NEW ZEALAND PROGRAMMING CONTEST

2013

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.

In most games, the more you score the better, but not in golf! In a single round of golf you play 18 holes, and the fewer times you have to hit the ball to complete the round the better. In a tournament, you may have to play 4 rounds and will do well to score less than around 280.

In this problem you will be given scores from a series of tournaments and will have to find the best (ie lowest) score in each. It does not matter how many times the low score occurs, we just need to know what that score is.

Input

Input consists of a number of lines representing different tournaments. The first line for a tournament will contain a single positive integer n ($0 < n <= 100$) representing the number of scores to process in that tournament. The last line will contain the number 0 – do not process that line.

N lines follow the first line for a tournament, each consisting of a single positive integer. Each integer represents the total score for a player who completed the tournament. You have to find the best (ie lowest) score.

Output

For each tournament in the input, produce one line of output containing a single integer, the best score found for the tournament.

[Turn over for sample data]

Sample Input

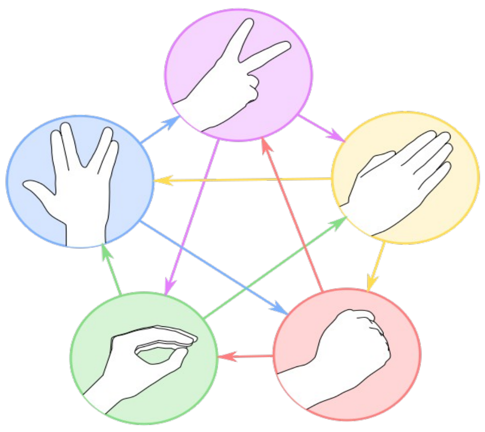
```
10

275

283

276

280

281

274

278

289

275

291

8

82

79

81

83

79

97

83

84

0
```

Output for Sample Input

```
274

79
```

This is a variation on the simple rock, paper, scissors game most of us have played. It was made famous by the television show *The Big Bang Theory*, but actually invented by software engineer Sam Kass and his wife, technical writer Karen Bryla.

Each player makes their hand into one of the five gestures as shown in the diagram from Wikipedia. At an agreed moment, both show which gesture they have chosen. Each gesture beats two gestures and loses to the other two. The rules are shown below. In each description, the gesture which comes first beats the other gesture.



Scissors cuts paper
Paper covers rock
Rock crushes lizard
Lizard poisons Spock
Spock smashes scissors
Scissors decapitates lizard
Lizard eats paper
Paper disproves Spock
Spock vaporizes rock
Rock crushes scissors

If both players present the same gesture, that game is a draw.

In this problem you will be given data for a number of sets of games. All you have to do is to work out and display the score for each set of games.

Input

The input begins with a single integer, S ($0 < S <= 25$), on a line by itself. This is the number of sets of games which follow.

Each set of games begins with a line containing the names of the two players, N1 and N2, separated by a space. The next line has a single integer, G ($0 < G < 100$), on a line by itself; this is the number of games for that set. There then follow G lines each containing two letters representing the gestures chosen by N1 and N2 respectively in that game, the letters being separated by a space. Each letter will be one of the upper case letters `R`, `P`, `S`, `L` or `K`. These are the first letters of rock, paper and lizard, with S representing scissors, and K representing Spock.

Output

Output consists of a single line for each set of games in the input. It is in the following format:

```
N1 x N2 y draws z
```

Where `N1` and `N2` are the names of the two players in the order they were presented in the input, and `x` and `y` are their respective scores. `z` is the number of drawn games.

[Turn over for sample data]

Sample Input

```
2

John Sally

5

R P

P S

S R

L S

K P

Siong Li

7

L L

P L

R S

R P

S L

K K

P P
```

Output for Sample Input

```
John 0 Sally 5 draws 0

Siong 2 Li 2 draws 3
```

In New Zealand we use coins for 10c, 20c, 50c, \$1 and \$2, plus notes for \$5, \$10, \$20, \$50 and \$100. In this problem you simply have to work out the monetary value of a series of notes and coins.

Input

Input will consist of a series of lines each containing 10 integers in the range 0 to 100 inclusive. Each number represents the quantity of one of the coins or notes listed above. The first number represents the number of 10c coins, the second the number of 20c coins and so on in the order given above, with the last number representing the number of \$100 notes. The last line will contain ten zeros – do not process this line.

Output

For each line of input, output a single monetary value in the form \$d.dd, ie a dollar sign, at least one digit for the dollars, a decimal point and two digts for the cents.

Sample Input

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

0 0 0 0 1 0 0 0 1 6

1 1 1 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0
```

Output for Sample Input

```
$80.70

$652.00

$0.80
```

Explanation

1 x 10c	\$0.10	1 x \$2	\$2.00	1 x 10c	\$0.10
3 x 20c	\$0.60	1 x \$50	\$50.00	1 x 20c	\$0.20
2 x \$1	\$2.00	6 x \$100	\$600.00	1 x 50c	\$0.50
4 x \$2	\$8.00	**Total**	**\$652.00**	**Total**	**\$0.80**
5 x \$10	\$50.00				
1 x \$20	\$20.00				
Total	**\$80.70**				

When I was 12, my grandmother taught me about binary numbers. Well, she was a maths teacher and wanted to have some fun on her 64th birthday. She put 7 candles on her cake and only lit the first one. She asked my sister and me if we knew why. After explaining the binary system, she asked us to work out how many candles we would need on our birthdays (the smallest number), and how many our parents would need. We then had to tell her how many of those candles had to be lit.

In this problem, you have to help anyone faced with the same question. On their birthday, with a lit candle representing a 1 and an unlit candle a 0, what is the smallest number of candles required to represent their age in binary, and how many should be lit.

Input

Input will consist of a series of lines each containing a single integer in the range 1 to 120 inclusive, representing a person's age. The last line will contain 0 – do not process this age.

Output

For each line of input, output two numbers separated by a space; the number of candles required followed by how many had to be lit.

Sample Input

```
12

15

36

38

64

0
```

Output for Sample Input	Explanation
4 2	1100
4 4	1111
6 2	100100
6 3	100110
7 1	1000000

Here is a simple mathematical function. Take any four positive integers w, x, y, z, for example

```
4 8 6 3
```

Now apply a simple subtraction function - work out w-x, x-y, y-z and z-w:

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

Remove the minus signs (so we are using the absolute values of the subtractions):

```
4 2 3 1
```

Now apply the same function to the new numbers

```
2 1 2 3
```

Keep applying the function until all 4 numbers are zero

```
1 1 1 1
0 0 0 0
```

So our sequence is:

```
4 8 6 3
4 2 3 1
2 1 2 3
1 1 1 1
0 0 0 0
```

So, starting with the numbers 4 8 6 3 we have to apply the function 4 times until we get all zeros. In this problem you will be given a series of sets of 4 numbers, and for each set must work out how many times the function has to be applied to get all 4 numbers down to zero.

You may be surprised to know that the number is usually quite small!

Input

Input will consist of a series of lines each containing 4 positive integers separated by spaces. The final line will contain all zeros – do not process that line. The maximum value for each number is 1000000.

Output

For each line of input, output a single integer, the number of times the subtraction function has to be applied until all 4 numbers are zero.

[Turn over for sample input and output]

Sample Input

```
4 8 6 3
3 5 9 1
10 11 12 13
0 0 0 0
```

Output for sample input

```
4
7
5
```

PROBLEM F — ANAGRAMS — 10 POINTS

A word is an anagram of another word if it contains the same letters but in a different order. For example `pat`, `tap` and `apt` are all anagrams of each other.

In this problem, you will be given a number of scenarios. Each will contain a dictionary of words, followed by a series of words for you to test. You have to decide whether or not each of the test words is an anagram of one of the words in the dictionary. No word in the dictionary will contain the same letters as any other word in the same dictionary.

Input

Input begins with a line containing a single integer, S (1 <= S <= 20) which is the number of scenarios that follow.

A scenario begins with a line containing a single integer, D (1 <= D <= 25) which is the number of words in the dictionary for the scenario. This is followed by D lines, each containing a single word.

The dictionary is followed by a line containing a single integer, T (1 <= T <= 20) which is the number of test words to follow. This is followed by T lines, each containing a single word. Words in the dictionary will not be repeated as test words, and test words will not be repeated in the same scenario.

In this problem a word is a collection of no more than 20 lower case letters (no spaces, hyphens or any other type of punctuation will occur).

Output

The output for each scenario will start with a line containing the scenario number, in the format

```
Scenario n
```

where n is the scenario number. Scenarios are to be output in the same order as they appear in the input, the first being number 1.

This will be followed by one line for each test word in the order they appear in the input. The line will start with the test word, followed by a space, followed by the word from the dictionary that is an anagram of the test word. If there is no anagram, `<none>` will be output instead.

Sample Input

```
1
2
apt
cat
4
pat
tap
dog
act
```

Output for sample input

```
Scenario 1
pat apt
tap apt
dog <none>
act cat
```

At an educational institution on New Zealand's west coast, they run a logic skills paper. During the course of a term, students take 4 tests, A, B, C and D. If they pass a test first time, they score 5 marks, second time 3 marks, and if it takes them three or more goes, only 1 mark. The tests are unrelated and may be taken in any order at any time during the term. As you can see, a perfect mark for the term would be 20 marks, but an A grade may be obtained for a score of 16 or more.

In this problem you will be given sets of test marks each for a different term. You have to calculate how many A grades were obtained for each term.

Input

Data for a term begins with a single integer, S (0 < S <= 25), on a line by itself. It represents the number of students taking the paper during the term. If this line contains 0 it marks the end of input.

S lines then follow, each giving data about one student. The line starts with a 3 digit ID number, which is followed by a space and the student's name containing no more than 32 characters.

The student list is followed by a set of results for the term. Results are in the order in which the tests were taken. A line containing just `0 # #` marks the end of input for the term.

A result consists of a student ID from the student list, followed by a space, followed by a test letter (A, B, C or D in upper case), followed by a result (an upper case letter P for pass, F for fail). Once a student passes a test, they will not take it again.

Output

For each term in the input, a single line will be output. It will be in the form

```
Term t A grades a
```

where `t` is the term number (in the order of the input, the first term being 1)

and a is the number of students in that term who scored 16 marks or more.

[Turn over for sample data]

Sample Input

```
4
106 Angela Carter
124 Barry McCaw
156 Josephine Henry
172 Thomas Hansen
156 A P
106 C F
124 B F
156 C F
124 B F
124 D F
172 A F
106 B F
172 B F
172 A P
124 A P
106 D F
156 D P
124 C P
172 C F
124 D P
156 C F
172 C P
156 B P
156 C P
124 B P
172 D F
172 B P
106 A F
106 B P
172 D P
0 # #
0
```

Output for sample input

```
Term 1 A grades 1
```

Explanation

Josephine Henry took 3 goes to pass Test C, scoring 1 mark, but passed the others first time scoring 15. This gave her 16 marks, and the only A grade.

Angela Carter passed Test B second time but did not pass any others for just 3 marks.

Barry McCaw passed Test A and Test C first time for 10 marks, but took 3 goes to pass Test B and 2 to pass Test D scoring 4 marks to make a total of 14.

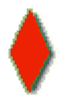
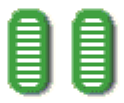
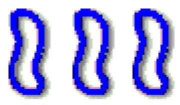
Thomas Hansen passed every test second time to score 12 marks.

PROBLEM H SET GAME 10 POINTS

Set is a card game played with a pack of 81 unique cards. Each card in the pack has 4 properties, having one of the three possible values for each property.

Colour – Red (R), Green (G), Blue (B)	**Number** 1, 2, 3,
Shape – Diamond (D), Oval (O), Squiggle (S)	**Fill** – Filled (F), Shaded (S), Empty (E)

This sample contains 3 cards and illustrates all the variations. As this is not in colour, note that the left card is red, the middle is green and the right card is blue.

		
RD1F	GO2S	BS3E

Below each card is its representation, being the value for its colour, shape, number and fill in that order.

The 3 cards illustrated form a set. A set is a collection of 3 cards such that for each property all 3 cards are either the same or different. The 3 cards above are different for all 4 properties. Here are some other examples.

RD1F	RD1S	RD1E	SET. Same colour, shape and number, different fill.
RD1F	RD1S	RD2E	NOT a set. Two are 1s but one is not.
RD1F	GO2F	BS3E	NOT a set. Two are filled but one is not.
BS3E	BD1E	BO2E	SET. Same colour and fill, different number and shape.

In this problem you will be given data for groups of 3 cards. You must decide whether or not the 3 cards in each group form a set.

Input

Input begins with a single integer, N, on a line by itself (0 < N <= 100). There then follow N lines, each line representing a group of 3 cards. Each card will be represented by 4 characters as described above. The cards will each be separated by a space.

Output

For each line of input, produce one line of output containing either the word "Set" or the phrase "Not a set".

[Turn over for sample data]

Sample Input

```
4

RD1F RD1S RD1E

RD1F RD1S RD2E

RD1F GO2F BS3E

BS3E BD1E BO2E
```

Output for Sample Input

```
Set

Not a set

Not a set

Set
```

PROBLEM I IKEA 30 POINTS

You've finally graduated and are moving overseas to take a lucrative offer with <insert big tech company>. You need to furnish your swanky new apartment so you decide to buy everything from IKEA, a Swedish furniture company infamous for its "some assembly required" products. Unfortunately, when you get home you drop all the instruction manuals, scattering the pages all over the floor. Sadly, IKEA doesn't number its pages.

In order to get through the process of assembling your furniture as quickly as possible, you decide sort the scattered pages into a new super manual. For speed, the pages need to be arranged such that any parts or subassemblies a page needs for assembly, are covered by some previous page in the new super manual.

Luckily it is clear from any given page, which others depend on it.

You decide to number all the pages in the order in which you picked them up, and write a computer program to figure out which order they should go in. In case the order of two pages is ambiguous, the one with the lower number should come first.

Input

The input will consist of a number of test cases. In each case, the first line will contain a single number,
P (1 <= P <= 1,000,000 (you bought a lot of furniture)), the number of pages. The following P lines will describe the dependencies between the pages. The i'th of those lines will begin with an integer, k_i, followed k space separated integers, d_i_j, sorted in ascending order. Each integer represents the fact the parts on page i are need for page d_i_j. 0 <= sum of k_i <= 10,000,000

The input will be terminated by test with 0 pages, i.e. P = 0.

Output

For each test case you will output a single line of P space-separated integers, the order of the pages in the super manual.

Sample Input

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

Sample Output

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

PROBLEM J MIXED ENGLISH/PERSIAN TEXT LAYOUT 30 POINTS

Most word processing software does text layout for a single language. Sometimes, however, we need to make documents that mix text from different languages. If the alphabets and layout conventions are similar, this is not especially difficult – for example when mixing French and English. However, mixed Persian/English text presents a particular problem. Persian is written right to left, where English is left to right.

كبوتر
من كبوترم
راه می روم و خیال می کنم
بابام داغم کرد
مامانم کبابم کرد
خواهر عزیزم
استخوان هایم را جمع کرد
سگ در کنارم
خونم را می لیسد!

Your task is to develop a text justification algorithm that allows for an arbitrarily mixed sequence of Persian and English words. There is no obviously correct solution. For this exercise it will be assumed that most text is English, with inserts of Persian. Accordingly, the text will have a generally left to right flow. Persian inserts will be in correct Persian word order, but their positions on the lines may not be what a Persian reader would expect.

We will adopt the convention that upper case letters represent English and lower case letters represent Persian, allowing development of the justification algorithm without having to manage different font systems. The input will be a series of letters and spaces in the order in which they are entered at the keyboard. Segments of text in 'Persian' will need to be reversed. For example:

The input: `this is ENGLISH and THIS IS persian`

should become: `si siht ENGLISH dna THIS IS niasrep`

Further, your output should be justified to a specified line width, spaces added as evenly as possible. For example, with a line width of 15 characters the sentence above should be output as:

```
si siht ENGLISH
dna   THIS   IS
        niasrep
```

The rules for justifying text are as follows.

- There is no hyphenation. Every word must fit completely on a line
- You may assume a monospaced font. I.e. all characters have the same width
- The final line in a block should not be filled to the edges. The words should be single spaced with a gap on the right (usually) or on the left as appropriate if all the words on the line are Persian.
- If only one word fits on a line it should be left aligned if English, and right aligned if Persian. If it is too long for the line width, then allow it to extend to the right regardless of whether it is English or Persian.
- When adding spaces to a line: if there are N word gaps, add spaces evenly until there are M < N spaces left to be added. Add one more space to each of the left hand M word spaces, regardless of language.
- Persian phrases should read from right to left. If a Persian phrase is split over two or more lines, the phrase should read in its correct order, reading from the top line to the bottom line involved, right to left along each (part) line. If the final part line of a multiline Persian phrase is followed by English on the same line, then the Persian should end at the left margin. (This is not natural for a Persian reader. It is ok where English is the predominant language.)

Input

Input consists of a series of text justification problems. Each problem begins with a line holding a single integer W in the range 1 to 100(inclusive). W is the required line width in characters. This is followed by a number of lines of text. Text consists of upper and lower case letters with single spaces separating words. There is no other punctuation. The total amount of text will not exceed 1000 characters (including spaces). The fact that the text is broken into lines is purely for convenience in preparing the input. Data on two successive lines should be treated as though it were on one line with a space character at the line break position. Words in the input will be wholly uppercase or wholly lowercase. I.e. English and Persian are not mixed within a word. Each problem finishes with a line holding a single # character. The # is not part of the text and should not be included in the output. End of input is indicated by a line holding a zero.

Output

Output for each problem should have a blank line, a line with the text “Block n”, where n is 1 for the first problem 2 for the second, etc. This should be followed by the justified text.

Sample Input

```
15
this is ENGLISH
and THIS IS persian
#
17
gnolootetiuqton gnoloothcumsidrowsihttub
AAA  BBB  CCC ddd eee fff AND
gg hh iii jjj kk
#
0
```

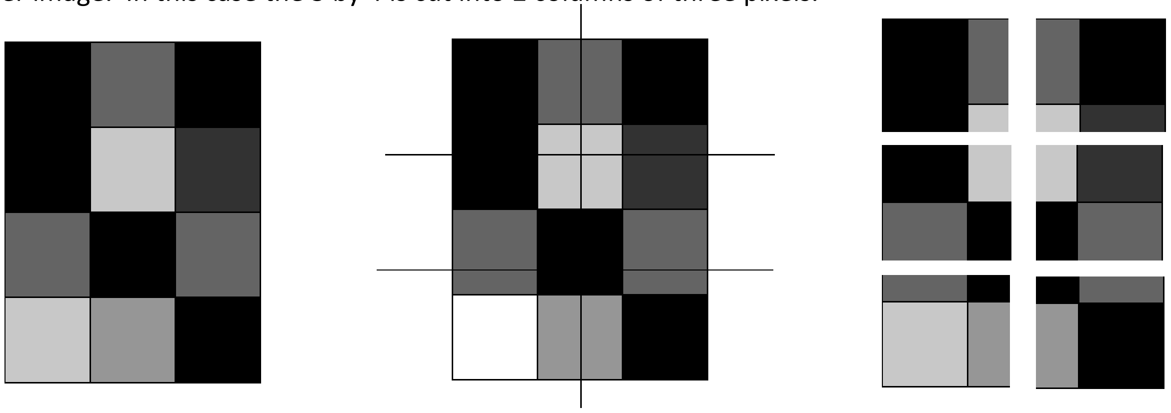
Sample Output

```

Block 1
si siht ENGLISH
dna   THIS   IS
        naisrep

Block 2
  notquitetoolong
butthiswordismuchtoolong
AAA  BBB  CCC ddd
fff eee AND hh gg
       kk jjj iii
```

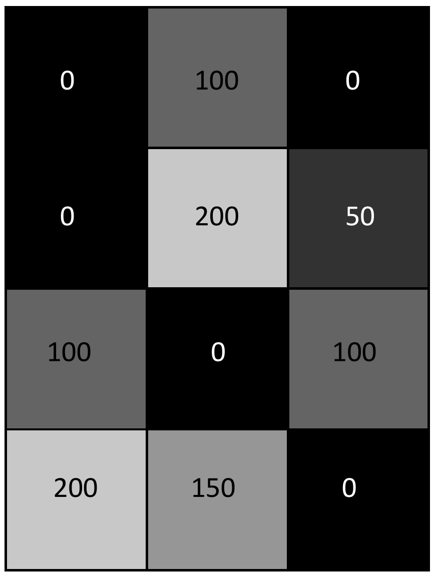
PROBLEM K **IMAGE SCALING** **30 POINTS**

Images in computer systems are often available at the wrong size (commonly very large, when captured by high resolution cameras), and so need scaling before use. Your task, in this problem is to write code to downsize an image. Given a greyscale image (pixel values in the range 0 to 255 inclusive) of size W by H pixels, you must generate a new greyscale image of size w by h where w <= W and h <= H. The smaller image usually cannot be a perfect reduction of the original, so some information will be lost. In computer graphics there are a number of ways of doing such scaling, each suited to different image uses. The method you are required to use is as follows. So that the arithmetic can be shown in full, the example given involves very small images. We will explain how a W=3 by H=4 image should be scaled to w=2 by h=3.

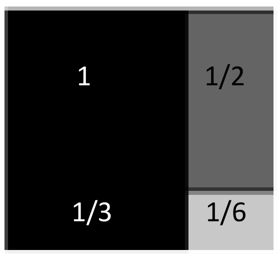
The method involves two steps. First, we cut the large image into the number and arrangement of pixels required by the smaller image. In this case the 3 by 4 is cut into 2 columns of three pixels.



Secondly, we calculate the average colour of each of each segment of the subdivided image, and use those values (truncated to integers) as the colours of smaller image. In the diagram below, we show the grey levels for each pixel of the original image and a close-up of the segments making up the top left pixel of the smaller image.



0	100	0
0	200	50
100	0	100
200	150	0



To calculate the colour of the top left pixel of the smaller image, we average the four contributions from the original image, weighted by area. In units of original pixels the areas of each of the four bits are 1, 1/2, 1/3, and 1/6 giving a total area of 2. The weighted sum of the colour contributions is

1 * 0 + 1/2 * 100 + 1/3 * 0 + 1/6 * 200 => 83 1/3.

The average colour is therefore 83 1/3 divided by 2, giving 41 2/3 The fractional part of the average is discarded, leaving an integer value – in this case 41. The sample output (below) shows the other output pixels.

Input
The input consists of a number of image scaling problems. For each problem the first line holds four integer values separated by spaces: W, H, w and h where 1 <= w <= W <= 100 and 1 <= h <= H <= 100. The following H lines each have W integer values, separated by spaces. These are (top to bottom) the colour (grey) values of the pixels (left to right) of the rows of the image to be scaled. Values lie in the range 0 to 255 inclusive. The end of input is indicated by a line with four zeros.

Output
For each scaling problem: Output one line with the problem number (starting from 1) as “Problem 1” or “Problem 2”, etc. This should be followed by h lines, each with w integer values, being the grey levels of the pixels of the scaled image.

Sample Input

```
3 4 2 3
0 100 0
0 200 50
100 0 100
200 150 0
1 1 1 1
128
0 0 0 0
```

Sample Output

```
Problem 1
41 50
66 83
154 54
Problem 2
128
```

PROBLEM L   DICTIONARY ATTACK   30 POINTS

One of the simplest forms of cypher is the deranged alphabet substitution cypher. The idea is that we rearrange the letters of the alphabet in some arbitrary (deranged) order, and write the alphabet in its normal order underneath.

```
p o n m t s r q w j u z y x d c b a h g f e l k v i
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

To cypher a message we replace each letter in the message with the one written above. So "`all is well`" will become "`pzz wh ltzz`". Note that there is no rule to prevent some letters being substituted as themselves. In the example above j is cyphered as j.

This is not a strong cypher, and there are many approaches to breaking it. A particular weakness in this case is the fact that the spaces between words have been preserved. This allows us to identify words, and know immediately the length of each word. The method of attack you will implement in this problem is based on this fact. It is a kind of dictionary attack.

Your task is to take a dictionary (a list of all possible words that might be used in a message), and use it to find all possible interpretations of an enciphered message. Be aware that the dictionary might be quite large (one supplied with the test set has 58,111 words).

For example, given a dictionary with just the four words: `aab   bbc   abb   cde`
The phrase: `zzq qqt` can be translated as `aab bbc`
The phrase: `prt` can be translated as `cde`
The phrase: `zzr` can be translated either as `aab` or as `bbc`

Input

Input consists of a series of problem sets. Each problem set consists of a dictionary and a number of cypher texts to translate. The first line of the dictionary holds an integer $0 < N > 60000$, the number of words in the dictionary. It is followed by N lines, each holding one word. A word is entirely in lower case letters with no special characters. No word is longer than 20 letters. Next is a line with a number $C > 0$ of texts to be translated, followed by C lines each holding a text. Texts are single spaced sequences of 'words'. No text is longer than 200 characters. End of input is signalled by a line with the number zero.

Output

For each text to be translated, output a line with 'Problem n' where n = 1, 2, 3, 4… Follow this with the possible translations you have found, one per line, lines being in alphabetic order. It is possible that some texts will have no translations. In that case you output the 'Problem n' header, but have no lines following it. The problem numbering should continue across problem sets.

Sample Input

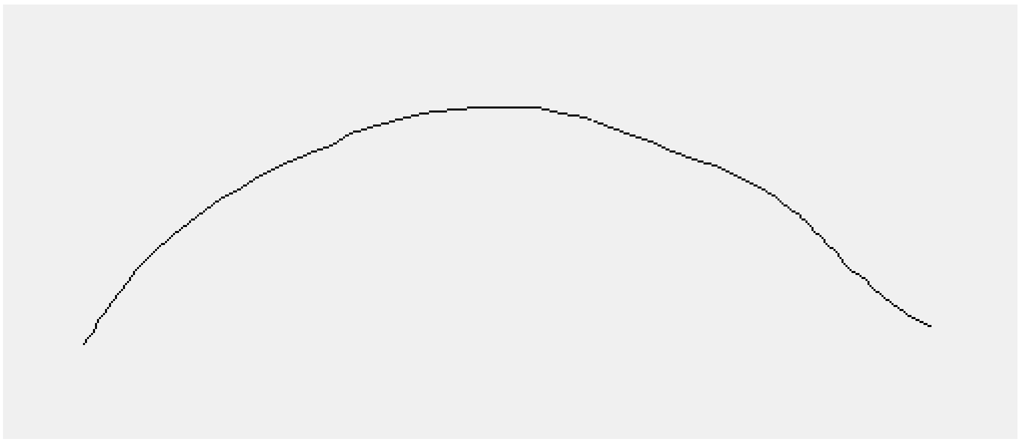
```
4
aab
bbc
abb
cde
3
zzq qqt
prt
zzr
1
soup
1
abcd
0
```

Sample Output

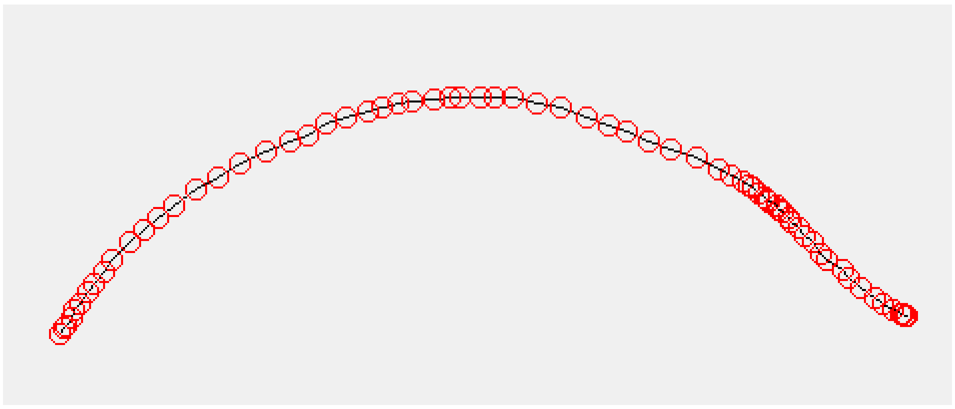
```
Problem 1
aab bbc
Problem 2
cde
Problem 3
aab
bbc
Problem 4
soup
```

PROBLEM M **CURVE SMOOTHING** **100 POINTS**

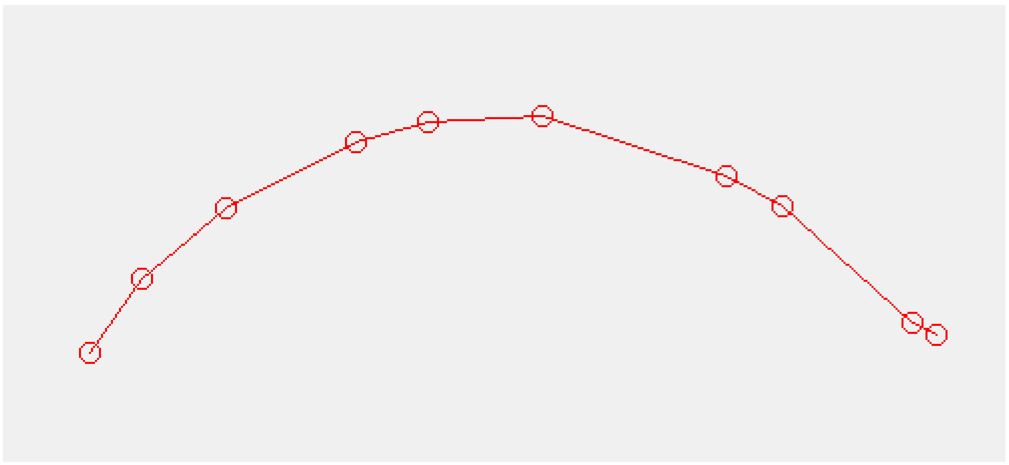
In computer sketch input we allow users to draw lines on a computer screen, using a mouse, a tablet, or some form of pen or finger touch system. Each of these input systems samples position quite frequently. For example, the following image shows a line drawn on screen using a mouse, by simply connecting successive mouse input locations by short straight line segments.



In the next image circles have been drawn around each of the points provided by the mouse system to the program.



The line doesn't look very good. In a drawing program for example, it would be better to smooth it to some extent. Smoothing is made difficult by the number of points, so smoothing algorithms usually work by reducing the number of points and then fitting a curve to those remaining. Reducing the 69 points in the sample above to 10 gives the following set (connected by straight lines)



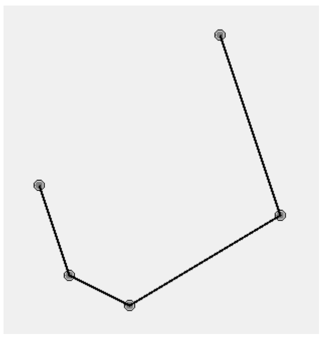
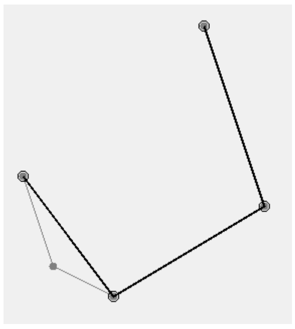
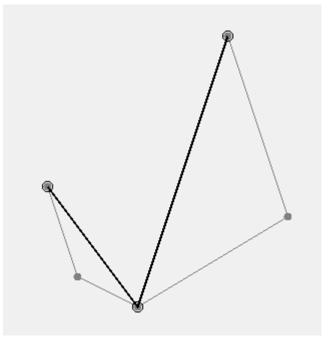
Your task is to write an algorithm to 'optimally' reduce the number of points on a line. You are not required to programming a subsequent smoothing step.

There are a number of ways to define 'optimal' in this problem. The one that you are required to use defines the optimal line as the one whose length (the sum of the straight line segments from point to point) is the closest to the original line length, defined as the sum of the distances between successive sample points. The first and last points of the original set should always be kept.

Removing any interior point will always reduce the line length. If two successive points are removed, the length reduction from each removal step sums to the total length reduction. This fact may prove helpful.

Deciding how many points to retain is an issue. In this problem, you will be given a required number of points to retain, and asked to report the (minimum) reduction in line length for an optimum choice of points.

Example. A line consists of 5 points, with coordinates `(100,250),(130,340),(190,370),(340,280),(280,100).` Note that the y coordinate measures down from the top of the image. The images below show the whole line, the best reduction to a 4 point line, and the best reduction to a three point line.

The best reduction to the four point line reduces the line length by roughly 150 – (94.87 + 67.08) = 11.95

Input

Input consists of a number of line reduction problems. Each problem starts with a line holding two integers, N and K where 2 <= K <= N <= 1000. It is followed by a line holding 2 * N integers (each in the range 0 to 1000), being the x and y coordinates of N points. End of input is indicated by a line holding two zeroes.

Output

One line per problem holding the minimum reduction in line length resulting from removing all but K of the N points in the manner discussed above. The value should be displayed rounded to two significant figures. Always display two digits after the decimal point, even if they are both zero.

Sample Input

```
5 4
100 250 130 340 190 370 340 280 280 100
0 0
```

Sample Output

```
11.95
```

PROBLEM N 100 POINTS

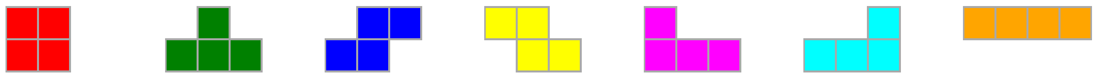
This problem has been withdrawn

PROBLEM O RTETRIS: TETRIS FOR REAL MEN, REAL WOMEN AND REAL RUSSIAN NERDS 100 POINTS

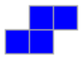
It is a little known fact that the rules of the modern Tetris game have been eased to accommodate the limitations and attitudes of soft modern people. Back in the day, the game, then known as rtetris (the rt being pronounced as in ancient Klingon) was tougher. Play was public, and players who did not 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.



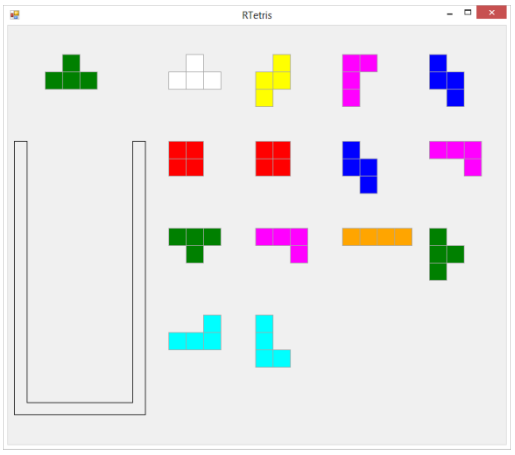
The game proceeds as a series of plays. On each play, a piece is positioned above the pit. The player can rotate it in multiples of 90 degrees; and move it left or right. When satisfied, they drop the piece into the pit. It moves down as far as possible, with the constraint that it cannot overlap the 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.

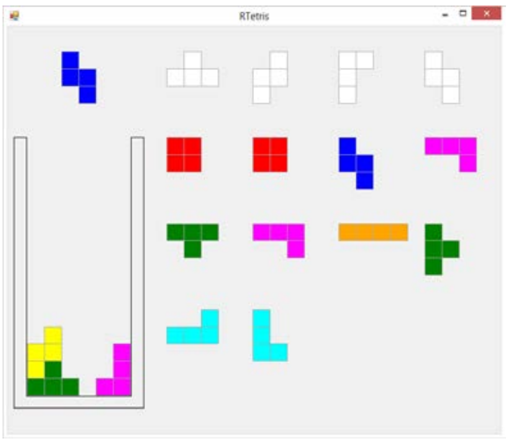
Visibility of all pieces leads to the game having deeper play strategies than modern Tetris. Your "helper" program must analyse game hands to find winning play strategies. Championship games were often played with very large hands.

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

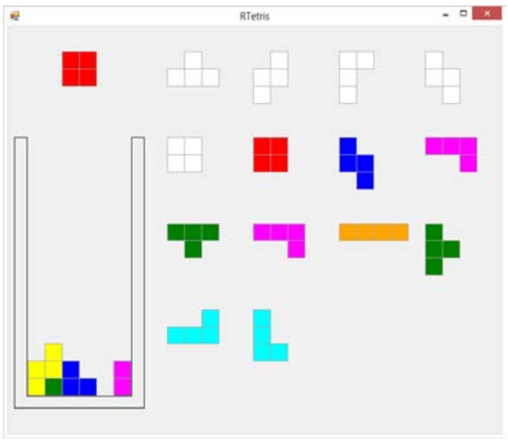
Here is an initial game state – playing with a 6 by 15 cell pit and a 14 piece hand. Note that the first piece has been moved over the pit, and is shown in outline in the hand. Pieces are played from the hand in the order dealt .



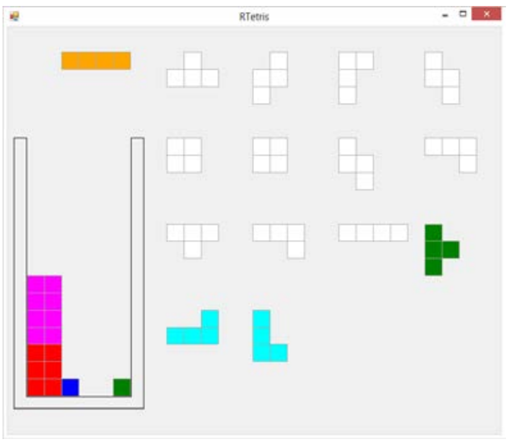
A little later, three pieces have been played and the fourth is in play (left below). Dropping the fourth piece will remove the bottom row, but leaves the game with a losing position (right below).



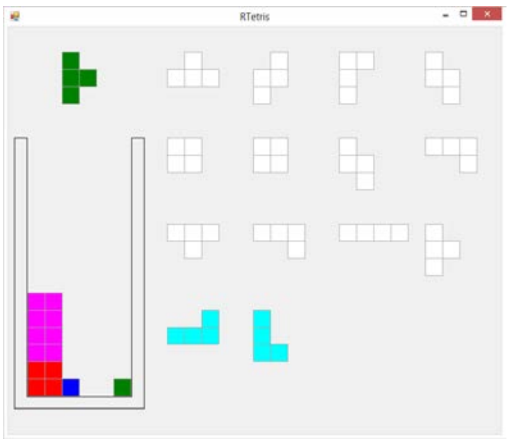
=>



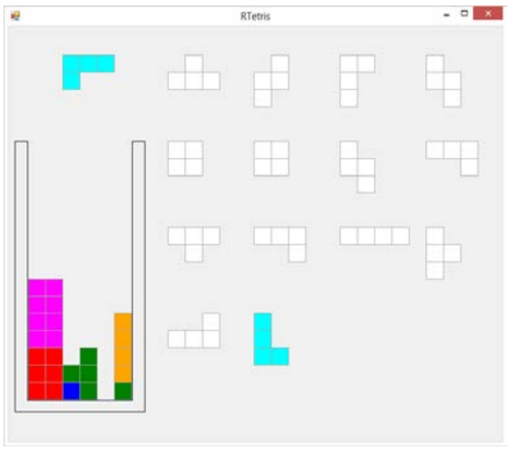
Further into the game, playing a different strategy, a new situation arises. Dropping the horizontal piece looks as though it might leave gaps, but the fact that it completes a row makes it legitimate.



=>

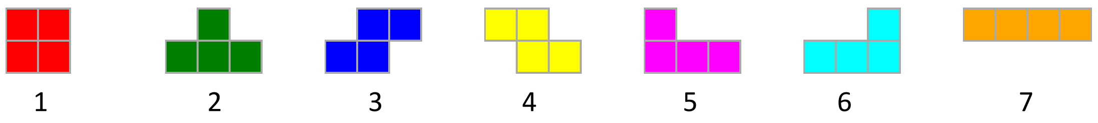


Here is another situation where a drop leads to a valid position, and a possible game win.



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
```

Sample Output

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

PROBLEM P NOHTYP 100 POINTS

Every decade there emerges a new programming language that captures the hearts and minds of the programming community – well … some decades, some hearts and some minds are captured. For this decade there is a new candidate design – a language that is incomplete (claiming a simplicity that can be rapidly abandoned once widespread adoption is achieved); unremarkable (nothing new to see), but with such a cleverly chosen name that many think it has a real chance. Nohtyp (the t is silent) is the language of the future.

One minor problem is that there are as yet no implementations. This is where you come in. The problem requires you to implement a Nohtyp interpreter. The language is described informally as follows. See sample input for examples of each construct.

Lexical Structure: Nohtyp has single letter (lower case a-z) identifiers (for variables, and functions). It supports strings – arbitrary sequences of characters enclosed in "double quotes". Empty strings are not allowed. It supports simple integer literals. Input is mostly free format – white space at the left of each line has meaning as described later, but otherwise programmers are free to use or not use spaces as thet wish. The only time a space is compulsory is where it separates tokens that would otherwise be hard to separate. It is not permissible to write "ifx>1". At least one space is required to separate the "if" and the "x", i.e. "if   x>1".

Expressions. Ordinary integer arithmetic expressions are supported with standard precedence. Factors can be parenthesised expressions (a + 2); function calls with single arguments f(2 + e); variable references; and positive integer literals. Integer binary operators supported are + - * and /. There is no unary minus.

Boolean expressions are allowed only in 'if' statements. The take the form <expr> <Boolean operator> <expr>. The Boolean operators supported are =, < and >.

Functions with single letter name and optional single parameter. Functions must return a result (integer value).

Statements supported are: function call a(<expr>); assignment statement a = <expr>; the return statement allowed only in functions (return <expr>); the print statement (print <comma separated list of <expr> and/or strings); the if statement.

```
if  <Boolean expression>
    <statement>
    <statement>
else
    <statement>
    <statement>
```

Functions

```
fun  x()                    or          fun x(p)
    <statement>                             <statement>
    <statement>                             <statement>
```

Programmes have names (string), and are written

```
program "name"
     <statement>
     <statement>
```

The block structure of programs, functions and statements is indentation controlled. For example, in the 'if' statement, nohtpy identifies the statements forming the 'then' block and the 'else' block by their indentation. Each of the statements in the 'then' block must have the same indentation (except those in deeper nested structures), which must be greater than that of the 'if' statement itself.

Unlike Python however, two lines have the same indentation their white space characters sum to the same level. It doesn't matter what mixture of tab and space characters are used. Tab characters add at least one white space character and round the current line character count up to a multiple of 8.

Notes: print without any items outputs a blank line
functions must be declared before use, functions cannot be redefined
variable and function names must be distinct (can't have a function and a variable with the same name)
variables must be assigned before access, and may be accessed globally in functions
function parameter names may mask access to global variables or functions
assignment to a parameter is not allowed
sample input will have no blank lines

Input

A sequence of nohtpy programs. End of input is indicated by 'exit' at or below the level of indentation of the last 'program' line. You may assume that all programs are syntactically correct, and obey semantic constraints – e.g. that functions with a parameter will be called with a parameter.

Output

For each program, output a blank line followed by a line with the text "program <name>" where name is the string supplied with the program. Follow this by the output generated by the program.

Sample Input

```
program "First"
    x = 4
    print "X is ", 4
  program     "Second"
      x = 4
      if x > 3
             print "Bigger"
      else
         print "Smaller or equal"
program "Third"
  fun g()
    print "Hello from g"
    return 1
  fun h(p)
    print "Hello from h with parameter ", p
    return p * 3
  x = g() + 3 * h(5)
  print x
exit
```

Sample Output

```

program "First"
X is 4

program "Second"
Bigger

program "Third"
Hello from g
Hello from h with parameter 5
46
```