Team Practice 2

21 October 2012: 1:00 – 6:00 PM

Contest Problem Set

The ten problems on this contest are referred to, in order, by the following names:

square, slides, persist, racket, clocks, goodbad, class, mines, tautology, rooks

All problem submissions will be done via the PC^2 system. Any questions about the problems should be addressed through the clarification system. Be sure that you are submitting your code under the correct problem name!

No matter what the individual problem statements say, your program should read its input from standard in and print its output to standard out.

You may print to the printer at any point during the contest.

Printed code libraries and online language references are permitted; all other aids, including other internet resources and any code you may have typed before the contest, are not.

Good Luck!

Problem D: Largest Square

There is an $N \times N$ mosaic of square solar cells ($1 \le N \le 2{,}000$). Each solar cell is either good or bad. There are W ($1 \le W \le 50{,}000$) bad cells. You need to find the largest square within the mosaic containing at most L ($0 \le L \le W$) bad cells.

Input Specification

The input will begin with a number $Z \le 20$, the number of test cases, on a line by itself. Z test cases then follow. The first line of the test case contains three space-separated integers: N, W, and L. W lines follow, each containing two space-separated integers representing the coordinates of a location of the bad solar cells.

Sample Input

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

Output Specification

For each input instance, the output will be a single integer representing the area of the largest square that contains no more than L bad solar cells.

Output for Sample Input

```
4
```

Explanation of Sample Output

The mosaic is 4× 4, and contains the following arrangement of good and bad cells ('G' represents good, and 'B' represents bad):

```
BGGG
GBBG
GGGG
GGGG
```

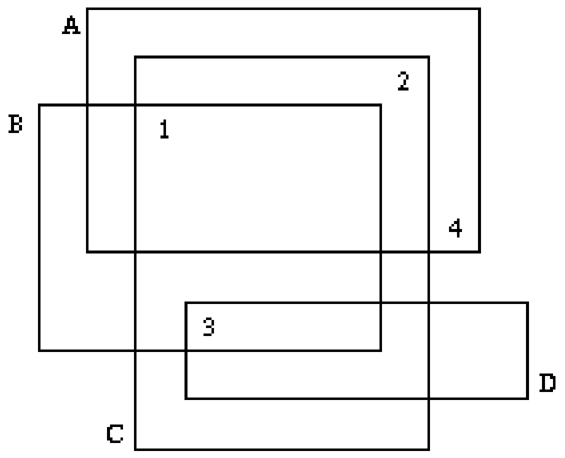
Several 2× 2 squares at the bottom contain no bad solar cells, but all 3 × 3 squares contain at least two bad solar cells.

Neal Wu

663 Sorting Slides

Professor Clumsey is going to give an important talk this afternoon. Unfortunately, he is not a very tidy person and has put all his transparencies on one big heap. Before giving the talk, he has to sort the slides. Being a kind of minimalist, he wants to do this with the minimum amount of work possible.

The situation is like this. The slides all have numbers written on them according to their order in the talk. Since the slides lie on each other and are transparent, one cannot see on which slide each number is written.



Well, one cannot see on which slide a number is written, but one may deduce which numbers are written on which slides. If we label the slides which characters A, B, C, ... as in the figure above, it is obvious that D has number 3, B has number 1, C number 2 and A number 4.

Your task, should you choose to accept it, is to write a program that automates this process.

Input

The input consists of several heap descriptions. Each heap descriptions starts with a line containing a single integer n, the number of slides in the heap. The following n lines contain four integers x_{min}, x_{max}, y_{min} and y_{max}, each, the bounding coordinates of the slides. The slides will be labeled as A,B,C,... in the order of the input.

This is followed by n lines containing two integers each, the x- and y-coordinates of the n numbers printed on the slides. The first coordinate pair will be for number 1, the next pair for 2, etc. No number will lie on a slide boundary.

The input is terminated by a heap description starting with $n = 0$, which should not be processed.

Output

For each heap description in the input first output its number. Then print a series of all the slides whose numbers can be uniquely determined from the input. Order the pairs by their letter identifier.

If no matchings can be determined from the input, just print the word `none` on a line by itself.

Output a blank line after each test case.

Sample Input

```
4
6 22 10 20
4 18 6 16
8 20 2 18
10 24 4 8
9 15
19 17
11 7
21 11
2
0 2 0 2
0 2 0 2
1 1
1 1
0
```

Sample Output

```
Heap 1
(A,4) (B,1) (C,2) (D,3)

Heap 2
none
```

Problem B: Persistent Numbers

The multiplicative persistence of a number is defined by Neil Sloane (Neil J.A. Sloane in *The Persistence of a Number* published in Journal of Recreational Mathematics 6, 1973, pp. 97-98., 1973) as the number of steps to reach a one-digit number when repeatedly multiplying the digits. Example:

```
      679 -> 378 -> 168 -> 48 -> 32 -> 6.
```

That is, the persistence of 679 is 5. The persistence of a single digit number is 0. At the time of this writing it is known that there are numbers with the persistence of 11. It is not known whether there are numbers with the persistence of 12 but it is known that if they exists then the smallest of them would have more than 3000 digits.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

The problem that you are to solve here is: what is the smallest number such that the first step of computing its persistence results in the given number?

For each test case there is a single line of input containing a decimal number with up to 1000 digits. A line containing -1 follows the last test case. For each test case you are to output one line containing one integer number satisfying the condition stated above or a statement saying that there is no such number in the format shown below.

Sample input

```
0
1
4
7
18
49
51
768
-1
```

Output for sample input

```
10
11
14
17
29
77
There is no such number.
2688
```

P. Rudnicki

Numbers Racket

Ollie has invented a new game for Stan to play. It works like this. First, Ollie chooses four integers, $1 \le n \le 1000$, $1 \le k \le 1000$, $0 \le x < n$, and $0 \le y < n$, and writes the number x on the blackboard. Stan's goal is to get y onto the blackboard. However, it is not so simple as just erasing x and writing y. Stan can only change the number on the board according to the following rules:

- Stan can only erase the number a and write the number b if $|a - b| \le k$ and $0 \le b < n$.
- In order to make such a change to the number on the board, Stan has to pay Ollie a fee. This fee depends on whether Stan's current turn is odd (first, third, fifth, etc) or even (second, fourth, etc). If it is an odd turn, then Stan has to pay Ollie $(a^2 + b^2)$ MOD n cents, and if it is an even turn, then Stan has to pay Ollie $(a^4 + b^4)$ MOD n cents. So, for instance, if $n = 9$, then to erase 1 and write 2 would cost Stan 5 cents on an odd turn, and 8 cents on an even turn.
- Stan cannot just skip a turn to change whether the current turn is even or odd, but if he wants, he can erase the number on the board and re-write the same number in the same way he could make any other move, by paying the same fee.

Ollie figures that this game will confound Stan, and earn him great riches! But of course he has to convince Stan to play, so he needs to offer him a reward for writing y on the board. Unfortunately, Ollie has confounded himself as well, and doesn't know what prize to offer! Help Ollie out by telling him the largest prize he can offer while still guaranteeing that he never loses money when Stan plays the game.

Input: The first line will contain a single integer, the number of cases (up to 20, no more than 5 of which have $n > 500$). There follows one line for each case, each containing four integers: n, k, x, and y, in that order. You are guaranteed that these integers will be within the bounds described above.

Output: For each test case, one line containing a single integer: the maximum reward that Ollie can offer, in cents, that still guarantees that Ollie will never lose money when Stan plays the game.

```
Sample Input:
2
9 1 3 7
17 3 0 3


Output for Sample Input:
17
4
```

Problem 5: Clock Repair

Mr. Horologia's House of Clocks contains various cuckoo clocks that customers have brought in for repair. Since they are in the clock shop, one might rightly assume that these clocks don't quite run as they should. In fact, a fast clock may take 3,500 seconds to advance one hour, instead of 3,600 seconds. A slow clock might take 3,750 seconds.

Every midnight on Sunday morning, Mr. Horologia sets all clocks to exactly 12:00. He has sufficient assistants awake at that hour that all clocks can be set simultaneously. Some time later, possibly that same day but quite possibly several days later, all clocks in the room will cuckoo at precisely the same instant in time. (All clocks initially would chime when they are set at midnight on Sunday, but the initial cuckooing does not count.) What is the first day, hour, minute, and second when all clocks simultaneously go off? The time might be several days in the future; also, midnight on the next Sunday morning might come around again before they ever cuckoo simultaneously. In this latter case, Mr. Horologia will set them all again, and the correct answer would be "Never".

Example

An easy example involves two clocks, one that advances an hour every 3,000 seconds, and a second clock that advances an hour every 4,500 seconds. For simplicity, note that these represent 50 minutes and 75 minutes, respectively. The first clock would reach 1:00 AM after 50 minutes, then 2:00 AM after 100 minutes, and 3:00 AM after 150 minutes. The second clock would reach 1:00 at 75 minutes and 2:00 at 150 minutes. Thus, 150 minutes after midnight, clock number one and clock number two both cuckoo. The correct answer is thus "Sunday at 2:30:00 AM".

A second example might involve four clocks, all of which are extremely fast. They advance at 600, 1200, 1800, and 600 seconds. After 30 minutes, all of them cuckoo except for the second clock. After 60 minutes, all will cuckoo. Thus the answer here is "Sunday at 1:00:00 AM".

Finally, suppose we have four clocks with speeds of 3601, 3559, 3600, and 3700. The answer in this case is "Never" – the next Sunday will occur before all clocks cuckoo.

Input

There will be multiple cases, sequentially numbered starting with 1. The input for each case is a single line that contains integers giving the number of clocks *N*, followed by *N* "seconds" measurements. A line containing the integer 0 follows the last case. Because the repair shop is limited, *N* will never be larger than 10.

Output

For each case, display the case number and either the first date and time when all clocks will cuckoo simultaneously, or the word "Never" as described above. Display a blank line after the output for each case.

Sample Input

```
2 3000 4500
4 600 1200 1800 600
3 3550 3650 3655
2 3525 3625
0
```

Output for the Sample Input

```
Case 1: Sunday at 2:30:00 AM

Case 2: Sunday at 1:00:00 AM

Case 3: Never

Case 4: Friday at 9:58:45 PM
```

Problem A — limit 5 seconds

Good or Bad?

Description

Bikini Bottom has become inundated with tourists with super powers. Sponge Bob and Patrick are trying to figure out if a given character is good or bad, so they'll know whether to ask them to go jelly-fishing, or whether they should send Sandy, Mermaid Man, and Barnacle Boy after them.

sponge bob: Wow, all these characters with super powers and we don't know whether they are good guys or bad guys.

patrick: Well, it's easy to tell. You just have to count up the number of g's and b's in their name. If they have more g's, they are good, if they have more b's, they are bad. Think about it, the greatest hero of them all, Algorithm Crunching Man is good since he has two g's and no b's.

sponge bob: Oh, I get it. So Green Lantern is good and Boba Fett is bad!

patrick: Exactly! Uh, who's Boba Fett?

sponge bob: Never mind. What about Superman?

patrick: Well he has the same number of g's as b's so he must be neutral.

sponge bob: I see, no b's and no g's is the same number. Very clever Patrick! Well what about Batman? I thought he was good.

patrick: You clearly never saw *The Dark Knight...*

sponge bob: Well what about Green Goblin? He's a baddy for sure and scary!

patrick: The Green Goblin is completely misunderstood. He's tormented by his past. Inside he's good and that's what counts. So the method works!

sponge bob: Patrick, you are clearly on to something. But wait, are you saying that Plankton is neutral after all the terrible things he's tried to do to get the secret Crabby Patty formula?

patrick: Have any of his schemes ever worked?

sponge bob: Hmmm, I guess not. Ultimately he's harmless and probably just needs a friend. So sure, neutral works for him.

patrick: Alright then, let's start taking names and figure this out.

sponge bob: But Patrick, if we start counting all day, Squidward will probably get annoyed and play his clarinet and make us lose count.

patrick: Well, let's hire a human to do it for us on the computer. We'll pay them with Crabby Patties!

sponge bob: Great idea Patrick. We're best friends forever!

Help Sponge Bob and Patrick figure out who is good and who is bad.

Input

The first line will contain an integer n ($n > 0$), specifying the number of names to process. Following this will be n names, one per line. Each name will have at least 1 character and no more than 25. Names will be composed of letters (upper or lower case) and spaces only. Spaces will only be used to separate multiple word names (e.g., there is a space between Green and Goblin).

Output

For each name read, display the name followed by a single space, followed by " is ", and then followed by either "GOOD", "A BADDY", or "NEUTRAL" based on the relation of b's to g's. Each result should be ended with a newline.

Sample Input	**Sample Output**

```
8
Algorithm Crunching Man
Green Lantern
Boba Fett
Superman
Batman
Green Goblin
Barney
Spider Pig
```
	```
	Algorithm Crunching Man is GOOD
	Green Lantern is GOOD
	Boba Fett is A BADDY
	Superman is NEUTRAL
	Batman is A BADDY
	Green Goblin is GOOD
	Barney is A BADDY
	Spider Pig is GOOD
	```

Problem E: Class Schedule

At Fred Hacker's school, there are $T \times C$ classes, divided into C catagories of T classes each. The day begins with all the category 1 classes being taught simultaneously. These all end at the same time, and then all the category 2 classes are taught, etc. Fred has to take exactly one class in each category. His goal is to choose the set of classes that will minimize the amount of ``energy'' required to carry out his daily schedule.

The energy requirement of a schedule is the sum of the energy requirement of the classes themselves, and energy consumed by moving from one class to the next through the schedule.

More specifically, taking the *j*th class in the *i*th category uses E_{ij} units of energy. The rooms where classes take place are located at integer positions (ranging from 0 to L) along a single hallway. The *j*th class in the *i*th category is located at position P_{ij}. Fred starts the day at position 0, moves from class to class, according to his chosen schedule, and finally exits at location L. Moving a distance d uses d units of energy.

Input Specification

The first line of the input is $Z \leq 20$ the number of test cases. This is followed by Z test cases. Each test case begins with three space-separated integers: C, T, and L. Each of the following $C \times T$ lines gives, respectively, the location and energy consumption of a class. The first T lines represent the classes of category 1, the next T lines represent the classes of category 2, and so on. No two classes in the same category will have the same location.

Bounds

$1 \leq C \leq 25$
$1 \leq T \leq 1000$
$1 \leq L \leq 1,000,000$
$1 \leq E_{ij} \leq 1,000,000$
$0 \leq P_{ij} \leq L$

Sample Input

```
1
3 2 5
2 1
3 1
4 1
1 3
1 4
3 2
```

Explanation of Sample Input

Fred must take 3 classes every day, and for each he has 2 choices. The hall has length 5. His first possible class is located at position 2 and will take 1 unit of energy each day, etc.

Output Specification

For each input instance, the output will be a single integer on a line by itself which is the minimum possible energy of a schedule satisfying the constraints.

Output for Sample Input

```
11
```

Explanation of Sample Output

Here is one way to obtain the minimum energy:
Go to the class at location 2. Energy used: 3
Next, go to the class at location 4. Energy used: 6
Then go to the class at location 3. Energy used: 9
Finally, leave the school at location 5. Energy used: 11

Neal Wu

Problem E: Data Mining?

Source file: `mines.{c, cpp, java, pas}`
Input file: `mines.in`
Output file: `mines.out`

A variation of the minesweeper game is available for almost every computer platform. Your employer wants to create yet another version that is targeted toward casual, as opposed to expert, players. Your task is to write a program that takes a minesweeper board and returns the minimum number of covered, unmined cells that remain after a casual player has tried his/her best. The details of the game and program are decribed below.

A minesweeper board consists of a rectangular grid of cells, with one or more cells containing a mine. The entire board is initially presented with all the cells covered, i.e., blank. The object of the game is to uncover all the cells that do not contain a mine. If a mine in uncovered, the game is over and the player loses. A cell can be in one of 3 states: *covered*, *cleared/uncovered*, or *flagged* as a mine.

When a player clears a cell that does not contain a mine, that cell displays the number of mines in cells that are adjacent to it. These numbers help the player determine where the mines are located. The adjacent cells are the cells that form a 3x3 square with the cleared cell in the center. Depending on a cell's location, it will have between 3 and 8 adjacent cells. The board in Figure 1 below shows two mines at locations (3,1) and (3,2), and the numbers of adjacent mines for each of the remaining cells.

A casual player makes use of this information in the following way. First the player selects one cell from a totally covered board. If it's a mine, the game is over. Otherwise, the player clears the cell and then applies the following two rules to cleared cells on the board until no further progress can be made. Let (x,y) be the location of a cleared cell, and let f, c, and m be the number of flagged, covered, and mined cells adjacent to (x,y).

1. If $f = m$, then clear all covered cells adjacent to (x,y).
2. If $f + c = m$, then flag all covered cells adjacent to (x,y).

Note that after successfully clearing the first cell, a casual player never clears or flags a cell except as dictated by rule 1 or 2, which means that the player may get "stuck". When a casual player is stuck, the game is over; no further guesses are made, and the player will not use more sophisticated rules that might allow him/her to safely clear additional cells.

Figure 2 below shows an application of these rules using the board from Figure 1.

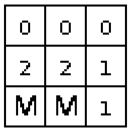


Figure 1

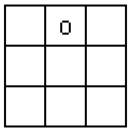


Figure 2a

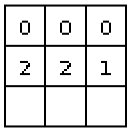


Figure 2b

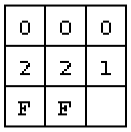


Figure 2c

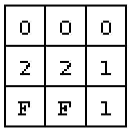


Figure 2d

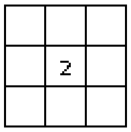


Figure 3

Figure 2a shows the board after a player initially clears cell (1,2). Rule 1 applies, since (0 flagged = 0 mined neighbors), so the player clears the adjacent cells at (1,1), (1,3), (2,1), (2,2), and (2,3), which leads to Figure 2b.

From the board in Figure 2b, the player can consider cell (2,1) and apply rule 2 (0 flagged + 2 covered = 2 mined) to flag cells (3,1) and (3,2) as mines. This generates Figure 2c.

Finally, by looking at cell (2,3), the player can again apply rule 1 to clear cell (3,3), since cell (2,3) has exactly 1 adjacent mine, and cell (3,2) is already flagged as a mine. Now, all the cells without mines have been cleared, so the game stops with the player winning.

As indicated above, these two rules are not sufficient to solve every game board from every starting position, so the player might get stuck. Again, considering the board in Figure 1, if the player instead first cleared cell (2,2), the resulting board appears as Figure 3. The player cannot make any further progress, since neither rule 1 nor rule 2 clears or flags any new cells. In this case the player is stuck with 6 covered cells that do not contain mines.

You must write a program that looks at a game board and determines the smallest number of covered, unmined cells that could possibly remain when a casual player plays the game as described. For the game board in Figure 1, the answer is 0.

The input file contains one or more game boards, followed by a final line containing only two zeros. A game board starts with a line containing two integers, *r* and *c*, the number of rows and columns in the game board; *r* and *c* will always be at least 3. The total number of cells in any board will never be greater than 40. The rest of the data set consists of a graphical representation of the game board, where an upper case '`M`' represents a mine and a period '`.`' represents an empty cell. There will always be at least one '`M`' and at least one '`.`' on each game board.

For each data set write one line with a single integer indicating the smallest number of covered, unmined cells for that board.

Example input:	**Example output:**

```
3 3
...
...
MM.
3 4
M.M.
.M.M
M.M.
7 5
.....
.....
MMM..
M.M..
MMM..
.....
.....
4 4
...M
....
....
M...
0 0
```
	```
	0
	5
	1
	0
	```

Last modified on October 23, 2003 at 8:34 PM.

Problem D: Tautology

WFF 'N PROOF is a logic game played with dice. Each die has six faces representing some subset of the possible symbols K, A, N, C, E, p, q, r, s, t. A Well-formed formula (WFF) is any string of these symbols obeying the following rules:

- p, q, r, s, and t are WFFs
- if *w* is a WFF, N*w* is a WFF
- if *w* and *x* are WFFs, K*wx*, A*wx*, C*wx*, and E*wx* are WFFs.

The meaning of a WFF is defined as follows:

- p, q, r, s, and t are logical variables that may take on the value 0 (false) or 1 (true).
- K, A, N, C, E mean *and, or, not, implies,* and *equals* as defined in the truth table below.

Definitions of K, A, N, C, and E					
w *x*	K*wx*	A*wx*	N*w*	C*wx*	E*wx*
1 1	1	1	0	1	1
1 0	0	1	0	0	0
0 1	0	1	1	1	0
0 0	0	0	1	1	1

A *tautology* is a WFF that has value 1 (true) regardless of the values of its variables. For example, *ApNp* is a tautology because it is true regardless of the value of *p*. On the other hand, *ApNq* is not, because it has the value 0 for *p=0, q=1*.

You must determine whether or not a WFF is a tautology.

Input consists of several test cases. Each test case is a single line containing a WFF with no more than 100 symbols. A line containing 0 follows the last case. For each test case, output a line containing *tautology* or *not* as appropriate.

Sample Input

```
ApNp
ApNq
0
```

Possible Output for Sample Input

```
tautology
not
```

Gordon V. Cormack

Problem A: Rooks

You have unexpectedly become the owner of a large chessboard, having fifteen squares to each side. Because you do not know how to play chess on such a large board, you find an alternative way to make use of it.

In chess, a rook attacks all squares that are in the same row or column of the chessboard as it is. For the purposes of this problem, we define a rook as also attacking the square on which it is already standing.

Given a set of chessboard squares, how many rooks are needed to attack all of them?

Input Specification

Input consists of a number of test cases. Each test case consists of fifteen lines each containing fifteen characters depicting the chess board. Each character is either a period (`.`) or a hash (`#`). Every chessboard square depicted by a hash must be attacked by a rook. After all the test cases, one more line of input appears. This line contains the word `END`.

Sample Input

```
...............
...............
...............
...............
...............
...............
...............
.......#.......
...............
...............
...............
...............
...............
...............
...............
END
```

Output Specification

Output consists of exactly one line for each test case. The line contains a single integer, the minimum number of rooks that must be placed on the chess board so that every square marked with a hash is attacked.

Output for Sample Input

```
1
```