Problem 1 How many invertible $n \times n$ matrices are there with coefficients in $\mathbb{Z}_p$? When $n = 2$ and $p = 2$ there are 16 matrices in total but only the following 6 of them are invertible:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$$

You may use the fact that a matrix is invertible iff it's determinant is nonzero. Note that the answer to this problem is the cardinality of the set of all automorphisms of $\mathbb{Z}_p^n$, $\mathrm{Aut}\,(\mathbb{Z}_p^n) = \mathrm{GL}(n, p)$.

Problem 2 The file `matrices.txt` on the website contains a list of 100 sets of up to 20 vectors in $\mathbb{Q}^n$ where $n \leq 10$. Each group occupies a single line. The first set looks like this:

```
[[-3,1,2,-5,8,4],[-6,6,7,-7,1,1],[-6,5,-4,7,-9,-9],[8,-6,-1,-6,0,5],[-7,-3,-5,-2,-4,-4]]
```

This set contains 5 linearly independent vectors in $\mathbb{Q}^6$ and the dimension of the vector space spanned by these vectors is therefore 5. The task is to compute the dimension of each vector space spanned by each set of vectors. All vectors have integral single-digit coefficients. The sum of all dimensions is a 3-digit integer. Find this number.

Below is a solution for a special case of this problem in `python`. It uses the functions in `ops.py` to find the dimension. The file `rank.py` contains a solution to this problem for the special case of n vectors in $\mathbb{Q}^2$. `solution.py` will not work on `matrices.txt` because it expects $n \times 2$ matrices.

To compute the dimension of the vectors space spanned by some vectors $v_1, v_2, \ldots v_n$, we first need some notation: Let $V = [v_1, v_2, v_3, \ldots v_n]$ stand for the vector space spanned by $v_1 \ldots v_n$. The three most important observations are

- If we replace v_k by λv_k, V does not change unless $\lambda = 0$.
- If we add λv_j to v_k, V does not change.
- If we swap v_j and v_k V does not change.

As an example, take $v_1, v_2, v_3 = (0, 1, 0), (1, 1, 0), (0, 1, 1)$. We write them above each other as a matrix,

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

Now swap v_1 and v_2. The matrix changes to

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

We have $V = [v_2, v_1, v_3]$. Now add -1 times v_1 to v_3:

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

This changes V to $[v_2, v_1, v_3 - v_1]$. But the three vectors $v_1, v_1, v_3 - v_1$ are obviously linearly independent so the dimension of V is 3.

The three operations **swap two rows**, **multiply row by nonzero scalar** and **add row times scalar to row** are implemented in `python` in `ops.py`. The file `rank.py` contains a function that solves the problem for n vectors of size 2. Your task is to generalize it to find the dimension for any size and number of vectors.

ops.py

```
from fractions import Fraction

def width(A):
    w = len(A[0])
    assert all(len(A[i]) == w for i in range(height(A)))
    return w

# height of matrix A
def height(A):  return len(A)

# swaps two rows in a matrix A.
def swap(rowj, rowk, A):
    assert(0 <= rowj < height(A))
    assert(0 <= rowk < height(A))
    A[rowj], A[rowk] = A[rowk], A[rowj]


# Adds row*factor to row 'to' in A
def addTo(row, factor, to, A):
    assert(0 <= row < height(A))
    assert(0 <= to < height(A))
    for i in range(width(A)):
        A[to][i] += A[row][i]*factor

# multiplies row by nonzero factor
def times(row, factor, A):
    assert(factor != 0)
    assert(0 <= row < height(A))
    for i in range(width(A)):
        A[row][i] *= factor

# prints the matrix nicely:
def print_mtx(A):
    print("\n")
    for row in A:
        print(" ".join(str(x) for x in row))
    print("\n")


# converts matrix to rational numbers:
def to_rational(A):
    for row_idx in range(height(A)):
        for col_idx in range(width(A)):
            A[row_idx][col_idx] = Fraction(A[row_idx][col_idx])
```

rank.py

```
from ops import *

# solves the problem for vectors in Q^2
```

```
# in a _very_ straight-forward way
# (A is nx2)
def solve_n_x_two(A, verbose = True):
    to_rational(A)

    # find something non-zero on the first column:
    idx = -1 # first non-zero index
    for i in range(height(A)):
        if A[i][0] != 0:
            idx = i
            break # stops iteration

    if verbose:
        if idx == -1:
            print("The first column is 0")
        else:
            print("The first nonzero index is {}".format(idx))

    if idx == -1:
        # then the first column is ZERO
        # we check if there is something != 0 in the 2:nd column:
        return 0 if all(A[i][1] == 0 for i in range(height(A))) else 1

    # otherwise A[idx][0] is nonzero. We move row idx to the top:
    swap(0, idx, A)

    if verbose:
        print_mtx(A)

    # now A[0][0] != 0.

    # make all the other rows start with 0:
    for row in range(1, height(A)):
        factor = -(1/A[0][0])*A[row][0]
        addTo(0, factor, row, A)

    if verbose:
        print("first row fixed:")
        print_mtx(A)

    # Now we have something like this:
    #
    #  X *
    #  0 *
    #  0 *
    #  ...
    #  0 *
    #

    # check if there is something != 0 on the 2:nd column starting with 2:nd entry
    return 1 if all(A[i][1] == 0 for i in range(1, height(A))) else 2
```

solution.py

```
from rank import solve_n_x_two

f = open("matrices.txt")
dimensions = sum(solve_n_x_two(eval(line)) for line in f.readlines())
print("The total dimension is {}".format(dimensions))
```