Self Assessment Assignments I

Assignment 1:

Determining the Type of a Triangle

Consider a triangle with three sides measuring a, b, and c units. A triangle is a **right-angled** triangle if
$a^2 + b^2 = c^2$

Allow a tolerance of 0.000001 in the comparison in the above case i.e
$a^2 + b^2 = c^2$ +/- 0.000001

A triangle is an **isosceles** triangle if any two of its sides are equal.
A triangle is an **equilateral** triangle if all the three sides are equal.
Three values can be the dimensions of a triangle if and only if the sum of every pair of values is greater than the third value. Otherwise, it is an **invalid** triangle.

Write a program that reads three real numbers and finds out whether they can be the sides of the triangle and if they do, prints what type of triangle it is. Even though all equilateral triangles are isosceles, your program should classify an equilateral triangle to be an equilateral only. Similarly isosceles right-angled triangles should be classified as right-angled and not isosceles. A valid triangle which does not belong to any of the special types belongs to the **notspecial** category. The three sides will be given as real numbers separated by blanks. The program should print the type of the triangle in words using lower case letters followed by eoln without any blanks. Thus the possible answers are

- invalid
- right-angled
- isosceles
- equilateral
- notspecial

Sample Input and Output:

Input:
```
4.0 5.0 3.0
```
Output:
```
right-angled
```
Input:
```
5.0 7.0 7.0
```
Output:
```
isosceles
```
Input:
```
3.0 3.0 8.0
```
Output:
invalid

Assignment: 2

Word Count

You have to write a program to count the number of words in the input. The program should read in the input text till end_of_file (**EOF**) and output the number of words found. A word can be taken to be a sequence of alphanumeric characters terminated by a space or by a newline. Assume that there will not be any characters other than alphanumeric (a-z, A-Z, 0-9) and white spaces (blank, tabs and newlines) in the input.

Sample Input and Output:

Input:

```
This is a sample line of text
  This is another line of text
    This line is the 3rd line
  This junk line contains 989902 99dsaWjJ8    015
This is the fifth and the last line of input
```

Output:

```
36
```

Input:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
```

Output:

```
1
```

Assignment: 3

Format a Positive Real Number

Consider a positive real number given in the form F.I (the float form). Treating F as the fractional part and I as the integral part, print the number in the form GH.J where H is a sequence of # (hashes) such that the decimal point always occurs at the sixth (6^{th}) position from the left and G is I with only the significant digits, and J is F with only the significant digits. Terminate your output with a newline.

Assume that the fractional part F always has no leading zeroes and both F and I have at most five (5) significant digits.

Sample Input and Output:

Input:
10000.6660
Output:
6660#.1

Input:
990.0080
Output:
80###.99

Input:
101.11111
Output:
11111.101

Input:
31644.0
Output:
#####.31644

Assignment: 4

Ackermann Function

The Ackermann function is the simplest example of a well-defined total function which is computable but not primitive recursive. The Ackermann function is one of the fastest growing functions, much faster than polynomials or exponentials. Its definition is:

1. If x = 0 then A(x, y) = y + 1
2. If y = 0 then A(x, y) = A(x-1, 1)
3. Otherwise, A(x, y) = A(x-1, A(x, y-1))

You have to write a program to read two integers x and y, and print the value of A(x, y).

Warning: Do not try values larger than x=5 and y=5.

Input specification:

Two integers x and y.

Output specification:

One integer corrseponding to A(x, y)

Assignment: 5

Game of Life

The game of life is a game which is played on a field of cells. Each cell has eight neighbours, i.e. adjacent cells. Each cell in the field is either occupied by an organism or is blank. Thus, given that a cell is occupied by an organism, this organism could have eight neighbours or less. The objective of the game is to simulate the life of the organisms present in the field, generation after generation. The only work of the organism is to either reproduce or die! The rules of reproduction are given as follows:

1. If an organism has 0 or 1 neighbours, it will die out of loneliness in the next generation
2. If an organism has 4 or more neighbours, it will die of crowding in the next generation
3. If an organism has 2 or 3 neighbours, it will prosper and survive into the next generation
4. If an unoccupied cell has exactly three neighbours, then an organism takes birth in the next generation

You are required to simulate the life of the organisms and output the state of the field after 'N' generations.

Input specification:

- The first line of input will be two integers, R and C, separated by a space, specifying the number of rows and columns that the field has. Assume that the maximum number of rows and columns will never be greater than 30.
- The next R lines of input contain C characters, consisting of '#' and '@' respectively. '#' means a blank cell, while '@' indicates an organism.
- The last line of input contains one integer N, which is the number of generations that are supposed to be simulated.

Output specification:

The output of your program should be a single integer indicating the number of living organisms at the end of simulation, i.e., after N generations.

Sample Input and Output:

Input:
```
3 3
#@#
#@@
@#@
3
```
Output:
```
3
```

Input:
```
3 3
#@#
#@@
@#@
4
```
Output:
```
4
```

Assignment: 6

Decimal/Roman Conversion

Roman numerals consist of the letters I, V, X, L, C, D, M which represent the values 1, 5, 10, 50, 100, 500, 1000. These letters are combined in an order-value system. Basically the rule is, if a letter is followed by an equal or lower valued one it counts as positive. If it is followed by a higher valued one it counts negative, i.e., in the number IX, I is followed by X, which is higher valued, hence I serves as a negative influence. This means that the value of I is subtracted from the value of X to yield NINE (9).

There are some refinements to the basic rule:

1. There are not more than three equal letters in a number.
2. Only I,X,C are used as negative values. For example it is not MLD for 1450 but MCDL.
3. The negating influence is only applied up to the next power of ten. I is only subtracted from V and X but not from L and C. Likewise X is only subtracted from L and C. So 9 is IX but 99 is XCIX not IC.

Given these rules you have to write a program which given a Roman number converts it into its equivalent Decimal number and vice-versa.

Note

1. The number 0 will not be present in the input or output
2. The subtraction rule should be applied only to the letter following immediately and should not be propagated i.e. numbers like IXC will not be valid

Input specification:

- First line has an integer N which specifies how many numbers follow
- There are N subsequent lines, each of which has either a valid decimal or roman number

Output specification:

N lines are output, each containing the corresponding converted number.

Sample Input and Output:

Input:
1
999
Output:
CMXCIX

Input:
3
2000
MCDL
MCM
Output:
MM
1450
1900

Assignment: 7

Removal of 'Zero' Rows and Columns in a Matrix

Given an integer matrix of size, say, M x N, you have to write a program to remove all the rows and columns consisting of all zeros. Your program must remove those rows and columns that consist of zero valued elements only. That is, if all the elements in a row (column) are zeros, then remove that row (column). All the remaining rows and columns should be output.

Input Specifiaction:

- The first line of input will have two integers, say, M and N. M specifies the number of rows in the matrix and N specifies the number of columns in the matrix.
- This will be followed by M lines, each consisting of N integers.

Note: Assume 1 <= M <= 10 and 1 <= N <= 10. Also, you may assume that there is at least one nonzero element in the given matrix.

Output Specification:

The output should be the resulting matrix. Each row of the output matrix should be terminated by a newline character. There should be exactly one space (not tab) between successive elements in each row.

Sample Input and Output:

```
Input:
4 4
1 2 3 4
0 0 0 0
5 6 7 8
0 0 0 3
Output:
1 2 3 4
5 6 7 8
0 0 0 3

Input:
4 5
1 2 0 3 5
0 0 0 0 0
4 5 0 6 9
7 8 0 9 1
Output:
1 2 3 5
4 5 6 9
7 8 9 1
```

Assignment: 8

N^{th} String Permutation

A permutation is a shuffling of the order of a sequence of elements. You are given a sequence of the first L alphabets starting from the letter 'a'. You have to write a program which generates the permutation of these alphabets in a lexicographic order, i.e. similar to the ordering in a dictionary.

For example, given that L=3, the resulting sequence being {a,b,c}, the lexicographic permutation produced would be:

1. abc
2. acb
3. bac
4. bca
5. cab
6. cba

The 4^{th} permutation from this sequence would then be: "bca".

Input specification:

The input consists of 2 integers. The first integer L indicates that the first L alphabets starting with the letter 'a' have to be considered. The second integer N specifies the index corresponding to which the permutation should be output. Assume that the permutations are indexed starting from 1.

Output specification:

The output contains the N^{th} permutation of the set.

Sample Input and Output:

Input:
3 4
Output:
bca

Input:
4 8
Output:
badc

Assignment: 9

Polynomial Addition

Write a program to perform polynomial addition. The program should read and print the polynomials in the format shown below.

Input Specification:

A polynomial will be represented by pairs of integers denoting its coefficients and powers. There will not be more than 15 terms in each polynomial. There is no limit on the power of each term. The polynomial will be terminated by an integer pair -1 -1.

Output Specification:

The output polynomial should be in the descending order of the power.

Sample Input and Output:

```
Input:
-2 2 1 3 3 0 -1 -1                    { - 2x^2 + x^3 + 3 }
0 0 1 4 -2 3 2 2 -1 -1          { x^4 - 2x^3 + 2x^2 }
Output:
1 4 -1 3 3 0                    { x^4 - x^3 + 3 }

Input:
2 -2 1 0 1789 9783 -1 -1              { 2x^-2 + 1 + 1789x^9783 }
-1 0 -1 -1                            { -1 }
Output:
1789 9783 2 -2                  { 1789x^9783 + 2x^-2 }
```

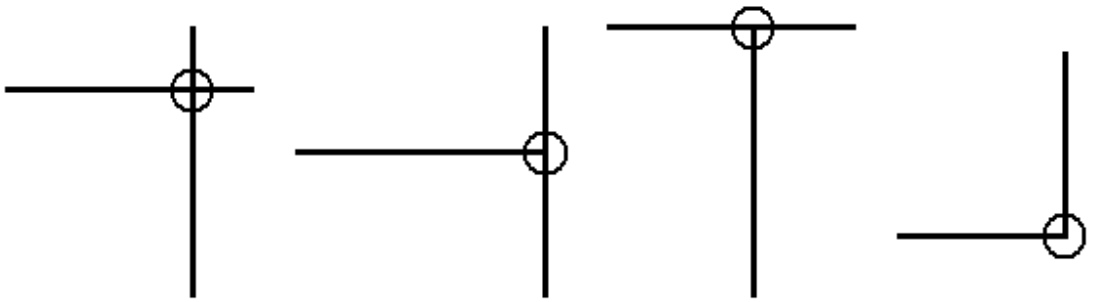
Assignment: 10

Horizontal/Vertical Line Crossings

Archeologists have discovered a very ancient cave, with many pictures on its wall. Each of the picture contains only vertical and horizontal line segments. Experts have analysed these pictures and came to the conclusion that, these are hieroglyphs (like Egyptian writings). To extract the meaning from these glyphs, it is necessary to find out how many of these vertical and horizontal line segments intersect.

Each picture glyph is inscribed in a square of side 100, and each end point of the line segment is assigned an X-Y coordinate pair. All the X and Y values are integers in the range 0 to 100. The vertical line segment is described by an X value and two Y values (X, Y1, Y2). Similarly, a horizontal line segment is described by a Y value and a pair of X values (Y, X1, X2). For example, a horizontal line segment described by a triplet (12, 8, 57) is a segment joining points (8, 12) and (57, 12).

You have to write a program to compute the total number of intersections, given the details of vertical and horizontal line segments. Only the intersections between a pair of horizontal and vertical line segments are counted. Even if one segment is touching another, it is counted as an intersection. (See figure below)



Input specification:

- The first line has two integers, V and H. V denotes the number of vertical line segments and H denotes the number of horizontal line segments in the picture.
- The next V lines describe V vertical line segments each as a triplet of integers (X, Y1, Y2).
- The next H lines describe H horizontal line segments each as a triplet of integers (Y, X1, X2).

Output specification:

The total number of intersections in the given picture.

Sample Input and Output:

Input:
1 1
50 20 60
30 40 80
Output:
1

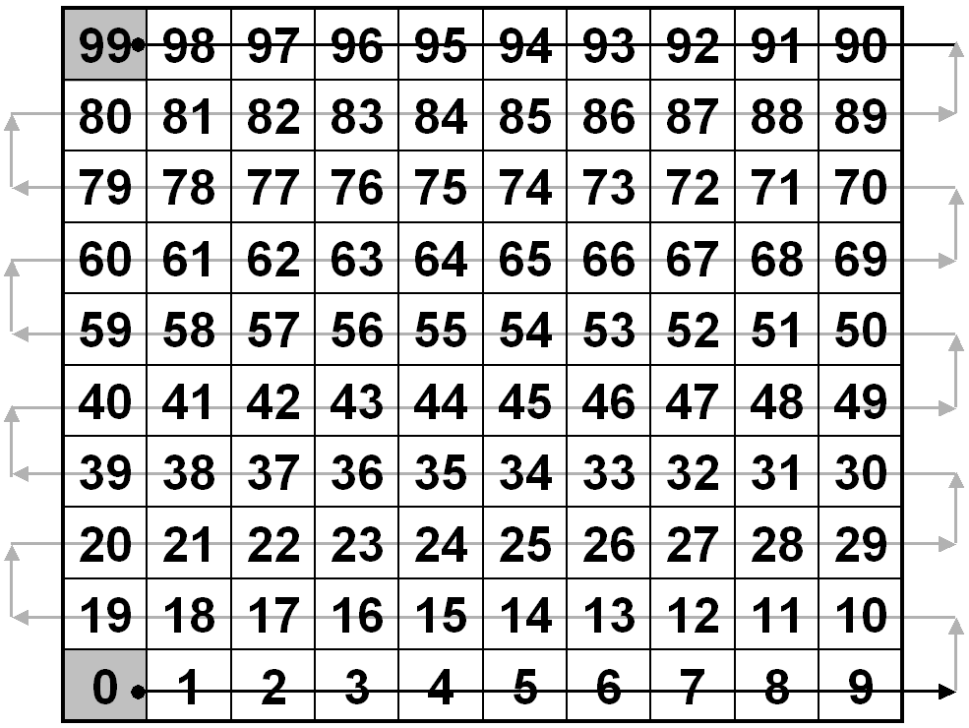
Input:
2 2
25 25 80
75 25 80
25 25 80
75 25 80
Output:
4

Input:
1 2
50 0 100
50 0 75
50 25 100
Output:
2

Assignment: 11

Snake and Ladder

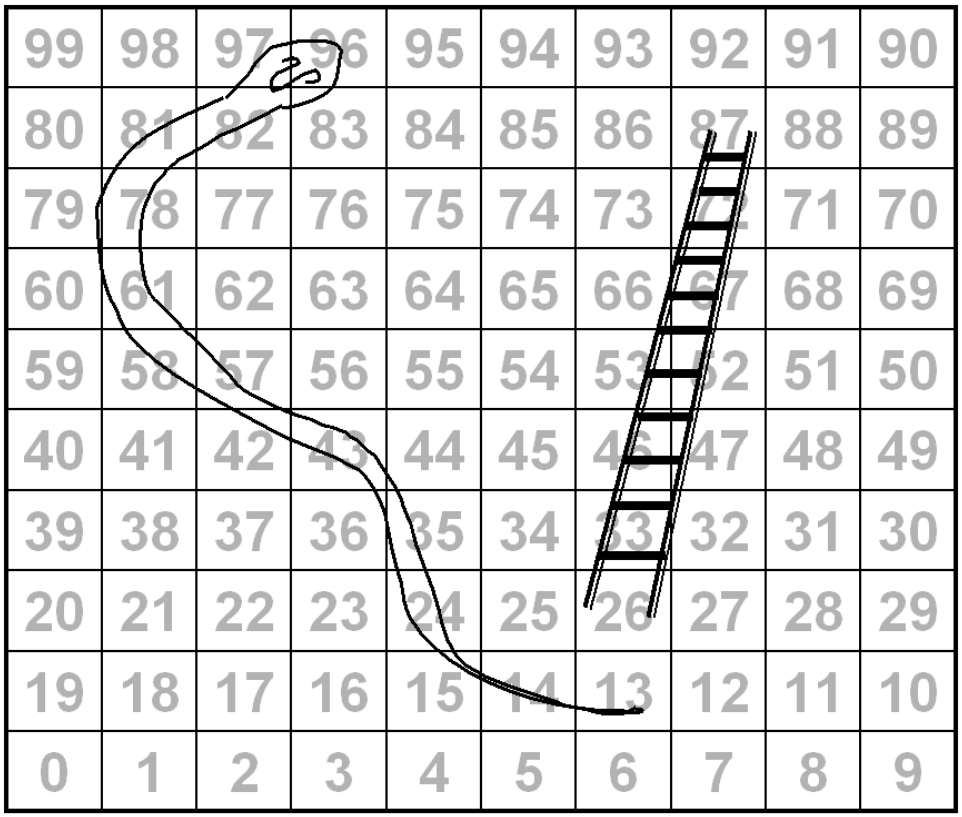
Snakes and ladders is a very popular game. It's a 10x10 maze, with cells numbered from 0 to 99, and filled with snakes and ladders, as shown below in the figures. Every player gets a chance to play and advance in the maze, depending on the number they get on the dice when thrown.



The advancement is in the increasing order from 0 to 99. All players start from cell numbered 0 (zero) and attempt to reach to cell numbered 99. The player who reaches first to cell number 99, is the winner. If the game is stopped in between, the player closest to cell 99 in number is the winner. Assume that both players will never be on the same cell when the game is stopped.

If a player after tossing the dice lands on a cell, where the snake opens it's mouth, then the player gets a penalty to retreat to the tail of the snake. While, if the player reaches the foot of a ladder, then the player climbs directly to the top of the ladder. In the example figure here, the snake has its mouth on cell 96 and its tail on cell 13, while the ladder has foot on cell 26 while its top is on cell 87.

If there is another snake (or ladder) waiting at the tail of a snake, a player slide down (or climb up) that second snake (or ladder), and so on as required. A cell will never have a mouth of a snake and a foot of a ladder simultaneously.



As such, the snake and ladder are similar, in that when a player lands on a particular cell, they transport the player to another cell. Except for the fact that, snake take a player away from finishing line, while ladder takes a player closer.

Write a program to simulate the game. There will be two players, named A and B, who toss the dice alternately. The program should output the name of the player at the highest position (closest to 99), along with the player's cell number.

Input specification:

- The first line of input will contain a single integer N, stating the number of snakes and ladders.
- Next N lines each will contain a pair of integers. Each pair will describe either a snake or ladder by its start cell and an end cell. For example, for a snake, start cell will be higher than end cell. Refer to the first sample input case which describes the board in the figure above.
- Following line of input has a single integer D, indicating the number of moves (dice throws) made by both players A and B.
- Last line contains D integers, each indicating a dice throw, alternately made by A and B. First throw is made by player A. Each dice throw is an integer between 1 to 6.

Output specification:

Your program should output the name of the winner (A or B) followed by the players position at the end of game, with a single space in between.

Sample Input and Output:

Input:
```
2
26 87
96 13
18
4 1 3 2 6 3 6 3 2 4 4 2 1 1 5 2 4 6
```
Output:
```
B 24
```

Input:
```
3
13 98
98 0
6 13
10
5 1 3 3 1 1 4 1 6 1
```
Output:
```
B 1
```

Assignment: 12

Escape from Maze

NCST has decided to create a maze, next to it's library, so that only intelligent people can go into the library. As is typical of mazes, this one too has several dead ends and only one correct start and end point. You have to write a program to navigate this maze and enter the library. Your starting point in the maze is marked by the character '#' and the entrance to the library (i.e., end point of the maze) is marked by the character '@', i.e. your program must start from '#' and end at '@'.

The maze is composed of a rectangular array of cells. Apart from the start (#) and end (@) cells, maze contains solid cells and empty cells. A valid path never passes via a solid cell. The solid cells are marked by 0s and empty cells are marked by 1s. Once on a cell, you can only move to any one of the neighbouring four (i.e., north, east, west, south) cells.

Note: Assume that no path will ever cross itself, i.e., there will be no loops in any path.

Input specification:

- The first line of input will be two integers, R and C, specifying the number of rows and columns.
- The next R lines contain C characters each separated by white space, thereby specifying the layout of the maze. It is guaranteed that the maze will contain only one starting and one ending character. Also, it is guaranteed that the maze will contain only one correct path, from the start point to the end point.

Output specification:

Your program must output on one line, the number of hops required to reach the end point.

Sample Input and Output:

Input:
```
5 5
0 0 0 0 0
0 0 0 0 0
0 # @ 0 0
0 0 0 0 0
0 0 0 0 0
```
Output:
```
1
```

Input:
```
6 6
0 0 0 0 @ 0
0 1 0 1 1 0
1 1 1 1 0 1
0 1 0 1 0 0
0 0 # 1 1 1
0 1 1 0 0 1
```
Output:
```
6
```

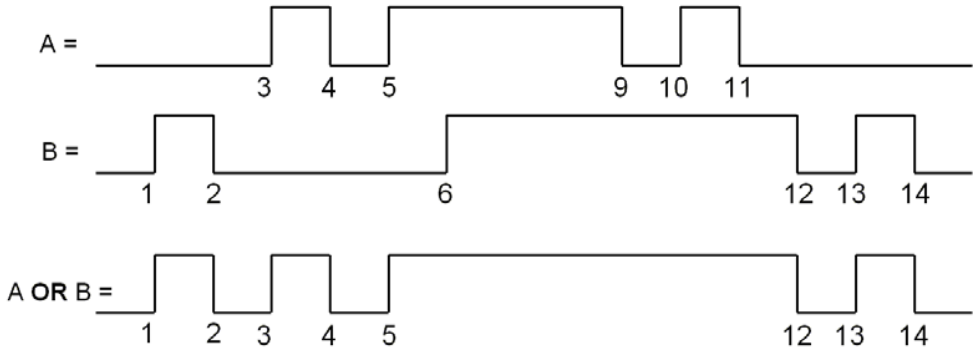
Assignment: 13

Boolean OR of Two Digital Signals

Write a program to generate a boolean OR of two digital signals. A signal has two states, a high state and a low state. It is given as an array of positive integers in ascending order. Each pair of integer indicates the range of the signal in a high state. The array size will be even, so that the signal starts with a low state and ends with a low state.

Consider two signals named A and B described as follows:
A = 3 4 5 9 10 11
and
B = 1 2 6 12 13 14

Their boolean OR will be a signal **A OR B** = 1 2 3 4 5 12 13 14
This is illustrated in the figure below.



Input Specification:

The input will describe two signals, one on each line. Every line will begin with an integer H, which specifies the number of high states of the signal. This is followed by 2*H positive integers describing the signal, as shown above. Assume that state change will never occur at same time (the integers can be thought of as a timeline) in the two given signals.

Output Specification:

The output signal should be the boolean OR of the two input signals, described in exactly the same manner as the input.

Sample Input and Output:

Input:
3 3 4 5 9 10 11
3 1 2 6 12 13 14
Output:
4 1 2 3 4 5 12 13 14

Input:
1 1 1000
1 999 10000
Output:
1 1 10000

Input:
5 2 5 6 10 20 23 38 45 51 52
5 1 4 7 9 14 19 21 24 30 31
Output:
7 1 5 6 10 14 19 20 24 30 31 38 45
51 52

Assignment: 14

Wordy Enough

A student, doing his apprenticeship as a linguist, is given a rather boring problem. His mentor has given him a list of words and a long string of characters. All the characters in the list of words and the long string of characters are in capital. The student has to look at the characters and decide how many words from the given list can be formed using the long string of characters. However, there are some rules of counting:

- Each character in the word must be accounted for in the character stream, i.e. if a character appears twice in a word, then it must also appear twice in the stream. For example, the word 'NEED' can only be formed if 'E' occurs twice in the long character string.
- Once a character is used up for a word, it cannot be used again for forming other words.

The student has to form the maximum number of words possible. You have to write a program to do this and help him to concentrate on other more interesting problems.

Input specification:

- The first line of input contains an integer 'w', which denotes the number of words in the list.
- The second line of input consists of 'w' words, each separated by a white space. The length of each word will never exceed 20 characters.
- The third line of input contains an integer 's' which denotes the number of characters in the long string of characters.
- The fourth line of input, which is a long character string, consists of 's' characters.

Output specification:

Your program has to output an integer 'm', denoting the maximum number of words possible.

Sample Input and Output:

Input:

```
9
SATAN SEES OASIS STRAIGHT FORWARD SENSE FROG DREW SHIRTS
20
AAEEIODFGHNRRRSSSTTW
```

Output:

```
4
```

Hint: This problem may be solved using recursion.

Assignment: 15

Paragraph Formatting

Given a stream of text, your task is to create a formatted paragraph out of it. The paragraph should be left justified and each line of text in the formatted paragraph should not exceed a given length. Each line should contain maximum possible characters.

In addition to the input text, a set of words with their possible hyphen positions will also be given. The hyphenation makes it possible sometimes to add incomplete words at the end of line, if the completed words overflow the line length limit. To indicate that this particular word is incomplete, a hyphen (-) is added at the end of such broken words. Remaining part of this word is placed at the beginning of the next line. The specified line length limit must be observed including the trailing hyphen character if any.

The input text will contain alphabets, digits, punctuation marks and spaces. The only white space character used is a blank space, which is used to separate words. Each line in a formatted paragraph begins with a non-blank character. Even if the input text has words separated by many spaces, your output should separate them by only single space or a line break.

Notes:

- The list of hyphenated words will not include any punctuation marks that may occur in the text to be formatted. You may assume that the punctuation marks will occur only at the end of each word (alpha-numeric characters). That is there will not be words like **They'll** or **Sita's** anywhere in the input.
- The word length will never exceed the line length limit.
- The input text can always be formatted using the given constraints.
- No word, hyphenated or otherwise will exceed a length of 20 characters.

Input specification:

- First line has an integer n denoting the number of hyphenated words.
- Next n lines give each of the hyphenated word with possible hyphen break positions in the word by a hyphen itself. For example, "hyphenate" is given as "hy-phen-ate" and "formatting" is given as "for-mat-ting".
- Next line will be an integer denoting the maximum number of characters per line in the formatted paragraph, i.e., the line length limit.
- Last line contains the text to be formatted into a paragraph. The entire text will be terminated by a new line. The length of text will not exceed 256 characters.

Output specification:

The output should be a sequence of characters, which begin each line of the formatted paragraph followed by a single new line character. Thus, there will be those many non-white space characters as there are lines in the formatted paragraph.

Sample Input and Output:

Input:

```
4
con-cept
pro-gram-ming
ob-vi-ous
im-pos-si-ble
25
Most people find the concept of programming obvious, but the doing impossible.
```

Output:

```
Mcos
```

Assignment: 16

Pattern Matching

You are required to write a program to do simple pattern matching, in a string. The string in which the pattern is to be found, henceforth referred to as **Master String**, can be composed of the following character sets:

- a-z {any character in the range a to z}
- A-Z {any character in the range A to Z}
- 0-9 {any digit in the range 0 to 9}

The pattern to be matched is also a string, henceforth referred to as **Scan String**. The **Scan String** can be composed of the above character sets, as well as two special characters, which are: '.' and '*'. The dot ('.') is to be interpreted as matching any one character from the above character sets and the star ('*') is interpreted as matching zero or more of the previously matched character. The **Scan String** may or may not contain the special characters.

A pattern is considered matched when the longest substring that satisfies the pattern is returned.

Input Specification

- The first line of input contains the **Master String**.
- The second line of input contains the **Scan String**.

Both **Master String** and **Scan String** will not be greater than 80 characters in length.

Output Specification

Your program must output the length of the longest substring matched. If there is no match, then the length of the matched substring is zero.

Sample Input and Output:

Input:
abcd23Abdaaaa4g9
.*Abd. { *dot-star*-A-b-d-*dot* }
Output:
10

Input:
aaadaaabbbb129cd
a.*d { a-*dot-star*-d }
Output:
16

Input:
0AbC1dEf2GhI3jKl4MnO5pQr6StU7vWx8Yz9
3JkL4 { 3-J-k-L-4 }
Output:
0

Input:
090m90mm90mmm90mm90m909
0m*9
Output:
5