**############################################################**

**1- Multiply the matrices**

*When dealing with matrices, you may, sooner or later, run into the elusive task of matrix multiplication. Here, we will try to multiply two matrices and hope to understand the process.*

Two matrices A[][] and B[][] can only be multiplied if number of columns in A is equal to number of rows in B. The dimensions of the resultant matrix will have A's row size and B's column size.

Given two matrices A and B having (n1 x m1) and (n2 x m2) dimensions respectively. Multiply A and B.

**Example 1:**

**Input:**

n1 = 3, m1 = 2

A[][] = {{4, 8},

{0, 2}

{1, 6}}

n2 = 2, m2 = 2

B[][] = {{5, 2},

{9, 4}}

**Output:** 92 40 18 8 59 26

**Explanation:**

Matrices are of size 3 x 2 and 2 x 2 which

results in 3 x 2 matrix with elements as:

res[][] = {{92, 40},

{18, 8}

{59, 26}}

**Example 2:**

**Input:**

n1 = 1, m1 = 1

A[][] = {2}

n2 = 1, m2 = 1

B[][] = {3}

**Output:** 6

**Explanation:** Both matrices are of size 1 x 1

which results in 1 x 1 matrix having element 6.

**Your Task:**  
You dont need to read input or print anything. Complete the function **multiplyMatrix()** that takes A and B as input parameters and returns a matrix containing their product. If the multiplication is not possible return an empty matrix.

**Expected Time Complexity:** O(N1 \* M1 \* M2)  
**Expected Auxiliary Space:**O(N1 \* M2) for resultant matrix.

**Constraints:**  
1 <= n1, m1, n2, m2 <= 30  
0 <= Ai, Bi <= 100

vector<vector<int>> multiplyMatrix( const vector<vector<int>>& A, const vector<vector<int>>& B)

{ int n1=A.size();

int m1=A[0].size();

int n2=B.size();

int m2=B[0].size();

vector<vector<int>> ans;

if(m1!=n2)

return ans;

int sum=0;

for(int i=0;i<n1;i++)

{ vector<int> v;

for(int j=0;j<m2;j++)

{ sum=0;

for(int k=0;k<m1;k++)

sum+=A[i][k]\*B[k][j];

v.push\_back(sum);

}

ans.push\_back(v);

}

return ans;

}

**############################################################**

**2- Determinant of a Matrix**

Given a square matrix of size N x N. The task is to find the [**determinant**](https://en.wikipedia.org/wiki/Determinant) of this matrix.  
  
**Example 1:**

**Input**:

N = 4

matrix[][] = {{1, 0, 2, -1},

  {3, 0, 0, 5},

  {2, 1, 4, -3},

  {1, 0, 5, 0}}

**Output**: 30

**Explanation**:

Determinant of the given matrix is 30.

**Example 2:**

**Input**:

N = 3

matrix[][] = {{1, 2, 3},

  {4, 5, 6},

  {7, 10, 9}}

**Output**: 12

**Explanation**:

Determinant of the given matrix is 12.

**Your Task:**  
You don't need to read input or print anything. Complete the function **determinantOfMatrix()**that takesmatrixand its size n as input parametersand returns the determinant of the matrix.  
  
**Expected Time Complexity:** O(N4)  
**Expected Auxiliary Space:** O(N2)  
  
**Constraints:**  
1 <= N <= 8  
-10 <= mat[i][j] <= 10

int determinantOfMatrix(vector<vector<int> > m, int n)

{ int det=0; // the determinant value will be stored here

if(m.size()==1)

return m[0][0]; // no calculation needed

else if(m.size()==2)

return (m[0][0]\*m[1][1]-m[0][1]\*m[1][0]);

else

{ for(int p=0;p<m[0].size();p++)

{ vector<vector<int>> sm;

for (int i=1;i<m.size();i++)

{ vector<int> v;

for (int j=0;j<m[i].size();j++)

{ if (j!=p)

v.push\_back(m[i][j]);

}

if(v.size()>0)

sm.push\_back(v);

}

det=det + m[0][p]\*pow(-1,p)\*determinantOfMatrix(sm,m.size());

}

return det;

}

}

**############################################################**

**3- Transpose of Matrix**

Write a program to find the transpose of a square matrix of size N\*N. Transpose of a matrix is obtained by changing rows to columns and columns to rows.  
  
**Example 1:**

**Input**:

N = 4

mat[][] = {{1, 1, 1, 1},

  {2, 2, 2, 2}

  {3, 3, 3, 3}

  {4, 4, 4, 4}}

**Output**:

{{1, 2, 3, 4},

 {1, 2, 3, 4}

 {1, 2, 3, 4}

 {1, 2, 3, 4}}

**Example 2:**

**Input**:

N = 2

mat[][] = {{1, 2},

  {-9, -2}}

**Output**:

{{1, -9},

 {2, -2}}

**Your Task:**  
You dont need to read input or print anything. Complete the function **transpose**() which takes matrix[][] and N as input parameter and finds the transpose of the input matrix. You need to do this in-place. That is you need to update the original matrix with the transpose.   
  
**Expected Time Complexity:** O(N \* N)  
**Expected Auxiliary Space:** O(1)  
  
**Constraints:**  
1 <= N <= 100  
-103 <= mat[i][j] <= 103

void transpose(vector<vector<int> >& matrix, int n)

{ for(int i=0;i<n;i++)

for(int j=i;j<n;j++)

swap(matrix[i][j],matrix[j][i]);

}

**############################################################**

**4- Rotate by 90 degree**

Given asquarematrix of size **N x N**. The task is to rotate it by**90 degrees in anti-clockwise** direction without using any extra space.   
  
**Example 1:**

**Input**:

N = 3

matrix[][] = {{1, 2, 3},

  {4, 5, 6}

  {7, 8, 9}}

**Output**:

Rotated Matrix:

3 6 9

2 5 8

1 4 7

**Example 2:**

**Input**:

N = 2

matrix[][] = {{1, 2},

  {3, 4}}

**Output**:

Rotated Matrix:

2 4

1 3

**Your Task:**  
You dont need to read input or print anything. Complete the function **rotateby90**() which takes the matrix as input parameter and rotates it by 90 degrees in anti-clockwise direction without using any extra space. You have to modify the input matrix **in-place**.   
  
**Expected Time Complexity**: O(N2)  
**Expected Auxiliary Space**: O(1)  
  
**Constraints:**  
1 ≤ N ≤ 100  
1 <= matrix[][] <= 1000

void rotateby90(vector<vector<int> >& matrix, int n)

{ for(int i=0;i<n;i++)

for(int j=i;j<n;j++)

swap(matrix[i][j],matrix[j][i]);

for(int i=0;i<n/2;i++)

for(int j=0;j<n;j++)

swap(matrix[i][j],matrix[n-1-i][j]);

}

**############################################################**

**5- Spirally traversing a matrix**

Given a matrix of size r\*c. Traverse the matrix in spiral form.

**Example 1:**

**Input**:

r = 4, c = 4

matrix[][] = {{1, 2, 3, 4},

  {5, 6, 7, 8},

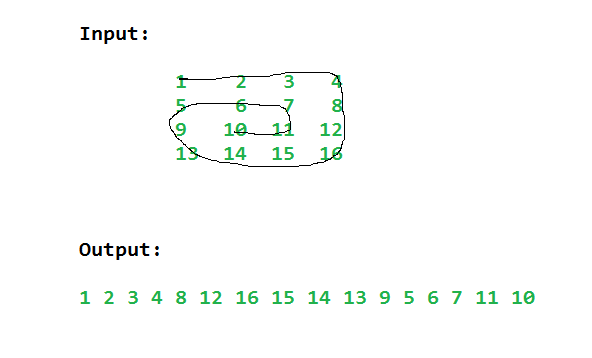
  {9, 10, 11, 12},

  {13, 14, 15,16}}

**Output**:

1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

**Explanation**:



**Example 2:**

**Input**:

r = 3, c = 4

matrix[][] = {{1, 2, 3, 4},

  {5, 6, 7, 8},

  {9, 10, 11, 12}}

**Output**:

1 2 3 4 8 12 11 10 9 5 6 7

**Explanation**:

Applying same technique as shown above,

output for the 2nd testcase will be

1 2 3 4 8 12 11 10 9 5 6 7.

**Your Task:**  
You dont need to read input or print anything. Complete the function **spirallyTraverse()**that takes **matrix, r**and**c**as input parametersand returns a list of integers denoting the spiral traversal of matrix.   
  
**Expected Time Complexity:** O(r\*c)  
**Expected Auxiliary Space:** O(r\*c), for returning the answer only.  
  
**Constraints:**  
1 <= r, c <= 100  
0 <= matrixi <= 100

vector<int> spirallyTraverse(vector<vector<int> > matrix, int m, int n)

{ vector<int> v;

int dir=0,l=0,r=n-1,u=0,d=m-1;

while(l<=r && u<=d)

{ if(dir==0)

{ for(int i=l;i<=r;i++)

v.push\_back(matrix[u][i]);

u++;

}

else if(dir==1)

{ for(int i=u;i<=d;i++)

v.push\_back(matrix[i][r]);

r--;

}

else if(dir==2)

{ for(int i=r;i>=l;i--)

v.push\_back(matrix[d][i]);

d--;

}

else if(dir==3)

{ for(int i=d;i>=u;i--)

v.push\_back(matrix[i][l]);

l++;

}

dir=(dir+1)%4;

}

return v;

}

**############################################################**

**6- Search in a row-column sorted Matrix**

Given a matrix of size n x m, where every row and column is**sorted in increasing order**, and a number **x.** Find whether element x is present in the matrix or not.  
  
**Example 1:**

**Input**:

n = 3, m = 3, x = 62

matrix[][] = {{ 3, 30, 38},

  {36, 43, 60},

  {40, 51, 69}}

**Output**: 0

**Explanation**:

62 is not present in the matrix,

so output is 0.

**Example 2:**

**Input**:

n = 1, m = 6, x = 55

matrix[][] = {{18, 21, 27, 38, 55, 67}}

**Output**: 1

**Explanation**: 55 is present in the matrix.

**Your Task:**  
You don't need to read input or print anything. Complete the function **search()** that takes **n, m, x,**and **matrix[][]** as input parameters and **returns a boolean value.** True if x is present in the matrix and false if it is not present.  
  
**Expected Time Complexity:** O(N + M)  
**Expected Auxiliary Space:** O(1)  
  
**Constraints:**  
1 <= N, M <= 1000  
1 <= mat[][] <= 105  
1 <= X <= 1000

bool search(vector<vector<int> > matrix, int n, int m, int k)

{ int l=0,r=m-1;

while(l<n && r>=0)

{ if(matrix[l][r]==k)

return 1;

else if(matrix[l][r]<k)

l++;

else

r--;

}

return 0;

}

**############################################################**

**7- Max rectangle**

Given a binary matrix. Find the maximum area of a rectangle formed only of 1s in the given matrix.

**Example 1:**

**Input:**

n = 4, m = 4

M[][] = {{0 1 1 0},

{1 1 1 1},

{1 1 1 1},

{1 1 0 0}}

**Output:** 8

**Explanation:** For the above test case the

matrix will look like

0 1 1 0

1 1 1 1

1 1 1 1

1 1 0 0

the max size rectangle is

1 1 1 1

1 1 1 1

and area is 4 \*2 = 8.

**Your Task:**  
Your task is to complete the function **maxArea** which returns the maximum size rectangle area in a binary-sub-matrix with all 1’s. The function takes 3 arguments the first argument is the Matrix M[ ] [ ] and the next two are two integers n and m which denotes the size of the matrix M.

**Expected Time Complexity** : O(n\*m)  
**Expected Auixiliary Space** : O(m)

**Constraints:**  
1<=n,m<=1000  
0<=M[][]<=1  
  
**Note:**The **Input/Ouput** format and **Example** given are used for system's internal purpose, and should be used by a user for **Expected Output** only. As it is a function problem, hence a user should not read any input from stdin/console. The task is to complete the function specified, and not to write the full code.

int area(vector<int> v, int n)

{ int ans=0;

vector<int> l;

vector<int> r;

stack<int> stk;

for(int i=0;i<n;i++)

{ if(stk.empty())

{ l.push\_back(0);

stk.push(i);

}

else

{ while(!stk.empty() && v[stk.top()]>=v[i])

stk.pop();

l.push\_back(stk.empty()?0:stk.top()+1);

stk.push(i);

}

}

while(!stk.empty())

stk.pop();

for(int i=n-1;i>=0;i--)

{ if(stk.empty())

{

r.push\_back(n-1);

stk.push(i);

}

else

{ while(!stk.empty() && v[stk.top()]>=v[i])

stk.pop();

r.push\_back(stk.empty()?n-1:stk.top()-1);

stk.push(i);

}

}

reverse(r.begin(),r.end());

for(int i=0;i<n;i++)

ans=max(ans, v[i]\*(r[i]-l[i]+1));

return ans;

}

int maxArea(int mat[MAX][MAX], int n, int m) {

// Your code here

vector<int> v;

stack<int> stk;

int ans=0;

for(int i=0;i<m;i++)

v.push\_back(mat[0][i]);

ans=area(v,m);

for(int i=1;i<n;i++)

{ for(int j=0;j<m;j++)

{ if(mat[i][j]==1)

v[j]+=mat[i][j];

else

v[j]=0;

}

ans=max(ans,area(v,m));

}

return ans;

}

**############################################################**

**8- Largest rectangle of 1s with swapping of columns allowed**

Given a matrix **mat** of size **R\*C** with 0 and 1’s, find the largest rectangle of all 1’s in the matrix. The rectangle can be formed by swapping any pair of columns of given matrix.

**Example 1:**

**Input:**

R = 3, C = 5

mat[][] = {{0, 1, 0, 1, 0},

{0, 1, 0, 1, 1},

{1, 1, 0, 1, 0}};

**Output:** 6

**Explanation:** The largest rectangle's area

is 6. The rectangle can be formed by

swapping column 2 with 3. The matrix

after swapping will be

0 0 1 1 0

0 0 1 1 1

1 0 1 1 0

**Example 2:**

**Input:**

R = 4, C = 5

mat[][] = {{0, 1, 0, 1, 0},

{0, 1, 1, 1, 1},

{1, 1, 1, 0, 1},

{1, 1, 1, 1, 1}};

**Output:** 9

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **maxArea()** which takes the 2D array of booleans **mat,** **r**and**c**as parameters and returns an integer denoting the answer.

**Expected Time Complexity:** O(R\*(R + C))  
**Expected Auxiliary Space:** O(R\*C)

**Constraints:**  
1<= R,C <=103  
0 <= mat[i][j] <= 1

int maxArea(vector<bool> mat[], int r, int c) {

// code here

vector<int> v;

vector<vector<int>> m;

for(int i=0;i<c;i++)

v.push\_back(mat[0][i]);

m.push\_back(v);

for(int i=1;i<r;i++)

{ for(int j=0;j<c;j++)

{ if(mat[i][j]==0)

v[j]=0;

else

v[j]+=mat[i][j];

}

m.push\_back(v);

}

for(int i=0;i<r;i++)

sort(m[i].begin(),m[i].end(),greater<int>());

/\*

for(int i=0;i<r;i++)

{ for(int j=0;j<c;j++)

cout<<m[i][j]<<" ";

cout<<endl;

}\*/

int ans=0;

for(int i=0;i<r;i++)

for(int j=0;j<c;j++)

ans=max(ans,m[i][j]\*(j+1));

return ans;

}

**############################################################**