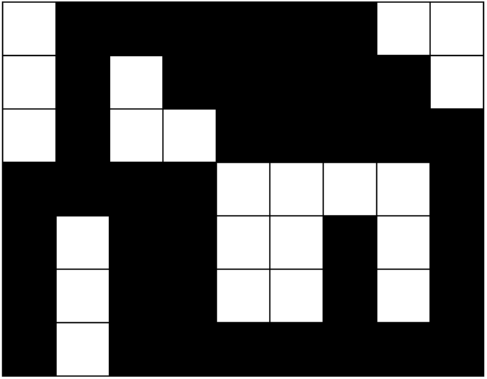
Taking into account that $F(n,n) = 0$ if $n > 1$ and $F(1,1) = 1$ we get full solution with $O\left(N^2\right)$ complexity.
There are more approaches with $O\left(N^2\right)$ complexity, but some of them use modulo multiplication instead of modulo addition and thus require about 10-15 times more time. Such approaches were rejected.

Problem F. Figure and Spots

Idea:	Ivan Metelsky
Solutions:	Ivan Metelsky, Pavel Irzhavski, Serge Kashkevich
Tests:	Ivan Metelsky
Analysis:	Ivan Metelsky
Statement:	Ivan Metelsky
Checker:	Ivan Metelsky

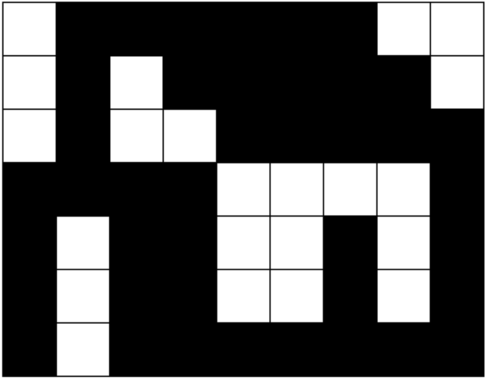
Problem F. Figure and Spots

First of all, we describe two transformations that, given a figure, allow to simplify it greatly, but preserving the figure's width, height and number of spots.



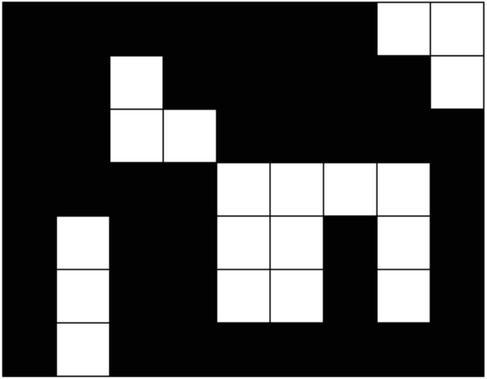
Problem F. Figure and Spots

If you consider a figure within its bounding rectangle, you see several white components that could be spots, but aren't really spots because they touch the rectangle's borders. On the picture below there are 3 of them. As they are not spots, we can fill them with black color without changing any important parameters of the figure. This is our first transformation.



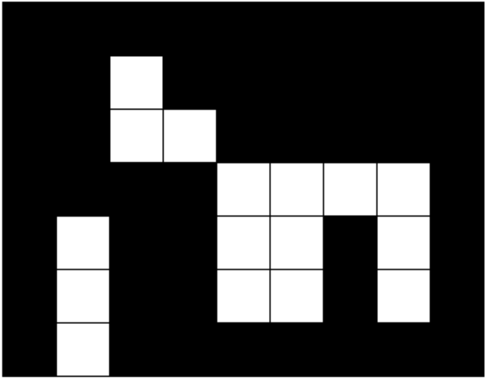
Problem F. Figure and Spots

If you consider a figure within its bounding rectangle, you see several white components that could be spots, but aren't really spots because they touch the rectangle's borders. On the picture below there are 3 of them. As they are not spots, we can fill them with black color without changing any important parameters of the figure. This is our first transformation.



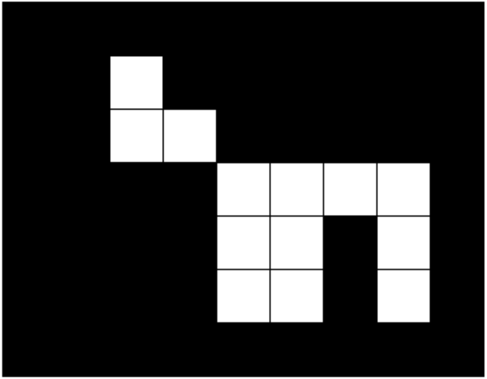
Problem F. Figure and Spots

If you consider a figure within its bounding rectangle, you see several white components that could be spots, but aren't really spots because they touch the rectangle's borders. On the picture below there are 3 of them. As they are not spots, we can fill them with black color without changing any important parameters of the figure. This is our first transformation.



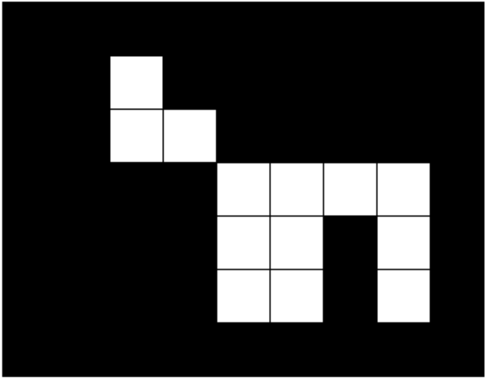
Problem F. Figure and Spots

If you consider a figure within its bounding rectangle, you see several white components that could be spots, but aren't really spots because they touch the rectangle's borders. On the picture below there are 3 of them. As they are not spots, we can fill them with black color without changing any important parameters of the figure. This is our first transformation.



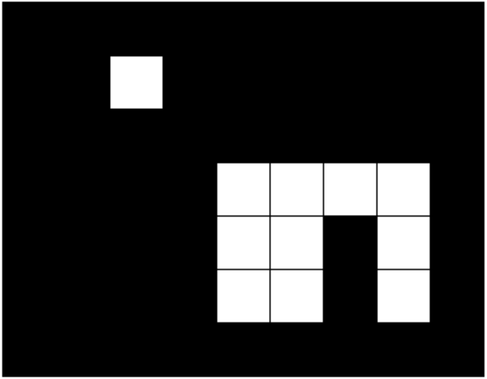
Problem F. Figure and Spots

Now comes the second transformation. Let's consider our spots and replace each spot by a single white cell (among all cells of a spot we choose any one and leave only this cell). It's easy to see that this transformation also leaves the important parameters of the figure unchanged.



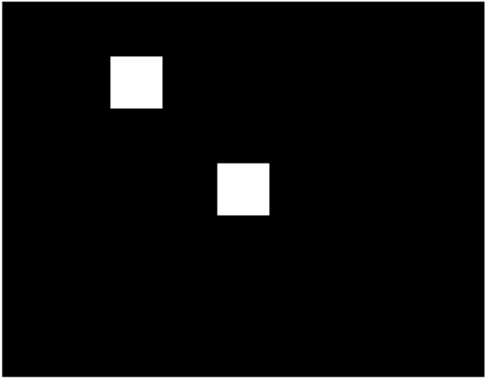
Problem F. Figure and Spots

Now comes the second transformation. Let's consider our spots and replace each spot by a single white cell (among all cells of a spot we choose any one and leave only this cell). It's easy to see that this transformation also leaves the important parameters of the figure unchanged.



Problem F. Figure and Spots

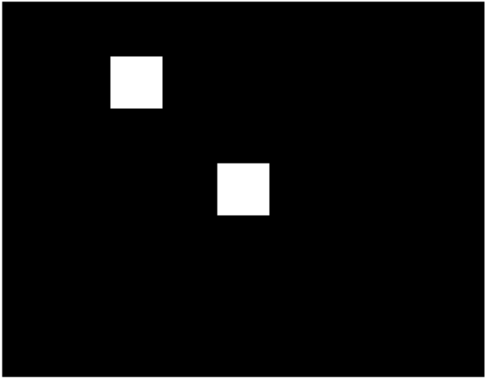
Now comes the second transformation. Let's consider our spots and replace each spot by a single white cell (among all cells of a spot we choose any one and leave only this cell). It's easy to see that this transformation also leaves the important parameters of the figure unchanged.



Problem F. Figure and Spots

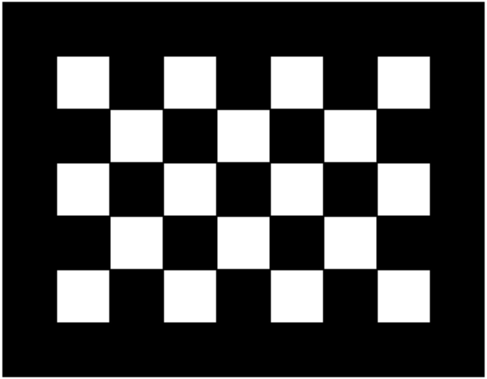
So without loss of generality we can consider only the figures with the following two properties:

1. The bounding rectangle's border is completely black.
2. Each spot consists of a single white cell.



Problem F. Figure and Spots

Under these conditions, it's easy to see how to fit the maximum number of spots into a figure with the given width and height. The border must be completely black and the inner cells must be filled according to a chessboard pattern shown below.



Problem F. Figure and Spots

Let's denote the maximum number of spots that can be fit for given W and H as M.

Problem F. Figure and Spots

Let's denote the maximum number of spots that can be fit for given W and H as M.

If $N = M$, the solution looks like described on the previous slide.

Problem F. Figure and Spots

Let's denote the maximum number of spots that can be fit for given W and H as M.

If $N = M$, the solution looks like described on the previous slide.

If $N > M$, there's no solution at all.

Problem F. Figure and Spots

Let's denote the maximum number of spots that can be fit for given W and H as M.

If $N = M$, the solution looks like described on the previous slide.

If $N > M$, there's no solution at all.

If $N < M$, the solution can be constructed as follows. First, take a solution with M spots and then remove any $M - N$ of them.

Problem F. Figure and Spots

The solution has linear complexity $O(W \cdot H)$.

Problem F. Figure and Spots

The solution has linear complexity $O(W \cdot H)$.

The constraints were deliberately set to low value in order to make it a little bit less obvious that the problem can be solved with a simple linear algorithm.

Problem G. Game for Little Johnny

Idea:	Ivan Metelsky, Pavel Irzhavski
Solutions:	Ivan Metelsky, Pavel Irzhavski
Tests:	Pavel Irzhavski
Analysis:	Ivan Metelsky
Statement:	Ivan Metelsky

Problem G. Game for Little Johnny

The first part of our solution is to find all divisors for all integers between 1 and N.

Problem G. Game for Little Johnny

The first part of our solution is to find all divisors for all integers between 1 and N.

The fastest way to do it is modified Eratosthenes Sieve algorithm.

Problem G. Game for Little Johnny

The first part of our solution is to find all divisors for all integers between 1 and N.

The fastest way to do it is modified Eratosthenes Sieve algorithm.

Let $DivList_i$ be the list of all divisors for integer i. Initially all these lists are empty.

Problem G. Game for Little Johnny

The first part of our solution is to find all divisors for all integers between 1 and N.

The fastest way to do it is modified Eratosthenes Sieve algorithm.

Let $DivList_i$ be the list of all divisors for integer i. Initially all these lists are empty.

For each $i = 1, 2, ..., N$ we iterate through all j such that i is a divisor of j. These j are i, $2i$, $3i$, $\cdots$, $k \cdot i$, where $k = \left\lfloor \frac{N}{i} \right\rfloor$. For all these j we add i to the end of $DivList_j$.

Problem G. Game for Little Johnny

Let's estimate the complexity of this step.

Problem G. Game for Little Johnny

Let's estimate the complexity of this step.

For each i we perform $\frac{N}{i}$ simple operations. So the total number of operations is:

$$\frac{N}{1} + \frac{N}{2} + ... + \frac{N}{N}$$

acm International Collegiate Programming Contest

Problem G. Game for Little Johnny

Let's estimate the complexity of this step.

For each i we perform $\frac{N}{i}$ simple operations. So the total number of operations is:

$$\frac{N}{1} + \frac{N}{2} + ... + \frac{N}{N} = N \cdot \left(\frac{1}{1} + \frac{1}{2} + ... + \frac{1}{N} \right)$$

Problem G. Game for Little Johnny

Let's estimate the complexity of this step.

For each i we perform $\frac{N}{i}$ simple operations. So the total number of operations is:

$$\frac{N}{1} + \frac{N}{2} + \ldots + \frac{N}{N} = N \cdot \left(\frac{1}{1} + \frac{1}{2} + \ldots + \frac{1}{N}\right) =$$
$$= N \cdot \left(\ln N + O(1)\right)$$

Problem G. Game for Little Johnny

Let's estimate the complexity of this step.

For each i we perform $\frac{N}{i}$ simple operations. So the total number of operations is:

$$\frac{N}{1} + \frac{N}{2} + ... + \frac{N}{N} = N \cdot \left(\frac{1}{1} + \frac{1}{2} + ... + \frac{1}{N}\right) =$$
$$= N \cdot \big(\ln N + O(1)\big) = O(N \cdot \log N)$$

Problem G. Game for Little Johnny

Now comes the solution itself. We consider all possible values of A, $1 \leq A \leq N$.

Problem G. Game for Little Johnny

Now comes the solution itself. We consider all possible values of A, $1 \leq A \leq N$.

For a given A, what values of B result in valid pairs (A, B)?

Problem G. Game for Little Johnny

Now comes the solution itself. We consider all possible values of A, $1 \leq A \leq N$.

For a given A, what values of B result in valid pairs (A, B)? Suppose that (A, B) is a valid pair. Then there are some $x, y \geq 1$ such that

$$A \cdot x + B \cdot y = N$$

Problem G. Game for Little Johnny

Now comes the solution itself. We consider all possible values of A, $1 \le A \le N$.

For a given A, what values of B result in valid pairs (A, B)? Suppose that (A, B) is a valid pair. Then there are some $x, y \ge 1$ such that

$$A \cdot x + B \cdot y = N$$

Let's express B from this equation:

$$B = \frac{N - A \cdot x}{y}$$

Problem G. Game for Little Johnny

So we see that B forms a valid pair with A if and only if $A < B$ and B is a divisor of $N - A \cdot x$.

Problem G. Game for Little Johnny

So we see that B forms a valid pair with A if and only if $A < B$ and B is a divisor of $N - A \cdot x$.

In order to calculate the number of B values for a given A, we check numbers $N - A$, $N - 2A$, $\cdots$, $N - k \cdot A$, where $k = \left\lfloor \frac{N}{A} \right\rfloor$, and select all their divisors that are greater than A.

Problem G. Game for Little Johnny

So we see that B forms a valid pair with A if and only if $A < B$ and B is a divisor of $N - A \cdot x$.

In order to calculate the number of B values for a given A, we check numbers $N - A$, $N - 2A$, $\cdots$, $N - k \cdot A$, where $k = \left\lfloor \frac{N}{A} \right\rfloor$, and select all their divisors that are greater than A.

The number of distinct divisors gives the number of valid (A, B) pairs for this A and must be added to the result.

Problem G. Game for Little Johnny

In order to select only distinct divisors efficiently, we can use an array of integers $last[1..N]$. The idea is to store in $last[i]$ the maximum value of A found such that $(last[i], i)$ is a valid pair.

Problem G. Game for Little Johnny

In order to select only distinct divisors efficiently, we can use an array of integers $last[1..N]$. The idea is to store in $last[i]$ the maximum value of A found such that $(last[i], i)$ is a valid pair.

When we are iterating through the divisors for a given value of A and have a divisor d, we need to check the value of $last[d]$. If $last[d] < A$, we have another pair (A, d), so $last[d]$ must be set to A and result must be increased by 1. However, if $last[d] = A$, we have already considered such value of d for this A, so it must be ignored.

Problem G. Game for Little Johnny

Estimating complexity of this solution is not trivial. It's not hard to see that we check divisors for $O(N \cdot \log N)$ numbers (argumentation is the same as we had when analyzing the Eratosthenes Sieve algorithm).

Problem G. Game for Little Johnny

Estimating complexity of this solution is not trivial. It's not hard to see that we check divisors for $O(N \cdot \log N)$ numbers (argumentation is the same as we had when analyzing the Eratosthenes Sieve algorithm).

When $N \leq 100\,000$, an integer between 1 and N can have at most 128 divisors, so we certainly have to check at most $128 \cdot N \cdot (\ln N + 1) < 150\,000\,000$ divisors and this is already not too much. Or course, the real working time is much smaller because most of numbers have much smaller amount of divisors.

Problem G. Game for Little Johnny

Estimating complexity of this solution is not trivial. It's not hard to see that we check divisors for $O(N \cdot \log N)$ numbers (argumentation is the same as we had when analyzing the Eratosthenes Sieve algorithm).

When $N \leq 100\,000$, an integer between 1 and N can have at most 128 divisors, so we certainly have to check at most $128 \cdot N \cdot (\ln N + 1) < 150\,000\,000$ divisors and this is already not too much. Or course, the real working time is much smaller because most of numbers have much smaller amount of divisors.

In average an integer between 1 and N has $O(\log N)$ divisors, and our solution tends to check smaller integers more frequently than larger integers (and of course, smaler integers have less divisors in average), so we expect the real working time for the solution to be $O\left(N \cdot \log^2 N\right)$.

Problem H. Hotel in Ves Lagos

Idea:	Rihards Opmanis
Solutions:	Rihards Opmanis, Pavel Irzhavski, Ivan Metelsky, Vladimir Kerus
Tests:	Ivan Metelsky
Analysis:	Pavel Irzhavski
Statement:	Ivan Metelsky

Problem H. Hotel in Ves Lagos

Suppose we can solve reverse problem. Given integer K, we have to calculate $F(K)$ that is quantity of correct room numbers in $[1, \cdots, K]$.

Problem H. Hotel in Ves Lagos

Suppose we can solve reverse problem. Given integer K, we have to calculate $F(K)$ that is quantity of correct room numbers in $[1, \cdots, K]$. Then since $F(K)$ is non-decreasing function we can solve initial problem using binary search to find the least such K that there are exactly N correct room numbers in $[1, \cdots, K]$.

Problem H. Hotel in Ves Lagos

Suppose we can solve reverse problem. Given integer K, we have to calculate $F(K)$ that is quantity of correct room numbers in $[1, \cdots, K]$. Then since $F(K)$ is non-decreasing function we can solve initial problem using binary search to find the least such K that there are exactly N correct room numbers in $[1, \cdots, K]$.

Let $G(K, i)$ be the quantity of numbers between 1 and K containing 13 as a substring at position i and not containing 13 in positions $0 \cdots i - 1$.

Problem H. Hotel in Ves Lagos

Suppose we can solve reverse problem. Given integer K, we have to calculate $F(K)$ that is quantity of correct room numbers in $[1, \cdots, K]$. Then since $F(K)$ is non-decreasing function we can solve initial problem using binary search to find the least such K that there are exactly N correct room numbers in $[1, \cdots, K]$.

Let $G(K, i)$ be the quantity of numbers between 1 and K containing 13 as a substring at position i and not containing 13 in positions $0 \cdots i - 1$. More formally $G(K, i)$ is number of such X that:

$$1 \le X \le K,$$
$$X \bmod 10^i \text{ doesn't contain } 13,$$
$$\left\lfloor \frac{X}{10^i} \right\rfloor \bmod 100 = 13$$

Problem H. Hotel in Ves Lagos

Suppose we can solve reverse problem. Given integer K, we have to calculate $F(K)$ that is quantity of correct room numbers in $[1, \cdots, K]$. Then since $F(K)$ is non-decreasing function we can solve initial problem using binary search to find the least such K that there are exactly N correct room numbers in $[1, \cdots, K]$.

Let $G(K, i)$ be the quantity of numbers between 1 and K containing 13 as a substring at position i and not containing 13 in positions $0 \cdots i-1$. More formally $G(K, i)$ is number of such X that:

$$1 \le X \le K,$$
$$X \bmod 10^i \text{ doesn't contain } 13,$$
$$\left\lfloor \frac{X}{10^i} \right\rfloor \bmod 100 = 13$$

Problem H. Hotel in Ves Lagos

Suppose $K = A \cdot 10^{i+2} + B \cdot 10^{i} + C,\ B < 100,\ C < 10^{i}$ and $X = D \cdot 10^{i+2} + 13 \cdot 10^{i} + E,\ E < 10^{i}$.

Problem H. Hotel in Ves Lagos

Suppose $K = A \cdot 10^{i+2} + B \cdot 10^i + C,\ B < 100,\ C < 10^i$ and $X = D \cdot 10^{i+2} + 13 \cdot 10^i + E,\ E < 10^i$. Then X may have one of the following forms:

$$\underbrace{\ldots\ldots}_{D}13\overbrace{\underbrace{\ldots\ldots}_{E}}^{i\text{ digits}},\ \text{where } 0 \le D < A,\ 0 \le E < 10^i$$

Problem H. Hotel in Ves Lagos

Suppose $K = A \cdot 10^{i+2} + B \cdot 10^i + C,\ B < 100,\ C < 10^i$ and $X = D \cdot 10^{i+2} + 13 \cdot 10^i + E,\ E < 10^i$. Then X may have one of the following forms:

$$\underbrace{\ldots\ldots}_{D}13\overbrace{\underbrace{\ldots\ldots}_{E}}^{i \text{ digits}}, \text{ where } 0 \le D < A,\ 0 \le E < 10^i$$

$$\underbrace{\ldots\ldots}_{A}13\underbrace{\ldots\ldots}_{E}, \text{ if } B = 13 \text{ and } 0 \le E <= C$$

Problem H. Hotel in Ves Lagos

Suppose $K = A \cdot 10^{i+2} + B \cdot 10^i + C,\ B < 100,\ C < 10^i$ and $X = D \cdot 10^{i+2} + 13 \cdot 10^i + E,\ E < 10^i$. Then X may have one of the following forms:

$$\underbrace{\ldots\ldots}_{D} 13 \underbrace{\overbrace{\ldots\ldots}^{i \text{ digits}}}_{E}, \text{ where } 0 \le D < A,\ 0 \le E < 10^i$$

$$\underbrace{\ldots\ldots}_{A} 13 \underbrace{\ldots\ldots}_{E}, \text{ if } B = 13 \text{ and } 0 \le E <= C$$

$$\underbrace{\ldots\ldots}_{A} 13 \underbrace{\ldots\ldots}_{E}, \text{ if } B > 13 \text{ and } 0 \le E < 10^i$$

Problem H. Hotel in Ves Lagos

Suppose $K = A \cdot 10^{i+2} + B \cdot 10^i + C,\ B < 100,\ C < 10^i$ and $X = D \cdot 10^{i+2} + 13 \cdot 10^i + E,\ E < 10^i$. Then X may have one of the following forms:

$$\underbrace{\ldots\ldots}_{D}13\overbrace{\underbrace{\ldots\ldots}_{E}}^{i\text{ digits}},\ \text{where } 0 \le D < A,\ 0 \le E < 10^i$$

$$\underbrace{\ldots\ldots}_{A}13\underbrace{\ldots\ldots}_{E},\ \text{if } B = 13 \text{ and } 0 \le E <= C$$

$$\underbrace{\ldots\ldots}_{A}13\underbrace{\ldots\ldots}_{E},\ \text{if } B > 13 \text{ and } 0 \le E < 10^i$$

So we have:

$$G(K,i) = A\Big(F\left(10^i - 1\right) + 1\Big) = A \cdot F\left(10^i\right) \text{ if } B < 13$$

Problem H. Hotel in Ves Lagos

Suppose $K = A \cdot 10^{i+2} + B \cdot 10^i + C,\ B < 100,\ C < 10^i$ and $X = D \cdot 10^{i+2} + 13 \cdot 10^i + E,\ E < 10^i$. Then X may have one of the following forms:

$$\underbrace{\ldots\ldots}_{D} 13 \overbrace{\underbrace{\ldots\ldots}_{E}}^{i \text{ digits}}, \text{ where } 0 \le D < A,\ 0 \le E < 10^i$$

$$\underbrace{\ldots\ldots}_{A} 13 \underbrace{\ldots\ldots}_{E}, \text{ if } B = 13 \text{ and } 0 \le E <= C$$

$$\underbrace{\ldots\ldots}_{A} 13 \underbrace{\ldots\ldots}_{E}, \text{ if } B > 13 \text{ and } 0 \le E < 10^i$$

So we have:

$$G(K,i) = A\Big(F\left(10^i - 1\right) + 1\Big) = A \cdot F\left(10^i\right) \text{ if } B < 13$$

$$G(K,i) = \qquad A \cdot F\left(10^i\right) + F(C) + 1, \qquad \text{if } B = 13$$

Problem H. Hotel in Ves Lagos

Suppose $K = A \cdot 10^{i+2} + B \cdot 10^i + C,\ B < 100,\ C < 10^i$ and $X = D \cdot 10^{i+2} + 13 \cdot 10^i + E,\ E < 10^i$. Then X may have one of the following forms:

$$\underbrace{\ldots\ldots}_{D}13\overbrace{\underbrace{\ldots\ldots}_{E}}^{i\text{ digits}},\ \text{where } 0 \le D < A,\ 0 \le E < 10^i$$

$$\underbrace{\ldots\ldots}_{A}13\underbrace{\ldots\ldots}_{E},\ \text{if } B = 13 \text{ and } 0 \le E <= C$$

$$\underbrace{\ldots\ldots}_{A}13\underbrace{\ldots\ldots}_{E},\ \text{if } B > 13 \text{ and } 0 \le E < 10^i$$

So we have:

$$G(K,i) = A\Big(F\left(10^i - 1\right) + 1\Big) = A \cdot F\left(10^i\right) \text{ if } B < 13$$

$$G(K,i) = A \cdot F\left(10^i\right) + F(C) + 1, \quad \text{if } B = 13$$

$$G(K,i) = (A+1) \cdot F\left(10^i\right), \quad \text{if } B > 13$$

Problem H. Hotel in Ves Lagos

To find A, B and C we can use following equations:

$$A = \left\lfloor \frac{K}{10^{i+2}} \right\rfloor$$

Problem H. Hotel in Ves Lagos

To find A, B and C we can use following equations:

$$A = \left\lfloor \frac{K}{10^{i+2}} \right\rfloor$$

$$B = \left\lfloor \frac{K}{10^{i}} \right\rfloor \bmod 100$$

Problem H. Hotel in Ves Lagos

To find A, B and C we can use following equations:

$$A = \left\lfloor \frac{K}{10^{i+2}} \right\rfloor$$

$$B = \left\lfloor \frac{K}{10^{i}} \right\rfloor \bmod 100$$

$$C = K \bmod 10^{i}$$

Problem H. Hotel in Ves Lagos

To find A, B and C we can use following equations:

$$A = \left\lfloor \frac{K}{10^{i+2}} \right\rfloor$$

$$B = \left\lfloor \frac{K}{10^{i}} \right\rfloor \bmod 100$$

$$C = K \bmod 10^{i}$$

Thus we can get $F(K)$ as $K - \sum\limits_{13 \cdot 10^i \le K} G(K, i)$.

Problem H. Hotel in Ves Lagos

To find A, B and C we can use following equations:

$$A = \left\lfloor \frac{K}{10^{i+2}} \right\rfloor$$

$$B = \left\lfloor \frac{K}{10^{i}} \right\rfloor \bmod 100$$

$$C = K \bmod 10^{i}$$

Thus we can get $F(K)$ as $K - \sum\limits_{13 \cdot 10^i \le K} G(K, i)$.

If we precalculate $F(10^i)$ for $0 \le i \le 18$ we can get $F(K)$ in $O\left(\log^2 N\right)$ time (memorizing $F\left(\left\lfloor \frac{K}{10^i} \right\rfloor\right)$).

Problem H. Hotel in Ves Lagos

To find A, B and C we can use following equations:

$$A = \left\lfloor \frac{K}{10^{i+2}} \right\rfloor$$

$$B = \left\lfloor \frac{K}{10^{i}} \right\rfloor \bmod 100$$

$$C = K \bmod 10^{i}$$

Thus we can get $F(K)$ as $K - \sum\limits_{13 \cdot 10^i \le K} G(K, i)$.

If we precalculate $F(10^i)$ for $0 \le i \le 18$ we can get $F(K)$ in $O\left(\log^2 N\right)$ time (memorizing $F\left(\left\lfloor \frac{K}{10^i} \right\rfloor\right)$). Taking into account binary search we have total complexity $O\left(\log^3 N\right)$.

Problem H. Hotel in Ves Lagos

To find A, B and C we can use following equations:

$$A = \left\lfloor \frac{K}{10^{i+2}} \right\rfloor$$

$$B = \left\lfloor \frac{K}{10^{i}} \right\rfloor \bmod 100$$

$$C = K \bmod 10^{i}$$

Thus we can get $F(K)$ as $K - \sum\limits_{13 \cdot 10^i \le K} G(K, i)$.

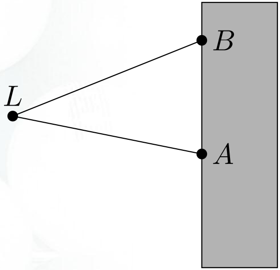
If we precalculate $F(10^i)$ for $0 \le i \le 18$ we can get $F(K)$ in $O\left(\log^2 N\right)$ time (memorizing $F\left(\left\lfloor \frac{K}{10^i} \right\rfloor\right)$). Taking into account binary search we have total complexity $O\left(\log^3 N\right)$.

There also exists an $O(\log N)$ approach.

Problem I. Illumination of Buildings

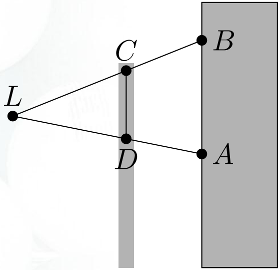
Idea:	Vladimir Kotov
Solutions:	Pavel Irzhavski, Ivan Metelsky
Tests:	Ivan Metelsky
Analysis:	Pavel Irzhavski
Statement:	Ivan Metelsky

Problem I. Illumination of Buildings



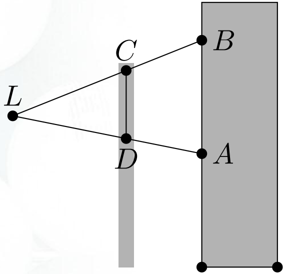
The first thing we should notice is that if some point A of building side is illuminated by light source located in point L than any higher point B on this side is also illuminated.

Problem I. Illumination of Buildings



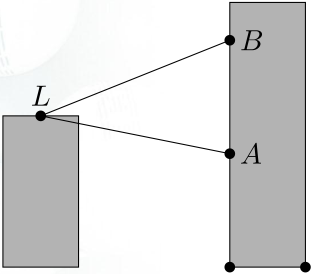
The first thing we should notice is that if some point A of building side is illuminated by light source located in point L than any higher point B on this side is also illuminated. If there is some internal point C of some rectangle on LB segment, then there would be some internal point D of the same rectangle with the same x-coordinate located on LA segment.

Problem I. Illumination of Buildings



The first thing we should notice is that if some point A of building side is illuminated by light source located in point L than any higher point B on this side is also illuminated. If there is some internal point C of some rectangle on LB segment, then there would be some internal point D of the same rectangle with the same x-coordinate located on LA segment. So we can assume we need illuminate only two points for each building (bottom points of side edges).

Problem I. Illumination of Buildings



The first thing we should notice is that if some point A of building side is illuminated by light source located in point L than any higher point B on this side is also illuminated. If there is some internal point C of some rectangle on LB segment, then there would be some internal point D of the same rectangle with the same x-coordinate located on LA segment. So we can assume we need illuminate only two points for each building (bottom points of side edges).

Now it is obvious that light source located inside the top edge is useless since it can't illuminate any bottom point.

Problem I. Illumination of Buildings

Consider following greedy approach.

- Sort rectangles from left to right ($L_1 < R_1 < L_2 < R_2 < \cdots < L_N < R_N$).

Problem I. Illumination of Buildings

Consider following greedy approach.

- Sort rectangles from left to right ($L_1 < R_1 < L_2 < R_2 < \cdots < L_N < R_N$).
- Let $A_i = (L_i, 0)$, $B_i = (L_i, H_i)$, $C_i = (R_i, H_i)$, $D_i = (R_i, 0)$.

Problem I. Illumination of Buildings

Consider following greedy approach.

- Sort rectangles from left to right ($L_1 < R_1 < L_2 < R_2 < \cdots < L_N < R_N$).
- Let $A_i = (L_i, 0)$, $B_i = (L_i, H_i)$, $C_i = (R_i, H_i)$, $D_i = (R_i, 0)$.
- We should put light sources into points B_1 and C_N since points A_1 and D_N can't be illuminated from any other possible locations.

Problem I. Illumination of Buildings

Consider following greedy approach.

- Sort rectangles from left to right ($L_1 < R_1 < L_2 < R_2 < \cdots < L_N < R_N$).
- Let $A_i = (L_i, 0)$, $B_i = (L_i, H_i)$, $C_i = (R_i, H_i)$, $D_i = (R_i, 0)$.
- We should put light sources into points B_1 and C_N since points A_1 and D_N can't be illuminated from any other possible locations. However, this two light sources doesn't illuminate any other desired point. Thus we can simply exclude A_1 and D_N from consideration and keep in mind to add 2 to the answer.

Problem I. Illumination of Buildings

Consider following greedy approach.

- Sort rectangles from left to right ($L_1 < R_1 < L_2 < R_2 < \cdots < L_N < R_N$).
- Let $A_i = (L_i, 0)$, $B_i = (L_i, H_i)$, $C_i = (R_i, H_i)$, $D_i = (R_i, 0)$.
- We should put light sources into points B_1 and C_N since points A_1 and D_N can't be illuminated from any other possible locations. However, this two light sources doesn't illuminate any other desired point. Thus we can simply exclude A_1 and D_N from consideration and keep in mind to add 2 to the answer.
- Then for each $2 \le i \le N$ we place light source into the leftmost position where A_i can be illuminated from if this position is not equal to C_{i-1}

Problem I. Illumination of Buildings

Consider following greedy approach.

- Sort rectangles from left to right ($L_1 < R_1 < L_2 < R_2 < \cdots < L_N < R_N$).
- Let $A_i = (L_i, 0)$, $B_i = (L_i, H_i)$, $C_i = (R_i, H_i)$, $D_i = (R_i, 0)$.
- We should put light sources into points B_1 and C_N since points A_1 and D_N can't be illuminated from any other possible locations. However, this two light sources doesn't illuminate any other desired point. Thus we can simply exclude A_1 and D_N from consideration and keep in mind to add 2 to the answer.
- Then for each $2 \le i \le N$ we place light source into the leftmost position where A_i can be illuminated from if this position is not equal to C_{i-1}
- Similarly for each $1 \le i \le N-1$ we place light source into the rightmost position where D_i can be illuminated from if this position is not equal to B_{i+1}.

Problem I. Illumination of Buildings

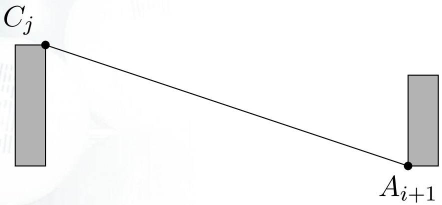
Consider following greedy approach.

- Sort rectangles from left to right $(L_1 < R_1 < L_2 < R_2 < \cdots < L_N < R_N)$.
- Let $A_i = (L_i, 0)$, $B_i = (L_i, H_i)$, $C_i = (R_i, H_i)$, $D_i = (R_i, 0)$.
- We should put light sources into points B_1 and C_N since points A_1 and D_N can't be illuminated from any other possible locations. However, this two light sources doesn't illuminate any other desired point. Thus we can simply exclude A_1 and D_N from consideration and keep in mind to add 2 to the answer.
- Then for each $2 \leq i \leq N$ we place light source into the leftmost position where A_i can be illuminated from if this position is not equal to C_{i-1}
- Similarly for each $1 \leq i \leq N-1$ we place light source into the rightmost position where D_i can be illuminated from if this position is not equal to B_{i+1}.
- And finally we place light source into C_i or B_{i+1} (into any of them) if D_i or A_{i+1} isn't illuminated.

Problem I. Illumination of Buildings

Let's prove algorithm is correct.

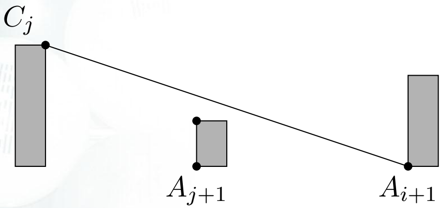
Problem I. Illumination of Buildings



Let's prove algorithm is correct.

Let the leftmost point A_{i+1} can be illuminated from is C_j, $j < i$.

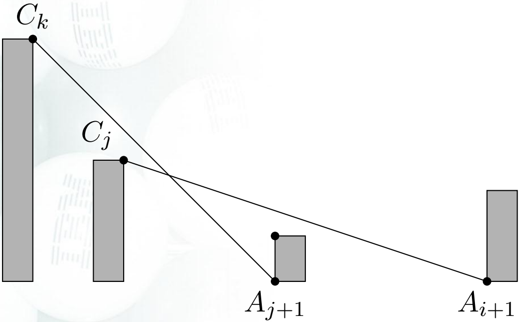
Problem I. Illumination of Buildings



Let's prove algorithm is correct.

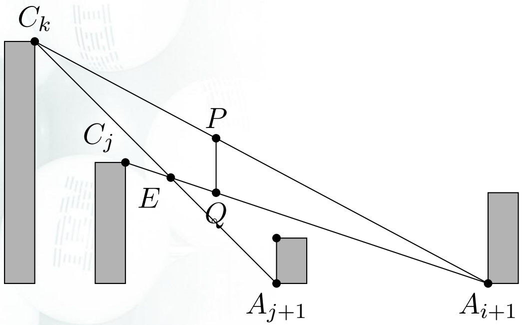
Let the leftmost point A_{i+1} can be illuminated from is C_j, $j < i$. Then A_{j+1} can be illuminated only from C_j and B_{j+1}.

Problem I. Illumination of Buildings



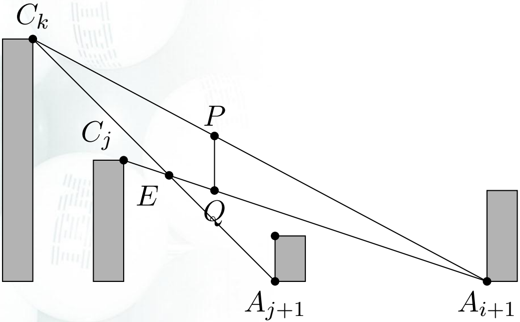
Suppose this is not true and C_k, $k < j$ is point A_{j+1} can be illuminated from.

Problem I. Illumination of Buildings



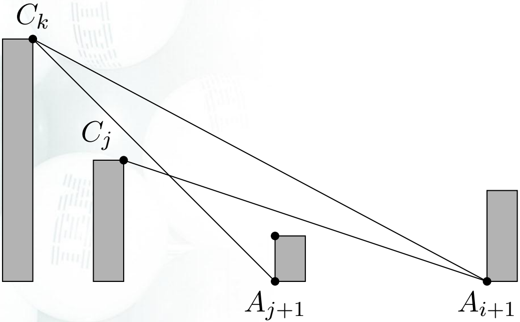
Suppose this is not true and C_k, $k < j$ is point A_{j+1} can be illuminated from. Let $E = A_{i+1}C_j \cap A_{j+1}C_k$. Then for any point P of segment C_kA_{i+1} we can find some point Q from C_kE or EA_{i+1} with same x-coordinate.

Problem I. Illumination of Buildings



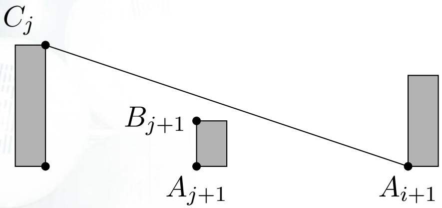
Suppose this is not true and C_k, $k < j$ is point A_{j+1} can be illuminated from. Let $E = A_{i+1}C_j \cap A_{j+1}C_k$. Then for any point P of segment C_kA_{i+1} we can find some point Q from C_kE or EA_{i+1} with same x-coordinate. Q is not an internal point of any rectangle and has y-coordinate not greater than P does, so P also is not an internal point of any rectangle.

Problem I. Illumination of Buildings



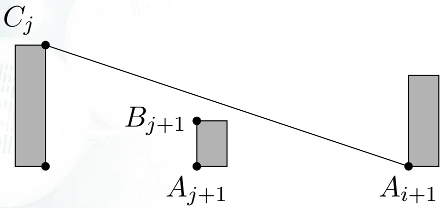
Suppose this is not true and C_k, $k < j$ is point A_{j+1} can be illuminated from. Let $E = A_{i+1}C_j \cap A_{j+1}C_k$. Then for any point P of segment C_kA_{i+1} we can find some point Q from C_kE or EA_{i+1} with same x-coordinate. Q is not an internal point of any rectangle and has y-coordinate not greater than P does, so P also is not an internal point of any rectangle. So A_{i+1} can be illuminated from C_k that is contrary with definition of j.

Problem I. Illumination of Buildings



A_{i+1} can be illuminated from C_j so $H_{j+1} < H_j$ and light source placed in B_{j+1} can illuminate only two points while light source placed in C_j can illuminate the same two points and point A_i.

Problem I. Illumination of Buildings



A_{i+1} can be illuminated from C_j so $H_{j+1} < H_j$ and light source placed in B_{j+1} can illuminate only two points while light source placed in C_j can illuminate the same two points and point A_i. So without loosing optimality we can state that light source anyway should be placed in C_j.

Problem I. Illumination of Buildings

Similarly we get that for each $1 \leq i \leq N-1$ light source should be placed in the rightmost position where D_i can be illuminated from if this position is not equal to B_{i+1}.

Problem I. Illumination of Buildings

Similarly we get that for each $1 \le i \le N-1$ light source should be placed in the rightmost position where D_i can be illuminated from if this position is not equal to B_{i+1}.

Finally if we have D_i or A_{i+1} not being illuminated then:

- This dark point(s) can't be illuminated from any point other than C_i and B_{i+1}.
- Light source placed in either C_i or B_{i+1} can't illuminate any dark point other than D_i and A_{i+1}.

Problem I. Illumination of Buildings

Similarly we get that for each $1 \leq i \leq N-1$ light source should be placed in the rightmost position where D_i can be illuminated from if this position is not equal to B_{i+1}.

Finally if we have D_i or A_{i+1} not being illuminated then:

- This dark point(s) can't be illuminated from any point other than C_i and B_{i+1}.
- Light source placed in either C_i or B_{i+1} can't illuminate any dark point other than D_i and A_{i+1}.

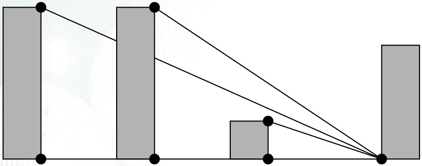
Thus we need to place one light source to illuminate D_i or A_{i+1} or both and doesn't matter where it will be placed.

Problem I. Illumination of Buildings

The only problem we still have is efficient implementation of this algorithm.

Problem I. Illumination of Buildings

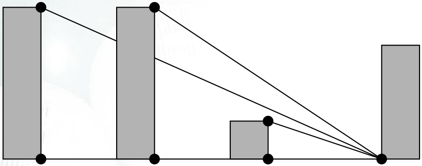
The only problem we still have is efficient implementation of this algorithm.



A_i can be illuminated from C_j if $\angle C_j A_i D_j \geq \angle C_k A_i D_k$ for any $j < k < i$.

Problem I. Illumination of Buildings

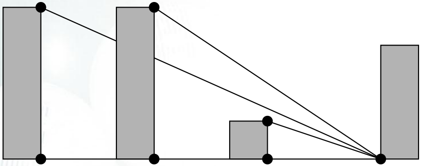
The only problem we still have is efficient implementation of this algorithm.



A_i can be illuminated from C_j if $\angle C_j A_i D_j \geq \angle C_k A_i D_k$ for any $j < k < i$. So to determine the leftmost position where A_i can be illuminated from we should iterate j from $i-1$ to 1, compare $\angle C_j A_i D_j$ to current maximum of such angles (if it is not less then C_j becomes current leftmost possible place to light source) and update this maximum.

Problem I. Illumination of Buildings

The only problem we still have is efficient implementation of this algorithm.

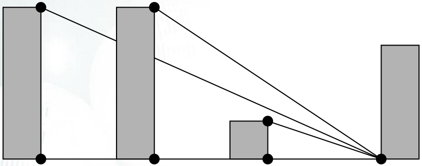


A_i can be illuminated from C_j if $\angle C_j A_i D_j \geq \angle C_k A_i D_k$ for any $j < k < i$. So to determine the leftmost position where A_i can be illuminated from we should iterate j from $i - 1$ to 1, compare $\angle C_j A_i D_j$ to current maximum of such angles (if it is not less then C_j becomes current leftmost possible place to light source) and update this maximum.

Similarly we can do the second part. And the third one is rather trivial.

Problem I. Illumination of Buildings

The only problem we still have is efficient implementation of this algorithm.



A_i can be illuminated from C_j if $\angle C_j A_i D_j \geq \angle C_k A_i D_k$ for any $j < k < i$. So to determine the leftmost position where A_i can be illuminated from we should iterate j from $i-1$ to 1, compare $\angle C_j A_i D_j$ to current maximum of such angles (if it is not less then C_j becomes current leftmost possible place to light source) and update this maximum.

Similarly we can do the second part. And the third one is rather trivial.

Complexity of this algorithm is $O\left(N^2\right)$.

Problem J. Journal

Idea:	Uldis Barbans
Solutions:	Uldis Barbans, Pavel Irzhavski, Ivan Metelsky, Vladimir Kerus
Tests:	Uldis Barbans, Ivan Metelsky
Analysis:	Ivan Metelsky
Statement:	Uldis Barbans, Ivan Metelsky

Problem J. Journal

We will describe two solutions for this problem - one with complexity $O(W \cdot N \cdot \log R)$ and a faster solution with complexity $O(W \cdot N)$.

Problem J. Journal

We will describe two solutions for this problem - one with complexity $O(W \cdot N \cdot \log R)$ and a faster solution with complexity $O(W \cdot N)$.

But first let's calculate some data that will be useful for both these solutions.

Problem J. Journal

Let's define the values $skip(pos, len)$, $1 \leq pos \leq N + 1$, $0 \leq len \leq W$ as follows. Suppose we have a single line that is len columns wide. We start filling this line with words numbered pos, $pos + 1$, $pos + 2$ and so on. Then $skip(pos, len)$ is defined as the number of the first word that doesn't fit within this line.

Problem J. Journal

Let's define the values $skip(pos, len)$, $1 \le pos \le N+1$, $0 \le len \le W$ as follows. Suppose we have a single line that is len columns wide. We start filling this line with words numbered pos, $pos+1$, $pos+2$ and so on. Then $skip(pos, len)$ is defined as the number of the first word that doesn't fit within this line.

If all words up to the N-th one fit within the line, then $skip(pos, len)$ is considered to be $N+1$. Also, by definition $skip(N+1, len) = N+1$ for any value of len.

Problem J. Journal

The values of $skip$ for a fixed len can be calculated in $O(N)$ time.
The idea is to use the fact that $skip(pos + 1, len) \geq skip(pos, len)$.

Problem J. Journal

The values of $skip$ for a fixed len can be calculated in $O(N)$ time. The idea is to use the fact that $skip(pos + 1, len) \geq skip(pos, len)$.

The value of $skip(1, len)$ is calculated directly. Each next value of $skip_{pos,len}$ is calculated from the value of $skip(pos - 1, len)$. We first remove the word with number $pos - 1$ from the beginning of the line (shifting the other words to the left) and then try adding words numbered $skip(pos - 1, len)$, $skip(pos - 1, len) + 1$, $\cdots$ to the end of the line until there's space.

Problem J. Journal

The values of $skip$ for a fixed len can be calculated in $O(N)$ time. The idea is to use the fact that $skip(pos+1, len) \geq skip(pos, len)$.

The value of $skip(1, len)$ is calculated directly. Each next value of $skip_{pos,len}$ is calculated from the value of $skip(pos-1, len)$. We first remove the word with number $pos-1$ from the beginning of the line (shifting the other words to the left) and then try adding words numbered $skip(pos-1, len)$, $skip(pos-1, len)+1$, $\cdots$ to the end of the line until there's space.

Of course, no real manipulation with words are made. We just need to store at each moment the total length of words currently in the line (including spaces between them) and modify this length accordingly.

Problem J. Journal

Two more values to be calculated are $BeginSkip(k)$, $k \geq 0$, and $EndSkip(pos)$, $1 \leq pos \leq N+1$.

Problem J. Journal

Two more values to be calculated are $BeginSkip(k)$, $k \geq 0$, and $EndSkip(pos)$, $1 \leq pos \leq N + 1$.

$BeginSkip(k)$ is defined as follows. Suppose we have k lines each with the width of W characters. Let's start laying out the text on these lines from the beginning. $BeginSkip(k)$ is the number of the first word that doesn't fit within these lines. It's equal to $N + 1$ if all words in the article can be fit within these lines.

Problem J. Journal

Two more values to be calculated are $BeginSkip(k)$, $k \geq 0$, and $EndSkip(pos)$, $1 \leq pos \leq N+1$.

$BeginSkip(k)$ is defined as follows. Suppose we have k lines each with the width of W characters. Let's start laying out the text on these lines from the beginning. $BeginSkip(k)$ is the number of the first word that doesn't fit within these lines. It's equal to $N+1$ if all words in the article can be fit within these lines.

$BeginSkip(k)$ is pretty easy to calculate:

$$BeginSkip(0) = 1$$
$$BeginSkip(k) = skip\big(BeginSkip(k-1), W\big), k > 0$$

Problem J. Journal

Two more values to be calculated are $BeginSkip(k)$, $k \geq 0$, and $EndSkip(pos)$, $1 \leq pos \leq N + 1$.

$BeginSkip(k)$ is defined as follows. Suppose we have k lines each with the width of W characters. Let's start laying out the text on these lines from the beginning. $BeginSkip(k)$ is the number of the first word that doesn't fit within these lines. It's equal to $N + 1$ if all words in the article can be fit within these lines.

$BeginSkip(k)$ is pretty easy to calculate:

$$BeginSkip(0) = 1$$
$$BeginSkip(k) = skip\big(BeginSkip(k-1), W\big), k > 0$$

Let's define K as the smallest k such that $BeginSkip(k) = N + 1$. K is the number of lines required to print the article without image and obviously $K \leq N$.

Problem J. Journal

$EndSkip(pos)$ is defined as the number of lines (each with the width of W characters) required to fit the ending part of the article starting with the word numbered pos (i.e. this part consists of words pos, $pos + 1, \cdots, N$).

Problem J. Journal

$EndSkip(pos)$ is defined as the number of lines (each with the width of W characters) required to fit the ending part of the article starting with the word numbered pos (i.e. this part consists of words pos, $pos + 1$, $\cdots$, N).

$EndSkip(pos)$ is also easy to calculate:

$$EndSkip(N + 1) = 0$$
$$EndSkip(pos) = EndSkip\big(skip(pos, W)\big) + 1, k \leq N$$

Problem J. Journal

At this moment we are ready to describe the slow $O(W \cdot N \cdot R)$ solution.

Problem J. Journal

At this moment we are ready to describe the slow $O(W \cdot N \cdot R)$ solution.

1. Iterate through all values of x, $0 \leq x \leq W - C$, where x is the number (0-based) of the leftmost column where the image is placed.

Problem J. Journal

At this moment we are ready to describe the slow $O(W \cdot N \cdot R)$ solution.

1. Iterate through all values of x, $0 \leq x \leq W - C$, where x is the number (0-based) of the leftmost column where the image is placed.
2. Iterate through all values of k, $0 \leq k \leq K$, where k is the number of lines to be skipped before inserting the image.

Problem J. Journal

At this moment we are ready to describe the slow $O(W \cdot N \cdot R)$ solution.

1. Iterate through all values of x, $0 \le x \le W - C$, where x is the number (0-based) of the leftmost column where the image is placed.
2. Iterate through all values of k, $0 \le k \le K$, where k is the number of lines to be skipped before inserting the image.
3. For each x and k calculate the number of lines required to locate the image and the text. Choose the best out of all layouts.

Problem J. Journal

For the given x and k, the calculation is as follows:

Problem J. Journal

For the given x and k, the calculation is as follows:

1. First, skip k lines. Set $curWord := BeginSkip(k)$.

Problem J. Journal

For the given x and k, the calculation is as follows:

1. First, skip k lines. Set $curWord := BeginSkip(k)$.
2. Then, simulate placing the text into the lines with the image. Each of them consists of two piece - the left one with the width of x and the right one with the width of $W - C - x$. So R times we need to do $curWord := skip(skip(curWord, x), W - C - x)$.

Problem J. Journal

For the given x and k, the calculation is as follows:

1. First, skip k lines. Set $curWord := BeginSkip(k)$.
2. Then, simulate placing the text into the lines with the image. Each of them consists of two piece - the left one with the width of x and the right one with the width of $W - C - x$. So R times we need to do $curWord := skip(skip(curWord, x), W - C - x)$.
3. At this point we are at word $curWord$ and are required $EndSkip(curWord)$ lines in order to place the rest of the article. So the total number of occupied lines is $k + R + EndSkip(curWord)$.

Problem J. Journal

Calculation for the fixed x takes $O(N \cdot R)$ time because of the slow second step (placing the text into the lines with the image).

Problem J. Journal

Calculation for the fixed x takes $O(N \cdot R)$ time because of the slow second step (placing the text into the lines with the image).

Let's formulate what we need to know in order to make it work faster.

Problem J. Journal

Calculation for the fixed x takes $O(N \cdot R)$ time because of the slow second step (placing the text into the lines with the image).

Let's formulate what we need to know in order to make it work faster.

Define $next(pos), 0 \le pos \le N$ as follows:

$$next(pos) = skip(skip(pos, x), W - c - x)$$

Problem J. Journal

Calculation for the fixed x takes $O(N \cdot R)$ time because of the slow second step (placing the text into the lines with the image).

Let's formulate what we need to know in order to make it work faster.

Define $next(pos), 0 \leq pos \leq N$ as follows:

$$next(pos) = skip(skip(pos, x), W - c - x)$$

The values of $next$ simulate placing the text into a single line with the image.

Problem J. Journal

Let's define the powers of $next$ as follows:

Problem J. Journal

Let's define the powers of $next$ as follows:

$$next^0(pos) = pos$$
$$next^i(pos) = next\left(next^{i-1}(pos)\right), i > 0$$

Problem J. Journal

Let's define the powers of $next$ as follows:

$$next^0(pos) = pos$$
$$next^i(pos) = next\left(next^{i-1}(pos)\right), i > 0$$

The value of $next^i$ simulate placing the text into i consecutive lines with the image.

Problem J. Journal

Let's define the powers of $next$ as follows:

$$next^0(pos) = pos$$
$$next^i(pos) = next\left(next^{i-1}(pos)\right), i > 0$$

The value of $next^i$ simulate placing the text into i consecutive lines with the image.

We need to know $next^R$. It would allow to perform placing the text into the lines with the image as follows:

Problem J. Journal

Let's define the powers of $next$ as follows:

$$next^0(pos) = pos$$
$$next^i(pos) = next\left(next^{i-1}(pos)\right), i > 0$$

The value of $next^i$ simulate placing the text into i consecutive lines with the image.

We need to know $next^R$. It would allow to perform placing the text into the lines with the image as follows:

$$curWord = next^R(curWord)$$

Problem J. Journal

Note that the defined powers of $next$ are similar to usual powers of integers in the following sense:

Problem J. Journal

Note that the defined powers of $next$ are similar to usual powers of integers in the following sense:

$$next^{i+j}(pos) = next^{i}\left(next^{j}(pos)\right)$$

Problem J. Journal

Note that the defined powers of $next$ are similar to usual powers of integers in the following sense:

$$next^{i+j}(pos) = next^{i}\left(next^{j}(pos)\right)$$

Therefore we can use a modification of “repeating square” fast exponentiation algorithm in order to get $next^{R}$.

Problem J. Journal

Note that the defined powers of $next$ are similar to usual powers of integers in the following sense:

$$next^{i+j}(pos) = next^{i}\left(next^{j}(pos)\right)$$

Therefore we can use a modification of “repeating square” fast exponentiation algorithm in order to get $next^{R}$.

We first calculate $next^{i}(pos)$ for all $i = 2^{t}, i \leq R$, using the following equation:

$$next^{2i}(pos) = next^{i}\left(next^{i}(pos)\right)$$

Problem J. Journal

Note that the defined powers of $next$ are similar to usual powers of integers in the following sense:

$$next^{i+j}(pos) = next^{i}\left(next^{j}(pos)\right)$$

Therefore we can use a modification of “repeating square” fast exponentiation algorithm in order to get $next^R$.

We first calculate $next^i(pos)$ for all $i = 2^t, i \leq R$, using the following equation:

$$next^{2i}(pos) = next^{i}\left(next^{i}(pos)\right)$$

Then we represent R in binary form and compose $next^R(pos)$ out the values of $next$ for the corresponding powers of 2.

Problem J. Journal

Note that the defined powers of $next$ are similar to usual powers of integers in the following sense:

$$next^{i+j}(pos) = next^{i}\left(next^{j}(pos)\right)$$

Therefore we can use a modification of "repeating square" fast exponentiation algorithm in order to get $next^R$.

We first calculate $next^i(pos)$ for all $i = 2^t, i \le R$, using the following equation:

$$next^{2i}(pos) = next^{i}\left(next^{i}(pos)\right)$$

Then we represent R in binary form and compose $next^R(pos)$ out the values of $next$ for the corresponding powers of 2.

It allows to calculate $next^R$ in $O(N \cdot \log R)$ time and thus achieve an $O(W \cdot N \cdot \log R)$ solution.

Problem J. Journal

The described solution is already enough to get accepted. But there's even a faster approach.

Problem J. Journal

The described solution is already enough to get accepted. But there's even a faster approach.

Consider the directed graph with vertices numbered 1 to $N+1$, inclusive.

Problem J. Journal

The described solution is already enough to get accepted. But there's even a faster approach.

Consider the directed graph with vertices numbered 1 to $N + 1$, inclusive.

For each pos, $1 \leq pos \leq N + 1$, such that $pos < next(pos)$, there's an arc from $next(pos)$ to pos.

Problem J. Journal

The described solution is already enough to get accepted. But there's even a faster approach.

Consider the directed graph with vertices numbered 1 to $N+1$, inclusive.

For each pos, $1 \leq pos \leq N+1$, such that $pos < next(pos)$, there's an arc from $next(pos)$ to pos.

This graph is a forest, i.e., each component of the graph is a rooted tree.

Problem J. Journal

The described solution is already enough to get accepted. But there's even a faster approach.

Consider the directed graph with vertices numbered 1 to $N + 1$, inclusive.

For each pos, $1 \leq pos \leq N + 1$, such that $pos < next(pos)$, there's an arc from $next(pos)$ to pos.

This graph is a forest, i.e., each component of the graph is a rooted tree.

$next^R(i)$ in terms of this graph is the R-th grandparent of the vertex i (or the root of the corresponding tree if i is located at a height less than R).

Problem J. Journal

So we can traverse each tree with DFS algorithm starting from the root.

Problem J. Journal

So we can traverse each tree with DFS algorithm starting from the root.

At each vertex, we can store the complete path from the root to this vertex in an array.

Problem J. Journal

So we can traverse each tree with DFS algorithm starting from the root.

At each vertex, we can store the complete path from the root to this vertex in an array.

This array works as a stack. When you enter a vertex, you add it to the end of the path, and when you leave it, you remove the vertex from the end of the path.

Problem J. Journal

So we can traverse each tree with DFS algorithm starting from the root.

At each vertex, we can store the complete path from the root to this vertex in an array.

This array works as a stack. When you enter a vertex, you add it to the end of the path, and when you leave it, you remove the vertex from the end of the path.

When we are at a vertex i, in order to calculate the value of $next^R(i)$, we simply look back on R steps in the array containing the path from the root to i.

Problem J. Journal

So we can traverse each tree with DFS algorithm starting from the root.

At each vertex, we can store the complete path from the root to this vertex in an array.

This array works as a stack. When you enter a vertex, you add it to the end of the path, and when you leave it, you remove the vertex from the end of the path.

When we are at a vertex i, in order to calculate the value of $next^R(i)$, we simply look back on R steps in the array containing the path from the root to i.

Calculation of $next^R$ becomes linear and therefore we have an $O(W \cdot N)$ algorithm.

Problem K. Kids and Prizes

Idea:	Pavel Irzhavski
Solutions:	Pavel Irzhavski, Ivan Metelsky
Tests:	Pavel Irzhavski
Analysis:	Pavel Irzhavski
Statement:	Ivan Metelsky

Problem K. Kids and Prizes

Let E_K be desired expectation for N prizes and K people.

acm International Collegiate Programming Contest

Problem K. Kids and Prizes

Let E_K be desired expectation for N prizes and K people.
Let $p_K(i)$ be probability that exactly i prizes will be given.

Problem K. Kids and Prizes

Let E_K be desired expectation for N prizes and K people.
Let $p_K(i)$ be probability that exactly i prizes will be given.

$$\sum_{i=0}^{N} i p_K(i) = E_K,$$

Problem K. Kids and Prizes

Let E_K be desired expectation for N prizes and K people.
Let $p_K(i)$ be probability that exactly i prizes will be given.

$$\sum_{i=0}^{N} i p_K(i) = E_K,$$

$$\sum_{i=0}^{N} p_K(i) = 1.$$

Problem K. Kids and Prizes

We can get some recurrence relation for $p_K(i)$.

Problem K. Kids and Prizes

We can get some recurrence relation for $p_K(i)$.

$$p_{K+1}(i) = p_K(i)\frac{i}{N} + p_K(i-1)\frac{N-(i-1)}{N} \quad \text{for } i, K \geq 1,$$
$$p_K(0) = 0 \quad \text{for } K \geq 1.$$

Problem K. Kids and Prizes

We can get some recurrence relation for $p_K(i)$.

$$p_{K+1}(i) = p_K(i)\frac{i}{N} + p_K(i-1)\frac{N-(i-1)}{N} \quad \text{for } i, K \geq 1,$$
$$p_K(0) = 0 \quad \text{for } K \geq 1.$$

Problem K. Kids and Prizes

Now it's time to calculate E_{K+1}.

Problem K. Kids and Prizes

Now it's time to calculate E_{K+1}.

$$E_{K+1} = \sum_{i=0}^{N} i p_{K+1}(i)$$

Problem K. Kids and Prizes

Now it's time to calculate E_{K+1}.

$$E_{K+1} = \sum_{i=0}^{N} i p_{K+1}(i) = \sum_{i=1}^{N} i p_{K+1}(i)$$

Problem K. Kids and Prizes

Now it's time to calculate E_{K+1}.

$$E_{K+1} = \sum_{i=0}^{N} i p_{K+1}(i) = \sum_{i=1}^{N} i p_{K+1}(i) =$$

$$= \sum_{i=1}^{N} i \left(p_K(i) \frac{i}{N} + p_K(i-1) \frac{N-(i-1)}{N} \right)$$

Problem K. Kids and Prizes

Now it's time to calculate E_{K+1}.

$$E_{K+1} = \sum_{i=0}^{N} i p_{K+1}(i) = \sum_{i=1}^{N} i p_{K+1}(i) =$$

$$= \sum_{i=1}^{N} i \left(p_K(i) \frac{i}{N} + p_K(i-1) \frac{N-(i-1)}{N} \right) =$$

$$= \sum_{i=1}^{N} i p_K(i) \frac{i}{N} + \sum_{i=1}^{N} i p_K(i-1) \frac{N-(i-1)}{N}$$

Problem K. Kids and Prizes

Now it's time to calculate E_{K+1}.

$$E_{K+1} = \sum_{i=0}^{N} i p_{K+1}(i) = \sum_{i=1}^{N} i p_{K+1}(i) =$$

$$= \sum_{i=1}^{N} i \left(p_K(i) \frac{i}{N} + p_K(i-1) \frac{N-(i-1)}{N} \right) =$$

$$= \sum_{i=1}^{N} i p_K(i) \frac{i}{N} + \sum_{i=1}^{N} i p_K(i-1) \frac{N-(i-1)}{N} =$$

$$= \sum_{i=0}^{N} i p_K(i) \frac{i}{N} + \sum_{i=0}^{N-1} (i+1) p_K(i) \frac{N-i}{N}$$

Problem K. Kids and Prizes

Now it's time to calculate E_{K+1}.

$$E_{K+1} = \sum_{i=0}^{N} i p_{K+1}(i) = \sum_{i=1}^{N} i p_{K+1}(i) =$$

$$= \sum_{i=1}^{N} i \left(p_K(i) \frac{i}{N} + p_K(i-1) \frac{N-(i-1)}{N} \right) =$$

$$= \sum_{i=1}^{N} i p_K(i) \frac{i}{N} + \sum_{i=1}^{N} i p_K(i-1) \frac{N-(i-1)}{N} =$$

$$= \sum_{i=0}^{N} i p_K(i) \frac{i}{N} + \sum_{i=0}^{N-1} (i+1) p_K(i) \frac{N-i}{N} =$$

$$= \sum_{i=0}^{N} p_K(i) \left(i \frac{i}{N} + (i+1) \frac{N-i}{N} \right)$$

Problem K. Kids and Prizes

Now it's time to calculate E_{K+1}.

$$E_{K+1} = \sum_{i=0}^{N} i p_{K+1}(i) = \sum_{i=1}^{N} i p_{K+1}(i) =$$

$$= \sum_{i=1}^{N} i \left(p_K(i) \frac{i}{N} + p_K(i-1) \frac{N-(i-1)}{N} \right) =$$

$$= \sum_{i=1}^{N} i p_K(i) \frac{i}{N} + \sum_{i=1}^{N} i p_K(i-1) \frac{N-(i-1)}{N} =$$

$$= \sum_{i=0}^{N} i p_K(i) \frac{i}{N} + \sum_{i=0}^{N-1} (i+1) p_K(i) \frac{N-i}{N} =$$

$$= \sum_{i=0}^{N} p_K(i) \left(i \frac{i}{N} + (i+1) \frac{N-i}{N} \right) =$$

$$= \sum_{i=0}^{N} p_K(i) \frac{Ni - i + N}{N}$$

Problem K. Kids and Prizes

Now it's time to calculate E_{K+1}.

$$E_{K+1} = \sum_{i=0}^{N} i p_{K+1}(i) = \sum_{i=1}^{N} i p_{K+1}(i) =$$

$$= \sum_{i=1}^{N} i \left(p_K(i) \frac{i}{N} + p_K(i-1) \frac{N-(i-1)}{N} \right) =$$

$$= \sum_{i=1}^{N} i p_K(i) \frac{i}{N} + \sum_{i=1}^{N} i p_K(i-1) \frac{N-(i-1)}{N} =$$

$$= \sum_{i=0}^{N} i p_K(i) \frac{i}{N} + \sum_{i=0}^{N-1} (i+1) p_K(i) \frac{N-i}{N} =$$

$$= \sum_{i=0}^{N} p_K(i) \left(i \frac{i}{N} + (i+1) \frac{N-i}{N} \right) =$$

$$= \sum_{i=0}^{N} p_K(i) \frac{Ni - i + N}{N} = \frac{N-1}{N} \sum_{i=0}^{N} i p_K(i) + \sum_{i=0}^{N} p_K(i).$$

Problem K. Kids and Prizes

Now it's time to calculate E_{K+1}.

$$E_{K+1} = \sum_{i=0}^{N} i p_{K+1}(i) = \sum_{i=1}^{N} i p_{K+1}(i) =$$

$$= \sum_{i=1}^{N} i \left(p_K(i) \frac{i}{N} + p_K(i-1) \frac{N-(i-1)}{N} \right) =$$

$$= \sum_{i=1}^{N} i p_K(i) \frac{i}{N} + \sum_{i=1}^{N} i p_K(i-1) \frac{N-(i-1)}{N} =$$

$$= \sum_{i=0}^{N} i p_K(i) \frac{i}{N} + \sum_{i=0}^{N-1} (i+1) p_K(i) \frac{N-i}{N} =$$

$$= \sum_{i=0}^{N} p_K(i) \left(i \frac{i}{N} + (i+1) \frac{N-i}{N} \right) =$$

$$= \sum_{i=0}^{N} p_K(i) \frac{Ni - i + N}{N} = \frac{N-1}{N} \sum_{i=0}^{N} i p_K(i) + \sum_{i=0}^{N} p_K(i).$$

Problem K. Kids and Prizes

$$E_{K+1} = \frac{N-1}{N}\sum_{i=0}^{N} i p_K(i) + \sum_{i=0}^{N} p_K(i) = \frac{N-1}{N} E_K + 1.$$

Problem K. Kids and Prizes

$$E_{K+1} = \frac{N-1}{N}\sum_{i=0}^{N} i p_K(i) + \sum_{i=0}^{N} p_K(i) = \frac{N-1}{N}E_K + 1.$$

This equation is enough to get E_M using $O(M)$ time.

Problem K. Kids and Prizes

$$E_{K+1} = \frac{N-1}{N}\sum_{i=0}^{N} i p_K(i) + \sum_{i=0}^{N} p_K(i) = \frac{N-1}{N}E_K + 1.$$

This equation is enough to get E_M using $O(M)$ time.
But we still can improve this result.

Problem K. Kids and Prizes

$$E_{K+1} = \frac{N-1}{N}\sum_{i=0}^{N} i p_K(i) + \sum_{i=0}^{N} p_K(i) = \frac{N-1}{N}E_K + 1.$$

This equation is enough to get E_M using $O(M)$ time.
But we still can improve this result.

$$E_{K+1} = \frac{N-1}{N}E_K + 1.$$

Problem K. Kids and Prizes

$$E_{K+1} = \frac{N-1}{N} \sum_{i=0}^{N} i p_K(i) + \sum_{i=0}^{N} p_K(i) = \frac{N-1}{N} E_K + 1.$$

This equation is enough to get E_M using $O(M)$ time.
But we still can improve this result.

$$E_{K+1} = \frac{N-1}{N} E_K + 1.$$

$$E_{K+1} - N = \frac{N-1}{N} (E_K - N)$$

Problem K. Kids and Prizes

$$E_{K+1} = \frac{N-1}{N} \sum_{i=0}^{N} i p_K(i) + \sum_{i=0}^{N} p_K(i) = \frac{N-1}{N} E_K + 1.$$

This equation is enough to get E_M using $O(M)$ time.
But we still can improve this result.

$$E_{K+1} = \frac{N-1}{N} E_K + 1.$$

$$E_{K+1} - N = \frac{N-1}{N} (E_K - N) = \left(\frac{N-1}{N}\right)^2 (E_{K-1} - N)$$

Problem K. Kids and Prizes

$$E_{K+1} = \frac{N-1}{N} \sum_{i=0}^{N} i p_K(i) + \sum_{i=0}^{N} p_K(i) = \frac{N-1}{N} E_K + 1.$$

This equation is enough to get E_M using $O(M)$ time.
But we still can improve this result.

$$E_{K+1} = \frac{N-1}{N} E_K + 1.$$

$$E_{K+1} - N = \frac{N-1}{N} (E_K - N) = \left(\frac{N-1}{N}\right)^2 (E_{K-1} - N) =$$

$$= \cdots = \left(\frac{N-1}{N}\right)^K (E_1 - N)$$

Problem K. Kids and Prizes

$$E_{K+1} = \frac{N-1}{N} \sum_{i=0}^{N} i p_K(i) + \sum_{i=0}^{N} p_K(i) = \frac{N-1}{N} E_K + 1.$$

This equation is enough to get E_M using $O(M)$ time.
But we still can improve this result.

$$E_{K+1} = \frac{N-1}{N} E_K + 1.$$

$$E_{K+1} - N = \frac{N-1}{N} (E_K - N) = \left(\frac{N-1}{N}\right)^2 (E_{K-1} - N) =$$

$$= \cdots = \left(\frac{N-1}{N}\right)^K (E_1 - N) = -\left(\frac{N-1}{N}\right)^K (N-1)$$

Problem K. Kids and Prizes

$$E_{K+1} = \frac{N-1}{N}\sum_{i=0}^{N} i p_K(i) + \sum_{i=0}^{N} p_K(i) = \frac{N-1}{N}E_K + 1.$$

This equation is enough to get E_M using $O(M)$ time.
But we still can improve this result.

$$E_{K+1} = \frac{N-1}{N}E_K + 1.$$

$$E_{K+1} - N = \frac{N-1}{N}(E_K - N) = \left(\frac{N-1}{N}\right)^2 (E_{K-1} - N) =$$

$$= \cdots = \left(\frac{N-1}{N}\right)^K (E_1 - N) = -\left(\frac{N-1}{N}\right)^K (N-1)$$

$$E_M = N - \left(\frac{N-1}{N}\right)^{M-1} (N-1)$$

Problem K. Kids and Prizes

$$E_{K+1} = \frac{N-1}{N}\sum_{i=0}^{N} ip_K(i) + \sum_{i=0}^{N} p_K(i) = \frac{N-1}{N}E_K + 1.$$

This equation is enough to get E_M using $O(M)$ time.
But we still can improve this result.

$$E_{K+1} = \frac{N-1}{N}E_K + 1.$$

$$E_{K+1} - N = \frac{N-1}{N}(E_K - N) = \left(\frac{N-1}{N}\right)^2 (E_{K-1} - N) =$$

$$= \cdots = \left(\frac{N-1}{N}\right)^K (E_1 - N) = -\left(\frac{N-1}{N}\right)^K (N-1)$$

$$E_M = N - \left(\frac{N-1}{N}\right)^{M-1} (N-1) = N\left(1 - \left(\frac{N-1}{N}\right)^M\right).$$

Problem K. Kids and Prizes

$$E_{K+1} = \frac{N-1}{N}\sum_{i=0}^{N} i p_K(i) + \sum_{i=0}^{N} p_K(i) = \frac{N-1}{N}E_K + 1.$$

This equation is enough to get E_M using $O(M)$ time.
But we still can improve this result.

$$E_{K+1} = \frac{N-1}{N}E_K + 1.$$

$$E_{K+1} - N = \frac{N-1}{N}(E_K - N) = \left(\frac{N-1}{N}\right)^2 (E_{K-1} - N) =$$

$$= \cdots = \left(\frac{N-1}{N}\right)^K (E_1 - N) = -\left(\frac{N-1}{N}\right)^K (N-1)$$

$$E_M = N - \left(\frac{N-1}{N}\right)^{M-1} (N-1) = N\left(1 - \left(\frac{N-1}{N}\right)^M\right).$$

So E_M might be found using $O(1)$ time.

Problem L. L-Shapes

Idea:	Serge Kashkevich, Ivan Metelsky
Solutions:	Pavel Irzhavski, Serge Kashkevich, Vladimir Kerus, Ivan Metelsky
Tests:	Pavel Irzhavski
Analysis:	Ivan Metelsky
Statement:	Serge Kashkevich, Ivan Metelsky, Pavel Irzhavski

acm International Collegiate Programming Contest

Problem L. L-Shapes

The first thing to note is that the number of pairs of segments is relatively small.

$$\frac{N(N-1)}{2} \leq 12\ 497\ 500.$$

Problem L. L-Shapes

The first thing to note is that the number of pairs of segments is relatively small.

$$\frac{N(N-1)}{2} \leq 12\ 497\ 500.$$

Therefore we can simply test all of them for being L-Shapes.

Problem L. L-Shapes

Let's consider a pair of segments AB and CD, where $A = (x_1, y_1)$, $B = (x_2, y_2)$, $C = (x_3, y_3)$, $D = (x_4, y_4)$.

Problem L. L-Shapes

Let's consider a pair of segments AB and CD, where $A = (x_1, y_1)$, $B = (x_2, y_2)$, $C = (x_3, y_3)$, $D = (x_4, y_4)$.

How to check whether AB and CD form an L-Shape?

Problem L. L-Shapes

Let's consider a pair of segments AB and CD, where $A = (x_1, y_1)$, $B = (x_2, y_2)$, $C = (x_3, y_3)$, $D = (x_4, y_4)$.

How to check whether AB and CD form an L-Shape?

There are two conditions to check:

1. They share an endvertex.
2. The angle between them is 90°.

Problem L. L-Shapes

The first condition is almost trivial to check. It holds when $A = C$ or $A = D$ or $B = C$ or $B = D$.

Problem L. L-Shapes

The first condition is almost trivial to check. It holds when $A = C$ or $A = D$ or $B = C$ or $B = D$.

Checking the second condition is a bit harder. There are several ways to do it, but the easiest one is probably using the scalar product of vectors.

Problem L. L-Shapes

Consider two vectors $\overrightarrow{AB}(x_2 - x_1, y_2 - y_1)$ and $\overrightarrow{CD}(x_4 - x_3, y_4 - y_3)$.

Problem L. L-Shapes

Consider two vectors $\overrightarrow{AB}(x_2 - x_1, y_2 - y_1)$ and $\overrightarrow{CD}(x_4 - x_3, y_4 - y_3)$.

The angle between the segments is 90° if and only if their scalar product is zero:

$$\overrightarrow{AB} \cdot \overrightarrow{CD} = 0.$$

Problem L. L-Shapes

Consider two vectors $\overrightarrow{AB}(x_2 - x_1, y_2 - y_1)$ and $\overrightarrow{CD}(x_4 - x_3, y_4 - y_3)$.

The angle between the segments is 90° if and only if their scalar product is zero:

$$\overrightarrow{AB} \cdot \overrightarrow{CD} = 0.$$

The scalar product of vectors $\overrightarrow{u}(x_u, y_u)$ and $\overrightarrow{v}(x_v, y_v)$ can be calculated as follows:

$$\overrightarrow{u} \cdot \overrightarrow{v} = x_u \cdot x_v + y_u \cdot y_v$$

Problem L. L-Shapes

Consider two vectors $\overrightarrow{AB}(x_2 - x_1, y_2 - y_1)$ and $\overrightarrow{CD}(x_4 - x_3, y_4 - y_3)$.

The angle between the segments is 90° if and only if their scalar product is zero:

$$\overrightarrow{AB} \cdot \overrightarrow{CD} = 0.$$

The scalar product of vectors $\overrightarrow{u}(x_u, y_u)$ and $\overrightarrow{v}(x_v, y_v)$ can be calculated as follows:

$$\overrightarrow{u} \cdot \overrightarrow{v} = x_u \cdot x_v + y_u \cdot y_v$$

So the angle between the given segments is 90 degrees if and only if

$$(x_2 - x_1) \cdot (x_4 - x_3) + (y_2 - y_1) \cdot (y_4 - y_3) = 0.$$

Problem L. L-Shapes

The complexity of the described approach is $O\left(N^2\right)$.

Problem L. L-Shapes

The complexity of the described approach is $O\left(N^2\right)$.

There is also a solution with complexity $O(N \cdot \log N)$. However it's harder to find and to implement this solution, so using it during the contest would be simply a waste of time.

Special thanks to:

Ahto Truu
for statements translation

Denis Kanonik
for testing system management and support