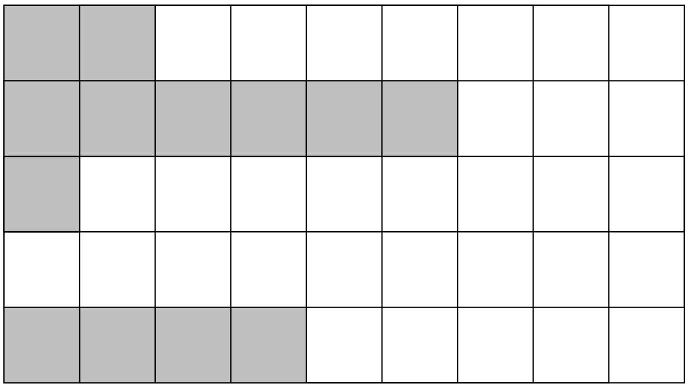
Bugs Integrated, Inc.

Author(s) of the problem: **Michal Forišek**
Contest-related materials by: **Dávid Pál, Martin Pál, Martin Macko**

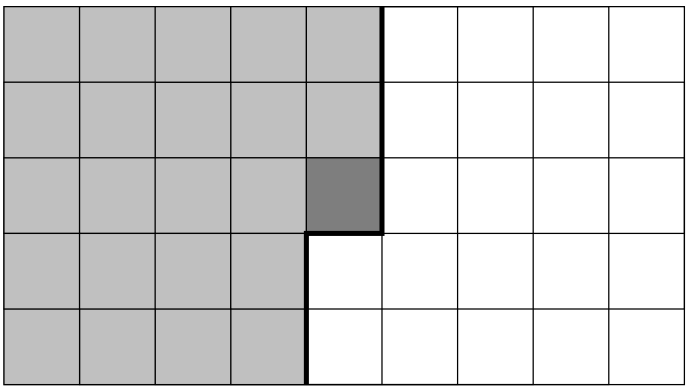
We present a solution based on dynamic programming with the time complexity $O(MN3^M)$. We note that all the coordinates $[x, y]$ of squares are decreased by one, that is $0 \le x < N$, $0 \le y < M$, as is common for C programmers.

Definition 1: Let $B = (b_0, b_1, \ldots, b_{M-1})$ be a vector of numbers (called the *border*). We define the *borderset* $S(B)$ as the set of all squares $[x, y]$ of the plate satisfying $x \le b_y$. Informally, it is the set of those squares that lie to the left of the border B. We usually won't distiguish between the border and its set.



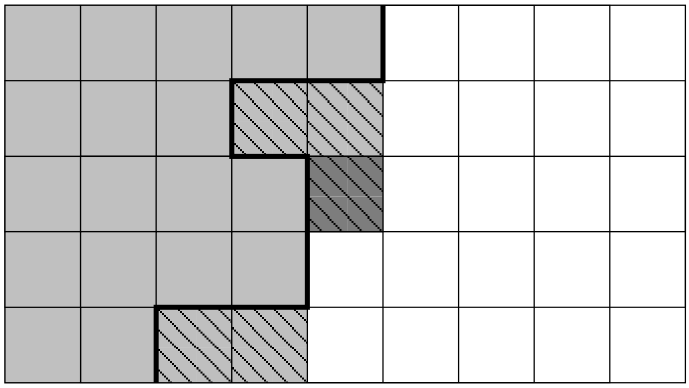
The set of the border $B = (1, 5, 0, -7, 3)$ $(N = 9,\ M = 5)$.

Definition 2: For each $[i, j]$ $(0 \le i < N,\ 0 \le j < M)$ we define an $[i, j]$-*border* as $B[i, j] = (b_0, b_1, \ldots, b_{M-1})$, where $b_0 = b_1 = \cdots = b_j = i$ and $b_{j+1} = \cdots = b_{M-1} = i - 1$. We will call the set $S(B[i, j])$ an $[i, j]$-*baseline*. In other words $S(B[i, j])$ is the set of the squares of the plate, lying to the left of the square $[x, y]$ or in the same column and above it (including the square $[x, y]$).



A $[4, 2]$-border $B[4, 2] = (4, 4, 4, 3, 3)$ and its $[4, 2]$-baseline.

Definition 3: Let's have a baseline with the border $B[i,j]$. Let $P = (p_0, p_1, \ldots, p_{M-1})$, $p_i \in \{0,1,2\}$ be a vector. We will denote the vector $(b_0 - p_0, \ldots, b_{M-1} - p_{M-1})$ as $B - P$. We define a *P-profile* for this baseline as the set $S(B[i,j] - P)$. The symbol $\mathbf{0}$ will denote the all-components-zero profile and e_j will denote a profile that has all components zero except for $p_j = 1$.



The profile $P = (0,2,1,0,2)$ of a $[4,2]$-baseline.

We can think of a profile as of a number between 0 and $3^M - 1$, written in base 3. We will sometimes use P as an index of an array in this sample solution. In the implementation, we first convert P from base 3 into base 10 and then use it as an index.

We can view the plate as the set of its good squares G. The original problem was to determine what is the maximum number of the chips that can be cut out of the plate G.

The basic idea how to solve the problem is to use dynamic programming in the following manner: For each baseline $B[i,j]$ and for each profile P we compute $A[i,j,P]$ – the maximum number of chips that can be cut out of the set (part of the plate) $G \cap S(B[i,j] - P)$. Note that $G \cap S(B[N-1][M-1] - \mathbf{0}) = G$, also the number $A[N-1, M-1, 0]$ gives us the answer to problem.

The part of the plate $G \cap S(B[0,j])$ is too thin to cut any chip out of it, so for the beginning of the table we have $A[0,j,P] = 0$, for any j and any P.

We will process all the other baselines $B[i,j]$ in the order from left to right (i.e. i increases) and the baselines with the same i from top to bottom (i.e. j increases). For each baseline we consider each profile P.

Let's have a fixed baseline $B[i,j]$ and a fixed profile P. There are two possibilities: either $p_j > 0$ or $p_j = 0$.

If $p_j > 0$, then $G \cap S(B[i,j] - P) = G \cap S(B' - P')$, where $B' = S(B[i',j'])$ is the previous baseline in the order we process them and $P' = P - e_j$. Because these two sets are equal, the maximum number of chips that can be cut out of them must also be equal. Therefore $A[i,j,P] = A[i',j',P']$.

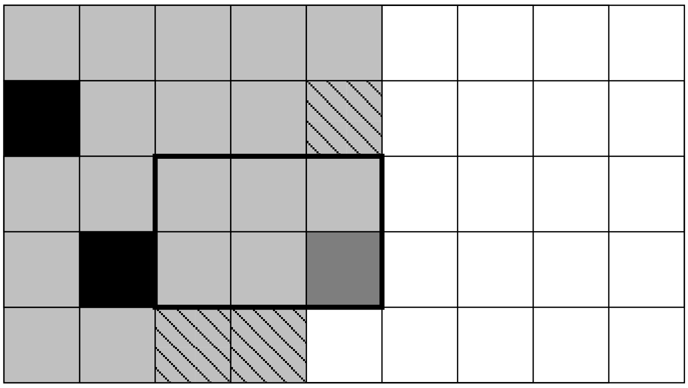
If $p_j = 0$, there are three possibilities how to obtain the desired maximum of chips, that can be cut out of $G \cap S(B - P)$.

- We cut no chip having the lower right corner at the position $[i,j]$.

- We cut a horizontal (3×2) chip having the lower right corner at the position $[i, j]$.
- We cut a vertical (2×3) chip having the lower right corner at the position $[i, j]$.

The maximum number of the chips corresponding to the first case is $A[i', j', P]$, where $S(B[i', j'])$ is the previous baseline in the order we process them.

The maximum number of the chips corresponding to the second case is $A[i'', j'', P + 2e_j + 2e_{j-1}]$, where $S(B[i'', j''])$ is the second previous baseline in the order we process them.



We are processing the baseline $S(B[4, 3])$ with the profile $P = (0, 1, 0, 0, 2)$.
It is possible to cut a horizontal chip here.
The second previous baseline is $S(B[4, 1])$, the new profile will be $P'' = (0, 1, 2, 2, 2)$.
Note that $S(B[4, 1] - P'')$ is exactly $S(B[4, 3] - P)$ without the horizontal chip.

The maximum number of the chips corresponding to the third case is $A[i''', j''', P + e_j + e_{j-1} + e_{j-2}]$, where $S(B[i''', j'''])$ is the third previous baseline in the order we process them.

Clearly $A[i, j, P]$ will be the maximum of these three numbers. (We consider the second and third case only if the corresponding chip can be cut out at this position.)

The test wheter a horizontal or a vertical chip can be cut out at some position is easy. A horizontal chip can be cut out iff there are no bad squares at those positions and $i \geq 2$, $j \geq 1$ and $p_j = p_{j-1} = 0$. Similarly a vertical chip can be cut out iff there are no bad squares at those positions and $i \geq 1$, $j \geq 2$ and $p_j = p_{j-1} = p_{j-2} = 0$. Thus this test takes only a constant amount of time.

It can be easily shown that we have to remember the values $A[i, j, P]$ only for the last four baselines (the one being computed and the three previous ones). There is only $3^M \leq 3^{10} < 60000$ profiles for each baseline, so this will easily fit into memory.

The time complexity of the algorithm presented here is as promised $O(MN3^N)$, because there are exactly 3^M profiles and $O(MN)$ baselines.