The Ninth Hunan Collegiate Programming Contest

Online Version

(with more input data and tighter time limits, especially for problem K)

13th October, 2013
You get 20 Pages
12 Problems
&
300 Minutes

A

Almost Palindrome

Input: Standard Input
Output: Standard Output

Given a line of text, find the longest almost-palindrome substring. A string S is almost-palindrome if

1. S begins and ends with a letter, and
2. a(S) and b(S) have at most 2k positions with different characters

Here a(S) is the string after removing all non-letter characters and converting all the letters to lowercase, b(S) is the reversed string of a(S).

For example, when k=1, "Race cat" is almost-palindrome, because a(S)="racecat" and b(S)="tacecar" differ at exactly 2 positions.

Input

There will be at most 25 test cases. Each test case contains two lines. The first line is k (0<=k<=200). The second line contains a string with at least one letter and at most 1,000 characters (excluding the newline character). The string will only contain letters, spaces and other printable characters like ("," or "." etc) and *will not start with a whitespace*.

Output

For each test case, print the length of the longest almost-palindrome substring and its starting position (starting from 1). If there is more than one such string, print the smallest starting position.

Sample Input

```
1
Wow, it is a Race cat!
0
abcdefg
0
Kitty: Madam, I'm adam.
```

Output for Sample Input

```
Case 1: 8 3
Case 2: 1 1
Case 3: 15 8
```

Problemsetter: Rujia Liu, Special Thanks: Feng Chen, Md. Mahbubul Hasan

B	Boxes in a Line **Input:** Standard Input **Output:** Standard Output	

You have n boxes in a line on the table numbered 1~n from left to right. Your task is to simulate 4 kinds of commands:

- 1 X Y: move box X to the left to Y (ignore this if X is already the left of Y)
- 2 X Y: move box X to the right to Y (ignore this if X is already the right of Y)
- 3 X Y: swap box X and Y
- 4: reverse the whole line.

Commands are guaranteed to be valid, i.e. X will be not equal to Y.

For example, if n=6, after executing 1 1 4, the line becomes 2 3 1 4 5 6. Then after executing 2 3 5, the line becomes 2 1 4 5 3 6. Then after executing 3 1 6, the line becomes 2 6 4 5 3 1. Then after executing 4, then line becomes 1 3 5 4 6 2

Input

There will be at most 10 test cases. Each test case begins with a line containing 2 integers n, m(1<=n, m<=100,000). Each of the following m lines contain a command.

Output

For each test case, print the sum of numbers at odd-indexed positions. Positions are numbered 1 to n from left to right.

Sample Input

```
6 4
1 1 4
2 3 5
3 1 6
4
6 3
1 1 4
2 3 5
3 1 6
100000 1
4
```

Output for Sample Input

```
Case 1: 12
Case 2: 9
Case 3: 2500050000
```

Problemsetter: Rujia Liu, Special Thanks: Feng Chen, Md. Mahbubul Hasan

C

Character Recognition?

Input: Standard Input
Output: Standard Output

Write a program that recognizes characters. Don't worry, because you only need to recognize three digits: 1, 2 and 3. Here they are:

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

Input

The input contains only one test case, consisting of 6 lines. The first line contains n, the number of characters to recognize (1<=n<=10). Each of the next 5 lines contains 4n characters. Each character contains exactly 5 rows and 3 columns of characters followed by an empty column (filled with '.').

Output

The output should contain exactly one line, the recognized digits in one line.

Sample Input

```
3
.*..***.***.
.*....*...*.
.*..***.***.
.*..*.....*.
.*..***.***.
```

Output for Sample Input

```
123
```

Problemsetter: Rujia Liu, Special Thanks: Feng Chen, Md. Mahbubul Hasan

D

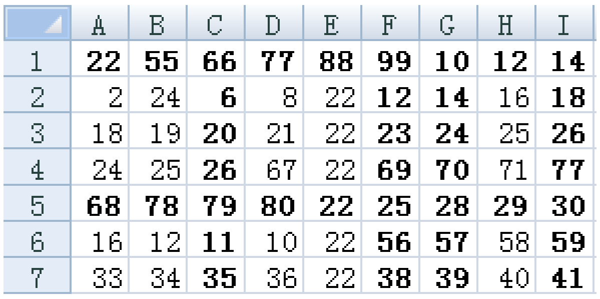
Damaging Your Spreadsheet (Spreadsheet Tracking II)

Input: Standard Input
Output: Standard Output

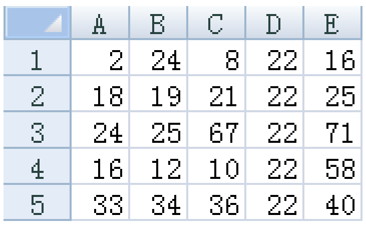
Data in spreadsheets are stored in cells, which are organized in rows (r) and columns (c). Some operations on spreadsheets can be applied to single cells (r,c), while others can be applied to entire rows or columns. Typical cell operations include inserting and deleting rows or columns and exchanging cell contents.

Some spreadsheets allow users to mark collections of rows or columns for deletion, so the entire collection can be deleted at once. Some (unusual) spreadsheets allow users to mark collections of rows or columns for insertions too. Issuing an insertion command results in new rows or columns being inserted before each of the marked rows or columns.

Suppose, for example, the user marks rows 1 and 5 of the spreadsheet on the left for deletion, then marks columns 3, 6, 7, and 9 for deletion, the spreadsheet shrinks to spreadsheet on the right:

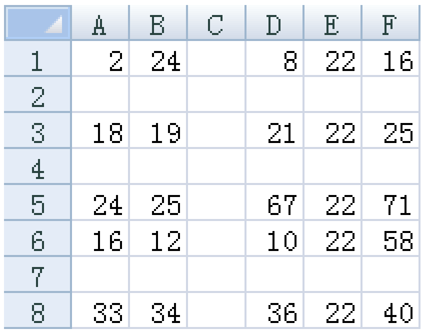


	A	B	C	D	E	F	G	H	I
1	**22**	**55**	**66**	**77**	**88**	**99**	**10**	**12**	**14**
2	2	24	**6**	8	22	**12**	**14**	16	**18**
3	18	19	**20**	21	22	**23**	**24**	25	**26**
4	24	25	**26**	67	22	**69**	**70**	71	**77**
5	**68**	**78**	**79**	**80**	**22**	**25**	**28**	**29**	**30**
6	16	12	**11**	10	22	**56**	**57**	58	**59**
7	33	34	**35**	36	22	**38**	**39**	40	**41**



	A	B	C	D	E
1	2	24	8	22	16
2	18	19	21	22	25
3	24	25	67	22	71
4	16	12	10	22	58
5	33	34	36	22	40

If the user marks rows 2, 3 and 5 for insertion, then marks column 3 for insertion, the spreadsheet grows to the one below:



	A	B	C	D	E	F
1	2	24		8	22	16
2						
3	18	19		21	22	25
4						
5	24	25		67	22	71
6	16	12		10	22	58
7						
8	33	34		36	22	40

Now that someone damaged your spreadsheet by issuing several insertion, deletion and exchange operations (details are described below).

Your task is to calculate the number of cells that are kept (not deleted), and the total distance between the original cells and their final locations. If a cell in (x1, y1) was moved to (x2, y2), the total distance is increased by |x1-x2|+|y1-y2|. You also need to determine the final locations of some important data.

Input

The input consists of a sequence of spreadsheets, operations on those spreadsheets, and queries about them. Each spreadsheet definition begins with a pair of integers specifying its initial number of rows (r) and columns (c), followed by an integer specifying the number (n) of spreadsheet operations. Row and column labeling begins with 1. The following n lines specify the desired operations. 1<=r,c<=5000, 1<=n<=50. There will be at least one row and one column at any time.

An operation to exchange the contents of cell (r1, c1) with (r2, c2) is given by: `EX r1 c1 r2 c2.` The four insert and delete commands--DC (delete columns), DR (delete rows), IC (insert columns), and IR (insert rows) are given by: `<command> A` x_1 x_2 `...` x_A, where <command> is one of the four commands; A is a positive integer not greater than 5, and x_1, …, x_A are the labels of the columns or rows to be deleted or inserted before. For each insert and delete command, the order of the rows or columns in the command has no significance. Within a single delete or insert command, labels will be unique.

The operations are followed by an integer Q(Q<=10000), which is the number of queries for the spreadsheet. Each query consists of positive integers r and c, representing the row and column number of a cell in the original spreadsheet. For each query, your program must determine the current location of the data that was originally in cell (r, c).

The end of input is indicated by a row consisting of a pair of zeros for the spreadsheet dimensions.

Output

For each spreadsheet, your program must output its sequence number (starting at 1). In the next line, your program must output the number of cells that are kept, and the total move distance.

For each query, your program must output the original cell location followed by the final location of the data or the word GONE if the contents of the original cell location were destroyed as a result of the operations.

Separate output from different spreadsheets with a blank line.

Sample Input

```
7 9
5
DR   2  1 5
DC  4  3 6 7 9
IC  1  3
IR  2  2 4
EX 1 2 6 5
4
4 8
5 5
7 8
6 5
0 0
```

Output for Sample Input

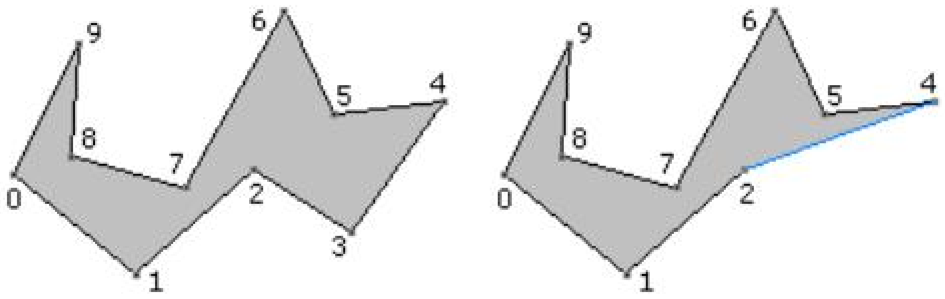
```
Spreadsheet #1
There are 25 cell(s) kept, total
distance = 29
Cell data in (4,8) moved to (4,6)
Cell data in (5,5) GONE
Cell data in (7,8) moved to (7,6)
Cell data in (6,5) moved to (1,2)
```

Modified from ACM/ICPC World Finals 1997: Spreadsheet Tracking.
Problemsetter: Rujia Liu, Special Thanks: Md. Mahbubul Hasan

E Ears Cutting

Input: Standard Input
Output: Standard Output

A famous way to cut polygon into triangles is ear cutting: each time cut off a triangle along a diagonal, after n-3 cuts only a single triangle remains. In the following picture, the ear {2,3,4} was cut off.



Find a way to cut ears of a simple polygon such that the sum of cut lengths is minimal.

Input

There will be at most 30 test cases. The first line of each case contains the number of vertices, n (4<=n<=100). Each of the following n lines contains the coordinates of a vertex, in clockwise *or* counter-clockwise order. Coordinates are integers whose absolute value does not exceed 10000.

Output

For each test case, print the minimal sum of cut lengths, rounded to 4 decimal digits.

Sample Input

```
4
0 0
3 0
1 1
0 3
4
0 0
10 0
10 1
0 1
```

Output for Sample Input

```
Case 1: 1.4142
Case 2: 10.0499
```

Problemsetter: Rujia Liu, Special Thanks: Md. Mahbubul Hasan

F

Funny Car Racing

Input: Standard Input
Output: Standard Output

There is a funny car racing in a city with *n* junctions and *m* directed roads.

The funny part is: each road is open and closed periodically. Each road is associate with two integers (*a*, *b*), that means the road will be open for *a* seconds, then closed for *b* seconds, then open for *a* seconds... All these start from the beginning of the race. You must enter a road when it's open, and leave it before it's closed again.

Your goal is to drive from junction *s* and arrive at junction *t* as early as possible. Note that you can wait at a junction even if all its adjacent roads are closed.

Input

There will be at most 30 test cases. The first line of each case contains four integers n, m, s, t (1<=n<=300, 1<=m<=50,000, 1<=s,t<=n). Each of the next m lines contains five integers u, v, a, b, t (1<=u,v<=n, 1<=a,b,t<=10^5), that means there is a road starting from junction u ending with junction v. It's open for a seconds, then closed for b seconds (and so on). The time needed to pass this road, by your car, is t. No road connects the same junction, but a pair of junctions could be connected by more than one road.

Output

For each test case, print the shortest time, in seconds. It's always possible to arrive at t from s.

Sample Input

```
3 2 1 3
1 2 5 6 3
2 3 7 7 6
3 2 1 3
1 2 5 6 3
2 3 9 5 6
```

Output for Sample Input

```
Case 1: 20
Case 2: 9
```

Problemsetter: Rujia Liu, Special Thanks: Md. Mahbubul Hasan

G

Good Teacher

Input: Standard Input
Output: Standard Output

I want to be a good teacher, so at least I need to remember all the student names. However, there are too many students, so I failed. It is a shame, so I don't want my students to know this. Whenever I need to call someone, I call his CLOSEST student instead. For example, there are 10 students:

```
A ? ? D ? ? ? H ? ?
```

Then, to call each student, I use this table:

Pos	Reference
1	A
2	right of A
3	left of D
4	D
5	right of D
6	middle of D and H
7	left of H
8	H
9	right of H
10	right of right of H

Input

There is only one test case. The first line contains n, the number of students (1<=n<=100). The next line contains n space-separated names. Each name is either ? or a string of no more than 3 English letters. There will be at least one name not equal to ?. The next line contains q, the number of queries (1<=q<=100). Then each of the next q lines contains the position p (1<=p<=n) of a student (counting from left).

Output

Print q lines, each for a student. Note that "middle of X and Y" is only used when X and Y are both closest of the student, and X is always to his left.

Sample Input

```
10
A ? ? D ? ? ? H ? ?
4
3
8
6
10
```

Output for Sample Input

```
left of D
H
middle of D and H
right of right of H
```

Problemsetter: Rujia Liu, Special Thanks: Feng Chen, Md. Mahbubul Hasan

H

High bridge, low bridge

Input: Standard Input
Output: Standard Output

Q: There are one high bridge and one low bridge across the river. The river has flooded twice, why the high bridge is flooded twice but the low bridge is flooded only once?
A: Because the lower bridge is so low that it's still under water *after* the first flood is over.

If you're confused, here's how it happens:

- Suppose high bridge and low bridge's heights are 2 and 5, respectively, and river's initial water level is 1.
- First flood: the water level is raised to 6(Both bridges are flooded), and then back to 2(high bridge is not flooded anymore, but low bridge is still flooded).
- Second flood: the water level is raised to 8(The high bridge is flooded *again*), and then back to 3.

Just a word game, right? The key is that if a bridge is still under water (i.e. the water level is no less than the bridge height) *after* a flood, then next time it will *not* be considered flooded again.

Suppose the i-th flood raises the water level to a_i and then back to b_i. Given n bridges' heights, how many bridges are flooded at least k times? The initial water level is 1.

Input

The input contains at most 25 test cases. Each test case begins with 3 integers n, m, k in the first line ($1<=n,m,k<=10^5$). The next line contains n integers h_i, the heights of each bridge ($2<=h_i<=10^8$). Each of the next m lines contains two integers a_i and b_i ($1<=b_i<a_i<=10^8$, $a_i>b_{i-1}$). The file size of the whole input does not exceed 5MB.

Output

For each test case, print the number of bridges that is flooded at least k times.

Sample Input

```
2 2 2
2 5
6 2
8 3
5 3 2
2 3 4 5 6
5 3
4 2
5 2
```

Output for Sample Input

```
Case 1: 1
Case 2: 3
```

Explanation

For the second sample, 5 bridges are flooded 1, 2, 3, 2, 0 times, respectively.

Problemsetter: Rujia Liu, Special Thanks: Feng Chen, Md. Mahbubul Hasan

I

Interesting Calculator

Input: Standard Input
Output: Standard Output

There is an interesting calculator. It has 3 rows of button.

Row 1: button 0, 1, 2, 3, ..., 9. Pressing each button *appends* that digit to the end of the display.
Row 2: button +0, +1, +2, +3, ..., +9. Pressing each button *adds* that digit to the display.
Row 3: button *0, *1, *2, *3, ..., *9. Pressing each button *multiplies* that digit to the display.

Note that it never displays leading zeros, so if the current display is 0, pressing 5 makes it 5 instead of 05. If the current display is 12, you can press button 3, +5, *2 to get 256. Similarly, to change the display from 0 to 1, you can press 1 or +1 (but not both!).

Each button has a positive cost, your task is to change the display from x to y with minimum cost. If there are multiple ways to do so, the number of presses should be minimized.

Input

There will be at most 30 test cases. The first line of each test case contains two integers x and y($0<=x<=y<=10^5$). Each of the 3 lines contains 10 positive integers (not greater than 10^5), i.e. the costs of each button.

Output

For each test case, print the minimal cost and the number of presses.

Sample Input

```
12 256
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1
12 256
100 100 100 1 100 100 100 100 100 100
100 100 100 100 100 1 100 100 100 100
100 100 10 100 100 100 100 100 100 100
```

Output for Sample Input

```
Case 1: 2 2
Case 2: 12 3
```

Problemsetter: Rujia Liu, Special Thanks: Md. Mahbubul Hasan, Feng Chen

J

Joking with Fermat's Last Theorem

Input: Standard Input
Output: Standard Output

Fermat's Last Theorem: no three positive integers a, b, and c can satisfy the equation $a^n + b^n = c^n$ for any integer value of n greater than two.

From the theorem, we know that $a^3 + b^3 = c^3$ has no positive integer solution.

However, we can make a joke: find solutions of $a^3 + b^3 = c3$. For example $4^3 + 9^3 = 793$, so a=4, b=9, c=79 is a solution.

Given two integers x and y, find the number of solutions where x<=a,b,c<=y.

Input

There will be at most 10 test cases. Each test case contains a single line: x, y ($1<=x<=y<=10^8$).

Output

For each test case, print the number of solutions.

Sample Input

```
1 10
1 20
123 456789
```

Output for Sample Input

```
Case 1: 0
Case 2: 2
Case 3: 16
```

Problemsetter: Rujia Liu, Special Thanks: Md. Mahbubul Hasan, Feng Chen

K	# Killer Puzzle **Input:** Standard Input **Output:** Standard Output	acm

Have you tried this horrible-looking puzzle?

1. Which question is the first question whose answer is b?
(a) 2; (b) 3; (c) 4; (d) 5; (e) 6;

2. The only question that has the same answer as its next question is (e.g. option e means question 6 and 7's answers are the same):
(a) 2; (b) 3; (c) 4; (d) 5; (e) 6;

3. Among the 5 options, which question has the same answer as this question (i.e. question 3)?
(a) 1; (b) 2; (c) 4; (d) 7; (e) 6;

4. How many questions' answer is a?
(a) 0; (b) 1; (c) 2; (d) 3; (e) 4

5. Which of the following questions has the same answer as this question?
(a) 10; (b) 9; (c) 8; (d) 7; (e) 6;

6. The number of questions whose answer is a, equals the number of questions whose answer is:
(a) b; (b) c; (c) d; (d) e; (e) none of above

7. What is the difference of this question's answer and the next question's answer (e.g. the difference of a and b is 1) ?
(a) 4; (b) 3; (c) 2; (d) 1; (e) 0;

8. How many questions' answer is a vowel? (only a and e are vowels. Others are consonants)
(a) 2; (b) 3; (c) 4; (d) 5; (e) 6

9. The number of questions whose answer is a consonant is:
(a) a prime; (b) a factorial; (c) a square number; (d) a cubic number; (e) a multiple of 5

10. The answer of this question is:
(a) a; (b) b; (c) c; (d) d; (e) e;

Note:
1. make sure that your answer is not self-contradicting. For example, the first question's answer can't be b.
2. make sure that for each question, *only* your answer is correct, all other options must be incorrect. For example, if your answer to question 5 is a, then none of question 9, 8, 7, 6's answers can be a!
3. make sure that your answer won't make any question invalid. For example, if question 2 and 3's answer are the same, and question 8,9's answers are also the same, question 2 would be invalid (because no question is "the only question" that satisfying the condition)

It's possible to solve this problem by hand, but as a programmer, solving it with a program is more fun!

How to Solve the Puzzle with a Program

Here's one way: enumerate all possible answers ($5^{10} = 9765625$), and for each question, check whether only your answer is correct. Pseudo-code:

```
forall(answer_list):
  bad = False
  for testing_question in [1,2,3,4,5,6,7,8,9,10]:
    for testing_option in ["a","b","c","d","e"]:
      # your answer should be correct
      if testing_option == answer_list[testing_question] and
check(testing_question, testing_option) == False:
        bad = True
      # other options must be incorrect
      if testing_option != answer_list[testing_question] and
check(testing_question, testing_option) == True:
        bad = True
  if not bad:
    print answer_list
```

Here "answer_list" is a list of letters (subscript is 1-based), where the i-th letter is the answer to the i-th question.

Believe or not, the *only* answer is: cdebeedcba (if you prefer to add the question index before each answer, it is 1c2d3e4b5e6e7d8c9b10a)

Amazing, huh? There's more. You wish that your program could solve some other puzzles, but first of all, you need to formulate the puzzle in a formal language.

Formalizing the Puzzle

This problem uses a LISP dialect to represent the puzzle. Don't worry if you don't know LISP, it has a very simple syntax. `(f a b)` means calling a function f with parameters a and b. That's like `f(a, b)` in C/C++/Java. Similarly, `(f a (g b c) d)` is like `f(a, g(b, c), d)` in C/C++/Java.

Here is an example of how to describe a question of the puzzle:

```
3. (equal (answer 3) (answer (option-value)))
a. 1
b. 2
c. 4
d. 7
e. 6
```

There are two very important built-in functions involved:

`(answer idx)`	returns answer_list[idx] in the pseudo-code above.
`(option-value)`	returns the "evaluation result" of testing_option's text, treated as an expression.

In the example above, if testing_option is "c", then (option-value) returns 4 (an integer) because 4 is the expression presented in the option "c" of this question. Note that testing_option's text can be a complex expression instead of a simple value. Refer to Sample Input.

The function `check(testing_question, testing_option)` above can be implemented as follows:

```
check(testing_question, testing_option):
  1. set-up the function (option-value) so that it returns the evaluation
result of testing_option of testing_question
  2. evaluate the lisp expression of testing_question (e.g. the expression
(equal (answer 3) (answer (option-value))) in the example above)
  3. if an unhandled exception is raised during the evaluation, returns False
  4. if the result of step 2 is boolean, return it; otherwise return False
```

There is one special option expression: "none-of-above". The result of "none-of-above" depends on other options' evaluation results. In this problem, there can be at most one "none-of-above" for each question, and it must be the last option.

Details

Here are the details of the LISP dialect used in this problem:

- There are four datatypes: integer, string, boolean and functions.
- There are only two boolean values: true and false. Note that there are no "boolean literal", so you don't care whether to use #t and #f (like in Scheme), or t and nil (like in Common Lisp) to represent boolean constants.
- Integers are always non-negative integers.
- String literals are always enclosed by double quotes, like "a string".
- There is no variable. All the so-called "identifiers" (consisting of letters and hyphens) are always pre-defined functions.

Below is a list of pre-defined functions. Functions starting with ! means it may throw an exception, and functions starting with @ means it can handle exceptions. Like C++/Java/Python, once an exception is raised, the evaluation process is stopped unless a function handles the exception. In the text below, iff means "if and only if".

Basic functions

`(equal a b)`	return true iff a and b are of the same type and are equal. In this problem, you'll never need to compare two functions.
`(option-value)`	discussed above.
`!(answer idx)`	discussed above. If idx is not an integer or is not in 1~n (where n is the number of questions), then raises an exception
`!(answer-value idx)`	Returns the "evaluation result" of option answer_list[idx] of question idx. Also raises an exception on error.

Predicates
Predicate is a special kind of function. It always takes a value of any type and returns a boolean value.

`prime-p`	returns true iff the parameter is a positive prime
`factorial-p`	self-explanatory
`square-p`	self-explanatory
`cubic-p`	self-explanatory
`vowel-p`	returns true iff the parameter is a single letter and is a vowel
`consonant-p`	self-explanatory

Queries and statistics

`!@(first-question pred)`	Return id of the first question that satisfies `pred`. Raises an exception if not found.
`!@(last-question pred)`	Return id of the last question that satisfies `pred`. Raises an exception if not found.
`!@(only-question pred)`	Return id of the only question that satisfies `pred`. Raises an

	exception if not found or more than one question found.
`@(count-question pred)`	Return the number of questions that satisfies `pred`.
`!(diff-answer idx1 idx2)`	The difference of answers of question idx1 and idx2. Raises an exception on error, otherwise the return value is always 0~m-1, where m is the number of options for each question.

Note that in the first four functions (those with a '@' flag), if an exception was raised when evaluating the predicate, the exception is handled and the predicate is not considered satisfied. For example, if answer_list is "abc", `(count-question (make-answer-diff-next-equal 0))` returns 0 and doesn't raise an exception, even though evaluating the predicate for question 3, i.e. `((make-answer-diff-next-equal 0) 3)` raised an exception. However, all other functions will not handle exceptions. For example, if there are only 3 questions, `(factorial-p (answer-value 5))` will raise an exception instead of returning false.

Predicate generators
There are also functions that can create predicates on-the-fly:

`!(make-answer-diff-next-equal num)`	returns a predicate (p idx) which evaluates (diff-answer idx idx+1) and returns true if the result equals to num. Raises an exception if num is not an integer.
`(make-answer-equal a)`	returns a predicate (p idx) which evaluates (answer idx) and returns true if the result equals a.
`(make-answer-is pred)`	returns a predicate (p idx) which evaluates (answer idx) and returns true if the result satisfies `pred`.
`(make-answer-value-equal a)`	self-explanatory. The predicate evaluates (answer-value idx)
`(make-answer-value-is pred)`	self-explanatory. The predicate evaluates (answer-value idx)
`!(make-is-multiple num)`	returns a predicate (p i) which returns true iff i is an integer and is a multiple of num. Raises an exception if num is not an integer.
`!(make-equal val)`	returns a predicate (p v) which returns true iff (equal v val) is true. Raises an exception if val is neither an integer nor a string.
`(make-not pred)`	returns a predicate (p v) which returns true iff (pred v) is false.
`(make-and pred1 pred2)`	returns a predicate (p v) which returns true iff (pred1 v) and (pred2 v) are both true. Both pred1 and pred2 need to be evaluated. No short-circuit operation should be done.
`(make-or pred1 pred2)`	returns a predicate (p v) which returns true iff at least one of (pred1 v) and (pred2 v) is true. Both pred1 and pred2 need to be evaluated. No short-circuit operation should be done.

For example, `(make-is-multiple 3)` returns a predicate "is a multiple of 3", so `((make-is-multiple 3) 6)` returns true and `((make-is-multiple 3) 10)` returns false. Similarly `(make-not (make-or square-p prime-p))` returns a predicate "neither a square nor a prime".

Input

There will be at most 50 test cases. Each test case begins with two integers n and m(2<=n<=10, 2<=m<=5), the number of questions and the number of options per question. Each question is described with m+1 lines: the question's expression and the options. Questions are numbered 1~n, and options are labeled a~e. Options are valid expressions and will not call option-value (calling option-value makes it recursive!). Each question is followed by a blank line. Most test cases are easy.

Output

For each test case, print the case number in the first line, and a list of answers, one per line, sorted in ascending order. There will always be at least one answer.

Sample Input

```
3 3
(equal (option-value) (count-question
(make-answer-equal "a")))
3
0
1

(equal (option-value) "a")
"c"
"b"
"a"

((option-value) (count-question (make-
answer-equal "c")))
(make-and (make-is-multiple 2) (make-or
factorial-p prime-p))
(make-not prime-p)
"none-of-above"

3 2
(equal (option-value) (answer 2))
"a"
"none-of-above"

(equal (option-value) (first-question
(make-answer-diff-next-equal 0)))
1
2

((option-value) (last-question (make-
answer-equal "b")))
(make-is-multiple 2)
(make-not (make-is-multiple 2))

3 2
(equal (option-value) (answer 1))
"a"
"b"

((option-value) (last-question (make-
answer-diff-next-equal 0)))
(make-equal 2)
"none-of-above"

((option-value) (only-question (make-
answer-equal "b")))
(make-is-multiple 2)
"none-of-above"

2 5
((option-value) (diff-answer 1 2))
factorial-p
prime-p
```

Output for Sample Input

```
Case 1:
bcb
cca
Case 2:
aab
Case 3:
aba
Case 4:
ab
ee
Case 5:
ba
```

```
(make-not square-p)
(make-not cubic-p)
"none-of-above"

(equal (only-question (option-value)) 1)
(make-answer-is consonant-p)
(make-answer-is vowel-p)
(make-answer-value-equal 1)
(make-answer-value-is square-p)
"none-of-above"

2 2
(option-value)
(equal (first-question (make-answer-diff-
next-equal 2)) (first-question (make-
answer-diff-next-equal 2)))
"none-of-above"

(equal (option-value) 1)
1
2
```

Problemsetter: Rujia Liu
Special Thanks: Zhiwei Li(Common Lisp), Md. Mahbubul Hasan (C++), Feng Chen (C#)
Peter Norvig's articles: http://norvig.com/lispy.html and http://norvig.com/lispy2.html
This problem is complex. If you have any questions, email me: rujia.liu@gmail.com

L

Last Blood

Input: Standard Input
Output: Standard Output

In many programming contests, special prizes are given to teams who solved a particular problem first. We call the first accepted solution "First Blood".

It's an interesting idea to set prizes for "Last Blood". Then people won't submit their solutions until the last minute. But this is dangerous: if the solution got "Wrong Answer" or even "Time limit exceeded", it may be too late to correct the solution.

You may argue that once a submission got "Accepted", the team can send it again, but in this problem, we only consider the earliest accepted solution of a team for each problem, so re-sending an accepted solution does NOT help!

Given all the submissions in a contest, your task is to find out the "Last Blood" prizes for each problem.

Input

There is only one test case. The first line contains three integer n, t, m (5<=n<=12, 10<=t<=100, 1<=m<=1000), the number of problems, teams and submissions. Each of the following m lines describes one submission: time (0<=time<=300), teamID(1~t), problem (A~L) and verdict("Yes" or "No"). Submissions are sorted in time order. That means for two submissions of the same "time" field, the submission that comes later in the input is received later in the contest (maybe only a few seconds later). No two submissions are received in exactly the same time.

Output

For each problem, print the last blood's time and teamID.

Sample Input

```
5 10 18
0 2 B No
11 2 B Yes
20 3 A Yes
35 8 E No
40 8 E No
45 7 E No
50 10 A Yes
100 4 A No
120 6 B Yes
160 2 E Yes
180 2 A Yes
210 3 B Yes
240 10 B No
250 10 B Yes
270 2 B Yes
295 8 E Yes
295 7 E Yes
299 10 D Yes
```

Output for Sample Input

```
A 180 2
B 250 10
C - -
D 299 10
E 295 7
```

Problemsetter: Rujia Liu, Special Thanks: Md. Mahbubul Hasan