Problemas: Representação de tipos e estrutura de dados
Data final de entrega dos códigos: **29/04/22**
Entrega via e-mail (arquivo zipado com todos códigos *.c ou *.cpp): rafael.queiroz@ufop.edu.br
Código em C e documentado

Discentes	**Problemas**
BRIHAN DE JESUS	1.6.4 - LCD Display e 3.8.3 - Common Permutation
CARLOS EDUARDO	1.6.5 - Graphical Editor e 3.8.5 - Automated Judge Script
DANIEL MONTEIRO	1.6.7 - Check the Check e 3.8.7 - Doublets
JOAO GABRIEL	1.6.8 - Australian Voting e 3.8.8 - Fmt
JOAO HENRIQUE	2.8.2 - Poker Hands e 4.6.2 - Stacks of Flapjacks
MATEUS PEVIDOR	2.8.3 – Hartals e 4.6.4 - Longest Nap
PEDRO LUIS	2.8.6 - Erdös Numbers e 4.6.7 - ShellSort
VINICIUS GABRIEL	2.8.7 - Contest Scoreboard e 4.6.8 - Football (aka Soccer)

Sugestão: antes de implementar computacionalmente, busque descrever os passos (algoritmo) da solução. Entenda bem o problema que irá resolver. Busque mais informações sobre o problema.

1.6.4 LCD Display

PC/UVa IDs: 110104/706, **Popularity:** A, **Success rate:** average **Level:** 1

A friend of yours has just bought a new computer. Before this, the most powerful machine he ever used was a pocket calculator. He is a little disappointed because he liked the LCD display of his calculator more than the screen on his new computer! To make him happy, write a program that prints numbers in LCD display style.

Input

The input file contains several lines, one for each number to be displayed. Each line contains integers s and n, where n is the number to be displayed ($0 \leq n \leq 99,999,999$) and s is the size in which it shall be displayed ($1 \leq s \leq 10$). The input will be terminated by a line containing two zeros, which should not be processed.

Output

Print the numbers specified in the input file in an LCD display-style using s "-" signs for the horizontal segments and s "|" signs for the vertical ones. Each digit occupies exactly $s + 2$ columns and $2s + 3$ rows. Be sure to fill all the white space occupied by the digits with blanks, including the last digit. There must be exactly one column of blanks between two digits.

Output a blank line after each number. You will find an example of each digit in the sample output below.

Sample Input

```
2 12345
3 67890
0 0
```

Sample Output

```
    --   --         --
  |    |    | |  | |
  |    |    | |  | |
     --   --   --   --
  | |       |    |    |
  | |       |    |    |
     --   --         --

 ---   ---   ---   ---   ---
|         | |   | |   | |   |
|         | |   | |   | |   |
|         | |   | |   | |   |
 ---         ---   ---
|   |     | |   |     | |   |
|   |     | |   |     | |   |
|   |     | |   |     | |   |
 ---         ---   ---   ---
```

1.6.5 Graphical Editor

PC/UVa IDs: 110105/10267, **Popularity:** B, **Success rate:** low **Level:** 1

Graphical editors such as Photoshop allow us to alter bit-mapped images in the same way that text editors allow us to modify documents. Images are represented as an $M \times N$ array of pixels, where each pixel has a given color.

Your task is to write a program which simulates a simple interactive graphical editor.

Input

The input consists of a sequence of editor commands, one per line. Each command is represented by one capital letter placed as the first character of the line. If the command needs parameters, they will be given on the same line separated by spaces.

Pixel coordinates are represented by two integers, a column number between $1 \ldots M$ and a row number between $1 \ldots N$, where $1 \leq M, N \leq 250$. The origin sits in the upper-left corner of the table. Colors are specified by capital letters.

The editor accepts the following commands:

I M N	Create a new $M \times N$ image with all pixels initially colored white (`O`).
C	Clear the table by setting all pixels white (`O`). The size remains unchanged.
L X Y C	Colors the pixel (X, Y) in color (`C`).
V X Y1 Y2 C	Draw a vertical segment of color (`C`) in column X, between the rows $Y1$ and $Y2$ inclusive.
H X1 X2 Y C	Draw a horizontal segment of color (`C`) in the row Y, between the columns $X1$ and $X2$ inclusive.
K X1 Y1 X2 Y2 C	Draw a filled rectangle of color `C`, where $(X1, Y1)$ is the upper-left and $(X2, Y2)$ the lower right corner.
F X Y C	Fill the region R with the color `C`, where R is defined as follows. Pixel (X, Y) belongs to R. Any other pixel which is the same color as pixel (X, Y) and shares a common side with any pixel in R also belongs to this region.
S Name	Write the file name in MSDOS 8.3 format followed by the contents of the current image.
X	Terminate the session.

Output

On every command `S NAME`, print the filename $NAME$ and contents of the current image. Each row is represented by the color contents of each pixel. See the sample output.

Ignore the entire line of any command defined by a character other than `I, C, L, V, H, K, F, S,` or `X`, and pass on to the next command. In case of other errors, the program behavior is unpredictable.

Sample Input

```
I 5 6
L 2 3 A
S one.bmp
G 2 3 J
F 3 3 J
V 2 3 4 W
H 3 4 2 Z
S two.bmp
X
```

Sample Output

```
one.bmp
OOOOO
OOOOO
OAOOO
OOOOO
OOOOO
OOOOO
two.bmp
JJJJJ
JJZZJ
JWJJJ
JWJJJ
JJJJJ
JJJJJ
```

1.6.7 Check the Check

PC/UVa IDs: 110107/10196, **Popularity:** B, **Success rate:** average **Level:** 1

Your task is to write a program that reads a chessboard configuration and identifies whether a king is under attack (in check). A king is in check if it is on square which can be taken by the opponent on his next move.

White pieces will be represented by uppercase letters, and black pieces by lowercase letters. The white side will always be on the bottom of the board, with the black side always on the top.

For those unfamiliar with chess, here are the movements of each piece:

Pawn (`p` or `P`): can only move straight ahead, one square at a time. However, it takes pieces diagonally, and that is what concerns you in this problem.

Knight (`n` or `N`) : has an L-shaped movement shown below. It is the only piece that can jump over other pieces.

Bishop (`b` or `B`) : can move any number of squares diagonally, either forward or backward.

Rook (`r` or `R`) : can move any number of squares vertically or horizontally, either forward or backward.

Queen (`q` or `Q`) : can move any number of squares in any direction (diagonally, horizontally, or vertically) either forward or backward.

King (`k` or `K`) : can move one square at a time in any direction (diagonally, horizontally, or vertically) either forward or backward.

Movement examples are shown below, where "`*`" indicates the positions where the piece can capture another piece:

```
Pawn        Rook        Bishop      Queen       King        Knight
........    ...*....    .......*    ...*...*    ........    ........
........    ...*....    *.....*.    *..*..*.    ........    ........
........    ...*....    .*...*..    .*.*.*..    ........    ..*.*...
........    ...*....    ..*.*...    ..***...    ..***...    .*...*..
...p....    ***r****    ...b....    ***q****    ..*k*...    ...n....
..*.*...    ...*....    ..*.*...    ..***...    ..***...    .*...*..
........    ...*....    .*...*..    .*.*.*..    ........    ..*.*...
........    ...*....    *.....*.    *..*..*.    ........    ........
```

Remember that the knight is the only piece that can jump over other pieces. The pawn movement will depend on its side. If it is a black pawn, it can only move one square diagonally down the board. If it is a white pawn, it can only move one square diagonally up the board. The example above is a black pawn, described by a lowercase "`p`". We use "*move*" to indicate the squares where the pawn can capture another piece.

Input

There will be an arbitrary number of board configurations in the input, each consisting of eight lines of eight characters each. A "." denotes an empty square, while upper- and lowercase letters represent the pieces as defined above. There will be no invalid characters and no configurations where both kings are in check. You must read until you find an empty board consisting only of "." characters, which should not be processed. There will be an empty line between each pair of board configurations. All boards, except for the empty one, will contain exactly one white king and one black king.

Output

For each board configuration read you must output one of the following answers:
`Game #`d`: white king is in check.`
`Game #`d`: black king is in check.`
`Game #`d`: no king is in check.`
where d stands for the game number starting from 1.

Sample Input

```
..k.....
PPP·PPPP
........
.R...B..
........
........
PPPPPPPP
K.......

rnbqk.nr
PPP..PPP
....p...
...p....
.bPP....
.....N..
PP..PPPP
RNBQKB.R

........
........
........
........
........
........
........
........
```

Sample Output

```
Game #1: black king is in check.
Game #2: white king is in check.
```

1.6.8 Australian Voting

PC/UVa IDs: 110108/10142, **Popularity:** B, **Success rate:** low **Level:** 1

Australian ballots require that voters rank all the candidates in order of choice. Initially only the first choices are counted, and if one candidate receives more than 50% of the vote then that candidate is elected. However, if no candidate receives more than 50%, all candidates tied for the lowest number of votes are eliminated. Ballots ranking these candidates first are recounted in favor of their highest-ranked non-eliminated candidate. This process of eliminating the weakest candidates and counting their ballots in favor of the preferred non-eliminated candidate continues until one candidate receives more than 50% of the vote, or until all remaining candidates are tied.

Input

The input begins with a single positive integer on a line by itself indicating the number of cases following, each as described below. This line is followed by a blank line. There is also a blank line between two consecutive inputs.

The first line of each case is an integer $n \leq 20$ indicating the number of candidates. The next n lines consist of the names of the candidates in order, each up to 80 characters in length and containing any printable characters. Up to 1,000 lines follow, each containing the contents of a ballot. Each ballot contains the numbers from 1 to n in some order. The first number indicates the candidate of first choice; the second number indicates candidate of second choice, and so on.

Output

The output of each test case consists of either a single line containing the name of the winner or several lines containing the names of all candidates who are tied. The output of each two consecutive cases are separated by a blank line.

Sample Input

```
1

3
John Doe
Jane Smith
Jane Austen
1 2 3
2 1 3
2 3 1
1 2 3
3 1 2
```

Sample Output

```
John Doe
```

2.8.2 Poker Hands

PC/UVa IDs: 110202/10315, **Popularity:** C, **Success rate:** average **Level:** 2

A poker deck contains 52 cards. Each card has a suit of either clubs, diamonds, hearts, or spades (denoted `C`, `D`, `H`, `S` in the input data). Each card also has a value of either 2 through 10, jack, queen, king, or ace (denoted `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`, `T`, `J`, `Q`, `K`, `A`). For scoring purposes card values are ordered as above, with 2 having the lowest and ace the highest value. The suit has no impact on value.

A poker hand consists of five cards dealt from the deck. Poker hands are ranked by the following partial order from lowest to highest.

High Card. Hands which do not fit any higher category are ranked by the value of their highest card. If the highest cards have the same value, the hands are ranked by the next highest, and so on.

Pair. Two of the five cards in the hand have the same value. Hands which both contain a pair are ranked by the value of the cards forming the pair. If these values are the same, the hands are ranked by the values of the cards not forming the pair, in decreasing order.

Two Pairs. The hand contains two different pairs. Hands which both contain two pairs are ranked by the value of their highest pair. Hands with the same highest pair are ranked by the value of their other pair. If these values are the same the hands are ranked by the value of the remaining card.

Three of a Kind. Three of the cards in the hand have the same value. Hands which both contain three of a kind are ranked by the value of the three cards.

Straight. Hand contains five cards with consecutive values. Hands which both contain a straight are ranked by their highest card.

Flush. Hand contains five cards of the same suit. Hands which are both flushes are ranked using the rules for High Card.

Full House. Three cards of the same value, with the remaining two cards forming a pair. Ranked by the value of the three cards.

Four of a Kind. Four cards with the same value. Ranked by the value of the four cards.

Straight Flush. Five cards of the same suit with consecutive values. Ranked by the highest card in the hand.

Your job is to compare several pairs of poker hands and to indicate which, if either, has a higher rank.

Input

The input file contains several lines, each containing the designation of ten cards: the first five cards are the hand for the player named "*Black*" and the next five cards are the hand for the player named "*White*".

Output

For each line of input, print a line containing one of the following:

```
Black wins.
White wins.
Tie.
```

Sample Input

```
2H 3D 5S 9C KD 2C 3H 4S 8C AH
2H 4S 4C 2D 4H 2S 8S AS QS 3S
2H 3D 5S 9C KD 2C 3H 4S 8C KH
2H 3D 5S 9C KD 2D 3H 5C 9S KH
```

Sample Output

```
White wins.
Black wins.
Black wins.
Tie.
```

2.8.3 Hartals

PC/UVa IDs: 110203/10050, **Popularity:** B, **Success rate:** high **Level:** 2

Political parties in Bangladesh show their muscle by calling for regular *hartals* (strikes), which cause considerable economic damage. For our purposes, each party may be characterized by a positive integer h called the *hartal parameter* that denotes the average number of days between two successive strikes called by the given party.

Consider three political parties. Assume $h_1 = 3$, $h_2 = 4$, and $h_3 = 8$, where h_i is the hartal parameter for party i. We can simulate the behavior of these three parties for $N = 14$ days. We always start the simulation on a Sunday. There are no hartals on either Fridays or Saturdays.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Days	Su	Mo	Tu	We	Th	Fr	Sa	Su	Mo	Tu	We	Th	Fr	Sa
Party 1			x			x			x			x		
Party 2				x				x				x		
Party 3								x						
Hartals			1	2				3	4			5		

There will be exactly five hartals (on days 3, 4, 8, 9, and 12) over the 14 days. There is no hartal on day 6 since it falls on Friday. Hence we lose five working days in two weeks.

Given the hartal parameters for several political parties and the value of N, determine the number of working days lost in those N days.

Input

The first line of the input consists of a single integer T giving the number of test cases to follow. The first line of each test case contains an integer N ($7 \leq N \leq 3,650$), giving the number of days over which the simulation must be run. The next line contains another integer P ($1 \leq P \leq 100$) representing the number of political parties. The ith of the next P lines contains a positive integer h_i (which will never be a multiple of 7) giving the *hartal parameter* for party i ($1 \leq i \leq P$).

Output

For each test case, output the number of working days lost on a separate line.

Sample Input

```
2
14
3
3
4
```

```
8
100
4
12
15
25
40
```

Sample Output

```
5
15
```

2.8.6 Erdös Numbers

PC/UVa IDs: 110206/10044, **Popularity:** B, **Success rate:** low **Level:** 2

The Hungarian Paul Erdös (1913–1996) was one of the most famous mathematicians of the 20th century. Every mathematician having the honor of being a co-author of Erdös is well respected.

Unfortunately, not everybody got the chance to write a paper with Erdös, so the best they could do was publish a paper with somebody who had published a scientific paper with Erdös. This gave rise to the so-called *Erdös numbers*. An author who has jointly published with Erdös had Erdös number 1. An author who had not published with Erdös but with somebody with Erdös number 1 obtained Erdös number 2, and so on.

Your task is to write a program which computes Erdös numbers for a given set of papers and scientists.

Input

The first line of the input contains the number of scenarios. Each scenario consists of a paper database and a list of names. It begins with the line P N, where P and N are natural numbers. Following this line is the paper database, with P lines each containing the description of one paper specified in the following way:

```
Smith, M.N., Martin, G., Erdos, P.: Newtonian forms of prime factors
```

Note that umlauts, like "ö," are simply written as "o". After the P papers follow N lines with names. Such a name line has the following format:

```
Martin, G.
```

Output

For every scenario you are to print a line containing a string "`Scenario` i" (where i is the number of the scenario) and the author names together with their Erdös number of all authors in the list of names. The authors should appear in the same order as they appear in the list of names. The Erdös number is based on the papers in the paper database of this scenario. Authors which do not have any relation to Erdös via the papers in the database have Erdös number "`infinity`."

Sample Input

```
1
4 3
Smith, M.N., Martin, G., Erdos, P.: Newtonian forms of prime factors
Erdos, P., Reisig, W.: Stuttering in petri nets
Smith, M.N., Chen, X.: First order derivates in structured programming
Jablonski, T., Hsueh, Z.: Selfstabilizing data structures
```

```
Smith, M.N.
Hsueh, Z.
Chen, X.
```

Sample Output

```
Scenario 1
Smith, M.N. 1
Hsueh, Z. infinity
Chen, X. 2
```

2.8.7 Contest Scoreboard

PC/UVa IDs: 110207/10258, **Popularity:** B, **Success rate:** average **Level:** 1

Want to compete in the ACM ICPC? Then you had better know how to keep score! Contestants are ranked first by the number of problems solved (the more the better), then by decreasing amounts of penalty time. If two or more contestants are tied in both problems solved and penalty time, they are displayed in order of increasing team numbers.

A problem is considered solved by a contestant if any of the submissions for that problem was judged correct. Penalty time is computed as the number of minutes it took until the first correct submission for a problem was received, plus 20 minutes for each incorrect submission prior to the correct solution. Unsolved problems incur no time penalties.

Input

The input begins with a single positive integer on a line by itself indicating the number of cases, each described as below. This line is followed by a blank line. There is also a blank line between two consecutive inputs.

The input consists of a snapshot of the judging queue, containing entries from some or all of contestants 1 through 100 solving problems 1 through 9. Each line of input consists of three numbers and a letter in the format *contestant problem time L*, where *L* can be `C`, `I`, `R`, `U`, or `E`. These stand for `C`orrect, `I`ncorrect, clarification `R`equest, `U`njudged, and `E`rroneous submission. The last three cases do not affect scoring.

The lines of input appear in the order in which the submissions were received.

Output

The output for each test case will consist of a scoreboard, sorted by the criteria described above. Each line of output will contain a contestant number, the number of problems solved by the contestant and the total time penalty accumulated by the contestant. Since not all contestants are actually participating, only display those contestants who have made a submission.

The output of two consecutive cases will be separated by a blank line.

Sample Input

```
1

1 2 10 I
3 1 11 C
1 2 19 R
1 2 21 C
1 1 25 C
```

Sample Output

```
1 2 66
3 1 11
```

3.8.3 Common Permutation

PC/UVa IDs: 110303/10252, **Popularity:** A, **Success rate:** average **Level:** 1

Given two strings a and b, print the longest string x of letters such that there is a permutation of x that is a subsequence of a and there is a permutation of x that is a subsequence of b.

Input

The input file contains several cases, each case consisting of two consecutive lines. This means that lines 1 and 2 are a test case, lines 3 and 4 are another test case, and so on. Each line contains one string of lowercase characters, with first line of a pair denoting a and the second denoting b. Each string consists of at most 1,000 characters.

Output

For each set of input, output a line containing x. If several x satisfy the criteria above, choose the first one in alphabetical order.

Sample Input

```
pretty
women
walking
down
the
street
```

Sample Output

```
e
nw
et
```

3.8.5 Automated Judge Script

PC/UVa IDs: 110305/10188, **Popularity:** B, **Success rate:** average **Level:** 1

Human programming contest judges are known to be very picky. To eliminate the need for them, write an automated judge script to judge submitted solution runs.

Your program should take a file containing the correct output as well as the output of submitted program and answer either `Accepted`, `Presentation Error`, or `Wrong Answer`, defined as follows:

Accepted: You are to report "`Accepted`" if the team's output matches the standard solution exactly. *All* characters must match and must occur in the same order.

Presentation Error: Give a "`Presentation Error`" if all *numeric* characters match in the same order, but there is at least one non-matching non-numeric character. For example, "`15 0`" and "`150`" would receive "`Presentation Error`", whereas "`15 0`" and "`1 0`" would receive "`Wrong Answer`," described below.

Wrong Answer: If the team output cannot be classified as above, then you have no alternative but to score the program a '`Wrong Answer`'.

Input

The input will consist of an arbitrary number of input sets. Each input set begins with a line containing a positive integer $n < 100$, which describes the number of lines of the correct solution. The next n lines contain the correct solution. Then comes a positive integer $m < 100$, alone on its line, which describes the number of lines of the team's submitted output. The next m lines contain this output. The input is terminated by a value of $n = 0$, which should not be processed.

No line will have more than 100 characters.

Output

For each set, output one of the following:

`Run #`x`: Accepted`
`Run #`x`: Presentation Error`
`Run #`x`: Wrong Answer`

where x stands for the number of the input set (starting from 1).

Sample Input

```
2
The answer is: 10
The answer is: 5
2
The answer is: 10
```

```
The answer is: 5
2
The answer is: 10
The answer is: 5
2
The answer is: 10
The answer is: 15
2
The answer is: 10
The answer is:  5
2
The answer is: 10
The answer is: 5
3
Input Set #1: YES
Input Set #2: NO
Input Set #3: NO
3
Input Set #0: YES
Input Set #1: NO
Input Set #2: NO
1
1 0 1 0
1
1010
1
The judges are mean!
1
The judges are good!
0
```

Sample Output

```
Run #1: Accepted
Run #2: Wrong Answer
Run #3: Presentation Error
Run #4: Wrong Answer
Run #5: Presentation Error
Run #6: Presentation Error
```

3.8.7 Doublets

PC/UVa IDs: 110307/10150, **Popularity:** C, **Success rate:** average **Level:** 3

A *doublet* is a pair of words that differ in exactly one letter; for example, "booster" and "rooster" or "rooster" and "roaster" or "roaster" and "roasted".

You are given a dictionary of up to 25,143 lowercase words, not exceeding 16 letters each. You are then given a number of pairs of words. For each pair of words, find the shortest sequence of words that begins with the first word and ends with the second, such that each pair of adjacent words is a doublet. For example, if you were given the input pair "booster" and "roasted", a possible solution would be ("booster," "rooster," "roaster," "roasted"), provided that these words are all in the dictionary.

Input

The input file contains the dictionary followed by a number of word pairs. The dictionary consists of a number of words, one per line, and is terminated by an empty line. The pairs of input words follow; each pair of words occurs on a line separated by a space.

Output

For each input pair, print a set of lines starting with the first word and ending with the last. Each pair of adjacent lines must be a doublet.

If there are several minimal solutions, any one will do. If there is no solution, print a line: "`No solution.`" Leave a blank line between cases.

Sample Input

```
booster
rooster
roaster
coasted
roasted
coastal
postal

booster roasted
coastal postal
```

Sample Output

```
booster
rooster
roaster
roasted

No solution.
```

3.8.8 Fmt

PC/UVa IDs: 110308/848, **Popularity:** C, **Success rate:** low **Level:** 2

The UNIX program *fmt* reads lines of text, combining and breaking them so as to create an output file with lines as close to 72 characters long as possible without exceeding this limit. The rules for combining and breaking lines are as follows:

- A new line may be started anywhere there is a space in the input. When a new line is started, blanks at the end of the previous line and at the beginning of the new line are eliminated.
- A line break in the input may be eliminated in the output unless (1) it is at the end of a blank or empty line, or (2) it is followed by a space or another line break. When a line break is eliminated, it is replaced by a space.
- Spaces must be removed from the end of each output line.
- Any input word containing more than 72 characters must appear on an output line by itself.

You may assume that the input text does not contain any tabbing characters.

Sample Input

```
   Unix fmt

The unix fmt program reads lines of text, combining
and breaking lines so as to create an
output file with lines as close to without exceeding
72 characters long as possible.  The rules for combining and breaking
lines are as follows.

   1.  A new line may be started anywhere there is a space in the input.
If a new line is started, there will be no trailing blanks at the
end of the previous line or at the beginning of the new line.

   2.  A line break in the input may be eliminated in the output, provided
it is not followed by a space or another line break.  If a line
break is eliminated, it is replaced by a space.
```

Sample Output

```
   Unix fmt

The unix fmt program reads lines of text, combining and breaking lines
so as to create an output file with lines as close to without exceeding
72 characters long as possible.  The rules for combining and breaking
```

lines are as follows.

1. A new line may be started anywhere there is a space in the input. If a new line is started, there will be no trailing blanks at the end of the previous line or at the beginning of the new line.

2. A line break in the input may be eliminated in the output, provided it is not followed by a space or another line break. If a line break is eliminated, it is replaced by a space.

4.6.2 Stacks of Flapjacks

PC/UVa IDs: 110402/120, **Popularity:** B, **Success rate:** high **Level:** 2

Cooking the perfect stack of pancakes on a grill is a tricky business, because no matter how hard you try all pancakes in any stack have different diameters. For neatness's sake, however, you can sort the stack by size such that each pancake is smaller than all the pancakes below it. The size of a pancake is given by its diameter.

Sorting a stack is done by a sequence of pancake "flips." A flip consists of inserting a spatula between two pancakes in a stack and flipping (reversing) *all* the pancakes on the spatula (reversing the sub-stack). A flip is specified by giving the position of the pancake on the bottom of the sub-stack to be flipped relative to the entire stack. The bottom pancake has position 1, while the top pancake on a stack of n pancakes has position n.

A stack is specified by giving the diameter of each pancake in the stack in the order in which the pancakes appear. For example, consider the three stacks of pancakes below in which pancake 8 is the top-most pancake of the left stack:

```
8       7       2
4       6       5
6       4       8
7       8       4
5       5       6
2       2       7
```

The stack on the left can be transformed to the stack in the middle via *flip(3)*. The middle stack can be transformed into the right stack via the command *flip(1)*.

Input

The input consists of a sequence of stacks of pancakes. Each stack will consist of between 1 and 30 pancakes and each pancake will have an integer diameter between 1 and 100. The input is terminated by end-of-file. Each stack is given as a single line of input with the top pancake on a stack appearing first on a line, the bottom pancake appearing last, and all pancakes separated by a space.

Output

For each stack of pancakes, your program should echo the original stack on one line, followed by a sequence of flips that results in sorting the stack of pancakes so that the largest pancake is on the bottom and the smallest on top. The sequence of flips for each stack should be terminated by a 0, indicating no more flips necessary. Once a stack is sorted, no more flips should be made.

Sample Input

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

Sample Output

```
1 2 3 4 5
0
5 4 3 2 1
1 0
5 1 2 3 4
1 2 0
```

4.6.4 Longest Nap

PC/UVa IDs: 110404/10191, **Popularity:** B, **Success rate:** average **Level:** 1

Professors lead very busy lives with full schedules of work and appointments. Professor P likes to nap during the day, but his schedule is so busy that he doesn't have many chances to do so.

He *really* wants to take one nap every day, however. Naturally, he wants to take the longest nap possible given his schedule. Write a program to help him with the task.

Input

The input consists of an arbitrary number of test cases, where each test case represents one day.

The first line of each case contains a positive integer $s \leq 100$, representing the number of scheduled appointments for that day. The next s lines contain the appointments in the format *time1 time2 appointment*, where *time1* represents the time which the appointment starts and *time2* the time it ends. All times will be in the `hh:mm` format; the ending time will always be strictly after the starting time, and separated by a single space.

All times will be greater than or equal to 10:00 and less than or equal to 18:00. Thus your response must be in this interval as well; i.e., no nap can start before 10:00 and last after 18:00.

The appointment can be any sequence of characters, but will always be on the same line. You can assume that no line is be longer than 255 characters, that $10 \leq hh \leq 18$ and that $0 \leq mm < 60$. You *cannot* assume, however, that the input will be in any specific order, and must read the input until you reach the end of file.

Output

For each test case, you must print the following line:

`Day #`d`: the longest nap starts at `$hh : mm$` and will last for [`H` hours and] `M` minutes.`

where d stands for the number of the test case (starting from 1) and $hh : mm$ is the time when the nap can start. To display the length of the nap, follow these rules:

1. If the total time X is less than 60 minutes, just print "X `minutes`."
2. If the total duration X is at least 60 minutes, print "H `hours and` M `minutes`," where

$$H = X \div 60 \quad \text{(integer division, of course) and} \quad M = X \bmod 60.$$

You don't have to worry about correct pluralization; i.e., you must print "`1 minutes`" or "`1 hours`" if that is the case.

The duration of the nap is calculated by the difference between the ending time and the beginning time. That is, if an appointment ends at 14:00 and the next one starts at 14:47, then you have 14:47 – 14:00 = 47 minutes of possible napping.

If there is more than one longest nap with the same duration, print the earliest one. You can assume the professor won't be busy all day, so there is always time for at least one possible nap.

Sample Input

```
4
10:00 12:00 Lectures
12:00 13:00 Lunch, like always.
13:00 15:00 Boring lectures...
15:30 17:45 Reading
4
10:00 12:00 Lectures
12:00 13:00 Lunch, just lunch.
13:00 15:00 Lectures, lectures... oh, no!
16:45 17:45 Reading (to be or not to be?)
4
10:00 12:00 Lectures, as everyday.
12:00 13:00 Lunch, again!!!
13:00 15:00 Lectures, more lectures!
15:30 17:15 Reading (I love reading, but should I schedule it?)
1
12:00 13:00 I love lunch! Have you ever noticed it? :)
```

Sample Output

```
Day #1: the longest nap starts at 15:00 and will last for 30 minutes.
Day #2: the longest nap starts at 15:00 and will last for 1 hours and 45 minutes.
Day #3: the longest nap starts at 17:15 and will last for 45 minutes.
Day #4: the longest nap starts at 13:00 and will last for 5 hours and 0 minutes.
```

4.6.7 ShellSort

PC/UVa IDs: 110407/10152, **Popularity:** B, **Success rate:** average **Level:** 2

King Yertle wishes to rearrange his turtle throne to place his highest-ranking nobles and closest advisors nearer to the top. A single operation is available to change the order of the turtles in the stack: a turtle can crawl out of its position in the stack and climb up over the other turtles to sit on the top.

Given an original ordering of a turtle stack and a required ordering for the same turtle stack, your job is to find a minimal sequence of operations that rearranges the original stack into the required stack.

Input

The first line of the input consists of a single integer K giving the number of test cases. Each test case consists of an integer n giving the number of turtles in the stack. The next n lines describe the original ordering of the turtle stack. Each of the lines contains the name of a turtle, starting with the turtle on the top of the stack and working down to the turtle at the bottom of the stack. Turtles have unique names, each of which is a string of no more than eighty characters drawn from a character set consisting of the alphanumeric characters, the space character and the period ("."). The next n lines in the input give the desired ordering of the stack, once again by naming turtles from top to bottom. Each test case consists of exactly $2n+1$ lines in total. The number of turtles (n) will be less than or equal to 200.

Output

For each test case, the output consists of a sequence of turtle names, one per line, indicating the order in which turtles are to leave their positions in the stack and crawl to the top. This sequence of operations should transform the original stack into the required stack and should be as short as possible. If more than one solution of shortest length is possible, any of the solutions may be reported.

Print a blank line after each test case.

Sample Input

```
2
3
Yertle
Duke of Earl
Sir Lancelot
Duke of Earl
Yertle
Sir Lancelot
9
```

```
Yertle
Duke of Earl
Sir Lancelot
Elizabeth Windsor
Michael Eisner
Richard M. Nixon
Mr. Rogers
Ford Perfect
Mack
Yertle
Richard M. Nixon
Sir Lancelot
Duke of Earl
Elizabeth Windsor
Michael Eisner
Mr. Rogers
Ford Perfect
Mack
```

Sample Output

```
Duke of Earl

Sir Lancelot
Richard M. Nixon
Yertle
```

4.6.8 Football (aka Soccer)

PC/UVa IDs: 110408/10194, **Popularity:** B, **Success rate:** average **Level:** 1

Football is the most popular sport in the world, even through Americans insist on calling it "*soccer*." A country such as five-time World Cup-winning Brazil has so many national and regional tournaments that is it very difficult to keep track. Your task is to write a program that receives the tournament name, team names, and games played and outputs the tournament standings so far.

A team wins a game if it scores more goals than its opponent, and loses if it scores fewer goals. Both teams tie if they score the same number of goals. A team earns 3 points for each win, 1 point for each tie, and 0 points for each loss.

Teams are ranked according to these rules (in this order):

1. Most points earned.
2. Most wins.
3. Most goal difference (i.e., goals scored – goals against)
4. Most goals scored.
5. Fewest games played.
6. Case-insensitive lexicographic order.

Input

The first line of input will be an integer N in a line alone ($0 < N < 1,000$). Then follow N tournament descriptions, each beginning with a tournament name. These names can be any combination of at most 100 letters, digits, spaces, etc., on a single line. The next line will contain a number T ($1 < T \leq 30$), which stands for the number of teams participating on this tournament. Then follow T lines, each containing one team name. Team names consist of at most 30 characters, and may contain any character with ASCII code greater than or equal to 32 (space), except for "#" and "@" characters.

Following the team names, there will be a non-negative integer G on a single line which stands for the number of games already played on this tournament. G will be no greater than 1,000. G lines then follow with the results of games played in the format:

team_name_1#goals1@goals2#team_name_2

For instance, *Team A*#3@1#*Team B* means that in a game between *Team A* and *Team B*, *Team A* scored 3 goals and *Team B* scored 1. All goals will be non-negative integers less than 20. You may assume that all team names mentioned in game results will have appeared in the team names section, and that no team will play against itself.

Output

For each tournament, you must output the tournament name in a single line. In the next T lines you must output the standings, according to the rules above. Should

lexicographic order be needed as a tie-breaker, it must be done in a case-insensitive manner. The output format for each line is shown below:

[*a*]) *Team_name* [*b*]p, [*c*]g ([*d*]-[*e*]-[*f*]), [*g*]gd ([*h*]-[*i*])

where [*a*] is team rank, [*b*] is the total points earned, [*c*] is the number of games played, [*d*] is wins, [*e*] is ties, [*f*] is losses, [*g*] is goal difference, [*h*] is goals scored, and [*i*] is goals against.

There must be a single blank space between fields and a single blank line between output sets. See the sample output for examples.

Sample Input

```
2
World Cup 1998 - Group A
4
Brazil
Norway
Morocco
Scotland
6
Brazil#2@1#Scotland
Norway#2@2#Morocco
Scotland#1@1#Norway
Brazil#3@0#Morocco
Morocco#3@0#Scotland
Brazil#1@2#Norway
Some strange tournament
5
Team A
Team B
Team C
Team D
Team E
5
Team A#1@1#Team B
Team A#2@2#Team C
Team A#0@0#Team D
Team E#2@1#Team C
Team E#1@2#Team D
```

Sample Output

```
World Cup 1998 - Group A
1) Brazil 6p, 3g (2-0-1), 3gd (6-3)
2) Norway 5p, 3g (1-2-0), 1gd (5-4)
3) Morocco 4p, 3g (1-1-1), 0gd (5-5)
4) Scotland 1p, 3g (0-1-2), -4gd (2-6)

Some strange tournament
1) Team D 4p, 2g (1-1-0), 1gd (2-1)
2) Team E 3p, 2g (1-0-1), 0gd (3-3)
3) Team A 3p, 3g (0-3-0), 0gd (3-3)
4) Team B 1p, 1g (0-1-0), 0gd (1-1)
5) Team C 1p, 2g (0-1-1), -1gd (3-4)
```