You decided to start a new life on a deserted island with your friend Isabelle. Your goal is to fill the island with buildings that have predetermined plot dimensions. For this, you need to divide the land into rectangular plots with the given dimensions, leaving the least possible m^2 of unusable land.

Isabelle tells you the *dimensions* of the island. She also tells you the dimensions of the rectangular plots that you can divide the land into. Any piece of land (the whole island or the already divided plots of the land) can be divided either horizontally or vertically into two rectangular plots with integer dimensions, dividing completely through that land. This is the only way to divide the land, and divided plots cannot be joined together. Since the buildings to be placed in these plots cannot be rotated, the dimensions of the plots also cannot be rotated. You can create zero or more plots with each given dimension. A piece of land is unusable if it is not of any of the desired dimensions after all dividing operations are completed.

Problem

In this exam, you are asked to calculate the most efficient way of dividing the land given the dimensions of the land *X* and *Y*, and the dimensions of the plots in a 2D array of booleans, then return the *m^2 of unusable land* by completing the *divide_land()* function defined below.

```
int divide_land(int X, int Y, bool** possible_plots);
```

- *X, Y*: dimensions X x Y of the total rectangular land of the island
- *possible_plots*: a 2D array of booleans, where each value stands for the existence of a possible plot with the indices of the array cell as dimensions (for example, if possible_plot[2][3] == true, then 2 x 3 is a possible plot dimension that can be used)
- to return: m^2 of unusable land, meaning the land that does not belong to any plot after all the dividing operations.

Constraints and Hints:

- *X* and *Y* are integers <= 600

- *possible_plots* is a 2D array with dimensions (X+1)x(Y+1). So, the maximum dimensions are 601x601. At any given time, there will be at most 200 "true" cells in the 2D array - but the complexity of the optimal solution does not depend on this value.
- You are expected to use a dynamic programming approach to reduce complexity.