1. Adding two matrices

vector<vector<int> > sumMatrix( vector<vector<int> >& A, vector<vector<int> >& B)

{

if(A.size()!=B.size() || A[0].size()!=B[0].size())

return {{-1}};

vector<vector<int>> C;

for(int i = 0; i<A.size(); i++)

{

cout<<"Test"<<i;

vector<int> ans;

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

{

ans.push\_back(A[i][j]+B[i][j]);

}

C.push\_back(ans);

}

return C;

}

1. Sum of upper and lower triangles

vector<int> sumTriangles(const vector<vector<int> >& matrix, int n)

{

vector<int> ans{0,0};

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

{

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

{

ans[0]+=matrix[i][j];

ans[1]+=matrix[n-i-1][n-j-1];

}

}

return ans;

}

1. Multiply the matrices

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

{

if(A[0].size()!=B.size())

return {{-1}};

vector<vector<int>> C(A.size(), vector<int> (B[0].size(), 0));

for(int i = 0; i<A.size(); i++)

{

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

{

for(int k=0;k<B.size();k++)

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

}

}

return C;

}

1. Print matrix in snake form

vector<int> snakePattern(vector<vector<int> > matrix)

{

vector<int> ans;

for(int i = 0;i<matrix.size();i++)

{

if(i%2==0)

{

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

{

ans.push\_back(matrix[i][j]);

}

}

else

{

for(int j = matrix[i].size()-1; j>=0; j--)

{

ans.push\_back(matrix[i][j]);

}

}

}

return ans;

}

1. Transpose of matrix

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

{

int swap = 0;

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

{

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

{

swap = matrix[i][j];

matrix[i][j]=matrix[j][i];

matrix[j][i]=swap;

}

}

}

1. Rotate by 90 degrees

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

{

int swap = 0;

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

{

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

{

swap = matrix[i][j];

matrix[i][j]=matrix[j][i];

matrix[j][i]=swap;

}

}

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

{

vector<int> swap2 = matrix[i];

matrix[i] = matrix[n-i-1];

matrix[n-i-1]=swap2;

}

}

1. Determinant of matrix

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

{

if(n==1)

return matrix[0][0];

int t = 1, det = 0;

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

{

vector<vector<int>> num;

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

{

vector<int> res;

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

{

if(k==i)

continue;

res.push\_back(matrix[j][k]);

}

num.push\_back(res);

}

det = det+t\*matrix[0][i]\*determinantOfMatrix(num, n-1);

t = -t;

}

return det;

}

1. Boundary traversal of matrix

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

{

vector<int> ans;

if(n==1)

return matrix[0];

if(m==1)

{

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

ans.push\_back(matrix[i][0]);

return ans;

}

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

{

ans.push\_back(matrix[0][i]);

}

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

{

ans.push\_back(matrix[i][m-1]);

}

for(int i = m-2;i>=0;i--)

{

ans.push\_back(matrix[n-1][i]);

}

for(int i = n-2;i>0;i--)

{

ans.push\_back(matrix[i][0]);

}

return ans;

}

1. Exchange matrix columns

void exchangeColumns(vector<vector<int> > &matrix)

{

for(int i=0;i<matrix.size();i++)

{

int n = matrix[i].size();

int swap = matrix[i][0];

matrix[i][0]=matrix[i][n-1];

matrix[i][n-1] = swap;

}

}

1. Spirally traversing a matrix

vector<int> spirallyTraverse(vector<vector<int> > matrix, int r, int c)

{

vector<int> ans;

if(r==1)

return matrix[0];

if(c==1)

{

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

ans.push\_back(matrix[i][0]);

return ans;

}

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

{

ans.push\_back(matrix[0][i]);

}

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

{

ans.push\_back(matrix[i][c-1]);

}

for(int i = c-2;i>=0;i--)

{

ans.push\_back(matrix[r-1][i]);

}

for(int i = r-2;i>0;i--)

{

ans.push\_back(matrix[i][0]);

}

if(c-2>0&&r-2>0)

{

vector<vector<int>> cpy;

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

{

vector<int> test;

for(int j = 1;j<c-1;j++)

{

test.push\_back(matrix[i][j]);

}

cpy.push\_back(test);

}

vector<int> res = spirallyTraverse(cpy,r-2,c-2);

ans.insert(ans.end(), res.begin(), res.end());

}

return ans;

}

1. Reversing columns of matrix

void reverseCol(vector<vector<int> > &matrix)

{

for(int i=0; i<matrix.size(); i++)

{

reverse(matrix[i].begin(), matrix[i].end());

}

}

1. Interchanging rows of matrix

void interchangeRows(vector<vector<int> > &matrix)

{

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

}

1. Search in a row column sorted matrix

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

{

int i = 0, j = m-1;

while(i<n&&j>=0)

{

if(matrix[i][j]==x)

return true;

else if(x>matrix[i][j])

i++;

else

j--;

}

return false;

}

};

1. Boolean matrix

void booleanMatrix(vector<vector<int> > &matrix)

{

vector<int> row, col;

for(int i=0;i<matrix.size();i++)

{

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

{

if(matrix[i][j])

{

row.push\_back(i);

col.push\_back(j);

}

}

}

for(int i=0;i<matrix.size();i++)

{

for(int j=0; j<col.size();j++)

{

matrix[i][col[j]]=1;

}

}

for(int i=0;i<row.size();i++)

{

matrix[row[i]] = vector<int> (matrix[row[i]].size(), 1);

}

}

1. Make matrix beautiful

int findMinOpeartion(vector<vector<int> > matrix, int n)

{

int count = 0, row[n], col[n];

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

{

row[i]=0, col[i]=0;

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

{

row[i]+=matrix[i][j];

col[i]+=matrix[j][i];

}

}

while(1)

{

bool found = true;

int rowmin=0,colmin=0, target = row[0];

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

{

if(row[rowmin]>row[i])

rowmin = i;

if(col[colmin]>col[i])

colmin = i;

}

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

{

if(row[i]!=target ||col[i]!=target)

{

found = false;

break;

}

}

if(found) break;

row[rowmin]++;

col[colmin]++;

count++;

}

return count;

}