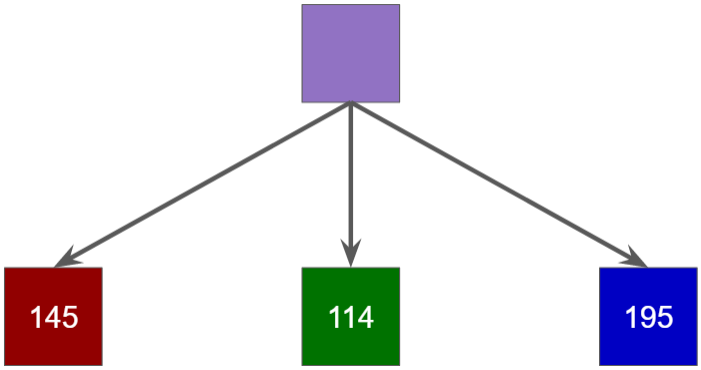
3 Colourblind

Time Limit: 2.0s
Memory Limit: 256MB

3.1 Problem Description

Whiterabbit has been trying to create nice, visually appealing designs for posters. His current poster design is an image H pixels high and W pixels wide. The top-left pixel is $(0, 0)$ and the bottom-right pixel is $(H - 1, W - 1)$. Each pixel's colour is represented by 3 numbers, indicating how red, blue and green the pixel is. To be precise, the colour of the pixel (i, j) is represented by the 3 numbers $R_{i,j}, G_{i,j}$ and $B_{i,j}$, representing the amount of red, blue and green in that pixel.



After finishing his design, Whiterabbit suddenly realised that some of the pixels had colours that were very similar to each other. He defines 2 pixels to be *k-similar* for a positive integer k if the maximum difference in red, blue and green values between the 2 pixels is at most k. More specifically, 2 pixels (a, b) and (c, d) are *k-similar* if:

$$\max\left(\left|R_{a,b} - R_{c,d}\right|, \left|G_{a,b} - G_{c,d}\right|, \left|B_{a,b} - B_{c,d}\right|\right) \leq k$$

where $|x|$ is the absolute value of x. Notably, (a, b) and (a, b) are *k-similar* for all k.

If the colours of adjacent pixels are too similar, viewers who are colourblind might see them as the same colour. They might be looking at one part of the image, look around to see pixels with the same colour before realising that what they are currently looking at is a different colour from what they started with, which can be very confusing.

As such, Whiterabbit defines 2 pixels (a, b) and (c, d) to be *k-confusing* if there exists some sequence of adjacent pixels (sharing a side) starting at pixel (a, b) and ending at pixel (c, d) such that every pair of consecutive pixels in the sequence are *k-similar*, but pixels (a, b) and (c, d) are **not** *k-similar*.

To assist him in improving his design, Whiterabbit will give you Q queries, the j^{th} one being a single integer C_j, and he wants you to determine, for each j, if there exists 2 pixels in the design which are C_j*-confusing*.

3.2 Interaction Protocol

This is a function call problem. You need to add `#include "colourblind.h"` at the start of your program, and implement the following function:

- `void init(int H, int W, int Q, int S, vector<vector<long long> > R, vector<vector<long long> > G, vector<vector<long long> > B)`
  - H is the height of the grid given.
  - W is the width of the grid given.
  - Q is the number of queries that Whiterabbit will ask.
  - S is the subtask this testcase belongs to.
  - R, G and B are 2 dimensional arrays with H rows and W columns. The entry in the i^{th} row and the j^{th} column represents the red, green and blue values of the pixel (i, j) respectively (0-indexed).
  - This function will be called exactly once per testcase, before any queries are given.
- `bool query(long long C)`
  - C is the query value asked by Whiterabbit.
  - This function should return true if there exists 2 pixels in the grid that are C-*confusing*, and false otherwise.

3.3 Sample Grader

In the Attachments, you can find a sample grader `grader.cpp`. This grader is similar **but not exactly identical** to the one used to grade your submitted program. It takes in input in the following format:

1. The first line of input will contain 4 spaced integers, H, W, Q and S respectively.
2. The next H lines of input will contain W spaced integers, the j^{th} number on the i^{th} line being $R_{i,j}$.
3. The next H lines of input will contain W spaced integers, the j^{th} number on the i^{th} line being $G_{i,j}$.
4. The next H lines of input will contain W spaced integers, the j^{th} number on the i^{th} line being $B_{i,j}$.
5. The next line will contain Q spaced integers, the i^{th} one being C_i.

The grader will output Q lines of output. The i^{th} line will be a "Yes" (without quotes) if your program returned 1 for the i^{th} query and "No" (without quotes) if your program returned 0.

A compile command to compile your program (`colourblind.cpp`) with the grader has been included in the Attachments.

3.4 Subtasks

Subtask	Score	H, W	Q	Extra Constraints
1	5	$1 \leq H, W \leq 2$	$Q = 1$	
2	11	$1 \leq H, W \leq 700$	$1 \leq Q \leq 500000$	$R_{i,j} = G_{i,j} = B_{i,j} = i + j$
3	24	$1 \leq H, W \leq 700$	$1 \leq Q \leq 5$	
4	28	$1 \leq H, W \leq 700$	$0 \leq Q \leq 500000$	$1 \leq R_{i,j}, G_{i,j}, B_{i,j} \leq 10^5$
5	32	$1 \leq H, W \leq 700$	$1 \leq Q \leq 500000$	
6	0	Sample Testcases		
For all subtasks: $1 \leq H, W \leq 700, 1 \leq Q \leq 500000$ $0 \leq R_{i,j}, G_{i,j}, B_{i,j}, C_i \leq 10^{17}$				

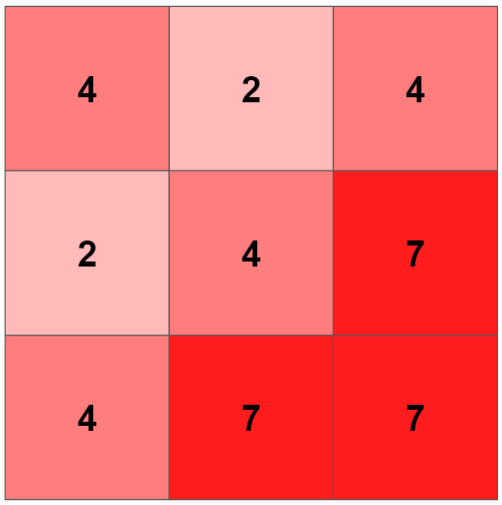
3.5 Sample Testcases

Do note that these inputs and outputs are meant for a sample grader compiled with your program. They are given as `sample1.txt` and `sample2.txt` in the Attachments.

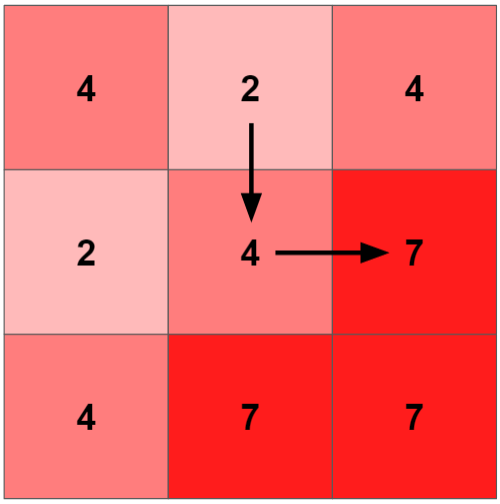
standard input	standard output
3 3 5 6 4 2 4 2 4 7 4 7 7 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2 3 4 5	No No Yes Yes No
2 2 5 6 1 4 3 2 4 3 7 1 5 3 3 4 1 2 3 5 8	No No Yes Yes No

3.6 Sample Testcase Explanation

Shown below are the red values of the image for the first sample, the blue and green values don't matter as they are all 0.



Note that the pixels $(1, 2)$ and $(2, 3)$ are both *3-confusing* and *4-confusing* as shown below.



$(1, 2)$ and $(1, 3)$ are *3-similar* and *4-similar* while $(1, 3)$ and $(2, 3)$ are also *3-similar* and *4-similar*, but $(1, 2)$ and $(2, 3)$ are neither *3-similar* nor *4-similar* as the difference in their red values is 5.