Program for Rank of Matrix

What is rank of a matrix?

Rank of a matrix A of size M x N is defined as

(a) Maximum number of linearly independent column vectors in the matrix or

(b) Maximum number of linearly independent row vectors in the matrix.

We strongly recommend that you click here and practice it, before moving on to the solution.

Example:

```
Input:  mat[][] = {{10,   20,   10},
                   {20,   40,   20},
                   {30,   50,   0}}
Output: Rank is 2
Explanation: Ist and IInd rows are linearly dependent.
             But Ist and  3rd or IInd and IIIrd  are
             independent.


Input:  mat[][] = {{10,   20,   10},
                   {-20, -30,   10},
                   {30,   50,   0}}
Output: Rank is 2
Explanation: Ist and IInd rows are linearly independent.
             So rank must be atleast 2. But all three rows
             are linearly dependent (the first is equal to
             the sum of the second and third) so the rank
             must be less than 3.
```

In other words rank of A is the largest order of any non-zero minor in A where order of a minor is the side-length of the square sub-matrix of which it is determinant.

So if M < N then maximum rank of A can be M else it can be N, in general rank of matrix can't be greater than min(M, N). The rank of a matrix would be zero only if the matrix had no non-zero elements. If a matrix had even one non-zero element, its minimum rank would be one. **How to find Rank?**

The idea is based on conversion to Row echelon form.

```
1) Let the input matrix be mat[][].  Initialize rank equals
   to number of columns

// Before we visit row 'row',  traversal of previous
// rows make sure that mat[row][0],....mat[row][row-1]
// are 0.
2) Do following for row = 0 to rank-1.

  a) If mat[row][row] is not zero, make all elements of
     current column as 0 except the element mat[row][row]
     by finding appropriate multiplier and adding a the
     multiple of row 'row'

  b) Else (mat[row][row] is zero). Two cases arise:
       (i) If there is a row below it with non-zero entry in
           same column, then swap current 'row' and that row.
       (ii) If all elements in current column below mat[r][row]
            are 0, then remove this column by swapping it with
            last column and  reducing number of rank by 1.
     Reduce row by 1 so that this row is processed again.

3) Number of remaining columns is rank of matrix.
```

Example:

```
Input:  mat[][] = {{10,   20,   10},
                   {-20, -30,   10},
                   {30,   50,   0}}

row = 0:
Since mat[0][0] is not 0, we are in case 2.a of above algorithm.
We set all entries of 0'th column as 0 (except entry mat[0][0]).
To do this, we subtract R1*(-2) from R2, i.e., R2 --> R2 - R1*(-2)
         mat[][] = {{10,   20,   10},
                    { 0,    10,   30},
                     {30,   50,   0}}
            And subtract R1*3 from R3, i.e., R3   --> R3 - R1*3
         mat[][] = {{10,   20,   10},
                    { 0,   10,   30},
                    { 0,  -10,  -30}}

row = 1:
Since mat[1][1] is not 0, we are in case 2.a of above algorithm.
We set all entries of 1st column as 0 (except entry mat[1][1]).
To do this, we subtract R2*2 from R1, i.e., R1 --> R1 - R2*2
         mat[][] = {{10,    0,  -50},
                    { 0,   10,   30},
                    { 0,  -10,  -30}}
           And subtract R2*(-1) from R3, i.e., R3   --> R3 - R2*(-1)
         mat[][] = {{10,   0,   -50},
                    { 0,   10,   30},
                    { 0,   0,     0}}

row = 2:
Since Since mat[2][2] is 0, we are in case 2.b of above algorithm.
Since there is no row below it swap. We reduce the rank by 1 and
keep row as 2.

The loop doesn't iterate next time because loop termination condition
row <= rank-1 returns false.
```

Below is C++ implementation of above idea.

```
// C++ program to find rank of a matrix
#include <bits/stdc++.h>
using namespace std;
#define R 3
#define C 3

/* function for exchanging two rows of
   a matrix */
```

```
void swap(int mat[R][C], int row1, int row2,
          int col)
{
    for (int i = 0; i < col; i++)
    {
        int temp = mat[row1][i];
        mat[row1][i] = mat[row2][i];
        mat[row2][i] = temp;
    }
}

// Function to display a matrix
void display(int mat[R][C], int row, int col);

/* function for finding rank of matrix */
int rankOfMatrix(int mat[R][C])
{
    int rank = C;

    for (int row = 0; row < rank; row++)
    {
        // Before we visit current row 'row', we make
        // sure that mat[row][0],....mat[row][row-1]
        // are 0.

        // Diagonal element is not zero
        if (mat[row][row])
        {
           for (int col = 0; col < R; col++)
           {
               if (col != row)
               {
                 // This makes all entries of current
                 // column as 0 except entry 'mat[row][row]'
                 double mult = (double)mat[col][row] /
                                       mat[row][row];
                 for (int i = 0; i < rank; i++)
                   mat[col][i] -= mult * mat[row][i];
              }
           }
        }

        // Diagonal element is already zero. Two cases
        // arise:
        // 1) If there is a row below it with non-zero
        //    entry, then swap this row with that row
        //    and process that row
        // 2) If all elements in current column below
        //    mat[r][row] are 0, then remvoe this column
        //    by swapping it with last column and
        //    reducing number of columns by 1.
        else
        {
            bool reduce = true;

            /* Find the non-zero element in current
                column  */
            for (int i = row + 1; i < R;  i++)
            {
                // Swap the row with non-zero element
                // with this row.
                if (mat[i][row])
                {
                    swap(mat, row, i, rank);
                    reduce = false;
                    break ;
                }
            }

            // If we did not find any row with non-zero
            // element in current columnm, then all
            // values in this column are 0.
            if (reduce)
```

```
            if (reduce)
            {
                // Reduce number of columns
                rank--;

                // Copy the last column here
                for (int i = 0; i < R; i ++)
                    mat[i][row] = mat[i][rank];
            }

            // Process this row again
            row--;
        }

       // Uncomment these lines to see intermediate results
       // display(mat, R, C);
       // printf("\n");
    }
    return rank;
}

/* function for displaying the matrix */
void display(int mat[R][C], int row, int col)
{
    for (int i = 0; i < row; i++)
    {
        for (int j = 0; j < col; j++)
            printf("  %d", mat[i][j]);
        printf("\n");
    }
}

// Driver program to test above functions
int main()
{
   int mat[][3] = {{10,   20,   10},
                   {-20,  -30,   10},
                    {30,   50,   0}};
    printf("Rank of the matrix is : %d",
         rankOfMatrix(mat));
    return 0;
}
```

Output:

```
Rank of the matrix is : 2
```

Since above rank calculation method involves floating point arithmetic, it may produce incorrect results if the division goes beyond precision. There are other methods to handle

Reference:
https://en.wikipedia.org/wiki/Rank_%28linear_algebra%29