DKC³ 2011 SHORT COMPUTING PROBLEMS

1. Budget Movie   (35 pts)

As director of the upcoming blockbuster movie Deep Code 9, your employer, 21st Century Fox, has assigned you the task of selecting the actors and actresses. The problem is that in the movie about 10 male and 10 female characters occur, and because of the tiny budget that you have been given, you simply cannot afford that many actors and actresses! Looking closely at the movie script, however, you wonder if some of the characters could be played by the same person. For example, if Mr. Programmer and Mr. Hero never appear together at the same time during the movie, one actor can play both roles! You tell your boss your idea, and he agrees, as long as you adhere to the following rules:

1. Male characters can only be played by male actors, and female characters can only be played by female actresses.
2. Each character must be played by one actor/actress throughout the whole movie. Changing the person who plays a given character during the movie is not allowed.
3. When two characters ever appear together at the same time during the movie, they must be played by different actors/ actresses.

Given these restrictions, your job is to determine the minimum number of actors and actresses needed to produce the movie.

Input
The input consists of several movie descriptions. Each description starts with one line that contains three integers M, F, S that specify the number of male (1<=M<=10) and female (1<=F<=10) characters occurring in the movie and the number of scenes that the movie has (1<=S<=100). On the second and third line, the names of the male and female characters are given. Then S lines follow, each line describing one scene. Each of these lines contains the number of people who occur in the scene and the list of their names. Input ends with an asterisk.

Output
For each movie description, output two lines:

1. A line saying: "Movie #*n*" where *n* is the number of the movie
2. A line saying: "You need *x* actors and *y* actresses." where *x* and *y* is the solution. Use singular (actor, actress), if *x* = 1 or *y* = 1.

Input File: C:\DKC3\MovieIn.txt
Output File: C:\DKC3\MovieOut.txt

Examples:

Input:

1 1 1
Donald
Daisy
2 Donald Daisy

4 3 6
Tarzan Jim John Tom
Lucy Cynthia Jane
3 Jim John Tom
2 Tarzan Lucy
2 Jane Cynthia
2 Jim Jane
2 Tarzan Jim
2 Tarzan Jane
*

Output:

Movie #1
You need 1 actor and 1 actress.
Movie #2
You need 3 actors and 2 actresses.

2. Binary Matrix   (20 pts)

Assume that you work for the Digital Processing Lab. They ask you to write a program that takes an input binary matrix A, which contains the pattern to search for on another binary matrix, matrix B. The input file includes the size and elements for both A and B. The recognition process consists in scanning row by row (horizontal scanning) matrix B. When a pattern is located on B you must mark this pattern. To mark a located pattern, change 1 to 2 and 0 to * on matrix B. The output file of your program will be the matrix B with the located patterns marked.

Input
The first line of the input contains the size of matrix A (columns by rows), the next lines contain the matrix A row by row, the next line contains the size of B (columns by rows) and next lines contain the matrix B row by row.

Each input test case is separated by an asterisk.

Output
The output is the matrix B with the located patterns marked. Separate each matrix with a blank line.

Input File: C:\DKC3\MatrixIn.txt
Output File: C:\DKC3\MatrixOut.txt

Examples:

Input: Output:

```
2 2                                       1 2 * 0 0
1 0                                       0 2 2 0 0
1 1                                       2 * 0 1 0
5 5                                       2 2 1 2 *
1 1 0 0 0                                 0 0 1 2 2
0 1 1 0 0
1 0 0 1 0                                 2 2 0 0 0
1 1 1 1 0                                 0 2 2 0 0
0 0 1 1 1                                 2 0 0 2 0
*                                         2 2 2 2 0
1 1                                       0 0 2 2 2
1
5 5
1 1 0 0 0
0 1 1 0 0
1 0 0 1 0
1 1 1 1 0
0 0 1 1 1
```

3. Porter's Algorithm   (25 pts)

Removing suffixes by automatic means is an operation which is especially useful in the field of information retrieval. In a typical IR environment, one has a collection of documents, each described by the words in the document title and possibly by words in the document abstract. Ignoring the issue of precisely where the words originate, we can say that a document is represented by a vector of words, or *terms.* Terms with a common stem will usually have similar meanings, for example:

  CONNECT
  CONNECTED
  CONNECTING
  CONNECTION
  CONNECTIONS

Frequently, the performance of an IR system will be improved if term groups such as this are conflated into a single term. This may be done by removal of the various suffixes -ED, -ING, -ION, - IONS to leave the single term CONNECT. In addition, the suffix stripping process will reduce the total number of terms in the IR system, and hence reduce the size and complexity of the data in the system, which is always advantageous.

To present the suffix stripping algorithm in its entirety we will need a few definitions.

A *consonant* in a word is a letter other than A, E, I, O or U, and other than Y preceded by a consonant. (The fact that the term `consonant' is defined to some extent in terms of itself does not make it ambiguous.) So in TOY the consonants are T and Y, and in SYZYGY they are S, Z and G. If a letter is not a consonant it is a *vowel*.

A consonant will be denoted by c, a vowel by v. A list ccc... of length greater than 0 will be denoted by C, and a list vvv... of length greater than 0 will be denoted by V. Any word, or part of a word, therefore has one of the four forms:
 CVCV ... C
 CVCV ... V
 VCVC ... C
 VCVC ... V
These may all be represented by the single form
 [C]VCVC ... [V]
where the square brackets denote arbitrary presence of their contents. Using $(VC)^m$ to denote VC repeated m times, this may again be written as
 $[C](VC)^m[V]$.
m will be called the measure of any word or word part when represented in this form. The case m = 0 covers the null word. Here are some examples:
 m=0 TR, EE, TREE, Y, BY.
 m=1 TROUBLE, OATS, TREES, IVY.
 m=2 TROUBLES, PRIVATE, OATEN, ORRERY.
The rules for removing a suffix will be given in the form
 (condition) S1 -> S2
This means that if a word ends with the suffix S1, and the stem before S1 satisfies the given condition, S1 is replaced by S2. The condition is usually given in terms of m, e.g.

(m > 1) EMENT ->

Here S1 is `EMENT' and S2 is null. This would map REPLACEMENT to REPLAC, since REPLAC is a word part for which m = 2.

The `condition' part may also contain the following:

      *S - the stem ends with S (and similarly for the other letters).

      v - the stem contains a vowel.

      *d - the stem ends with a double consonant (e.g. -TT, -SS).

      *o - the stem ends cvc, where the second c is not W, X or Y (e.g. -WIL, -HOP).

And the condition part may also contain expressions with *and*, *or* and *not*, so that

   (m>1 and (*S or *T))

tests for a stem with m>1 ending in S or T, while

   (*d and not (*L or *S or *Z))

tests for a stem ending with a double consonant other than L, S or Z. Elaborate conditions like this are required only rarely.

In a set of rules written beneath each other, only one is obeyed, and this will be the one with the longest matching S1 for the given word. For example, with

   SSES -> SS

   IES -> I

   SS   -> SS

   S   ->

(here the conditions are all null) CARESSES maps to CARESS since SSES is the longest match for S1. Equally CARESS maps to CARESS (S1=`SS') and CARES to CARE (S1=`S').

In the rules below, examples of their application, successful or otherwise, are given on the right in lower case. The algorithm now follows:

Step 1a

```
SSES -> SS                        caresses  ->  caress
IES   -> I                        ponies    ->  poni
                                  ties      ->  ti
SS    -> SS                       caress    ->  caress
S     ->                          cats      ->  cat
```

Step 1b

```
(m>0)  EED -> EE                  feed       ->  feed
                                  agreed     ->  agree
(*v*)  ED  ->                     plastered  ->  plaster
                                  bled       ->  bled
(*v*)  ING ->                     motoring   ->  motor
                                  sing       ->  sing
```

If the second or third of the rules in Step 1b is successful, the following is done:

```
AT -> ATE                         conflat(ed)   ->   conflate
BL -> BLE                         troubl(ed)    ->   trouble
IZ -> IZE                         siz(ed)       ->   size
(*d and not (*L or *S or *Z)) -> single letter
                                  hopp(ing)     ->   hop
                                  tann(ed)      ->   tan
```

```
                                    fall(ing)      ->  fall
                                    hiss(ing)      ->  hiss
                                    fizz(ed)       ->  fizz
(m=1 and *o) -> E                   fail(ing)      ->  fail
                                    fil(ing)       ->  file
```

Step 1c

```
(*v*) Y -> I                        happy          ->  happi
                                    sky            ->  sky
```

Step 2

```
(m>0) ATIONAL ->  ATE       relational      ->  relate
(m>0) TIONAL  ->  TION      conditional     ->  condition
                            rational        ->  rational
(m>0) ENCI    ->  ENCE      valenci         ->  valence
(m>0) ANCI    ->  ANCE      hesitanci       ->  hesitance
(m>0) IZER    ->  IZE       digitizer       ->  digitize
(m>0) ABLI    ->  ABLE      conformabli     ->  conformable
(m>0) ALLI    ->  AL        radicalli       ->  radical
(m>0) ENTLI   ->  ENT       differentli     ->  different
(m>0) ELI     ->  E         vileli          - >  vile
(m>0) OUSLI   ->  OUS       analogousli     ->  analogous
(m>0) IZATION ->  IZE       vietnamization  ->  vietnamize
(m>0) ATION   ->  ATE       predication     ->  predicate
(m>0) ATOR    ->  ATE       operator        ->  operate
(m>0) ALISM   ->  AL        feudalism       ->  feudal
(m>0) IVENESS ->  IVE       decisiveness    ->  decisive
(m>0) FULNESS ->  FUL       hopefulness     ->  hopeful
(m>0) OUSNESS ->  OUS       callousness     ->  callous
(m>0) ALITI   ->  AL        formaliti       ->  formal
(m>0) IVITI   ->  IVE       sensitiviti     ->  sensitive
(m>0) BILITI  ->  BLE       sensibiliti     ->  sensible
```

Step 3

```
(m>0) ICATE ->  IC          triplicate      ->  triplic
(m>0) ATIVE ->              formative       ->  form
(m>0) ALIZE ->  AL          formalize       ->  formal
(m>0) ICITI ->  IC          electriciti     ->  electric
(m>0) ICAL  ->  IC          electrical      ->  electric
(m>0) FUL   ->              hopeful         ->  hope
(m>0) NESS  ->              goodness        ->  good
```

Step 4

```
(m>1) AL    ->              revival         ->  reviv
(m>1) ANCE  ->              allowance       ->  allow
(m>1) ENCE  ->              inference       ->  infer
(m>1) ER    ->              airliner        ->  airlin
(m>1) IC    ->              gyroscopic      ->  gyroscop
(m>1) ABLE  ->              adjustable      ->  adjust
```

```
(m>1) IBLE  ->                    defensible   ->  defens
(m>1) ANT   ->                    irritant     ->  irrit
(m>1) EMENT ->                    replacement  ->  replac
(m>1) MENT  ->                    adjustment   ->  adjust
(m>1) ENT   ->                    dependent    ->  depend
(m>1 and (*S or *T)) ION ->       adoption     ->  adopt
(m>1) OU    ->                    homologou    ->  homolog
(m>1) ISM   ->                    communism    ->  commun
(m>1) ATE   ->                    activate     ->  activ
(m>1) ITI   ->                    angulariti   ->  angular
(m>1) OUS   ->                    homologous   ->  homolog
(m>1) IVE   ->                    effective    ->  effect
(m>1) IZE   ->                    bowdlerize   ->  bowdler
```

Step 5a

```
(m>1) E     ->                    probate      ->  probat
                                  rate         ->  rate
(m=1 and not *o) E ->             cease        ->  ceas
```

Step 5b

```
(m > 1 and *d and *L) -> single letter
                                  controll     ->  control
                                  roll         ->  roll
```

Input File: C:\DKC3\PortersIn.txt
Output File: C:\DKC3\PortersOut.txt

Examples:

Input:

irreplaceable
Activation

Output:

irreplac
Activ

4. Fractional Sequences   (5 pts)

For any positive integer k, a finite sequence a_i of fractions x_i/y_i is defined by:
$a_1 = 1/k$ and $a_i = (x_{i-1}+1)/(y_i-1-1)$ reduced to lowest terms for $i > 1$.

When a_i reaches some integer n, the sequence stops. (That is, when $y_i = 1$.)

Define f(k) = n.
For example, for k = 20:

$1/20 \rightarrow 2/19 \rightarrow 3/18 = 1/6 \rightarrow 2/5 \rightarrow 3/4 \rightarrow 4/3 \rightarrow 5/2 \rightarrow 6/1 = 6$

So f(20) = 6.

Write a program to read 10 test cases of input (k) and print the output.

Input File: C:\DKC3\FractionIn.txt
Output File: C:\DKC3\FractionOut.txt

Examples:

Input:

3
20

Output:

1
6

5. Round Table   (20 pts)

There are countA representatives of a company A and countB representatives of a company B that are having a meeting at a round table. There are chairs around the table and they want to sit down in such a way that no representative of company A is sitting closer than minDistance to a representative of company B (minDistance=1 means any seating is allowed, minDistance=2 means there must be at least one empty chair between representatives of different companies, etc.).

Return the number of ways that such a seating arrangement is possible. Two seating arrangements are different if at least one person is sitting in different chairs in the two arrangements (i.e., include in the count all possible rotations of each seating arrangement). Note also that two different representatives of the same company are to be treated as different persons, see example 5.

Definition
    Class: RoundTable
    Method: arrangements
    Parameters: int, int, int, int
    Returns: long
    Method signature: long arrangements(int countA, int countB, int chairs, int minDistance)

Notes
- The return value will fit in a 64 bit signed int.
- The input values are not guaranteed to allow a valid seating arrangement; if there is no seating arrangement that satisfies the input, return 0.

Constraints
- countA and countB will each be between 1 and 5, inclusive.
- chairs will be between 1 and 50, inclusive.
- minDistance will be between 1 and 50, inclusive.

Input File: C:\DKC3\RoundIn.txt
Output File: C:\DKC3\RoundOut.txt

Examples:

Input:

1, 1, 10, 1
1, 1, 10, 2
1, 2, 7, 3
5, 3, 11, 3
2, 1, 3, 1

Output:
90
70
14
0
6

Explanation of output:

90   Each company sends one representative. The first one can sit in any of the 10 chairs, the
        second one in any of the remaining 9 chairs (since minDistance=1 he may sit arbitrarily
        close to the other one). So there are a total of 10 * 9 = 90 arrangements.

70   The same as example 0, but now the second one can not sit next to the first one, so only 7
        chairs remain that he can choose from. This gives a total of 10 * 7 = 70 arrangements.

14   Company A sends one representative, who sits down in one of the 7 possible chairs.
        Since minDistance=3, two chairs next to him at each side must remain empty. This leaves
        only 2 chairs that the 2 representatives from company B have to share (2 possible
        combinations for which one takes which of the two chairs). This gives a total of 7 * 2 =
        14 arrangements.

0   Now there would be enough chairs, but since minDistance=3, we have to keep 2 chairs
        empty between the groups at each side, thus requiring at least 5 + 3 + 2 + 2 = 12 chairs.

6   Note that the two representatives of company A are treated as different persons, so there
        are a total of 3! = 6 possibilities to assign the three representatives to the three available
        chairs.

6. Seconds   (5 pts)

Write a program that takes the time as three integer arguments (for hours, minutes, and seconds), and returns the number of seconds since the last time the clock "struck 12". For example, a time of 1:35:04 will be specified in the test case as 1 35 4. Since 1:35:04 is 5,704 seconds since the last time the clock "struck 12", the answer is 5704.

Input
For each test case, there will be three integer arguments, each of which will be separated by a single space. There will be 10 test cases.

Output
The output for each test case will be on its own line in the output file.

Input File: C:\DKC3\SecondsIn.txt
Output File: C:\DKC3\SecondsOut.txt

Examples:

Input:

12 5 3
7 0 1

Output:

303
25201

7. Multiples   (5 pts)

If we list all of the natural numbers less than 10 that are multiples of 3 or 5, we get 3, 5, 6, and 9. The sum of these multiples is 23. In this problem, there are three variables: 10, 3, and 5. Write a program that takes these three variables as input and calculates the sum of the *unique* multiples.

Input
For each test case, there will be three integer arguments, each of which will be separated by a single space. The first integer is the limit, the next two are the multiples to match. There will be 10 test cases.

Output
The output for each test case will be on its own line in the output file.

Input File: **C:\DKC3\MultiplesIn.txt**
Output File: **C:\DKC3\MultiplesOut.txt**

Examples:

Input:

20 4 5
30 5 7

Output:

70
145

8. Base 9 **(10 pts)**

Normally, you use base 10 to do arithmetic. In computer science, you also deal with binary (base 2), octal (base 8), and hex (base 16). In this problem, we'll work with base 9, which uses the digits 0..8.

Input
Each line of input will contain two numbers separated by a space. Each number is in base 9. If converted to base 10, the numbers would be greater than 0, and smaller than 65000. There will be 10 test cases.

Output
For each line of input, your program should add the two numbers together and output the result (in base 9).

Input File: C:\DKC3\BaseIn.txt
Output File: C:\DKC3\BaseOut.txt

Examples:

Input:

148 765
111 888

Output:

1024
1110

9. Macro Expansion (30 pts)

The C programming language has a rich preprocessor that supports the use of macros. Macros are identified by the "#define" directive and may be of two forms, *object-like* and *function-like*. Object-like macros do not take parameters, function-like macros do. The syntax of a macro definitions are as follows:

#define <identifier> <replacement token list>
#define <identifier>(<parameter list>) <replacement token list>

Note that the *function-like* macro declaration must **not** have any whitespace between the identifier and the first, opening parenthesis. If whitespace is present, the macro will be interpreted as object-like with everything starting from the first parenthesis added to the token list.

After the C preprocessor has read a macro, whenever that identifier appears in the source code it is replaced with the replacement token list, which can be empty. For an identifier declared to be a function-like macro, it is only replaced when the following token is also a left parenthesis that begins the argument list of the macro invocation. This process is called macro expansion.

Write a program that will read in a C program and simulate object-like and function-like macro expansion of the C preprocessor.

Input
Input will consist of one C program, along with ten preprocessor macros (one for each test case). You may assume that the macro identifiers will not appear in string literals or comments. You may assume macro definitions will not be on more than 1 line, and will be the only thing on the line.

Output
Output should be the C program with the preprocessor macros removed and the macro identifiers in the remaining program replaced with the expanded macro.

Input File: C:\DKC3\MacroIn.txt
Output File: C:\DKC3\MacroOut.txt

Examples:

```
Input:    #define ARRAY_LEN_PRINT_STRING "Array Length = %d\n"
          #define ARRAY_LEN(array) (sizeof(array) / sizeof(array)[0])

          int main(int argc, char **argv)
          {
              long nums[50];
              printf(ARRAY_LEN_PRINT_STRING, ARRAY_LEN(nums));
          }
```

```
Output:   int main(int argc, char **argv)
          {
              long nums[50];
              printf("Array Length = %d\n", (sizeof(nums) / sizeof(nums)[0]));
          }
```

10. Morse Code   (10 pts)

Write a program that will translate a sentence in Morse code into English. The following table describes the translations:

Character	Code	Character	Code		Digits	
A	.-	N	-.		1	.----
B	-...	O	---		2	..---
C	-.-.	P	.--.		3	...--
D	-..	Q	--.-		4	....-
E	.	R	.-.		5	.....
F	..-.	S	...		6	-....
G	--.	T	-		7	--...
H	....	U	..-		8	---..
I	..	V	...-		9	----.
J	.---	W	.--		0	-----
K	-.-	X	-..-			
L	.-..	Y	-.--			
M	--	Z	--..			

Input
Input will consist of a list of Morse coded English sentences. A single space represents the end of a letter, and 3 spaces represent the end of a word. The end of the line ends both. There will be 1 sentence per line. Do not worry about punctuation. The Morse code uses the period . and dash - characters.

Output
Output should have the English translation of the Morse code. One sentence per line.

Input File: C:\DKC3\MorseIn.txt
Output File: C:\DKC3\MorseOut.txt

Examples:

Input:

```
.... . .-.. .-.. ---   .-- --- .-. .-.. -..
..   .- --   .-   -.-. --- -.. . .-.
```

Output:

```
HELLO WORLD
I AM A CODER
```

11. Conquering Applications   (45 pts)

You are Sir Gilliam Bates of the kingdom of Softshire, Great Britain. The kingdom to the north is called iland, and run by the brutal, yet very gifted and creative, Sir Jeven Stobs. You wish to conquer Sir Stobs' kingdom. To do this you must capture and hold all of the applications within his kingdom.

A certain minimum number of developers are required to capture an application. Some developers are expected to die during the attacks. After capturing the applications, some developers are required to remain with the application to defend it against attacks from Sir Stobs' army of developers. Of course, these numbers are different for different applications. Commanders of the developers are obliged to consider the number of developers required for victory. For example, there are five applications in the kingdom map shown in Figure 1. The application at the lower right requires at least 20 developers to wage a winning attack. None are expected to perish during the attack, and 10 developers must be left with the application when the army moves on.

10	5

20, 10, 5	10, 5, 10	5, 5, 5

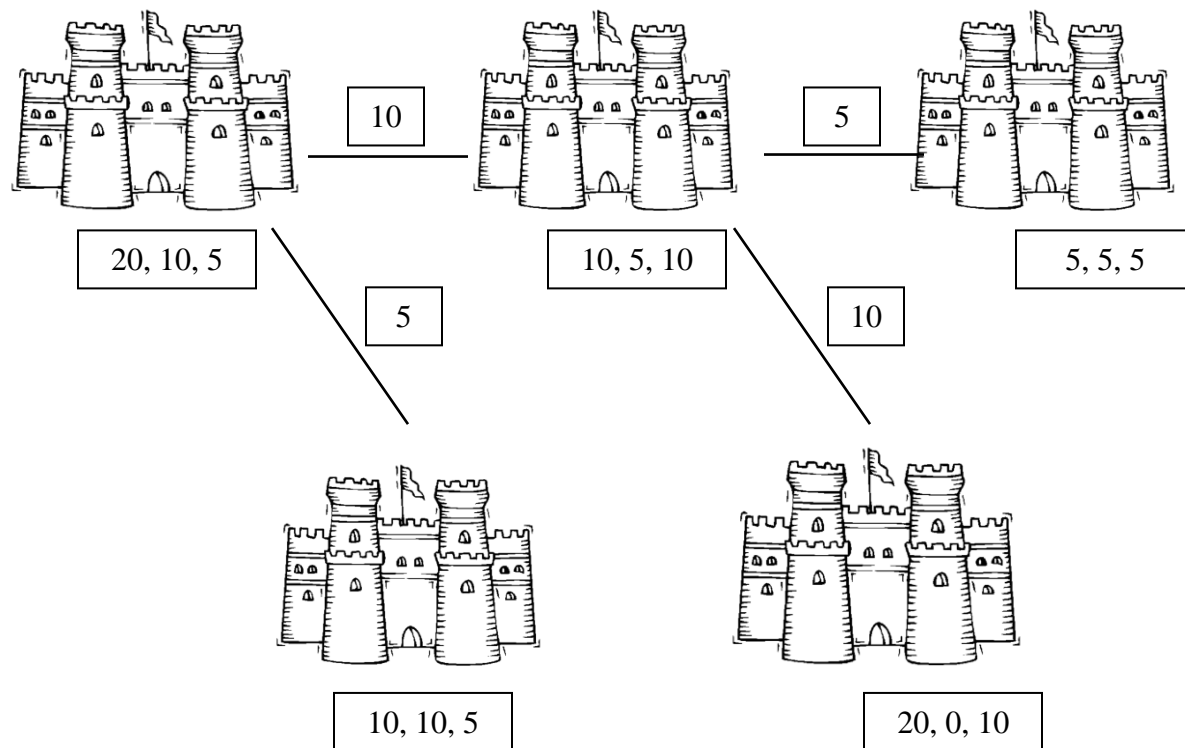| 5	10

10, 10, 5	20, 0, 10

Figure 1

For reasons of security, there is exactly one bi-directional network route between any pair of applications in the region. Moving into the neighborhood of an un-captured application begins

an attack on that application. Any application can serve as the first application to be attacked, without regard for how the army got there. Once an application has been captured, the requisite number of developers is left with the application to defend it, while the remainder of the developers move on to do battle at another application, if any remain un-captured. The army of developers must move as a whole unit, it may not be split up to attack separate applications in parallel. Also, because of the usual casualties of war on the road, such as data-packet loss, dysentery, or overdosing on Mountain Dew and Doritos, an army will lose developers along the network. Each time an army crosses over a network link from one application to another, fellow developers will die along the way. As can be seen in Figure 1, each network section has a number associated with it that indicates the number of developers that will perish each time an army crosses over it.

Input
There will be multiple lines per test case. Line 1 of a test case contains an integer $n < 100$ that is the number of applications in the kingdom. Each of the next n lines contains three integers a, b, c. Where a is the minimum number of developers required to successfully attack an application, b is the number of developers expected to perish in the attack and c is the number of developers that must be left with the application to defend it. The applications are numbered 1 to n, and the input lines describing them are given in increasing order of application numbers. The remaining $n - 1$ lines describe the network links connecting the applications. Each of these lines contains integers x, y, z. Integers x and y are the application numbers and z is the number of developers that will perish each time this network link is followed.

Output
For each test case give the minimum number of developers in the army needed to conquer and hold all of the applications in iland.

Input File: C:\DKC3\ConquerIn.txt
Output File: C:\DKC3\ConquerOut.txt

Examples:
Input: Output:

3   37
5 5 5   100
10 5 5
5 1 1
1 3 5
2 3 10
5
20 10 5
10 5 10
5 5 5
10 10 5
20 0 10
1 2 10
2 3 5
1 4 5
2 5 10

12. Mapping Mars   (45 pts)

In 2015 the new Mars rover, Enterprise, makes a startling discovery. Using radar and other imaging technology the rover discovers what appears to have been a small outpost or settlement at the base of Olympus Mons. The site is buried several feet below the surface and is 1 kilometer square. NASA has launched 3 new archeological rovers to dig to the buried artifacts at the site. NASA has mapped out the site and divided it into a 10 by 10 grid, with each grid being 1 square hectometer. Since the rovers will have to remove several feet of Martian soil to get to the artifacts the maximum area of coverage of each rover will be 4 square hectometers. Based on the imaging from Enterprise each square hectometer has been assigned a number from 0 to 100 to indicate the level of interest of the scientists at NASA, with a 0 indicating no interest and 100 indicating "ludicrous" interest. Your job is to calculate the 3 grid points at which the 3 archeological rovers need to start their digging in order to maximize the amount of interest points covered. As NASA doesn't want to risk the rovers damaging each other during the dig the 4 square hectometers covered by each rover must not overlap.

Input
There will be 10 lines per test case. Each line will be a list of 10 numbers from 0 to 100 indicating the interest level for that grid square. The grid is laid out such that coordinate (0,0) is the lower left corner, and coordinate (9,9) is the upper right corner. The first coordinate is the column number and the second coordinate in the pair is the row. There will be no ties in level of interest between the three top areas.

Each test case will be separated by an asterisk.

Output
For each test case print out the three starting coordinates (lower left corner of the 4 square hectometer coverage) of the 3 rovers that will maximize the interest points covered.

Input File: C:\DKC3\MappingIn.txt
Output File: C:\DKC3\MappingOut.txt

Examples:

Input:

```
100 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 40 0 0
0 0 10 0 0 0 0 0 10 0
0 0 0 20 0 0 0 50 0 0
0 0 0 0 0 0 0 0 0 0
0 0 5 0 0 0 0 60 0 0
0 0 0 0 0 0 50 0 0 0
0 0 10 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 5 0
0 0 0 0 2 0 0 0 0 0
*
100 0 0 0 0 0 0 0 100 0
0 0 0 100 0 0 0 40 0 0
0 0 10 0 0 0 0 0 10 0
```

0 0 0 20 0 0 0 50 0 0
0 0 0 0 0 0 0 0 0 0
0 0 5 0 0 0 0 60 0 0
0 0 0 0 0 0 50 0 0 0
0 0 10 0 0 100 0 0 0 0
0 0 0 0 0 0 0 0 5 0
0 0 0 0 2 0 0 0 0 0
*

Output:

(0,8) (7,6) (6,3)
(2,7) (5,2) (7,8)

13. Love at First Byte   (10 pts)

Fred is a single guy looking for love in all the wrong places. He finally met someone who was smart, funny, interesting, and, most importantly, a computer scientist. Fred had tried dating other computer science women before, but none of them could write a properly formatted linked list in C without consulting a reference manual!

Fred quickly typed up a SMTP client to send her a plaintext email to ask her out on a date. No later than 4.3 minutes after he sent it (he was eagerly timing it), he receives a strange response:

```
Fred,
Each single dot is 1 away from a space and each stacked dot is 5 away
from a space, but a space is a blank line.  A squiggle means you're
done.
```

Attached to the email was a regular text file where each line is a series of colons (:) and periods (.). Fred, being so brilliant, quickly realized that each line must represent a single character in her response. The dots must represent the ASCII value greater than the ASCII value of a space (which of course is 32). When calculating how much further the character is from a space, each colon would be worth five, and each space would be worth one. The squiggle must mean a tilde (~) which has a value of 126 in ASCII. Naturally, being a computer scientist, Fred decided to write a program to decrypt her response.

Input

Each test case will contain several lines of colons (:) and periods (.). Each line (delimited by a newline character) will represent a character in the message. Each encoded word/punctuation will be separated by a blank line. Each test case will end with an encoded tilde (~), which is represented as ":::::::::::::::::::....". There will be 10 test cases.

Output

For each input set, output the message as a single line *including the ending ~*. Each set should be separated by a newline character.

Input File: C:\DKC3\LoveIn.txt
Output File: C:\DKC3\LoveOut.txt

Examples:

Input:

```
::::::::
:::::::::::::::::....
::::::::::::::::::..

:::::::::::::::...
:::::::::::::::::....
::::::::::::::....
::::::::::::::::::...
```

```
::::::::::.
::::::::::::
::::::::::::::....
:::::::::::::::
:::::::::::::::..
::::::::::::::...
:::::::::::::
:::::::::::::::::....

:::::::::::::::::...
:::::::::::::::....
:::::::::::::::::
:::::::::::::::...
::::::::::::::...
::::::.
:::::::::::::::::....

:::::::::::
:::::::::::::....

:::::::::::::::::...
::::::::::::::..
::::::::::::::::....
:::::::::::::::::
::::::::::::::.
::::::::::::::...

::::::::::::::::..
::::::::::::::....
::::::::::::::....
:::::::::::::::::...

::::::::::::
::::::::::::::::....

::::....
::....
::::::::::::::::::....
```

Output:

How does Saturday sound?~
We should meet at 8.~

14. Bit Flipper **(15 pts)**

Bit Flipper is a game played by Digi-Key software engineers in their spare time. In this game, all the players are given a binary string of some fixed length and the goal is to convert this binary string to a string containing all 0's.

In a turn, an engineer is allowed to perform only one operation: Replace a 1 by a 0 or vice-versa. But each such operation will flip the states of all the bits following the bit you changed.
Take for example, the string: 1001010. You decide to flip the 1 located at the 4th position. The new string after the operation will be: 1000101. (Note that 5th to 7th bits flipped as a result of flipping the 4th bit.)

Give the minimal moves necessary to convert the string to all 0's.

Input
There are going to 10 test cases. Each test case consists of a single line which is the initial bit-string.

Output
A single line/number representing the smallest number of moves necessary for conversion to all 0's.

Input File: C:\DKC3\FlipperIn.txt
Output File: C:\DKC3\FlipperOut.txt

Examples:

Input:

0101
10000

Output:

3
2

15. Dome Builder Dennis   (15 pts)

Dennis is a home builder, but he doesn't like square structures so he builds geodesic domes instead. A geodesic dome is made up of triangles where the vertices of each triangle lie on an imaginary sphere (e.g. the Epcot Center is a geodesic dome). The resulting structure uses far fewer materials than an equivalent rectangular house and is significantly stronger. However, there are a few extra challenges when building a dome. One of those is calculating how much of each building material to buy.

Dennis' domes are of the hub-and-strut type. Each triangle in the dome framework is made up of 3 wooden struts that bolt into a hub at each vertex. A piece of plywood is nailed to the outside of each triangle forming the "skin" of the dome. Depending on the size of the dome, there can be from 1 to 6 different lengths of struts. Each is given a letter designation (A = the smallest strut, B = the next biggest, etc.). Each triangle can then be designated by the struts that make it up (e.g. a CCB triangle has a B length strut at its base and C length struts on its other two sides). The diagram to the right illustrates the framework of a "2 frequency" dome, which is made up of two strut lengths.
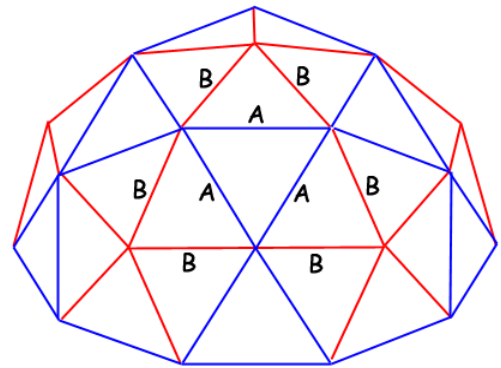
Dennis has a few domes to build and is trying to figure out how many nails to buy (he can get a sizable discount if he buys them up front in bulk). Write a program that will accept the length of each type of strut and the number of each type of triangle for a planned dome and calculate how many nails will be required to attach plywood to all of those triangles. Strut lengths will be in inches and nails are driven in every 4" along each strut. Note that there is no nail at the very end of each strut (the first nail along a strut will be 4" from each end).

Input File: C:\DKC3\DomeIn.txt
Output File: C:\DKC3\DomeOut.txt

Examples:

Input:

A:79 B:89 AAB:30 BBB:10
A:81 B:94 C:103 D:108 AAB:15 CCB:31 CCD:32

Output:

2460
5640