1. *Counting in a sequence*? Write a Java program that calculates the number of special numbers whose squares are in a sequence of integers from ***m*** through ***n*** inclusive. A number's square is an integer that is the square of an integer. In other words, it is the product of some integer with itself. For example, 16 is such a special number, since it can be written as 4 x 4. You may assume that ***m*** and ***n*** are positive integers and that ***n*** is greater than or equal to ***m***.

Input: the first line gives an integer ***k*** which is the number of test cases and is followed by ***k*** lines of inputs, each of which consists of two integers for a test case, ***m*** and ***n*** (separated with a whitespace).

- Example:

```
4
1 4
1 100
10 100
45 45678
```

Output: ***k*** lines of output, each of which is a single integer (for a test case) as the number of special numbers for the input in the test case defined above.

- Example (the output for the above inputs):

```
2
10
1
169
```

Hints:

- For an *int* number, ***n***, the product, ***n*** x ***n***, might overflow.
- You may need to use Java method "**static double Math.sqrt(double a)**"

2. *Sum of a numeric sequence*. Write a Java program that calculates the sum of all the even or odd sequence numbers from ***m*** through ***n***. The selection of even or odd numbers is controlled by a flag, ***f***. If *f* = 1, the odd numbers will be added; if *f* = -1, the even numbers will be added. You may assume that ***n*** is greater than or equal to ***m***. You may also assume that the sum can be represented with Java's *int* variable.

Input: the first line gives an integer ***k*** which is the number of test cases and is followed by ***k*** lines of inputs, each of which consist of three integers for a test case ***f***, ***m*** and ***n*** (separated with whitespaces).

- Example:
  7
  1 1 5
  1 1 999
  1 -3 -1
  -1 1 4
  1 1 1
  1 2 2
  -1 2 4

Output: ***k*** lines of output, each of which is a single integer (for a test case) as the sum for the test case defined above.

- Example (the output for the above inputs):
  9
  250000
  -4
  6
  1
  0
  6

3. *Vigenere Cipher*. Long before the Internet years, encryption algorithms were invented for communication. One is called the Caesar cipher, in which each letter of the alphabet is shifted along some number of places; for example, in a Caesar cipher of shift 3, "A" would become "D", "B" would become "E", "Y" would become "B" (wrap around), and so on.

The Vigenere cipher consists of several Caesar ciphers in sequence with different shift values. The following gives an example on how Vigenere cipher works:

1. Suppose that the *plaintext* to be encrypted is: **ATTACKATDAWN**.
2. Choose a keyword (for example, "*LEMON*") and repeats it until it matches the length of the plaintext, **LEMONLEMONLE**.
3. Each letter in the plaintext is shifted *according to* the corresponding letter in the keyword. Thus,
    a. The first letter "A" is shifted *according to* "L" (*not* encoded to "L");
    b. The second letter "T" is shifted according to "E",
    c. The third letter "T" is shifted according to "M", and so on.
4. The shifting itself works as follows (with the use of Caesar cipher):
    a. If we shift according to, say, "L" (a keyword letter), then: "A" becomes "L", "B" becomes "M", "C" becomes "N", and so on. If we reach the end of the alphabet, then we wrap around to the beginning. So "O" becomes "Z", and "P" becomes "A".
    b. If the keyword letter is "E", then each letter is shifted by 4 (the distance between "A" and "E", i.e., "A" would become "E") so "T" becomes "X" and so on.
5. The plaintext above is thus encrypted to *ciphertext*: **LXFOPVEFRNHR** .
6. (You may use the examples below to help your understanding of the algorithm.)

Now you are to write a Java program to encrypt messages using the Vigenere technique.

Input: the first line gives an integer ***k*** which is the number of test cases and is followed by ***k*** lines of inputs, each of which consists of 2 strings (in upper case) separated with a white-space: the first string is the *keyword* and the second one is the *plaintext*.

- Example:

```
5
AB  ABCDE
BBC  ABCXYZ
AB  BEIFANG
SSU  SALEMSTATE
LEMON  ATTACKATDAWN
```

Output: ***k*** lines of output, each of which is a string (in upper case) for a ciphertext.

- Example (the output for the above inputs):

```
ACCEE
BCEYZB
BFIGAOG
KSFWEMLSNW
LXFOPVEFRNHR
```

Hints: You may use Scanner's *next*() method to read the next string.

4. *Rectangular Map*. You are a commuter student and go to school by buses and trains. All the stations are marked on a rectangular map with their coordinates. This is a problem for you to solve: compute the total number of stations in the smallest rectangle that contains your trip.

You are given the locations (as (*x, y*) coordinates, *x* and *y* are integers.) for all stations in the public transit system on the map and the list of stations (in order) that your line/trip takes you through, in order to solve this problem.

Input: the first line gives an integer ***k*** which is the number of test cases and is followed by ***k*** data sets for the test cases. Each data set is defined in the following format:

a. The first line contains two integers, ***n*** (the total number of stations in the system) and ***m*** (the number of stations you travel through). You may assume that **2 <= *n* <= 500** and **2 <= *m* <= *n***.
b. The next ***n*** lines give ***n*** pairs of integers, x_i and y_i, the coordinates for the i^{th} station (***i*** is a positive number).
c. The following line contains ***m*** integers for the list of stations you visit in order.

- Example:

```
1
8 4
0 0
6 3
0 2
7 2
0 -1
4 4
1 4
3 5
1 7 6 2
```

Output: ***k*** lines of output, each of which is a single integer, the total number of stations visible in the *smallest rectangle* containing *your entire trip*.

- Example (the output for the above inputs):

```
5
```

5. *Searching for new living things or new materials in the universe* (***Part I***). Scientists have been using telescopes in the search for possible new materials and new living things in the universe. The photos taken with a telescope consist of pixels and the "*values*" of the pixels reflect the features/properties of the cosmological bodies or materials on such bodies. In this problem, we use upper-case letters ('A' to 'Z') to represent pixel values (instead of integers for color intensities in a picture). Each picture is a rectangle of ***h*** x ***w*** pixels, with 1<= ***h*** <= **50** and 1<= ***w*** <= **50**. Given a set of *special pixel* values which indicate a clue for new materials, we are to find the maximum number of such special pixels in a certain area on a picture. That certain area is defined as a square with side, ***m***, such that 1 <= ***m*** <= min(***h, w***)) and can be in any place within the picture. Write a Java program to solve this problem.

Input: the first line gives an integer ***k*** which is the number of test cases (i.e., the *number of pictures*) and is followed by ***k*** data sets for the test cases. Each data set is defined in the following format:

1. The first line contains three integers for ***h*** (the image height), ***w*** (the image width), and ***m*** (the square side), which are separated with white spaces.
2. Next comes *a line of a string* of 1-26 letters from 'A'-'Z' for the *special* (indicative) pixel values, with no repetitions (but not necessarily sorted).
3. This line is followed by ***h*** lines, each consisting of ***w*** upper-case letters, each describing one row of the image.

- Example (inputs; the comment texts marked with "//…" are not the input parts, only for the explanation):

```
3        // number of test cases
3 4 2    // h, w, m
AC       // special indicator values
ABCD     // first row
ABDE     // second row
CDEF     // third row
3 4 1
AC
ABCD
ABDE
CDEF
3 4 3
AC
ABCD
ABDE
CDEF
```

Output: ***k*** lines of output, each being a single integer for the solution for a test case.

- Example (the output for the above inputs):

```
2
1
4
```

Hints:

1) You may imagine that the square is just a sliding window and you are sliding that window over (within) the picture. Each time the window covers different areas. You are to calculate the number of special pixels in these areas. Select the maximum as your solution.

6. *Searching for new living things or new materials in the universe* (***Part II***). Scientists have been using telescopes in the search for possible new materials and new living things in the universe. The photos taken with a telescope consist of pixels and the "*values*" of the pixels reflect the features/properties of the cosmological bodies or materials on such bodies. In this problem, we use upper-case letters ('A' to 'Z') to represent pixel values (instead of integers for color intensities in a picture). Each picture is a rectangle of ***h*** x ***w*** pixels, with 1<= ***h*** <= **50** and 1<= ***w*** <= **50**. Given a set of *special pixel* values which indicate a clue for new materials, we are to find the largest density of special letters in any subrectangle of size at least ***m*** x ***m*** (the density is defined as the number of special pixels divided by the area of the subrectangle, i.e., its total number of pixels). You may assume that 1 <= ***m*** <= min(***h, w***).

Input: the first line gives an integer ***k*** which is the number of test cases (i.e., the *number of pictures*) and is followed by ***k*** data sets for the test cases. Each data set is defined in the following format:

1. The first line contains three integers for ***h*** (the image height), ***w*** (the image width), and ***m*** (the square side), which are separated with white spaces.
2. Next comes *a line of a string* of 1-26 letters from 'A'-'Z' for the *special*, indicative, pixels, with no repetitions (but not necessarily sorted).
3. This line is followed by ***h*** lines, each consisting of ***w*** upper-case letters, each describing one row of the image.

- Example (inputs):

```
3
3 4 2
AC
ABCD
ABDE
CDEF
3 4 1
AC
ABCD
ABDE
CDEF
3 4 3
AC
ABCD
ABDE
CDEF
```

Output: ***k*** lines of output, each being the maximum density in any ***m*** x ***m*** or *large* rectangle. This should be output as a fraction ***x / y***, where ***x*** is the number of special pixels, and ***y*** is the number of total pixels in the maximizing rectangle. If there are multiple rectangles of different sizes giving the same ratio, then output the ration with the largest number ***y*** of pixels in the rectangle.

- Example (the output for the above inputs):

```
3/6
3/3
4/9
```