The CENTRE for EDUCATION
in MATHEMATICS and COMPUTING

2006 Canadian Computing Competition: Senior Division

Sponsor:

University of
Waterloo

Canadian Computing Competition
Student Instructions for the Senior Problems

1. You may only compete in one competition. If you wish to write the Junior paper, see the other problem set.

2. Be sure to indicate on your **Student Information Form** that you are competing in the **Senior** competition.

3. You have three (3) hours to complete this competition.

4. You should assume that

   - all input is from a file. The filename will be `s`N**.in**, where N is the problem number (i.e, the input file for Problem S1 is `s1.in`).
   - all output is to the screen

   For some problems, you may be asked for prompting: please provide this for the user. If no prompting is required, you do not need to provide any. Be sure your output matches the output in terms of order, spacing, etc. IT SHOULD MATCH EXACTLY!

5. Do your own work. Cheating will be dealt with harshly.

6. Do not use any features that the judge (your teacher) will not be able to use while evaluating your programs.

7. Books and written materials are allowed. Any machine-readable materials (like other programs which you have written) are *not* allowed. However, you are allowed to use "standard" libraries for your programming languages; for example, the STL for C++, java.util.*, java.io.*, etc. for Java, and so on.

8. Applications other than editors, compilers, debuggers or other standard programming tools are **not** allowed. Any use of other applications will lead to disqualification.

9. Please use file names that are unique to each problem: for example, please use `s1.pas` or `s1.c` or `s1.java` (or some other appropriate extension) for Problem S1. This will make the evaluator's task a little easier.

10. Your program will be run against test cases other than the sample ones. Be sure you test your program on other test cases.

11. Some problems will require an efficient solution in order to gain full marks (specifically, Problem S5).

12. Note that the top 2 Senior competitors in each region of the country will get a plaque and $100, and the schools of these competitors will also get a plaque. The regions are:

- West (BC to Manitoba)
- Ontario North and East
- Metro
- Ontario Central and West
- Quebec and Atlantic

13. If you finish in the top 20 competitors on this competition, you will be invited to participate in CCC Stage 2, held at the University of Waterloo in early May 2006. Should you finish in the top 4 at Stage 2, you will be a member of the Canadian IOI team at IOI 2006, held in Mexico. Note that you will need to know C, C++ or Pascal if you are invited to Stage 2. But, first, do well on this contest!

14. Check the CCC website at the end of March to see how you did on this contest, how the problems were meant to be solved, and to see who the prize winners are. The CCC website is:

`www.cemc.uwaterloo.ca/ccc`

Problem S1: Maternity

Problem Description

Alice and Bob are fruit flies. They are now the proud parents of a new baby girl fly. Unfortunately, due to a slight mix-up, the nurses in the maternity ward at the hospital aren't sure which baby fly is their daughter. Luckily they've hired you to help them out. Using full genetic profiles of the parents, can you figure out which baby fly is theirs?

Recall from biology that attributes (like eye colour, hair colour, etc) are inherited, or passed, from generation to generation. A single *gene* controls each such attribute. Alternative versions of each gene, called *alleles*, account for the variation in the attributes. For example, the gene for eye colour exists in two versions, one for brown eyes and one for blue eyes. For each attribute, a fruit fly has two alleles, one inherited from each parent.

If two alleles differ, then one, the *dominant allele*, will appear in the fruit fly's appearance. The other, or *recessive allele*, does not affect the fruit fly's appearance. Conventionally, the dominant allele is represented by an uppercase letter, and the recessive allele by a lowercase letter. For example, consider the gene for eye colour, B. Each fruit fly has two alleles, each is either that of brown eyes (B, dominant) or blue eyes (b, recessive). If the fruit fly has BB or Bb, she will have brown eyes; if the fruit fly has bb, she will have blue eyes.

When reproducing, each parent fruit fly passes exactly one allele for each gene to its child. We can draw a *Punnett square* to see all possible allele combinations for the child. For example, here is the *Punnett square* for possible offspring of Alice (Bb) and Bob (Bb):

		Bob	
		B	**b**
Alice	**B**	BB	Bb
	b	Bb	bb

Unfortunately, the full genetic profiles of the babies were not available (as they take weeks to process). All we have are the attributes of the baby – whether or not they have brown eyes, hair colour, etc. Can you use this information, with the full genetic profile of the parents, to determine which babies could not possibly be theirs?

Input Specifications

Luckily for you, our fruit flies are simple and have exactly five genes, labelled A through E. The input begins with two lines, describing the mother and father, respectively. Each line consists of five pairs of letters, one pair for each gene. Each pair describes the two alleles the parent has for a particular gene. These alleles are listed in order (from A through E).

Next is a line with the number X, $X <= 10$, the number of babies to check. Following are X lines describing the *attributes* of the baby. Each line consists of five letters in order (from A through E). An uppercase letter denotes the baby shows the attribute of the dominant allele, and a lowercase letter denotes the baby shows the attribute of the recessive allele. For example, if the baby has

brown eyes, the letter B will be written; if the baby has blue eyes the letter b will be written.

Output Specifications

For each baby test, output the line “Possible baby.”, if the baby could possibly be their offspring, or “Not their baby!” if impossible.

Sample Input

```
AABbCcddEe
AaBBccddee
5
ABCdE
aBcdE
ABcdE
ABCde
ABcDe
```

Output for Sample Input

```
Possible baby.
Not their baby!
Possible baby.
Possible baby.
Not their baby!
```

Explanation of the Sample Output

Baby 2 could not be their child. The baby has attribute *a*, which is recessive; therefore the baby must have alleles *aa*. The first parent has *AA*, and could not have given her baby an *a*. Baby 5 could not be their child. The baby has attribute *D*, but both parents have alleles *dd* and could not have given their baby a *D*. All other babies could possibly be their children!

Problem S2: Attack of the CipherTexts

Problem Description

Ruby is a code-breaker. She knows that the very bad people (Mr. X and Mr. Z) are sending secret messages about very bad things to each other.

However, Ruby has managed to intercept a *plaintext* message and the corresponding *ciphertext* message. By plaintext, we mean the message before it was encrypted (i.e., readable English sentences), and by ciphertext, we mean the message after it was encrypted (i.e., gibberish). To encrypt a message, each letter is changed to a new letter, so that if you read the ciphertext message, it is not obvious what the plaintext message is.

However, Ruby being the outstanding code-breaker she is, knows the algorithm that Mr. X and Mr. Z use. She knows they simply map one letter to another (possibly different) letter when they encrypt their messages. Of course, this map must be "one-to-one", meaning that each plaintext letter must correspond to exactly one ciphertext letter, as well as "onto", meaning that each ciphertext letter has exactly one corresponding plaintext letter.

You job is to automate Ruby's codebreaking and help save the world.

Input Specifications

The input consists of 3 strings, with each string on a separate line. The first string is the plaintext message which Ruby knows about. The second string is the ciphertext message which corresponds to the plaintext message. The third string is another ciphertext message. You may assume that all strings have length of at least 1 character and at most 80 characters. You can also assume that there are only 27 valid characters: the uppercase letters ('A' through 'Z') as well as the space character (' '). That is, there will be no punctuation, lowercase letters, or special characters (like '!' or '@') in either the plaintext or ciphertext messages.

Output Specifications

The output is a (plaintext) string which corresponds to the second ciphertext input. It may not be possible to determine each character of the second ciphertext string, however. If this is the case, the output should have a period ('.') character for those letters which cannot be determined.

Sample Input 1

```
THE QUICK BROWN FOX JUMPS OVER THE LAZY DOG
UIFARVJDLACSPXOAGPYAKVNQTAPWFSAUIFAMB ZAEPH
XFABSFAWFSZACBEAQFPQMFAEPJOHAWFSZACBEAUIJOHTAIBAIB
```

Output for Sample Input 1

```
WE ARE VERY BAD PEOPLE DOING VERY BAD THINGS HA HA
```

Explanation for Sample Output 1

Notice that every plaintext character is in the first message, and so, the entire mapping can be computed.

Sample Input 2

```
THERE ARE NOT ENOUGH LETTERS
XQAZASEZASNYXSANYLWQSTAXXAZM
JSCENNYXSIACYIASXQJM
```

Output for Sample Input 2

```
. .ANNOT .E.O.E TH.S
```

Explanation for Sample Output 2

Notice that the characters that cannot be determined are the (ciphertext) letters `J, C, I`, since these ciphertext letters do not appear in the earlier ciphertext message. It turns out that the plaintext message for the last ciphertext was `I CANNOT DECODE THIS`. All other letters should correspond between plaintext and ciphertext.

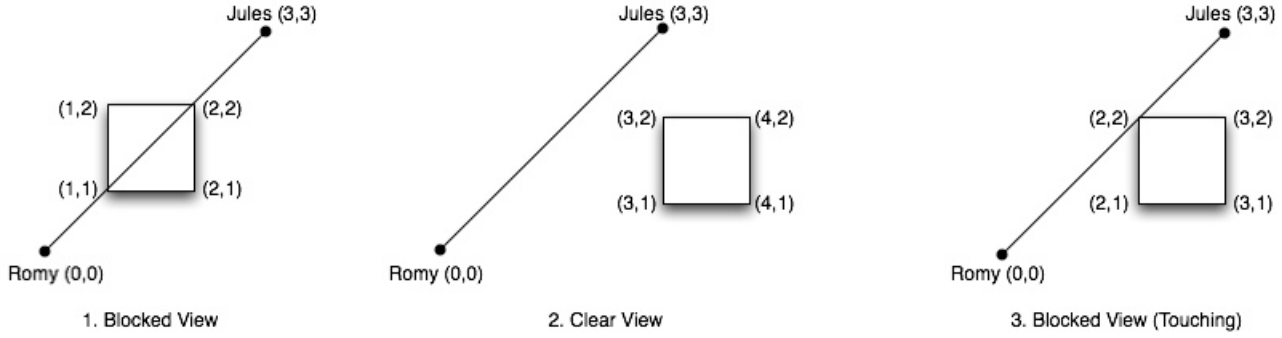
Problem S3: Tin Can Telephone

Problem Description

Romy and Jules have been talking with each other on their cell phones. Unfortunately, their parents dislike each other and have decided that they can no longer call each other. In fact, their parents have taken their cell phones away. So, Romy and Jules must find some other way to communicate. After searching the web for ideas, they have decided to build a "tin can" telephone.

Simply, a tin can telephone is two empty soup cans attached to each other with a string. To use it, the string must be stretched tight and then one person speaks while the other person listens. It is important that nothing touches the string so that it can vibrate and transfer sound from one can to the other.

To successfully set up a tin can telephone, Romy and Jules are going to need a clear line of sight between their two bedroom windows. To determine if they can run the string between their rooms, they get out a map that uses simple integer coordinates. Now consider the following three situations.



In these figures, "Romy" is Romy's window (the grid coordinates 0,0) and "Jules" is Jules' window (grid coordinates 3,3). In the first figure, there is a building between their windows, and it blocks the line of sight between them. In the second figure, the building doesn't block their view and they can successfully set up a tin can phone. In the third figure, a line drawn from Romy's coordinates to Jules' coordinates would touch the corner of the building. Since the string cannot touch anything, they cannot set up a tin can telephone and the view is considered as blocked.

Input Specifications

The input begins with a line containing four integer coordinates representing the locations of Romy's and Jules' windows. That is, the input $x_R\ y_R\ x_J\ y_J$ represents the coordinates (x_R, y_R) for Romy's window, and the coordinates (x_J, y_J) for Jules' window. You may assume that $-1000 \leq x_R, x_J \leq 1000$ and $-1000 \leq y_R, y_J \leq 1000$. The next line contains a single integer, n ($0 \leq n \leq 100$), identifying the number of buildings that will follow on the next n lines. Each building is on a separate line and begins with an integer specifying the number of corners that the building has. This integer is followed by the integer coordinates of the building's corners, in either clockwise

or counter-clockwise order. No building has more than 32 corners. The sample input and output, shown below, corresponds to the first “blocked” situation described previously.

Output Specifications

For the input data, output a single number identifying the number of buildings that touch or block the line of sight.

Sample Input

```
0 0 3 3
1
4 1 2 2 2 2 1 1 1
```

Output for Sample Input

```
1
```

Problem S4: Groups

Problem Description

In mathematics, a *group*, G, is an object that consists of a set of elements and an operator (which we will call $\times$) so that if x and y are in G so is $x \times y$. Operations also have the following properties:

- Associativity: For all x, y and z in G, $x \times (y \times z) = (x \times y) \times z$.
- Identity: the group contains an "identity element" (we can use i) so that for every x in G, $x \times i = x$ and $i \times x = x$.
- Inverse: for every element x there is an inverse element (we denote by x^{-1}) so that $x \times x^{-1} = i$ and $x^{-1} \times x = i$.

Groups have a wide variety of applications including the modeling of quantum states of an atom and the moves in solving a Rubik's cube puzzle. Clearly the integers under addition from a group (0 is the identity, and the inverse of x is $-x$, and you can prove associativity as an exercise), though that group is infinite and this problem will deal only with finite groups.

One simple example of a finite group is the integers modulo 10 under the operation addition. That is, the group consists of the integers $0, 1, \ldots, 9$ and the operation is to add two keeping only the least significant digit. Here the identity is 0. This particular group has the property that $x \times y = y \times x$, but this is not always the case. Consider the group that consists of the elements a, b, c, d, e and i. The "multiplication table" below defines the operations. Note that each of the required properties is satisfied (associativity, identity and inverse) but, for example, $c \times d = a$ while $d \times c = b$.

	i	a	b	c	d	e
i	i	a	b	c	d	e
a	a	i	d	e	b	c
b	b	e	i	d	c	a
c	c	d	e	i	a	b
d	d	c	a	b	e	i
e	e	b	c	a	i	d

Your task is to write a program which will read a sequence of multiplication tables and determine whether each structure defined is a group.

Input Specifications

The input will consist of a number of test cases. Each test case begins with an integer n ($0 \leq n \leq 100$). If the test case begins with $n = 0$, the program terminates. To simplify the input, we will use the integers $1, \ldots, n$ to represent n elements of the candidate group structure; the identity could be any of these (i.e., it is not necessarily the element 1). Following the number n in each test case are

n lines of input, each containing n integers in the range $[1, \ldots, n]$. The qth integer on the pth line of this sequence is the value $p \times q$.

Output Specifications

If the object is a group, output `yes` (on its own line), otherwise output `no` (on its own line). You should not output anything for the test case where $n = 0$.

Sample Input

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

Output for Sample Input

```
yes
yes
no
no
```

Explanation of Output for Sample Input

The first two collections of elements are in fact groups (that is, all properties are satisfied). For the third candidate, it is not a group, since $3 \times (2 \times 2) = 3 \times 1 = 3$ but $(3 \times 2) \times 2 = 1 \times 2 = 2$. In the last candidate, there is no identity, since 1 is not the identity, since $2 \times 1 = 3$ (not 2), and 2 is not the identity, since $2 \times 1 = 3$ (not 1) and 3 is not the identity, since $1 \times 3 = 3$ (not 1).

Problem S5: Origin of Life

Problem Description

Conway's *Game of Life* is not really a game, but a *cellular automaton* — a set of rules describing interactions among adjacent cells on a grid. In our game, we have an n by m rectangular grid of cells identified by integer coordinates (x, y).

The game progresses through a series of steps; at each step a new *generation* is computed from the current *generation*. The game begins with the *first generation*. In any given generation, which we shall call the current generation, each cell is either *live* or *dead*. In the next generation, each cell's status may change, depending on the status of its immediate neighbours in the current generation. Two distinct cells (x_1, y_1) and (x_2, y_2) are immediate neighbours if they are horizontally, vertically, or diagonally adjacent; that is, if $|x_1 - x_2| \leq 1$ and $|y_1 - y_2| \leq 1$. Each cell that is not on the border of the grid has eight immediate neighbours.

There are three integer parameters (a, b, c) which affect the game. The rules of the game are:

- If a live cell has fewer than a live neighbours in the current generation it dies of loneliness. That is, it is dead in the next generation.
- If a live cell has more than b live neighbours in the current generation it dies of overcrowding. That is, it is dead in the next generation.
- If a dead cell has more than c live neighbours in the current generation it is born. That is, it is live in the next generation.
- Otherwise, a cell's status is unchanged from the current generation to the next.

This process continues indefinitely. Eventually, a generation may be repeated in which case life goes on forever. Or all the cells may die. Similarly, if we explore previous generations that may have led to the current one, we may find one that is necessarily a first generation; that is, it could not have been created from a previous generation by following the rules. Such a generation is known as a Garden of Eden.

Given the game parameters and the current generation, you are to determine whether or not the game might have started with a Garden of Eden. If so, output the number of steps necessary to reach the current generation from the Garden of Eden. If there are several possible answers, find the smallest. If there is none, output -1.

Input Specifications

There are $m + 1$ lines of input in total. The first line of input contains the game parameters, which are five integers m, n, a, b, c each separated by one space. The constraints are $1 \leq m \leq 4$, $1 \leq n \leq 5$, $1 \leq a < b \leq 8$, $1 \leq c \leq 8$. The next m lines each contain a string of n characters representing a row of the current generation. In the string, an asterisk ("`*`") indicates *live* while a period ("`.`") indicates *dead*.

Sample Input

```
4 5 2 3 2
.****
.****
.****
.****
```

Output for Sample Input

```
2
```

Explanation of Output for Sample Input

Assume the sample input is the “current” generation. A possible previous generation is

```
**.**
..*.*
....*
*****
```

The above generation can be derived from the following previous generation

```
.****
**.*.
*****
*..*.
```

This generation cannot be derived from any other generation. Furthermore, there is no shorter series of generations that has these properties.