CMU Fall Programming Contest: Round 2

October 1, 2011

- There are 8 problems. Do as many as you can in 4 hours.
- You *can* bring paper notes, code printouts, and books.
- You can consult these API documentation web sites:
  `http://download.oracle.com/javase/6/docs/api/`
  `http://cplusplus.com/doc`
- You *cannot* make use of any code that's in electronic form – from the internet, from the computer, or anywhere else – whether you wrote it yourself or not. You cannot use websites other than those mentioned on this page.
- The running time limit is 30 seconds, and the memory limit is 250MB. (Please note that these limits are for the server machine. The running time on your laptop might be faster or slower.)
- Input is from standard input, output is to standard output.
- Input files might consist of multiple test cases, read the input description for detials.
- The URL for the scoreboard and submitting solutions is
  `http://www.link.cs.cmu.edu/contest/current`

The Problems		
	Problem Name	Source (to be added after contest)
A	Milkshakes	
B	$\{a, b, c\}$ Forbidden Take-Away	
C	Sound	
D	Power Plants	
E	Elegant Diamond	
F	Colored Cubes	
G	Watch Tower	
H	EZ-Sokoban	

This contest was compiled and administered by Siyoung Oh, Richard Peng, Danny Sleator, and Kevin Waugh. Prizes were generously funded by IMC Financial Markets, Inc.

Carnegie Mellon

imc financial markets & asset management

A — Milkshakes

You own a milkshake shop. There are N different flavors that you can prepare, and each flavor can be prepared "malted" or "unmalted". So, you can make $2N$ different types of milkshakes.

Each of your customers has a set of milkshake types that they like, and they will be satisfied if you have at least one of those types prepared. At most one of the types a customer likes will be a "malted" flavor.

You want to make N batches of milkshakes, so that:

- There is exactly one batch for each flavor of milkshake, and it is either malted or unmalted.
- For each customer, you make at least one milkshake type that they like.
- The minimum possible number of batches are malted.

Find whether it is possible to satisfy all your customers given these constraints, and if it is, what milkshake types you should make. If it is possible to satisfy all your customers, there will be only one answer which minimizes the number of malted batches.

Input

One line containing an integer C ($1 \leq C \leq 5$), the number of test cases in the input file. For each test case, there will be:

- One line containing the integer N ($1 \leq N \leq 2,000$), the number of milkshake flavors.
- One line containing the integer M ($1 \leq M \leq 2,000$), the number of customers.
- M lines, one for each customer, each containing:
  - An integer $T \geq 1$, the number of milkshake types the customer likes, followed by
  - T pairs of integers "X Y", one for each type the customer likes, where X is the milkshake flavor between 1 and N inclusive, and Y is either 0 to indicate unmalted, or 1 to indicated malted. Note that:
    - No pair will occur more than once for a single customer.
    - Each customer will have at least one flavor that they like ($T \geq 1$).
    - Each customer will like at most one malted flavor. (At most one pair for each customer has $Y = 1$).
    - All of these numbers are separated by single spaces.

Output

C lines, one for each test case in the order they occur in the input file, each containing the string `Case #X:` where `X` is the number of the test case, starting from 1, followed by:

- The string `IMPOSSIBLE`, if the customers' preferences cannot be satisfied; OR
- N space-separated integers, one for each flavor from 1 to N, which is 0 if the corresponding flavor should be prepared unmalted, and 1 if it should be malted.

Example

Input	Output
2 5 3 1 1 1 2 1 0 2 0 1 5 0 1 2 1 1 0 1 1 1	Case #1: 1 0 0 0 0 Case #2: IMPOSSIBLE

Example

Input	Output
2 5 3 1 1 1 2 1 0 2 0 1 5 0 1 2 1 1 0 1 1 1	Case #1: 1 0 0 0 0 Case #2: IMPOSSIBLE

In the first case, you must make flavor #1 malted, to satisfy the first customer. Every other flavor can be unmalted. The second customer is satisfied by getting flavor #2 unmalted, and the third customer is satisfied by getting flavor #5 unmalted. In the second case, there is only one flavor. One of your customers wants it malted and one wants it unmalted. You cannot satisfy them both.

B — The Game of $\{a, b, c\}$ Forbidden Take-Away

The game starts with a pile of n stones. Players alternate removing one or more stones from the pile. Any number can be removed except a, b, or c. The winner is the player who makes the last move.

The game, like nim, is not that interesting with just one pile. To be able to figure out how to play with multiple piles, you need to compute the *nimber* of a pile. The nimber of a game is the *mex* of the nimbers of the positions you can reach in one move. Mex is a function that takes a set of non-negative integers and retuns the minimum non-negative integer *not* in the set. So, for example, mex of the set $\{0, 1, 2, 4, 8\}$ is 3, and the nimber of $\{5, 7, 8\}$ is 0. (By the way, the nimber of a bunch of piles is the `xor` of the nimbers of the individual piles. If it's your turn your goal is to make a move to a position with a nimber of 0. But this is beyond the scope of this problem.)

To give a concrete example, let's compute the nimbers of different pile sizes in $\{2, 3\}$ forbidden take-away.

```
   pile size    nimber
------------------------
       0          0         no moves
       1          1         moves have nimbers {0}
       2          0         moves have nimbers {1}
       3          1         moves have nimbers {0}
       4          2         moves have nimbers {1,0}
       5          3         moves have nimbers {2,1,0}
       6          2         moves have nimbers {3,0,1,0}
```

In this problem you will write a program that takes as input a,b,c, and some pile sizes, and will compute the nimbers of these pile sizes in $\{a, b, c\}$ forbidden take-away.

Input

The input consists of up to ten problem instances. Each one begins with a line containing four space-separated positive integers: a,b,c, and k. This is followed by a line containing k space separted pile sizes (the first of which is the largest), for which the nimber is to be computed. a, b, and c are all at most 200,000, as are the pile sizes. $k \leq 1000$. The input is terminated by a line with four zeros.

Output

For each input instance, your program should produce a line of output containing k space-separated numbers. These are the nimbers of the k pile sizes listed in the input, when playing $\{a, b, c\}$ forbidden take-away.

Example

Input	Output
2 3 3 1 6 5 7 9 3 100 76 10 45 78 102 2 500 400 0 0 0 0	2 50 36 5 230 220

C — Sound

In digital recording, sound is described by a sequence of numbers representing the air pressure, measured at a rapid rate with a fixed time interval between successive measurements. Each value in the sequence is called a sample. An important step in many voice-processing tasks is breaking the recorded sound into chunks of non-silence separated by silence. To avoid accidentally breaking the recording into too few or too many pieces, the silence is often defined as a sequence of m samples where the difference between the lowest and the highest value does not exceed a certain treshold c. Write a program to detect silence in a given recording of n samples according to the given parameter values m and c.

Input

The input begins with three integers: n ($1 \leq n \leq 1,000,000$), the number of samples in the recording; m ($1 \leq m \leq 10,000$), the required length of the silence; and c ($0 \leq c \leq 10,000$), the maximal noise level allowed within silence. The next line contains n integers a_i ($0 \leq a_i \leq 1,000,000$ for $1 \leq i \leq n$), separated by single spaces: the samples in the recording.

Output

You should list all values of i such that

$$\max\{a_i, a_{i+1}, \ldots, a_{i+m-1}\} - \min\{a_i, a_{i+1}, \ldots, a_{i+m-1}\} \leq c$$

The values should be listed in increasing order, each on a separate line. If there is no silence for a test case, output `NONE`.

Example

Input	Output
7 2 0 0 1 1 2 3 2 2	2 6

D — Power Plants

Our hero, Peng, has just woken to a horrible discovery! While sleeping at work, several of the power plants in the state have lost power. Even worse, his boss is on the way to his office, and if he doesn't have the situation fixed in time, he'll be fired. He's called you, desperately asking for help, and you've agreed to help him figure things out. More specifically, Peng is managing N ($1 \leq N \leq 16$) power plants and he knows the cost to restart power plant j using power plant i for every possible pair. Also he knows which plants are down currently. (At least one of plants is alive.) He wants you to figure it out how to restart power plants so that M ($0 \leq M \leq N$) of them are alive with minimum cost.

Input

The input contains multiple test cases and each case starts with a line containing a integer N explained above. Then, the following N lines have information about the cost to restart the power plants:

- Each line contains exactly N integers from 0 to 35 inclusive.
- The ith number in the first line represents the cost to restart power station i using the first power plant. In general, ith number in the jth line represents the cost to restart power station i using the jth power plant.

Then, the next line is a string of N "`Y`" and "`N`" characters, the ith of which is a "`Y`" iff the ith power station is alive. The last line of the test case contains an integer M.

The input is terminated by a line containing 0.

Output

For each test case, your output should start with two numbers C (the minimum cost needed to power at least M plants), and K (the number of orders). The following K lines each contain an order, which is two space-separated numbers a and b (indicating that the ath power station restarted bth power station). The order should be valid. i.e. ath power station should not be down when the order is executed. If there are many possible orders just print any of them.

Example

Input	Output
3 0 2 4 2 0 3 4 3 0 YNN 3 0	5 2 1 2 2 3

E — Elegant Diamond

The king has hired you to make him an elegant diamond. An elegant diamond is a two-dimensional object made out of digits that's symmetric about a horizontal and a vertical axis. For example, the following four shapes are elegant diamonds:

```
  2        8       3      7
 3 3      8 8     2 2
4 1 4      8       3
 3 3
  2
```

These three shapes are diamonds, but are not elegant:

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

These three shapes are not diamonds:

```
  1      2      8   8
 1 1    222       0
         2      00000
```

The king will start by giving you a diamond, which may not be elegant. Your job is to make it elegant by enhancing it, adding digits on to make a bigger diamond. Because you don't want to spend too much money, you want to do it with as little cost as possible.

Definitions

A diamond of size k is $2k-1$ lines of digits, 0-9, separated by single spaces, organized in the following way:

- Line i ($1 \leq i \leq k$) contains $k-i$ spaces, then i digits separated by single spaces.
- Line i ($k < i < 2k$) contains $i-k$ spaces, then $2k-i$ digits separated by single spaces.

An *elegant diamond of size* k is a diamond of size k with the following two symmetry properties:

- Horizontal symmetry: Reflection of the diamond about the center (tallest) column results in the same diamond.
- Vertical symmetry: Reflection of the diamond about the center (longest) row results in the same diamond.

A diamond of size k can be enhanced by adding digits to it. The result of enhancing a diamond of size k has the following properties:

- The result is a diamond of size $\geq k$.
- An identical copy of the original diamond occurs somewhere within the enhanced diamond.

The cost of an enhancement is the number of digits in the enhancement minus the number of digits in the original diamond (i.e. the number of digits that are added in the enhancing process.)

Input

The first line of the input gives the number of test cases, T ($1 \leq T \leq 100$). T test cases follow. Each test case consists of a single integer k in a line on its own, followed by a diamond of size k. ($1 \leq k \leq 10$).

Output

For each test case, output one line containing `Case #X: Y`, where `X` is the case number (starting from 1) and `Y` is the minimum cost required to enhance the given diamond into an elegant diamond. If the diamond is already elegant, `Y` = 0.

Example

Input	Output

4
1
0
2
 1
2 2
 1
2
 1
1 2
 1
3
  1
 6 3
9 5 5
 6 3
  1
	Case #1: 0
	Case #2: 0
	Case #3: 5
	Case #4: 7

Explanation

There are four cases. The first two cases start as elegant diamonds of size 1 and 2, respectively, and don't need to be enhanced; so the cost is 0. The third case can be enhanced to look like:

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

There are several possible enhancements, but this is one with the lowest possible cost, which is 5. In the fourth case we can enhance the diamond into the following elegant diamond:

```
   9
  1 1
 6 3 6
9 5 5 9
 6 3 6
  1 1
   9
```

...for a cost of 7.

F — Colored Cubes

There are several colored cubes. All of them are of the same size but they may be colored differently. Each face of these cubes has a single color. Colors of distinct faces of a cube may or may not be the same. Two cubes are said to be identically colored if some suitable rotations of one of the cubes give identical looks to both of the cubes. For example, two cubes shown in Figure 1 are identically colored. A set of cubes is said to be identically colored if every pair of them are identically colored.

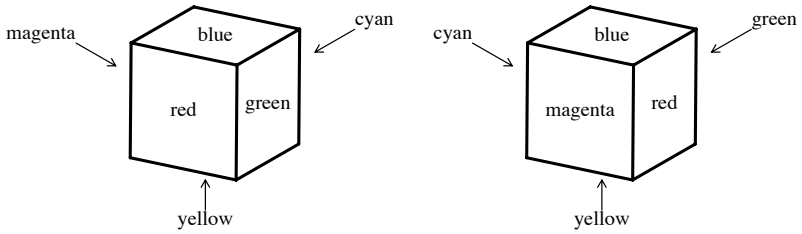


Figure 1. Identically colored cubes

A cube and its mirror image are not necessarily identically colored. For example, two cubes shown in Figure 2 are not identically colored.

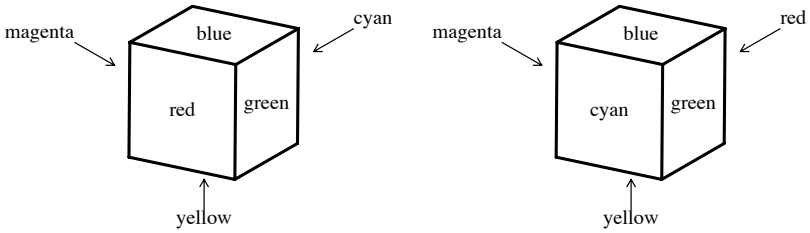


Figure 2. Not identically colored cubes

You can make a given set of cubes identically colored by repainting some of the faces, whatever colors the faces may have. In Figure 3, repainting four faces makes the three cubes identically colored and repainting fewer faces will never do. Your task is to write a program to calculate the minimum number of faces that need to be repainted for a given set of cubes to become identically colored.

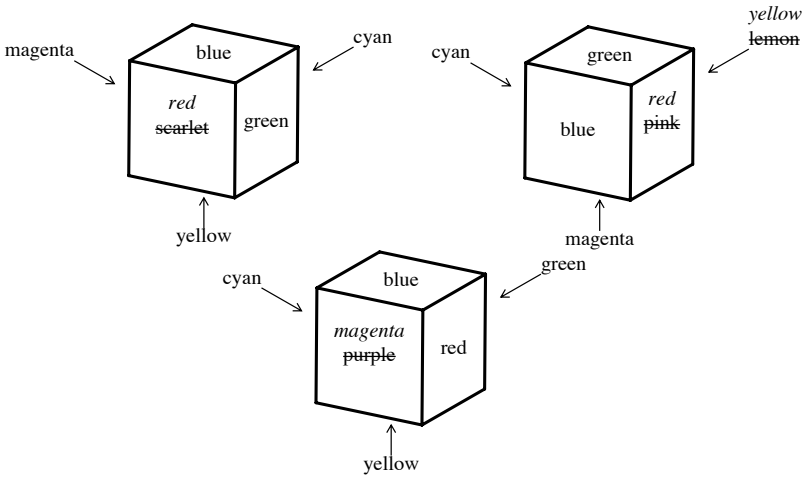


Figure 3. An example of recoloring

Input

The input is a sequence of datasets. A dataset consists of a header and a body appearing in this order. A header is a line containing one positive integer n and the body following it consists of n lines. You can assume that $1 \leq n \leq 4$. Each line in a body contains six color names separated by a space. A color name consists of a word or words connected with a hyphen (-). A word consists of zero or more lowercase letters. You can assume that a color name is at most 24-characters long including hyphens. A dataset corresponds to a set of colored cubes. The integer n corresponds to the number of cubes. Each line of the body corresponds to a cube and describes the colors of its faces. Color names in a line is ordered in accordance with the numbering of faces shown in Figure 4. A line of the form:

color1 color2 color3 color4 color5 color6

corresponds to a cube colored as shown in Figure 5. The end of the input is indicated by a line containing a single zero. It is not a dataset nor a part of a dataset.

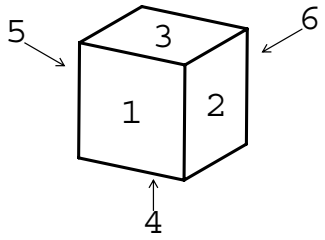


Figure 4. Numbering of faces

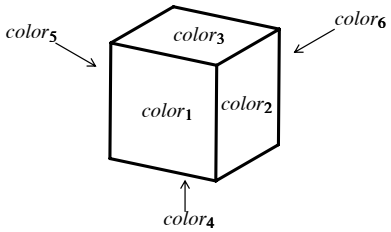


Figure 5. Coloring

Output

For each dataset, output a line containing the minimum number of faces that need to be repainted to make the set of cubes identically colored.

Example

Input	Output
3 scarlet green blue yellow magenta cyan blue pink green magenta cyan lemon purple red blue yellow cyan green 2 red green blue yellow magenta cyan cyan green blue yellow magenta red 2 red green gray gray magenta cyan cyan green gray gray magenta red 2 red green blue yellow magenta cyan magenta red blue yellow cyan green 3 red green blue yellow magenta cyan cyan green blue yellow magenta red magenta red blue yellow cyan green 3 blue green green green green blue green blue blue green green green green green green green green sea-green 3 red yellow red yellow red yellow red red yellow yellow red yellow red red red red red red 4 violet violet salmon salmon salmon salmon violet salmon salmon salmon salmon violet violet violet salmon salmon violet violet violet violet violet violet salmon salmon 1 red green blue yellow magenta cyan 4 magenta pink red scarlet vermilion wine-red aquamarine blue cyan indigo sky-blue turquoise-blue blond cream chrome-yellow lemon olive yellow chrome-green emerald-green green olive vilidian sky-blue 0	4 2 0 0 2 3 4 4 0 16

G — Watch Tower

In a one-dimensional mountainous landscape you want to build a watchtower. This watchtower must fulfill one condition: it must be possible to oversee the whole landscape from it. The landscape is described by pairs of integer *positions* and *heights* and the intermediate heights are obtained by interpolation.

You are allowed to build the watchtower on the landscape wherever you like. To save expenses, you want to build it at a position such that the necessary height of the watchtower needed to oversee the whole landscape is minimized. Your goal is to compute this minimal height.

Input

The data contains multiple test cases and each case starts with N ($2 \leq N \leq 50$) representing the number of pairs. The following N lines have an integer pair position_i height_i. The positions are sorted in increasing order (and no two positions are equal). Furthermore $0 \leq$ positions, heights $\leq 1,000,000$. The input is terminated by a line containing 0.

Output

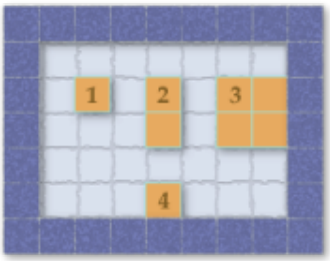
Print out the minimal height described above. For the precision issue, relative or absolute error less than 1e-6 is considered to be right.

Example

Input	Output
6 1 1 2 2 4 2 5 4 6 2 7 1 4 10 0 20 10 49 10 59 0 0	1.0 14.5

H — EZ-Sokoban

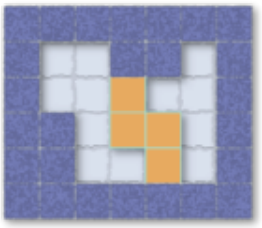
Sokoban is a famous Japanese puzzle game. Sokoban is Japanese for "warehouse keeper". In this game, your goal is to push boxes to their designated locations in the warehouse. To push a box, the area behind the box and in front of the box must be empty. This is because you stand behind the box when pushing and you can push only one box at a time. You cannot push a box out of the board and you cannot stand outside the board when pushing a box. For example, in this picture: Box 1 can be pushed in any of the



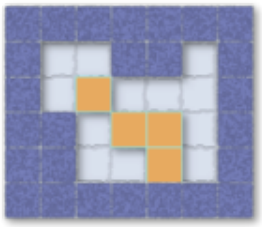
four directions because the four spaces adjacent to it are empty. Box 2 can only be pushed east or west; it cannot be pushed north or south because the space to its south is not empty. Box 3 cannot be pushed in any direction. Box 4 can only be pushed east or west because there is a wall south of it.

Sokoban was proved to be a P-Space complete problem, but we deal with an easier variation here. In our variation of Sokoban, boxes have strong magnets inside and they have to stick together almost all the time. Under "stable" conditions, all boxes should be connected, edge to edge. This means that from any box we can get to any other box by going through boxes that share an edge. If you push a box and boxes are no longer connected, you are in "dangerous mode". In dangerous mode, the next push must make the boxes connected again.

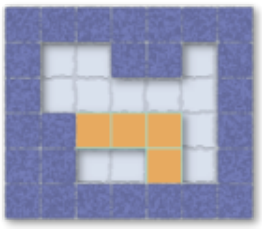
For example, in this picture: The situation is stable, since all 4 boxes are connected, edge to edge. Let's



assume that you decided to push the northmost box west: For example, in this picture: Now, we are in



dangerous mode since the northmost box is not connected to any other boxes. The next push must return us to a stable position. For example, we can push that northmost box south:



Making the boxes stable again.

A Sokoban puzzle consists of a board, initial configuration of the boxes and the final configuration (where we want the boxes to be at the end). Given an EZ-Sokoban puzzle, find a solution that makes the minimum number of box moves, or decide that it can't be solved. The final and initial configurations will not be in dangerous mode. To simplify things, we will assume that you, the warehouse keeper, can jump at any time to any empty spot on the board.

Input

The first line in the input file contains the number of cases, T ($1 \leq T \leq 50$). Each case consists of several lines. The first line contains R and C ($1 \leq R, C \leq 12$), the number of rows and columns of the board, separated by one space. This is followed by R lines. Each line contains C characters describing the board:

- `.` is an empty spot
- `#` is a wall
- `x` is a goal (where a box should be at the end)
- `o` is a box
- `w` is both a box and a goal

The number of boxes (between 1 and 5 inclusive) will be equal to the number of goals.

Output

For each test case, output `Case #X: K` where `X` is the test case number, starting from 1, and `K` is the minimum number of box moves that are needed to solve the puzzle or `-1` if it cannot be solved.

Example

Input	Output
6 5 4 .... #..# #xx# #oo# #..# 7 7 .###### .x....# .x....# ..#oo.# ..#...# .###### .###### 7 7 ####### #x....# #xx.o.# #..oo.# #.#...# ####### ####### 4 10 ########## #.x...o..# #.x...o..# ########## 7 7 ####### #x....# #x..o.# #x#oo.# #.#...# ####### ####### 3 4 .#x. .oww	Case #1: 2 Case #2: 8 Case #3: 12 Case #4: 8 Case #5: -1 Case #6: 2