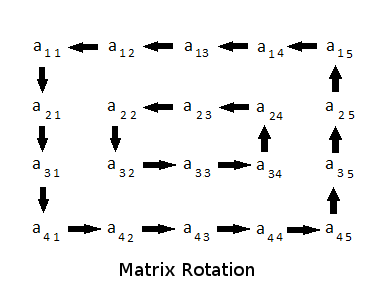
Difficulty: Hard Points: 80

Problem: Matrix Rotation Algo

You are given a 2D matrix, a, of dimension MxN and a positive integer R. You have to rotate the matrix R times and print the resultant matrix. Rotation should be in anti-clockwise direction.

Rotation of a 4x5 matrix is represented by the following figure. Note that in one rotation, you have to shift elements by one step only (refer sample tests for more clarity).



It is guaranteed that the minimum of M and N will be even.

**Input Format**   
First line contains three space separated integers, M, N and R, where M is the number of rows, N is number of columns in matrix, and R is the number of times the matrix has to be rotated.   
Then M lines follow, where each line contains N space separated positive integers. These M lines represent the matrix.

**Constraints**   
2 <= M, N <= 300   
1 <= R <= 109   
min(M, N) % 2 == 0   
1 <= aij <= 108, where i ∈ [1..M] & j ∈ [1..N]

**Output Format**   
Print the rotated matrix.

**Sample Input #00**

4 4 1

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

**Sample Output #00**

2 3 4 8

1 7 11 12

5 6 10 16

9 13 14 15

**Sample Input #01**

4 4 2

1 2 3 4

5 6 7 8

9 10 11 12

13 14 15 16

**Sample Output #01**

3 4 8 12

2 11 10 16

1 7 6 15

5 9 13 14

**Sample Input #02**

5 4 7

1 2 3 4

7 8 9 10

13 14 15 16

19 20 21 22

25 26 27 28

**Sample Output #02**

28 27 26 25

22 9 15 19

16 8 21 13

10 14 20 7

4 3 2 1

**Sample Input #03**

2 2 3

1 1

1 1

**Sample Output #03**

1 1

1 1

**Explanation**   
Sample Case #00: Here is an illustration of what happens when the matrix is rotated once.

1 2 3 4 2 3 4 8

5 6 7 8 1 7 11 12

9 10 11 12 -> 5 6 10 16

13 14 15 16 9 13 14 15

Sample Case #01: Here is what happens when to the matrix after two rotations.

1 2 3 4 2 3 4 8 3 4 8 12

5 6 7 8 1 7 11 12 2 11 10 16

9 10 11 12 -> 5 6 10 16 -> 1 7 6 15

13 14 15 16 9 13 14 15 5 9 13 14

Sample Case #02: Following are the intermediate states.

1 2 3 4 2 3 4 10 3 4 10 16 4 10 16 22

7 8 9 10 1 9 15 16 2 15 21 22 3 21 20 28

13 14 15 16 -> 7 8 21 22 -> 1 9 20 28 -> 2 15 14 27 ->

19 20 21 22 13 14 20 28 7 8 14 27 1 9 8 26

25 26 27 28 19 25 26 27 13 19 25 26 7 13 19 25

10 16 22 28 16 22 28 27 22 28 27 26 28 27 26 25

4 20 14 27 10 14 8 26 16 8 9 25 22 9 15 19

3 21 8 26 -> 4 20 9 25 -> 10 14 15 19 -> 16 8 21 13

2 15 9 25 3 21 15 19 4 20 21 13 10 14 20 7

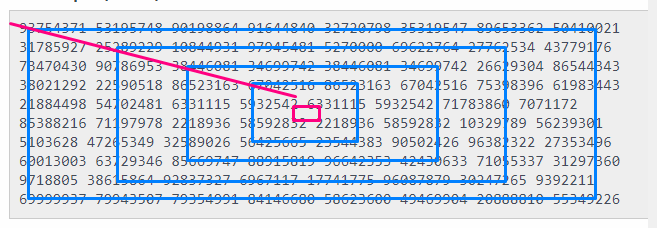
1 7 13 19 2 1 7 13 3 2 1 7 4 3 2 1

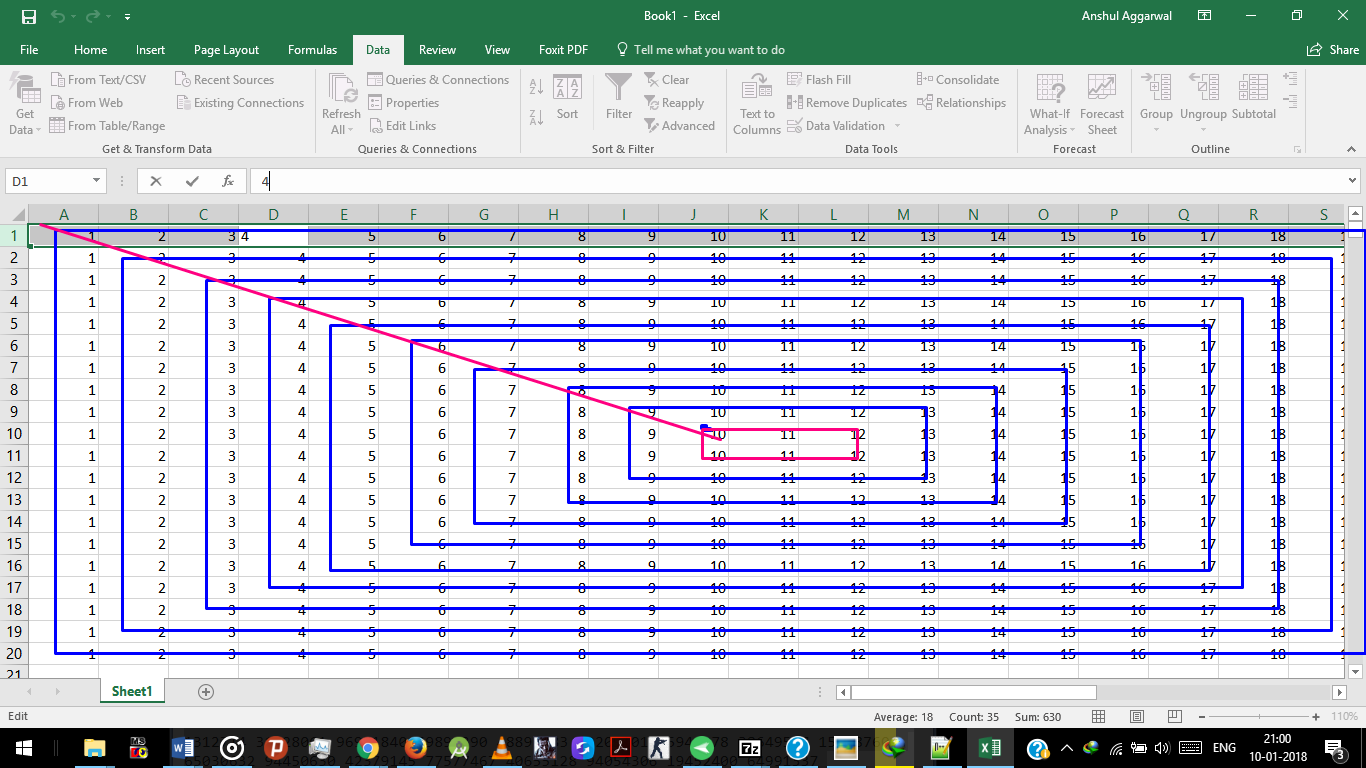
Sample Case #03: As all elements are same, any rotation will reflect the same matrix.

Adopted Approach

* Bifurcate the matrix into different layers each forming a rectangle
* Start from the upper right corner for each layer and read the matrix layer’s elements into an array clockwise
* Process the array for left rotation to compensate for the clockwise rotations of the matrix
* Similar to 2 step follow the same loop and order but this time the processed left rotated array’s elements are fed to the matrix layer
* Repeat until all the layers are processed.

Visual Aid:





Solution

#include <cmath>

#include <cstdio>

#include <vector>

#include <iostream>

#include <algorithm>

using namespace std;

int main()

{

/\*Feeding the data\*/

int rows, cols;

long i,j,k;

long rotations;

cin>>rows >>cols>>rotations;

long array[rows][cols];

for(long i=0; i<rows; i++)

{for(long j=0; j<cols;j++)

{cin>>array[i][j];}

}

/\*------------- The MEAT -----------------\*/

long const fixrows=rows;

long const fixcols=cols;

vector <long> vec;

rows+=2; cols+=2; //Equates the decrement during the first pass

for(long k=0; k<min(fixcols/2,fixrows/2); k++)

//runs for each layer of the matrix

//min() formulates total layers to process

{

vec.erase(vec.begin(),vec.end()); //resets the vector for each layer nextr=0+k; long nextc=fixcols-1-k;

//nextr nextc ---> coordinates of the upper right corner of the layer

long counter=0; // ---- > the total elements in the layer of the matrix

rows-=2; cols-=2; //decrements per loop to move to next layer

while(counter<=(rows\*2+(cols-2)\*2)-1)

//while total elements of layer are not listed

{

//nextr and nextc are the index of next element of matrix layer's

Element from the upper right corner in clockwise fashion

//based on counter value, nextr and nextc calculated for each element position in the matrix layer

counter++;

if(counter<=(rows-1))

{nextr+=1; nextc+=0;}

else if(counter>(rows-1) && counter<=(rows+cols-2))

{nextr+=0; nextc-=1;}

else if(counter>(rows+cols-2) && counter<=(2\*rows+cols-3))

{nextr-=1; nextc+=0;}

else if(counter>(2\*rows-3+cols) && counter<=(2\*rows-4+2\*cols))

{nextr+=0; nextc+=1;}

else if(counter>2\*rows-4+2\*cols-1)

{nextr+=1; nextc+=1;}

vec.push\_back(array[nextr][nextc]);

//places the elements of the layer into vector vec

//starting from the right vertical column moving clockwise

// ^ ---> |

// | <--- v

}

//vector feeding complete

//---------repeat the upper loop now to feed the value to matrix layer's indexes from the array instead of feeding to array

nextr=0+k; nextc=fixcols-1-k; //nextr nextc find the upper right corner of the next matrix layer to process

counter=0; long size=vec.size();

while(counter<=(rows\*2+(cols-2)\*2)-1) //while all the elements of the matrix layer are not re fed.

{

counter++;

if(counter<=(rows-1))

{nextr+=1; nextc+=0;}

else if(counter>(rows-1) && counter<=(rows+cols-2))

{nextr+=0; nextc-=1;}

else if(counter>(rows+cols-2) && counter<=(2\*rows+cols-3))

{nextr-=1; nextc+=0;}

else if(counter>(2\*rows-3+cols) && counter<=(2\*rows-4+2\*cols))

{nextr+=0; nextc+=1;}

else if(counter>2\*rows-4+2\*cols-1)

{nextr+=1; nextc+=1;}

array[nextr][nextc]=vec[((counter-1)+(size+1)\*rotations)%size];

//the elements are left rotated in the vector

//and then inserted to matrix layer's elements

//to compensate for the clockwise rotations

}

}

//Printing the final processed matrix

for(i=0; i<fixrows; i++)

{ for(j=0; j<fixcols; j++)

{cout<<array[i][j]<<" ";}

cout<<endl;

}

}

- `’Anshul AgGarwal