2016

PROBLEM SET

PREAMBLE

Please note the following very important details relating to input and output:

- Read all input from the keyboard, i.e. use `stdin`, `System.in`, `cin` or equivalent. Input will be redirected from a file to form the input to your submission.

- Do NOT prompt for input as this will appear in your output and cause a submission to be judged as wrong.

- Write all output to the screen, i.e. use `stdout`, `System.out`, `cout` or equivalent. Do not write to `stderr`. Do NOT use, or even include, any module that allows direct manipulation of the screen, such as `conio`, `Crt` or anything similar.

- Output from your program is redirected to a file for later checking. Use of direct I/O means that such output is not redirected and hence cannot be checked. This could mean that a correct program is rejected! You have been warned.

- Unless otherwise stated, all *integers* will fit into a standard 32-bit computer word. If more than one integer appears on a line, they will be separated by white space, i.e. spaces or tabs.

- An *uppercase letter* is a character in the sequence 'A' to 'Z'. A *lower case letter* is a character in the sequence 'a' to 'z'. A *letter* is either a lower case letter or an upper case letter.

- Unless otherwise stated, a *word* or a *name* is a continuous sequence of letters.

- Unless otherwise stated, a *string* is a continuous sequence of visible characters.

- Unless otherwise stated, words and names will contain no more than 60 characters, and strings will contain no more than 250 characters.

- If it is stated that ‘a line contains no more than *n* characters’, this does not include the character(s) specifying the end of line.

- Input files are often terminated by a ‘sentinel’ line, followed by an end of file marker. This line should not be processed.

Please also note that the filenames of your submitted programs may need to follow a particular naming convention, as specified by the contest organisers at your site.

PROBLEM A **PRESENTS** **3 POINTS**

The Bloggs family keep getting invited to birthday parties, and they feel they must buy the person who invites them a present. The trouble is they do not have much money!

The family has come up with a strategy. Whenever they go to a shop to buy a present, they will not buy the cheapest present (that would seem mean!), so they buy the second cheapest gift they can find. Your task here is to help them quickly find the second least expensive gift in the shop.

Input

Input will consist of a single scenario which contains a list of prices from a shop, each on a separate line. The first line will contain N, the number of prices (2 <= N <= 100). No price will be duplicated. The next N lines will each contain a single price in the form of a decimal number with 2 places of decimals (ie dollars and cents).

Output

Output the second lowest price on a line by itself. It must be in the same format as with the input.

Sample Input 1

```
6
62.95
18.50
17.49
26.30
58.95
22.25
```

Sample Input 2

```
8
115.90
129.99
106.95
99.95
136.18
117.85
109.56
99.99
```

Output for Sample Input 1

```
18.50
```

Output for Sample Input 2

```
99.99
```

PROBLEM B **HOLES** **3 POINTS**

Mike wrote some text on a piece of paper and now he wants to know how many holes are in the text.

What is a hole in this context?

If you think of the paper as the plane and a letter as a curve on the plane, then each letter divides the plane into regions. For example letters "A" and "O" divide the plane into two regions so we say these letters each have one hole. Similarly, letter "B" has two holes and letters such as "C", "E", "F" and "K" have no holes. Spaces, of course, have no holes.

We say that the number of holes in the text is equal to the total number of holes in the letters of the text. Help Mike to determine how many holes are in the text.

Input

Input starts with an integer, N, on a line of its own (0 < N <= 30). It tells you how many lines of text follow. There then follow N lines of text. Each line contains a string of text composed only of upper case letters and spaces; it will contain at least 1 letter. The length of the text is no more than 250 characters.

Output

For each line of input, output a single line containing the number of holes in the corresponding text.

Sample Input

```
3
CHEAT CODE
HAVE A CUP OF TEA
NEW ZEALAND
```

Output for Sample Input

```
3
5
3
```

PROBLEM C **OLYMPIC GAMES** **3 POINTS**

The Olympic Games in Rio cannot have escaped your attention! 2016 is a summer Olympics year.

The modern summer Olympic Games were first held in 1896, and have been held at 4 yearly intervals since then, except during the two world wars (1914 to 1918 and 1939 to 1945). The next games are scheduled to be held in Tokyo in 4 years time.

Input

Input will consist of a list of years, one per line, in the range 1860 to 2030 inclusive. The final year will be 0 – do not process that year.

Output

Output is one line per year. The year is given followed by:

`Summer Olympics`	if the summer Olympic Games were held, or are scheduled to be held in that year.
`Games cancelled`	if the summer games should have been held but there was a war.
`No city yet chosen`	if it is a future summer Olympic year but no city has yet been awarded the games
`No summer games`	otherwise.

Sample Input

```
1896
1902
1916
2024
2016
0
```

Output for Sample Input

```
1896 Summer Olympics
1902 No summer games
1916 Games cancelled
2024 No city yet chosen
2016 Summer Olympics
```

PROBLEM D **PALINDROMES** **3 POINTS**

A palindrome is a word, phrase, number or other sequence of characters which reads the same backwards or forwards. Given a string of lower case letters, can you make it a palindrome by deleting exactly one character? Note that the size of the string after deletion would be one less than it was before.

Input

Up to 30 lines each containing from 3 to 30 lower case letters. The final line will just contain # - do not process this line.

Output

If it is possible to make a palindrome from the text by deleting one letter, display the palindrome text. If it is not possible, display `not possible.`

It may be possible to make more than one palindrome by deleting a single letter. For example, `madmam` will become `mamam` if the `d` is deleted, or `madam` if the middle `m` is deleted. In such a case, display the palindrome formed by deleting the earliest letter from the text.

Sample Input

```
radars
rayon
madmam
#
```

Output for Sample Input

```
radar
not possible
mamam
```

PROBLEM E **VIRUS** **10 POINTS**

Shroeder and Patsy heard from a friend whose computer had been attacked by a ransom virus. A nasty message appeared telling their friend that all his files had been encrypted and that it would cost him to have them restored.

Being keen programmers, Shroeder and Patsy, just for fun you understand, wondered how that worked. They tried a simple encryption which would depend of a key which was a single number.

They thought they could use a sort of Caesar cipher (where letters are shifted a number of places through the alphabet which is considered circular so A follows Z). They would start with a shift somewhere between 1 and 25 (the key), then increase it by one each letter. It would cycle round so after 25 would become 1 again.

To decipher, if you know the key it is quite easy, as long as you remember to shift in the opposite direction!

Input

Input consists of a single integer, K, the key, followed by a line of encrypted text. The text will be no more than 250 characters long, and will consist of lower case letters, spaces, digits and punctuation marks only. Only letters are to by encrypted.

Output

Output the line of text correctly decrypted.

Sample Input

```
24
ym fpfvdvaqxx adbpaud xhl odb ldx bheqgul ybzseobczfz wjjqcru.
```

Output for Sample Input

```
an encryption problem for the new zealand programming contest.
```

PROBLEM F **ALPHA PUZZLE** **10 POINTS**

An alpha puzzle is a type of crossword puzzle where each letter square (ie one that is not black) contains a number to represent a letter. Throughout the puzzle, a particular letter is always represented by the same number. All letters of the alphabet are used, so numbers range from 1 to 26.

In this problem you will be given a solution to an alpha puzzle and have to assign numbers to the letters. Some alpha puzzles assign numbers to letters randomly, but the one that appears in the New Zealand Herald every day uses a non-random method which you have to implement.

The answer grid is read left to right, top to bottom; spaces (which represent black squares) are ignored. The first letter encountered is assigned the number 1, the next number 2 and so on. Only the first occurrence of a letter is processed, so only 26 numbers are allocated.

Input

You will be presented with a single puzzle solution which starts with a single integer, S, on a line by itself (10 <= S <=20). S is the size of the puzzle – the number of rows and columns of squares that it contains.

There will then follow S lines, each containing S characters, all upper case letters or spaces. Every letter of the alphabet will appear at least once.

Output

Follow the rules for letter allocation described above then output the 26 letters of the alphabet in their assigned numerical order so that, for example, the first letter is the one to which the number 1 is assigned, the last the one to which 26 was assigned.

[Turn over for sample input and output]

Sample Input

```
13
CITADEL POPPY
H O O U A L I
ARROW MASSAGE
S S N B S C L
SHOULDER BARD
I   O R T T
SQUEAL FRIEZE
  N D A O   X
FILE CLAPTRAP
E U J T I A L
VACCINE CARGO
E K V R A E D
RHYME SPLURGE
```

Output for Sample Input

```
CITADELPOYHURWMSGNBQFZXJVK
```

PROBLEM G **HUNT THE RABBIT** **10 POINTS**

Mr Farkle was brought up on a farm and spent quite a bit of time in his youth hunting rabbits! He now teaches maths and computing, and came up with a hunting game to help his students learn about the binary search.

He put 50 envelopes at the front of the room, numbered sequentially from 1 to 50. Inside one he hid a rabbit – not a real one, of course, just a card with a rabbit picture on it! He then put cards in all the other envelopes so that if an envelope was chosen with a number lower than the one holding the rabbit, the card would read "*Try a higher number*", otherwise the card would read "*Try a lower number*".

Students have to find the rabbit using a binary search, and write down the numbers of all the envelopes they open (in order) during their search. Remember, in a binary search you have to pick the middle envelope in the range you are searching. This is easy to find if there is an odd number of envelopes, but with an even number, you have to choose the lower numbered of the two middle envelopes. That means 25 will always be the first envelope checked.

Input

Each line of input will be a single number in the range 1 to 50 inclusive, it being the number of the envelope in which Mr Farkle has hidden the rabbit. The final input will be 0 which should not be processed.

Output

For each line of input, output the numbers of all envelopes opened, in the order they were opened, until the rabbit is found. Each number must be on the same line separated by a space from the previous number.

Sample Input

```
25
17
31
0
```

Output for Sample Input

```
25
25 12 18 15 16 17
25 38 31
```

PROBLEM H **PARLIAMENTARY RANKINGS** **10 POINTS**

A group of journalists decided it would be quite fun to rank the performance of MPs in parliament each week, and have asked you to write a program to help them.

The journalists were convinced that most MPs did not do very much at all, but came up with a list of codes that would identify actions taken by MPs which they considered noteworthy. They associated points with each action, some positive and some negative. The current table of actions and points follows. If your program is a success and the scheme catches on, more will be added later.

Code	Action	Points
S	Made a speech lasting at least 5 minutes	+10
Q	Asked a question during Question Time	+5
A	Answered a question during Question Time	+7
L	Spent less than an hour in the chamber	-8
F	Made a funny remark that caused laughter in the chamber	+4
D	Made a derisory comment about another party	-5
E	Was asked to leave the chamber	-10

Input

Input will contain data for one week. It will start with a line containing a positive integer N (0 < N <= 120), the number of MPs who attended the debating chamber of parliament in the week in question. There then follow N lines, each giving data on 1 MP. Data will be a unique identifying number, I (0 < I <= 120) followed by a space, followed by the name of the MP.

The list of MPs will be followed by a positive integer, A (0 < A <200), the number of action entries that complete the data. Each of the A lines following will contain data on 1 recorded action of an MP. It will consist of the MP's unique identifying number, followed by a space, followed by one of the letter codes from the table above. The points for each MP have to be added to give their points score for the week.

Output

Output the points score and name of the best scoring MP, and the points score and name of the worst scoring MP each on a separate line. In the case of equal scores, list on the same line all MPs with those scores in order of their unique identifying number, and separated by a space.

Sample Input 1

```
3
1 Bill Bloggs
3 Sara Quentin
2 Jo Jones
4
1 S
2 D
1 L
3 Q
```

Output for Sample Input 1	Explanation
`5 Sara Quentin`	Asked a question during Question Time (+5).
`-5 Jo Jones`	Made a derisory comment about another party (-5).
	Bill Bloggs made a speech lasting at least 5 minutes (+10), but countered that by spending less than an hour in the chamber (-8, total score +2).

Sample Input 2

```
2
1 Joe Smith
2 Sally Baynes
2
1 S
2 S
```

Output for Sample Input 2

```
10 Joe Smith Sally Baynes
10 Joe Smith Sally Baynes
```

Explanation: Both MPs have the same score so are both best scoring and worst scoring.

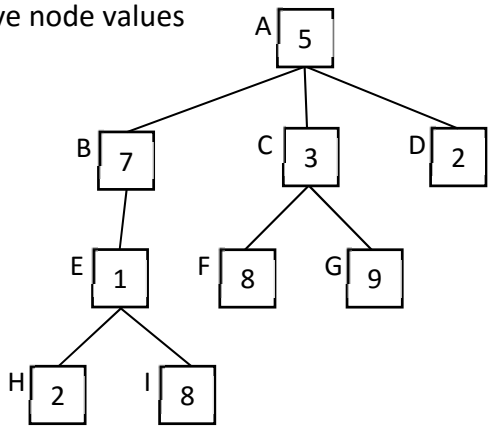
New Zealand Programming Contest 2016

PROBLEM I **OBFUSTICATION** **30 POINTS**

The NZPC data service has a need to secretly transmit trees of numbers from office to office (obviously, we can't tell you why). Of course the transmission is encrypted, but in order to further confound would-be spies, the service decided to obfusticate the tree before encryption.

Trees are linearized to text. The chosen obfustication is done by randomly varying the sub-tree ordering between pre-order, in-order and post-order on a per-node basis. For, example consider the following tree of integers. If we use pre-ordering on node A, post-ordering on B and E, and in-order on node C, the obfusticated tree data (OTD) will have node values in order

5 2 8 1 7 8 3 9 2



Note that sub-trees are output in left to right order.

The tree cannot be reconstructed from this form. To make reconstruction possible, additional information is added. The output for each non-leaf node is preceded by one or two values: first a negative number to specify ordering

-1 for pre -2 for in -3 for post

and then for pre or post ordering, a positive number, being the number of sub-trees of the node. The second number isn't needed for in-order nodes as they always have exactly two sub-trees.

For the sample tree this gives: -1 3 5 -3 1 -3 2 2 8 1 7 -2 8 3 9 2

Your task is to read obfusticated trees, and display the tree data in pre-order sequence.

For the sample tree the output should be 5 7 1 2 8 3 8 9 2

Input

Input will consist of a sequence of problems. There will be one line for each problem. A problem line will hold a space separated series of integers: the first will be the number of code values in the problem C: $0 < C <= 2000$; followed by C integer values V_i: $-3 <= V_i <= 9$, being the obfusticated tree data. Note that the numbers in the tree are always in the range 0 to 9. Negative values are only present for coding the ordering. End of input will be denoted by a line with a single 0.

Output

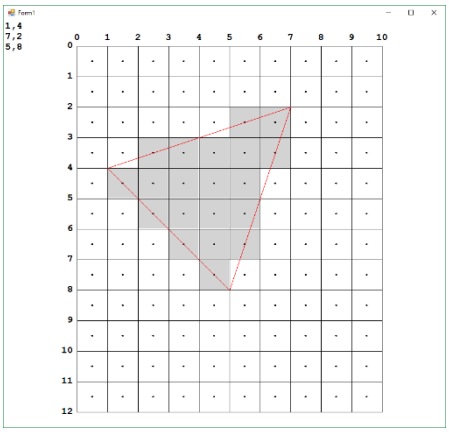
Output will be a single line per problem, with space separated numbers, being the pre-order sequence of the data in the tree.

Sample Input	**Output for Sample Input**
16 -1 3 5 -3 1 -3 2 2 8 1 7 -2 8 3 9 2 0	5 7 1 2 8 3 8 9 2

New Zealand Programming Contest 2016

PROBLEM J **RASTERIZATION** **30 POINTS**

In computer graphics, one of the fundamental operations is rasterization of a triangle. Given pixel coordinates of the corners of a triangle, the rasterization algorithm decides which pixels to colour. Consider the following image. It shows a 'screen' with 12 rows of 10 columns of pixels. The coordinate system used for triangle vertices is as shown – integer values at the edges of the pixels. Pixel centres are at half integral coordinates. The coordinates of the corners of the triangle are shown at the top left of the image, with the x or horizontal coordinate first and y coordinate second. Note that the y coordinate increases downward. The coordinates of the triangle follow a standard graphics convention in that they are listed in clockwise order. Any cyclic permutation of the three corners is considered equivalent.



The rasterization technique chosen is to colour a pixel if and only if its centre is inside the triangle, or on an edge of the triangle. With integer coordinates for the corners it is possible (and required) to decide whether a pixel centre lies exactly on the line or not.

Your task is, given the coordinates of the corners of a triangle, calculate the number of pixels coloured following the technique described. In the example shown 20 pixels are coloured.

Input

Input will consist of a sequence of problems. There will be one line for each problem. A problem line will hold a space separated series of integers: x1, y1, x2, y2, x3, y3. All x and y coordinates will be in the range 0 to 2000 inclusive. You may assume that the screen size is 2000 by 2000. End of input will be denoted by a line with six zeroes (which should not be processed).

Output

Output will be a single line per problem, with the number of coloured pixels.

Sample Input

```
1 4 7 2 5 8
0 0 0 0 0 0
```

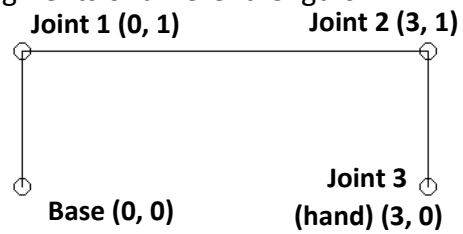
Output for Sample Input

```
20
```

New Zealand Programming Contest 2016

PROBLEM K **IK ROBOT** **30 POINTS**

Moving a robot arm is a surprisingly complex process. If the arm has a number of segments, the angles of each joint combine to determine the final position of the 'hand'. In most industrial robotics, the setting of the individual joint angles is left to software and the robot's operator just specifies where they want the hand to be. The process for determining suitable angles is called Inverse Kinematics. This problem asks you to find an Inverse Kinematic solution for a robot arm with up to 30 segments of different lengths.



Fortunately your robot arm has limitations on its joint angles. The arm is anchored at its starting point or base – bottom left joint in the diagram – and can pivot freely about that anchor. All the other joints are required to have identical angles. In the diagram the arm is anchored at coordinate (0, 0). The first segment is pivoted by 90 degrees (anticlockwise from the horizontal). The angles at joints 1 and 2 are also 90 degrees (anticlockwise). Segment lengths are 1, 3, and 1 respectively. As a result the hand is at location (x = 3, y = 0).

Your task, given an arm description and a location for the hand is to determine the two angles (pivot of the base) and common angle used by the remaining joints.

Input

Input will consist of a sequence of problems. The first line for each problem will have three space separated numbers: N, X and Y. N is the number of segments: 2 <= N <= 30. X and Y are the required coordinates for the hand and will be floating point values. This line will be followed by N lines, holding the lengths of the segments (floating point values). Segment lengths and the X and Y coordinates will all be in the range 1 to 10 (inclusive). End of input will be denoted by a line with three zeroes (which should not be processed).

Output

Output will be a single line per problem, two space separated values – the pivot angle at the base and the common angle of the remaining joints. Angles will be rounded floating point values output with exactly 3 decimal digits. They will represent angles in degrees. Input data has been chosen to avoid marginal rounding values on the last digit. Angles will always be in the range 0 to 180 degrees (inclusive)

Sample Input

```
3 3.0 0.0
1
3
1
3 0.0 2.0
1
2
1
0 0 0
```

Output for Sample Input

```
90.000 90.000
180.000 90.000
```

New Zealand Programming Contest 2016

PROBLEM L **POKEMON** **30 POINTS**

There are many different kinds of pokemon. The NZPCA (New Zealand Pokemon Credential Authority) is building image processing software and hardware for pokemon identification. They have decided on an agent architecture. An agent is a device with a camera and a specialized algorithm, designed to recognize a single pokemon feature (eg: has wings, has a tail, etc). Agent devices have been designed and built for each of the features deemed important. Unfortunately, the algorithms are not 100% reliable, in that they don't work for all orientations and movement states of pokemon. Fortunately, the algorithms never produce false positives – they either correctly recognize their feature or return 'false'. When a number of identical agents are deployed to analyse a single pokemon, it turns out that their chances of false positives are statistically independent. As a result, reliability can be improved by using several of each kind of agent, and the joint reliability can be calculated reliably. The pokemon system requires at least one reliable result from each kind of agent in order to successfully complete an identification.

Given a budget B, a number F of features, and a cost c_f and reliability r_f for each of the F agent types, your task is to decide on the number of each kind of agent to deploy in order to maximize the overall reliability of the identification system.

Example

The system has three detectors (wings, tail, legs – back in the day when there were only 8 types of pokemon). Your budget is $105. The detectors cost $30, $15 and $20, and have reliabilities of 0.9, 0.8, and 0.5 respectively.

The best configuration, with a cost of $100 and reliability of 0.648 has one of the first detector and two of each of the others. Cost is $30 * 1 + $15 * 2 + $20 * 2 => $100. Reliability is calculated as follows. The chance of the detector failing for the first feature is (1 – 0.9) => 0.1. The chance of both detectors failing for the second feature is (1 – 0.8)^2 => 0.04. For the third (1 – 0.5)^2 => 0.25. This gives reliabilities for the systems at each stage of (1 – 0.1), (1 – 0.04), (1 – 0.25) => 0.9, 0.96 and 0.75. The overall reliability is therefore 0.9 * 0.96 * 0.75 => 0.648.

Input

Input consists of a sequence of problems. For each problem, there will be a line with integers B (budget in dollars: 0 < B <= 10000) and F (number of features: (0 < F <= 30), followed by F lines each holding an integer c_f (cost of the f^{th} detector in dollars) and a real number r_f.(reliability of the f^{th} detector). Reliability is the probability that the detector will return true if the pokemon being examined does have the feature for which that detector was designed. You may assume that the budget is sufficient to pay for at least one of each kind of detector. Input is terminated by a line with two zero values.

Output

One line per problem with the cost of an optimal system and its overall reliability. If there are multiple solutions with the same highest reliability, then report one with the lowest cost.Overall reliability should be rounded and reported to exactly 4 decimal places. The problem set has been chosen to ensure the number is such that the 4^{th} decimal digit is not close to 5.

Sample Input

```
105 3
30 0.9
15 0.8
20 0.5
0 0
```

Output for Sample Input

```
100 0.6480
```

New Zealand Programming Contest 2016

PROBLEM M **HASHING** **30 POINTS**

A string hashing function is an algorithm that turns arbitrary strings into numbers. If S is a string of length N, one simple hashing function is

```
S[0] * 31^(N-1) + S[1] * 31^(N-2) + ... + S[N-1]
```

where S[i] is the ASCII code of the i'th character and ^ indicates exponentiation. The computation is done using big integer arithmetic (it never overflows).

Note: In most programming languages ^ does NOT perform exponentiation.

Here are some example strings and the corresponding values computed by this hashing function:

1. "ab" (N = 2, ASCII codes of characters: 97 98)
   hash = 97 * 31 + 98 = 3105

2. "Hi!" (N = 3, ASCII codes of characters: 72 105 33)
   hash = 72 * 31^2 + 105 * 31 + 33 = 72480

3. "IJ!" (N = 3, ASCII codes of characters: 73 74 33).
   hash = 73 * 31^2 + 74 * 31 + 33 = 72480

Sometimes different strings have the same hash (like the second and third strings above).

A string is considered valid if all its characters have ASCII codes in the range 32 to 126 inclusive.

Your task is to count the number of valid strings that have the same hash as a given input string.

Input

Input consists of several lines. Each line describes a string using a sequence of space-separated integers in the
format

```
N S[0] S[1] ... S[N-1]
```

N is the length of the string (1 <= N <= 1000) and S[i] is the ASCII code of the i'th character (32 <= S[i] <= 126).

A value of N = 0 indicates the end of the input and should not be processed.

Output

For each test case, print a single integer: the number of valid strings that have the same hash as the input string (including the input string itself), modulo 1,000,000,007 (10^9 + 7).

[Turn over for sample input and output]

Sample Input

```
2 97 98
3 72 105 33
3 32 32 32
0
```

Output for Sample Input

```
3
12
1
```

Explanation

The first line in the input describes the string "ab" which hashes to 3105. There are two other valid strings with the same hash: "bC" (ASCII: 98 67) and "c$" (ASCII: 99 36), so in total there are three strings. The string which in ASCII is 100 5 also hashes to 3105 but is not valid because 5 is outside the range 32 to 126.

The second line corresponds to the string "Hi!" which hashes to 72480. There are 12 valid strings with this hash.

The third line corresponds to a string with three spaces which hashes to 31776. There are no other valid strings that hash to 31776.

New Zealand Programming Contest 2016

PROBLEM N **WHAT NEXT** **100 POINTS**

When you are entering program text into a development environment editor, there is usually a prediction system which will use your current context to help you by listing completions of the symbol you are typing or a symbol you could type next. Your task is to write a program to provide such predictions for the language "NZPC Speak"

The rules for writing NZPC Speak programs are as follows

- A **program** in NZPC Speak is a **block**.
- A **block** is a sequence of **declaration**s and **statement**s enclosed in braces '{' '}'.
- A **declaration** is 'int' or 'float' followed by a comma separated list of names (there must be at least one name) terminated by a semicolon.
- A **statement** is either a **block** or an **assignment_statement**
- An **assignment_statement** is a name, followed by '=', followed by an **expression**, terminated by a semicolon. There is no type checking on assignment.
- **Expression**s are names or positive integers with operators '+', '-', '*' and '/' only. Integers and floats may be mixed without restriction. (There are no parentheses.)
- Names are made from upper and lower case letters (considered distinct), digits and underscores. The first character in a name must not be a digit.
- Names used in expressions must have been declared before they are used.
- Names in inner scopes (nested blocks) mask names in their enclosing scope(s) from their point of declaration until the '}' that closes the scope. If a name is declared more than once in a block, the second… declaration replaces the first… from the point of declaration.
- Integers are sequences of digits.
- Reserved words 'int' and 'float' may not be used as names.
- Programs may optionally have white space (spaces and newlines only) between tokens.
- Reserved words 'int' and 'float' must be followed by at least one white space character.

The text in the box is an NZPC Speak program. Your program will be presented with truncated syntactically correct NZPC Speak programs and will determine the 'next' symbol.

```
{
    int abc, abcd, abcdef;
    float ab, x, xy, xz;
    abc = 12 * xy + 99;
    {
        float abc;
        int xy;
        xy = abc + abcd;
    }
}
```

- Next symbols include punctuation and operators
- When input is truncated on a name or reserved word, such that further characters of the name could be typed (ie: no space or punctuation yet entered), then the next symbols to be listed are just the names that are valid at that point. This rule applies even when the name is already complete (see sample input).
- When entering a (possibly) new name in a declaration, no next symbols are listed.

Input

Input is a single line holding the number of problems to be solved. Each problem takes the form of a number of lines (between 1 and 100 inclusive) of program text. The final line of text for each problem will end with an '@' character.

Output

For each problem the output will be a line with 'Problem # where # is the problem number (1, 2, …) followed by zero or more lines, each holding a predicted 'next' symbol. Next symbols will be listed in alphabetic order, where alphabetic is interpreted as the ASCII code ordering with upper and lower case characters interleaved as follows.
This order for the characters used in NZPC Speak is: <space> * , / ; _ { } + = 0…9 a A b B … z Z
Names will be output followed by their types (int or float) in parentheses.

Sample Input

```
6
{@
{in@
{int@
{int @
{int abc; abc @
{int abc, ab;
 a@
```

Output for Sample Input

```
Problem 1
{
}
float
int
Problem 2
int
Problem 3
int
Problem 4
Problem 5
=
Problem 6
ab(int)
abc(int)
```

New Zealand Programming Contest 2016

PROBLEM O **THE RESISTANCE** **100 POINTS**

"The Resistance pits a small group of resistance fighters against a powerful and corrupt government. The resistance has launched a series of bold and daring missions to bring the government to its knees. Unfortunately, spies have infiltrated the resistance ranks, ready to sabotage the carefully crafted plans. Even a single spy can take down a resistance mission team, choose your teams carefully or forever lose your chance for freedom."

In this simplified version of the board game there are N players (2 <= N <= 15) of which S are randomly chosen to be spies (1 <= S <= N-1). A player knows whether or not he/she is a spy, but not what the other players are.

A game consists of 5 missions. On each mission, a number of players are chosen (the number varies depending on the mission). Any spies among the chosen players can secretly choose to sabotage the mission. The number of sabotages is publicly revealed at the end of each mission, but not which players were responsible for them. For this problem, each spy on the mission decide whether to sabotage or not using a fair coin (i.e. with probability 1/2).

You will be given the description of M missions that have already been played (1 <= M <= 4). The i'th mission had C[i] players (1 <= C[i] <= N) and F[i] sabotages (0 <= F[i] <= min(C[i], S)). You will also be given the ID's of the players on each mission (each player has an ID between 1 and N).

Your task is to choose a team of Q players for the next mission (1 <= Q <= N-S) and print the probability that none of them are spies. You must choose the team so that it maximises this probability.

Input

Input will consist of multiple game descriptions.

The first line of each description will have N, S, M, and Q separated by spaces (the number of players, spies, missions, and players you'll need to choose).

The second line will have M space-separated integers, the values of C[i] (number of players on each mission).

The third line will also have M space-separated integers, the values of F[i] (number of sabotages on each mission).

The following M lines will describe the players that were chosen on each mission. The i'th line have C[i] space-separated integers, the ID's of the players chosen for the i'th mission.

Input will be terminated by the line "0 0 0 0", which should not be processed.

Output

Print the probability that the optimal team of Q players does not have any spies. Print your answer as a floating point number with exactly 5 decimal places. The input has been constructed so that an error of up to 10^-6 will not affect the rounded answer. Do not print the members of the team.

Sample Input

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

Output for Sample Input

```
1.00000
0.30769
0.25000
```

Explanation

In the first game there are 4 players of which 2 are spies. There was 1 mission, and the task is to choose 2 non-spies. The first mission had 2 players and 2 sabotages. Players 1 and 2 were on the mission, so they must be the spies. Players 3 and 4 should be chosen for the next mission, and are guaranteed not to be spies (probability = 1, i.e. 100%).

The second game is similar, except there were no sabotages on the mission. Choosing the optimal team is a bit harder:

The prior probability that players 1 and 2 are spies but neither sabotages is 1/6 * 1/2 * 1/2 = 1/24 (there are 6 possible pairs of spies, the probability that player 1 doesn't sabotage is 1/2, and the probability that player 2 doesn't sabotage is 1/2).

The prior probability that players 1 and 3 are spies but player 1 doesn't sabotage is 1/6 * 1/2 = 1/12. The same applies for the other three cases where one spy is on the mission and the other spy is player 3 or player 4.

The prior probability of players 3 and 4 being spies and having no sabotages is 1/6 * 1 = 1/6 (six spy configurations, and if players 3 and 4 are the spies there can't be any sabotages on that mission).

The conditional probability that players 3 and 4 are the spies given that there were no sabotages on the mission is

(1/6) / (1/24 + 4 * 1/12 + 1/6) = 4/13

So if players 1 and 2 are chosen for the next team, the probability that the team is free spies is 4/13 ~= 0.3076923. This is the best choice.

In the third game there were two missions, and both had one sabotage. The best choice is to take one player from the first mission and one player from the second mission (e.g. player 1 and player 3). Each player has a 1 in 2 chance of being a spy, so the probability that neither is a spy is 1/4.

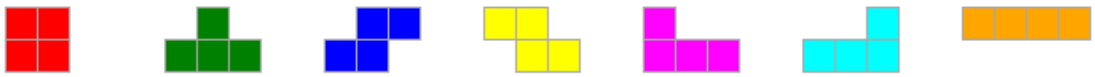
New Zealand Programming Contest 2016

PROBLEM P **RTETRIS** **100 POINTS**

It is a little known fact that the rules of the modern Tetris game have been eased to accommodate the limitations and attitudes of soft modern people. Back in the day, the game, then known as rtetris (the rt being pronounced as in ancient Klingon) was tougher. Play was public, and players who did not do well were subject to ritual humiliation, and occasionally (if things got out of hand) internet deprivation by the so-called 'denial of rtetris' attack (DOR for short). Rewards for winners, on the other hand were great. As always in sports, the intense pressure led to a culture of cheating. Cheating took two main forms: the ubiquitous steroid abuse in which players took steroid inhibiting drugs to prevent their overworked fingers developing muscle that would slow keyboard action; and the use of "helper" algorithms to generate game play strategies.

There is to be a revival of Rtetris and to assist top players (by which we mean cheats) you (a programmer with a lust for glory, and minimal ethics) have been asked to recreate one of the ancient "helper" algorithms.

The game of Rtetris works as follows. Like ordinary Tetris, there is a pit, into which pieces are dropped. The pit has integer width and height. Pieces are made of four unit blocks. There are seven distinct piece types.



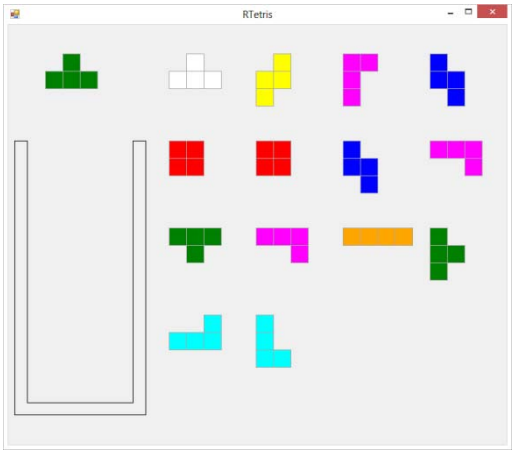
The game proceeds as a series of plays. On each play, a piece is positioned above the pit. The player can rotate it in multiples of 90 degrees; and move it left or right. When satisfied, they drop the piece into the pit. It moves down as far as possible, with the constraint that it cannot overlap the bottom of the pit or any other piece already in the pit. If the piece cannot fall to a level completely inside the pit, the game ends. Next, the piece can be thought of as disassembling into its four constituent unit blocks. The game then checks for full rows of blocks. Full rows are removed and the blocks above them drop down a level to fill the vacated spaces. To this point the game is just like modern Tetris. Next however, there is a new test. No vertical column of blocks is permitted to have gaps in it after a move has completed. If there is a column with a gap the game ends.

RTetris also differs from modern Tetris, in that there are a fixed number of pieces available for play (the hand), generated randomly before the start of play, and the whole hand for a game is visible to the player. It is therefore possible to win a game, by placing the whole set of pieces under the conditions specified above. There are also games where the player loses immediately. If for example the first piece is  then there is no way of dropping it into the pit without violating the "columns may not have gaps" rule.

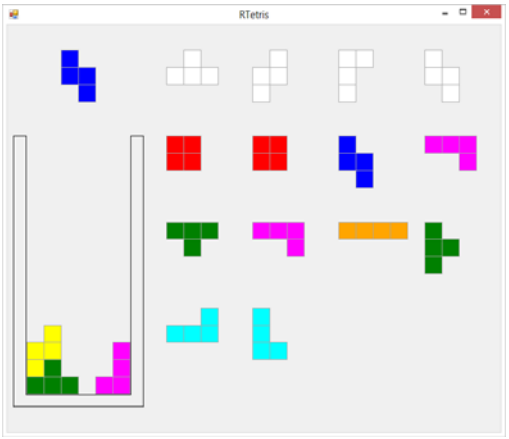
Visibility of all pieces leads to the game having deeper play strategies than modern Tetris. Your "helper" program must analyse game hands to find winning play strategies. Championship games were often played with very large hands.

Note: See 'Input' below for the actual pit size in use. Knowing this may be helpful in algorithm development.

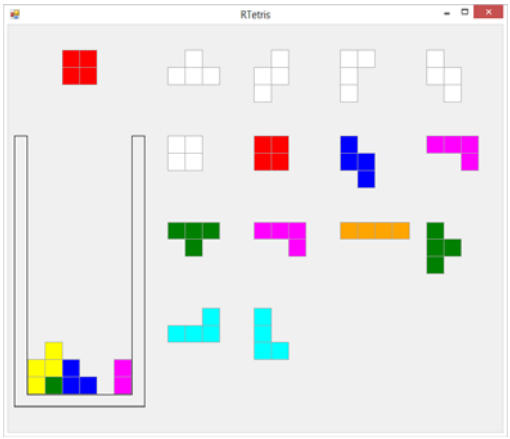
Here is an initial game state – playing with a 6 by 15 cell pit and a 14 piece hand. Note that the first piece has been moved over the pit, and is shown in outline in the hand. Pieces are played from the hand in the order dealt .



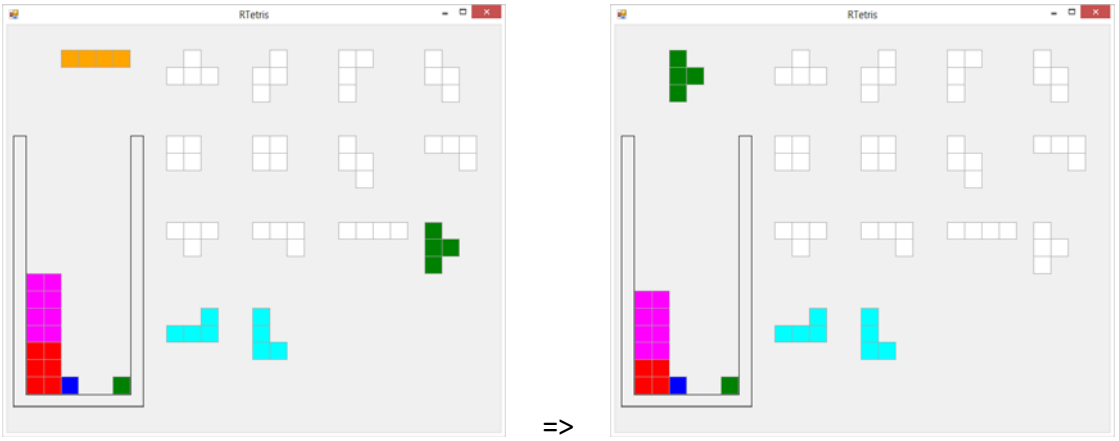
A little later, three pieces have been played and the fourth is in play (left below). Dropping the fourth piece will remove the bottom row, but leaves the game with a losing position (right below).



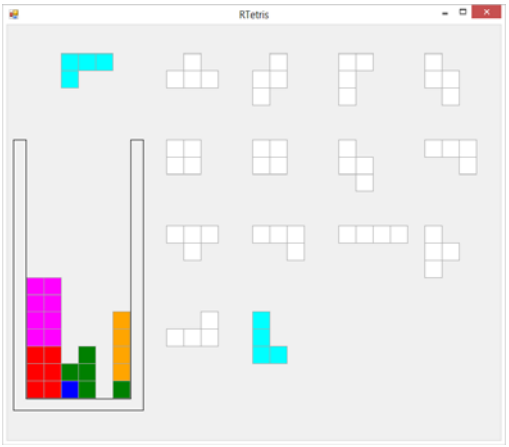
=>



Further into the game, playing a different strategy, a new situation arises. Dropping the horizontal piece looks as though it might leave gaps, but the fact that it completes a row makes it legitimate.



Here is another situation where a drop leads to a valid position, and a possible game win.



Input

A series of games. Each game starts with a line having three integers separated by spaces: N being hand size (1 <= N <= 200), W and H being pit width and height respectively. Your team has bribed a referee, and you know that W will always be 6, and H will always be 7. Next is a line with the pieces of the hand. Each piece is identified by a digit from 1 to 7, as shown below. Digits are separated by spaces. End of input is signalled by a line with three zeroes.

Output

One line per game with either “Game cannot be won”, or “Game can be won with N lines removed”, where N is the largest possible number of lines removed for a winning solution. Ie: you are required to find the number of lines removed in a best winning solution, where best means ‘most lines removed’.

Sample Input

```
14 6 7
2 4 5 3 1 1 3 5 2 5 7 2 6 6
0 0 0
```

Output for Sample Input

```
Game can be won with 9 lines removed
```