Problem B
Image Scaling

Time Limit: 3 seconds
Memory Limit: 512 Megabytes

Problem description

Scaling of image is one frequently used task in any decent image processing software. Nearest neighbor is the simplest and fastest implementation of image scaling technique. It is very useful when speed is the main concern, for example when zooming image for editing or for a thumbnail preview. The principle in image scaling is to have a reference image and using this image as the base to construct a new scaled image. The constructed image will be smaller, larger, or equal in size depending on the scaling ratio. When enlarging an image, we are actually introducing empty spaces in the original base picture. From the image below, an image with dimension (w1 = 4, h1 = 4) is to be enlarged to (w2 = 8, h2 = 8). The black pixels represent empty spaces where interpolation is needed, and the complete picture is the result of nearest neighbor interpolation.

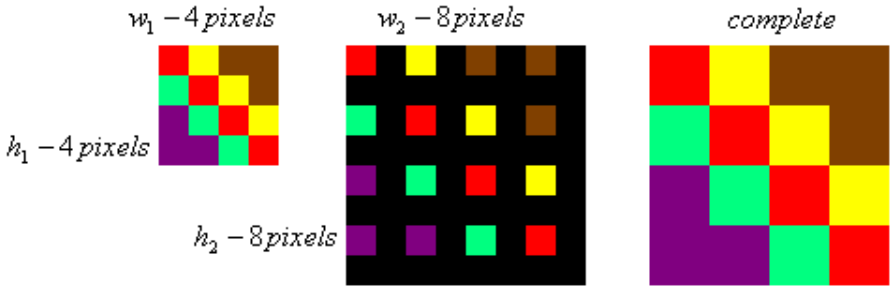

Figure B. 1. Nearest Neighbor Interpolation

Scaling algorithm is to find appropiate spot to put the empty spaces inside the original image, and to fill all those spaces with livelier colors. For the nearest neighbor technique, the empty spaces will be replaced with the nearest neighboring pixel, hence the name. This results in a sharp but jaggy image, and if the enlarge scale is two, it would seem each pixel has doubled in size. Shrinking, in the other hand involves reduction of pixels and it means lost of irrecoverable information. In this case scaling algorithm is to find the right pixels to throw away.

Good scaling algorithm is one that can do up and down scalling without introducing too many conditions (the ifs) in its implementation code, even better if there is none. Nearest neighbor is a no if, up down scaling algorithm. What information it needs are both the horizontal and vertical ratios between the original image and the (to be) scaled image. Consider again the diagram above, w1 and h1 are the width and height of an image, whereas w2 and h2 are the

width and height when enlarged (or shrinked). Calculating the ratio for both horizontal and vertical plane is given by,

$$\left.\begin{aligned} x\_ratio &= \frac{w_1}{w_2} \\ y\_ratio &= \frac{h_1}{h_2} \end{aligned}\right\} w_2, h_2 \neq 0$$

Of course, the not equal to zero is a condition, and will eventually be translated as ifs in coding implementation. Once ratio has been calculated prepare a buffer, or array, or whatever that can store information for the to be constructed image. The size should be enough to store w2*h2 of pixels.

Viet is a software engineerer and he got a job to implement an image resizing function with nearest neighbor technique for his company project. He doesn't have any experience in image processing, so he asked you to help him to implement it.

Input

The first line contains four integers h_1, w_1, h_2 and w_2 ($1 < w_1, h_1, w_2, h_2 \leqslant 1024$), the number rows and columns of the original image matrix and scaled one.

The next h_1 lines contain w_1 integer numbers a_{ij} where $0 \leqslant a_{ij} \leqslant 255$ - the grey value of the pixels of the original greyscale image that are separated by spaces ($0 \leqslant i \leqslant h_1$, $0 \leqslant j \leqslant w_1$).

Output

The next h_2 lines contain w_2 integer numbers b_{ij} where $0 \leqslant b_{ij} \leqslant 255$ - the grey value of the pixels of the constructed image that are separated by spaces ($0 \leqslant i \leqslant h_2$, $0 \leqslant j \leqslant w_2$).

Example:

Input	Output
2 2 4 4 1 0 0 1	1 1 0 0 1 1 0 0 0 0 1 1 0 0 1 1

Input	Output
2 2 3 3 1 0 0 1	1 1 0 1 1 0 0 0 1