2ª Prova do CPUP'2007

4ª Prova do TIUP'2007

Conjunto de Problemas

Universidade do Porto
30 de Maio de 2007
17:00 às 20:00

Conteúdo:

Informação:

Limites impostos:

- Dados do programa: ***32Mb***
- Stack: ***4Mb***
- Código do Programa: ***100Kb***
- Tempo: ***dependente do problema, mas sempre no mínimo 3x maior que solução modelo***

Compiladores:

Uma submissão com "Compile Time Error" é clicável para se saber qual o erro obtido ao compilar. Para não obter erros é fundamental compilar como no servidor do Mooshak. Relembra que não pode haver nenhum "warning".

Linguagem	Compilador	Extensão	Comando de Compilação
C	gcc 3.3.2	.c	`gcc -Wall -lm $file`
C++	gcc 3.3.2	.cpp	`g++ -Wall $file`
Java	jdk 1.5.0_06	.java	`javac $file`
Pascal	fpc 2.0.4	.pas	`fpc -v0w -oprog $file`

Comissão Científica:

- **DCC / FCUP**
  - Pedro Ribeiro
  - Fernando Silva
  - José Paulo Leal
- **FEUP**
  - André Restivo
  - Augusto Sousa
  - Cristina Ribeiro
  - Gabriel David
  - Lígia Ribeiro

Problem A - Extra Terrestrial Bacterium

Since humans mastered interstellar travel, a lot of new worlds have been visited. One of these worlds, imaginatively called BXP-167, has revealed a curious bacteria specie like no other seen on earth. These bacterium are in fact 4 different types of bacterium that scientists like to call *cardinal*, *asterisk*, *null* and *cross* because of their unusual shapes (see Figure 1).

Fig. 1 - The first shot taken of asterisk bacterium.

What's funny about these bacterium is that, besides always being organized in a grid like form, they seem to be able to change from one type to another following a set of rules:

- *null* bacterium always transform in *cross* bacterium.
- *cross* bacterium transform into *cardinal* bacterium when they have more than 1 *cross* bacterium near them.
- *cardinal* bacterium transform into *asterisk* bacterium when they have at least 1 *cross* bacterium near them.
- *asterisk* bacterium transform into *null* bacterium when they have less than 2 *null* bacterium near them.

Note: For a bacteria to be near another it has to be directly above it, below it or at its left or right.

These rules are automatically triggered every second. These are the rules that scientist found so far but they suspect more can exist.

The Problem

Your task is to write a program that will aid scientists to find if more rules exist. Your program should be capable of, knowing the initial bacterium configuration, anticipating its final configuration after a number of seconds.

Input

The first line of the input will contain three integers **W**, **H** and **S** ($1 \leq W,H \leq 100$, $0 \leq S \leq 100$) representing the number of columns and lines of the bacteria population and also the number of seconds to be elapsed before the final configuration is found.

The next **H** lines will contain **W** characters, each one representing a single bacteria: # for *cardinals*, * for *asterisks*, 0 (zero) for *nulls* and X (capital x) for *crosses*.

Output

The output will have **H** lines each one of them containing **W** characters. These lines will represent the final configuration after **S** seconds.

Sample Input 1

```
10 5 1
#####**####
##000000##
##00XX00##
##000000##
#####**####
```

Sample Output 1

```
####00####
##XXXXXX##
##XXXXXX##
##XXXXXX##
####00####
```

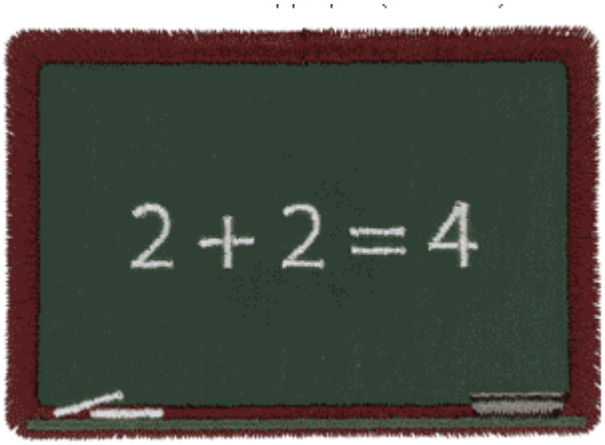
Sample Input 2

```
10 5 10
#####**####
##000000##
##00XX00##
##000000##
#####**####
```

Sample Output 2

```
X#0X**X0#X
#*#0000#*#
*X*XXXX*X*
#*#0000#*#
X#0X**X0#X
```

Problem B - Unattainable Numbers



In the fall of 1989 Paul Loomis was a sophomore at Wabash College. During an uninspiring lecture, he tried to find an interesting number sequence. And he found something he liked! Imagine the following function:

```
f(n) = n + (the product of the
nonzero digits of n)
```

If we begin with the number 1, by iterating f, we obtain the following sequence:

```
1, 2, 4, 8, 16, 22, 26, 38, 62, 74, 102, 104, 108, (...)
```

For example, from 22, we get `f(22) = 22 + 2*2 = 22 + 4 = 26`. And then, `f(26) = 26 + 2*6 = 26 + 12 = 38`. When he have zeros, we ignore them in the product, as we said before. For example, `f(102) = 102 + 1*2 = 104`. If we start with another number, we have another sequence that eventually hits a number in the first sequence we gave. For example, if we start with 5 or 19:

```
5, 10, 11, 12, 14, 18, 26, (...)
19, 28, 44, 60, 66, 102, (...)
```

However, some numbers are "unattainable" in the sense that for obtaining them using the function described before, we must use them as the starting number. More formally, a number `n` is unattainable if there exists no `m > 0`, such that `f(m) = n` (the same as saying there exists no `m > 0` such that `m + (the product of the nonzero digits of m) = n`). For example, 1, 5 and 19 are unattainable numbers.

The Problem

For a given number, you have to discover if it is an unattainable one.

Input

In the first line of input comes an integer number **C**, indicating the quantity of numbers to evaluate ($1 \leq C \leq 10\,000$).

Then follow exactly **C** lines, each one with one integer **N**, indicating the numbers you should evaluate ($1 \leq N \leq 1\,000\,000$).

Output

The output has exactly **C** lines, each one indicating if the respective number is unattainable, in the following format:

```
N is unattainable
```
or
```
N is not unattainable
```

See the sample input and output for a concrete example.

Sample Input

```
7
1
74
11
5
19
63
66
```

Sample Output

```
1 is unattainable
74 is not unattainable
11 is not unattainable
5 is unattainable
19 is unattainable
63 is unattainable
66 is not unattainable
```

Problem C - Mighty-Hero Competition

Super Heros from all over the galaxy are coming to town for their annual Mighty-Hero Competition(tm). This year's competition will be held as a race where Heroes jump from building to building. The race will take part in a very well organized part of the city, where buildings are distributed in a grid like structure. The only known property of each building is its height.

The race always start in the top left building of the selected town area and ends in the bottom right one. Super Heros are free to choose their path but have to be careful because when they jump down they lose energy and also because they have a limit on how much they can climb. If an hero reaches a **negative energy level** he dies.

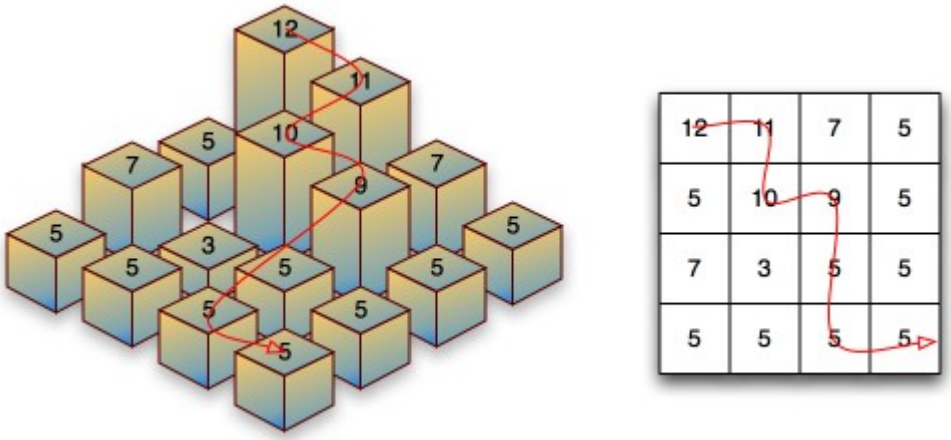


Fig 1. - One possible scenario.

As usual, as the competition stirs a lot of enthusiasm, there are lot of betting offers for people wishing to guess the overall winner. *Dr. What* is a evil villain who is trying to take advantage of this, by determining the winner beforehand. For that he invented a machine that is capable of scanning the overall characteristics of each Super Hero:

- **Energy Level** - The initial energy of each Super Hero.
- **Climbing Capability** - A Super Hero with climbing capability of *c* can only jump from one building to a taller one if the height difference is less or equal to *c*.
- **Descent Capability** - If a Super Hero jumps from one building to a smaller one and the difference in height between the two buildings is greater than its descent capability *d*, then he will lose energy. The energy lost will be equal to the difference between *d* and the height difference.

Example: An hero with climbing capability of 3 can climb from a building of height 1 to one of height 4 but not to one of height 5. An hero with descent capability of 3 can jump from a building of height 7 to a building of height 4 without losing any energy, but if he jumps from the same building to a building of height 2 he will lose 2 points of energy.

The Problem

Dr. What wants you to create a program that can calculate the number of jumps each Super Hero takes to finish the race and the energy he will have at the end. You should assume that all Super Heroes have Super Intelligence and will always choose the quickest path. If two paths take the same number of jumps the Hero will chose the one that drains less energy.

Input

The first line of the input file will contain information about the hero as three integers **E**, **C** and **D**: energy ($0 \leq E \leq 300$), climb ability ($0 \leq C \leq 100$) and descent ability ($0 \leq D \leq 100$). The second line will contain two integers **C** and **L**: the number of buildings in each grid line ($1 \leq C \leq 200$) and in each grid column ($1 \leq L \leq 200$). The next **L** lines will describe each line of the city. Each one of these lines will contain **C** integers containing the height **H** ($1 \leq H \leq 300$) of each building.

Output

The output will be a single line containing two integers. The minimum number of jumps the Super Hero must use to reach the final building and the remaining energy. If two paths exist that require the same number of moves you should print the one that requires less energy spending. If the hero cannot complete the course without depleting his energy the output should be a single line containing the word **impossible**.

Sample Input 1

```
10 5 2
4 4
12 11 7 5
5 10 9 5
7 3 5 5
5 5 5 5
```

Sample Output 1

```
6 8
```

Sample Input 2

```
2 5 0
4 4
12 11 7 5
5 10 9 5
7 3 5 5
5 5 5 5
```

Sample Output 2

```
impossible
```

Problem D - Finding UFOs

It is known that a certain kind of UFOs are there. They look like a disk and, when they are seen from the bottom side, they present a variable number of thin and thick concentric circles. It is also known that the UFO presenting the largest number of consecutive thick circles is the leader of a group.

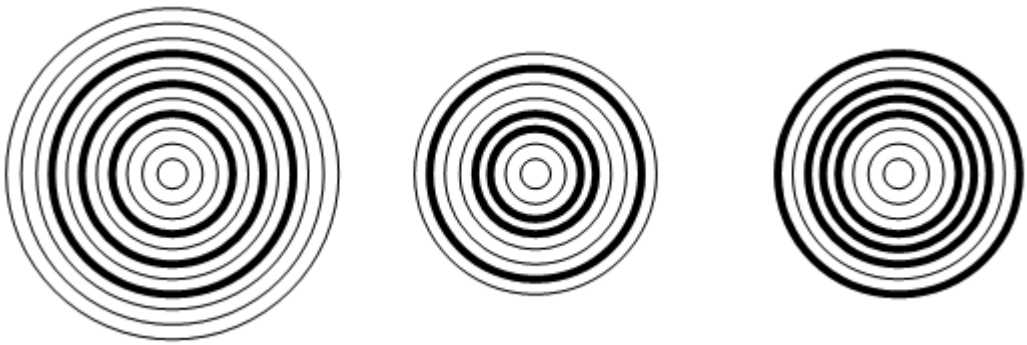


Fig. 1 - Three UFOs; the last one, on the right side, is the leader.

One got several images from these UFOs and processed them, trying to know where the leader is located. Unfortunately, the processing system used was able to detect, from each circle line, only the thickness and the (x, y) coordinates of three points. Furthermore, data resulted completely unsorted, so it is now necessary to reconstruct their positions.

The Problem

The problem you have to solve is to evaluate the (x, y) centre coordinates of one UFO group leader. To do it, you will have access to a list of circles, without any order guaranteed. Each circle contains its thickness (0=thin; 1=thick) and the coordinates (x, y) of the detected three points that belong to the circle line.

Input

The first line of the input contains the number of circles defined in the next lines, one circle per line. Each one of the subsequent lines is composed by a character "0" or "1" and a sequence of six decimal values in the order x1 y1 x2 y2 x3 y3. All the values are separated by a space, and can be negative or positive.

The maximum number of circles is 1000 and there may be UFOs with only one circle each. Coordinates of centres are integer values. Circles from different UFOs may overlap.

Output

The output must be a text line containing two integer values corresponding to the resulting coordinates x and y. Remember that these should be the coordinates of the leader UFO. There will always be an unique leader (there will be an UFO with an unique maximum of consecutive thick circles).

Sample Input

```
9
1 185.0000 64.0192 200.0000 60.0000 230.0000 90.0000
1 100.0000 50.0000 94.6410 70.0000 80.0000 84.6410
1 70.0000 50.0000 68.6603 55.0000 60.0000 40.0000
0 217.3205 100.0000 180.0000 90.0000 200.0000 110.0000
0 40.0000 50.0000 80.0000 50.0000 60.0000 30.0000
0 90.0000 50.0000 45.0000 75.9808 45.0000 24.0192
0 225.0000 133.3013 243.3013 115.0000 150.0000 90.0000
0 200.0000 100.0000 190.0000 90.0000 210.0000 90.0000
1 180.0000 124.6410 200.0000 130.0000 240.0000 90.0000
```

Sample Output

```
200 90
```

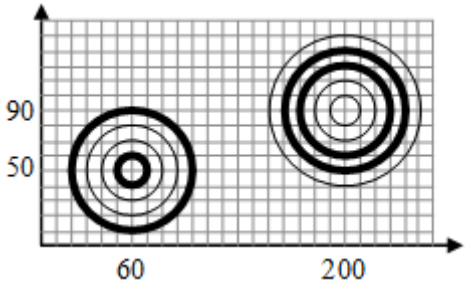
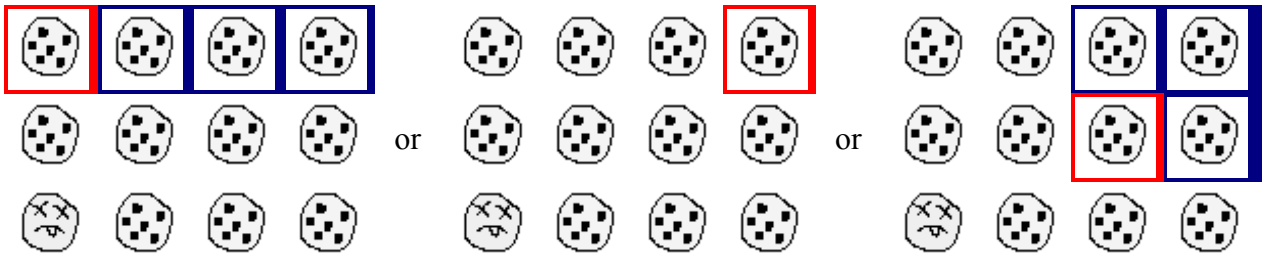


Fig. 2 - The two UFOs of the sample input.

Problem E - Chomp

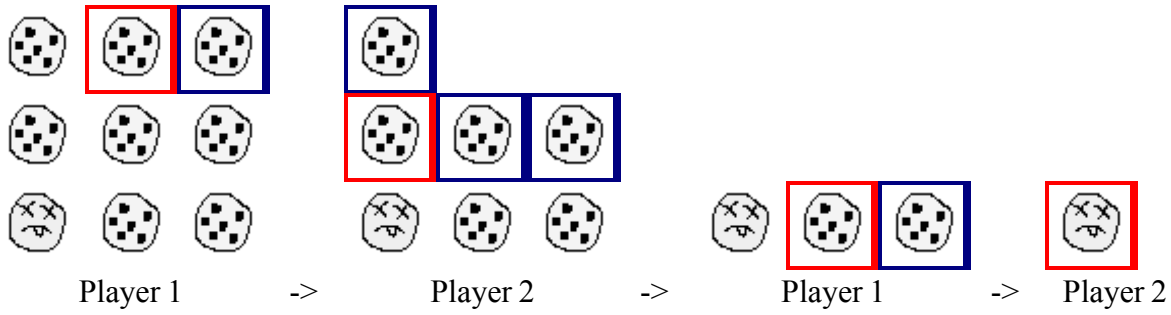
As you know, the Cookie Monster from Sesame Street is always interested in eating a lot of cookies. Recently, he even invented a game which, of course, was about... cookies. He wanted to have fun with Elmo, the furry red creature, while still being able to eat his favourite food. The game is for 2 players and he called it "Chomp". It starts with an `MxN` grid table of cookies. For example, it can be a 3x4 grid:

All the cookies are good, except the one in lower leftposition, which is a poisonous one, and nobody wants to eat it. The players take turns to choose one cookie from the table, eating it (that is, also removing it from the table), together with the cookies that are above it and to its right. For example, given the initial table above, these are 3 possible moves for the first player (eating and removing the selected cookies). Note there are also other possible moves.



The game continues like that until one player is forced to eat the poisonous cookie. That player looses the game (nobody wants that cookie!)

For example, this would be a valid game in a `3x3` table, with player 2 losing because it can only eat the poisonous cookie:



You have to help Cookie Monster, because we wants to win all games against Elmo (and then challenge Oscar The Grouch, Kermit the Frog, Bert and Ernie).

The Problem

For a given table position of the Chomp game, if you have to tell if it is a winning position or not. That is, given that all players play the best moves, is it a guaranteed win for the player playing on that position. Or, in other words, no matter the other player responses, there is always a winning strategy that ensures the win for the player who is going to play first on that position.

Input

In the first line of input comes an integer number **C**, indicating the number of Chomp positions (tables) to evaluate ($1 \leq C \leq 10$).

Then follow **C** tables, each one starting by a line with two space separated integers **R** and **C**, indicating respectively the number of rows and columns of the table ($1 \leq R,C < 10$). After that come exactly **R** lines, each one with **C** characters, indicating the content of a valid table (where `'.'` means an empty position and `'C'` means a cookie - the poisonous cookie is always on the lowest left position and is indicated by `'P'`).

Output

For each table in the input you should output a line with the following format (where NUM is the table number in the input):

```
Case #NUM: winning position
```
or
```
Case #NUM: losing position
```
Of course the output should reflect the fact that the position is a winning one (guaranteeing a win) or not.

Sample Input

```
4
3 3
...
...
P..
3 3
CCC
CCC
PCC
3 4
....
C...
PCCC
2 4
CCC.
PCCC
```

Sample Output

```
Case #1: losing position
Case #2: winning position
Case #3: winning position
Case #4: losing position
```