1. Perfection

From the article Number Theory in the 1994 Microsoft Encarta: "If a, b, c are integers such that a = bc, a is called a multiple of b or of c, and b or c is called a divisor or factor of a. Even integers, which include 0, are multiples of 2, for example, -4, 0, 2, 10; an odd integer is an integer that is not even, for example, -5, 1, 3, 9. A perfect number is a positive integer that is equal to the sum of all its positive, proper divisors; for example, 6, which equals 1 + 2 + 3, and 28, which equals 1 + 2 + 4 + 7 + 14, are perfect numbers. A positive number that is not perfect is imperfect and is deficient or abundant according to whether the sum of its positive, proper divisors is smaller or larger than the number itself. Thus, 9, with proper divisors 1, 3, is deficient; 12, with proper divisors 1, 2, 3, 4, 6, is abundant."

Problem Statement: Given a number, determine if it is perfect, abundant, or deficient.

Input

The input should be read from system in. A list of N positive integers (none greater than 60,000), with $1 \leq N \leq 100$. A 0 will mark the end of the list.

Output

Output should be sent to system out. The first line of output should read PERFECTION OUTPUT. The next N lines of output should list for each input integer whether it is perfect, deficient, or abundant, as shown in the example below. The final line of output should read END OF OUTPUT .

Input

```
15
28
6
56
60000
22
496
0
```

Output

```
PERFECTION OUTPUT
15  DEFICIENT
28  PERFECT
6  PERFECT
56  ABUNDANT
60000  ABUNDANT
22  DEFICIENT
496  PERFECT
END OF OUTPUT
```

2. Password Security

Password security is a tricky thing. Users prefer simple passwords that are easy to remember (like buddy), but such passwords are often insecure. Some sites use random computer-generated passwords (like xvtpzyo), but users have a hard time remembering them and sometimes leave them written on notes stuck to their computer. One potential solution is to generate "pronounceable" passwords that are relatively secure but still easy to remember.

A company is developing such a password generator. You work in the quality control department, and it's your job to test the generator and make sure that the passwords are acceptable. To be acceptable, a password must satisfy these three rules:

1. It must contain at least one vowel.
2. It cannot contain three consecutive vowels or three consecutive consonants.
3. It cannot contain two consecutive occurrences of the same letter, except for 'ee' or 'oo'.

(For the purposes of this problem, the vowels are 'a', 'e', 'i', 'o', and 'u'; all other letters are consonants.) Note that these rules are not perfect; there are many common/pronounceable words that are not acceptable.

Input

The input should be read from system in. The input consists of one or more potential passwords, one per line, followed by a line containing only the word 'end' that signals the end of the file. Each consists only of lowercase letters.

Output

Output should be sent to system out. For each password, output whether or not it is acceptable, using the precise format shown in the example.

Example input:

```
a
tv
ptoui
bontres
zoggax
wiinq
eep
houctuh
end
```

Example output:

```
<a> is acceptable.
<tv> is not acceptable.
<ptoui> is not acceptable.
<bontres> is not acceptable.
<zoggax> is not acceptable.
<wiinq> is not acceptable.
<eep> is acceptable.
<houctuh> is acceptable.
```

3. Up and Down Sequences

The quality of pseudo random-number generators used in some computations, especially simulation, is a significant issue. Proposed generation algorithms are subjected to many tests to establish their quality, or, more usually, their lack of it. One of the common tests is the run test.

In this test, sequences are tested for "runs up" and "runs down."

We will examine series of data values for the "Up" and "Down" sequences each series contains. Within a series, an "Up" sequence continues as long as each data-value received is not less than the previous data-value. An "Up" sequence terminates when a data-value received is less than the previous data-value received. A "Down" sequence continues as long as each data-value received is not greater than the previous data-value.

A "Down" sequence terminates when a data-value received is greater than the previous data-value received. An "Up" sequence can be initiated by the termination of a "Down" sequence and vice versa. (Sequences initiated in this manner have length one at this initiation point.) All the initial data-values are part of an "Up" sequence, and contribute to its length, if the first deviation of the data-values is upwards. All the initial data-values are part of a "Down" sequence, and contribute to its length, if the first deviation of the data-values is downwards.

If the data-values received don't allow classification as either an "Up" or a "Down" sequence, the data should be considered to have neither sequence. Find the average length of both the "Up" and the "Down" sequences encountered for each input line in the data file. Report these average lengths as each input line is processed.

Input

The input should be read from system in. Each of the separate series to be examined is contained on a single line of input.

Each series to be analyzed consists of at least one and no more than 30 unsigned, non-zero integers. Each integer in a series has at least one digit and no more than four digits. The integers are separated from each other by a single blank character. Each of the series will be terminated by a single zero (0) digit. This terminator should not be considered as being part of the series being analyzed. The set of series to be analyzed is terminated by a single zero (0) digit as the input on a line. This terminator should not be considered to be a series, and no output should be produced in response to its encounter.

Output

Output should be sent to system out. A line with two real values is to be emitted for each input data set encountered. First print, the average "Up" run length, and then the average "Down" run length. Separate these values with a space. The default real/float output format should be used. Your output must be formatted exactly as what is shown below.

EXAMPLE

Input File Contents (All Single-Digit Integers Shown)	Output File (Average Run Lengths)
1 2 3 0 3 2 1 0 1 2 3 2 10 2 2 2 2 3 0 4 4 4 4 3 0 4 4 4 3 3 3 3 0 4 4 4 3 3 3 4 0 5 5 5 5 0 1 2 3 2 3 4 5 0 0	2.000 0.000 0.000 2.000 2.000 2.000 4.000 0.000 0.000 4.000 0.000 6.000 1.000 5.000 0.000 0.000 2.500 1.000

4. Word Index

Encoding schemes are often used in situations requiring encryption or information storage/transmission economy. Here, we develop a simple encoding scheme that encodes particular types of words with five or fewer (lower case) letters as integers. Consider the English alphabet {a,b,c,...z}. Using this alphabet, a set of valid words are to be formed that are in a strict lexicographic order. In this set of valid words, the successive letters of a word are in a strictly ascending order; that is, later letters in a valid word are always after previous letters with respect to their positions in the alphabet list {a,b,c...,z}.
For example,

abc aep gwz

are all valid three-letter words, whereas

aab are cat

are not.

For each valid word associate an integer which gives the position of the word in the alphabetized list of words. That is:
a → 1
b → 2
.
.
z → 26
ab → 27
ac → 28
.
.
az → 51
bc → 52
.
.
vwxyz → 83681

Your program is to read a series of input lines. Each input line will have a single word on it that will be from one to five letters long. For each word read, if the word is invalid give the number 0. If the word read is valid, give the word's position index in the above alphabetical list.

Input
The input should be read from system in. The input consists of a series of single words, one per line. The words are at least one letter long and no more than five letters. Only the lower case alphabetic {a,b,...,z} characters will be used as input. The first
letter of a word will appear as the first character on an input line.
The input will be terminated by end-of-file.

Output
Output should be sent to system out. The output is a single integer, greater than or equal to zero (0) and less than or equal 83681. The first digit of an output value should be the first character on a line. There is one line of output for each input line.

EXAMPLE

Input	Output
z	26
a	1
cat	0
vwxyz	83681

5. Bulk Mailing

An organization that wishes to make a large mailing can save postage by following U.S. Postal Service rules for a bulk mailing. Letters in zip code order are bundled into packets of 10 or 15 letters each. Bundles may consist of letters in which all 5 digits of the zip code are the same (5-digit bundles), or they may consist of letters in which only the first 3 digits of the zip code are the same (3-digit bundles). If there are fewer than 10 letters to make up a bundle of either type, those letters are mailed first class.

You are to write a program to read a data set of 5-digit zip codes, one per line, until end of file.

Input
The input should be read from system in. Your program should count the number of 5-digit bundles, 3-digit bundles, and first class letters. You should include as many letters as possible in 5-digit bundles first, then as many as possible in 3-digit bundles, with as few bundles of 10 or 15 letters as possible. For example, if there are 31 letters with the same zip code, they must be combined into exactly two 5-digit bundles. Not all zip codes in the data set will be valid. A valid zip code consists of exactly 5 digits (0-9), all of which cannot be 0. Non-numeric characters are not allowed.

Output
Output should be sent to system out. Print a report that lists 5-digit zip code bundles first, with the number of letters and number of bundles for each zip code. Next list all 3-digit zip code bundles with the same two counts, followed by all zip codes that are not bundled and to be sent first class. Within each group output should be sorted by zip code in ascending order. (i.e. 66509, 66612, 90210, etc) At the end print totals of letters and bundles, followed by the number of invalid zip codes and a list of these. Print blank lines following the heading, before the total line, and between the three groups of zip codes. For 3-digit bundles, print the zip codes in the form dddxx, where ddd represents the three significant digits and xx represents the last two digits to be omitted. Delimit your columns with a comma (,). Your output should be similar to that shown in the sample below.

For example (input continues on 2nd page):

SAMPLE INPUT:

```
95864
95864
95864
95867
95920
9j876
95616
95616
95747
95814
95818
95818
8976
95818
95818
95819
95819
00000
95819
95819
95819
95819
95825
95825
95825
95825
95825
95826
95826
95826
95826
95826
95827
8976
95833
95833
95833
95833
95819
95819
95819
95819
95833
95833
95833
95864
95864
95864
123456
95864
95864
95864
95864
```

PRODUCES THIS OUTPUT:

```
ZIP,LETTERS,BUNDLES

95819,11,1
95864,10,1

958xx,25,2

95616 ,2,0
95747 ,1 ,0
95920,1, 0

TOTALS,50,4

INVALID ZIP CODES

9j876
8976
00000
123456
```

6. Anagram

You are to write a program that has to generate all possible words from a given set of letters.
Example:
Given the word “abc”, your program should – by exploring all different combination of the three letters – output the words “abc”, “acb”, “bac”, “bca”, “cab” and “cba”. In the word taken from the input file, some letters may appear more than once. For a given word, your program should not produce the same word more than once, and the words should be output in alphabetically ascending order.

Input

The input file consists of several words. Each line contains one word. A word consists of uppercase or lowercase letters from A to Z. Uppercase and lowercase letters are to be considered different.

Output

For each word in the input file, the output file should contain all different words that can be generated with the letters of the given word. The words generated from the same input
word should be output in alphabetically ascending order.

Example:

Input

abc
acba

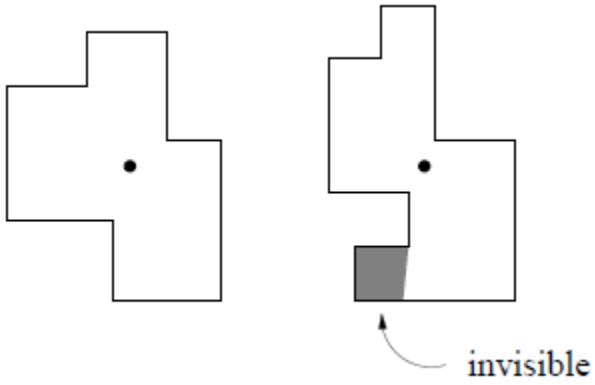
Output

abc
acb
bac
bca
cab
cba
aabc
aacb
abac
abca
acab
acba
baac
baca
bcaa
caab
caba
cbaa

7. Surveillance

A friend of yours has taken the job of security officer at the Star-Buy Company, a famous department store. One of his tasks is to install a video surveillance system to guarantee the security of the customers (and the security of the merchandise of course) on all of the store's countless floors. As the company has only a limited budget, there will be only one camera on every floor. But these cameras may turn around to look in every direction.

The first problem is to choose where to install the camera for every floor. The only requirement is that every part of the room must be visible from there. In the following figure the left floor can be completely surveyed from the position indicated by a dot, while for the right floor, there is no such position, the given position failing to see the lower left part of the floor.



Before trying to install the cameras, your friend first wants to know whether there is indeed a suitable position for them. He therefore asks you to write a program that, given a ground plan, determines whether there is a position from which the whole floor is visible. All floor ground plans form rectangular polygons, whose edges do not intersect each other and touch each other only at the corners.

Input

The input file contains several floor descriptions. Every description starts with the number n of vertices that bound the floor ($4 \le n \le 100$). The next n lines contain two integers each, the x and y coordinates for the n vertices, given in clockwise order. All vertices will be distinct and at corners of the polygon. Thus the edges alternate between horizontal and vertical. A zero value for n indicates the end of the input.

Output

For every test case first output a line with the number of the floor, as shown in the sample output. Then print a line stating "Surveillance is possible." if there exists a position from which the entire floor can be observed, or print "Surveillance is impossible." if there is no such position. Print a blank line after each test case.

Sample Input

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

Sample Output

```
Floor #1
Surveillance is possible.

Floor #2
Surveillance is impossible.
```

8. Pythagorean Perfection Pt. II

Pythagoras defined a Perfect Number as a number A whose divisors add up to A itself. When this sum is larger than A it is called *excessive*, when it is smaller than A it is called *defective*.

Example 1: divisors of 10 = {1 2 5}, Sum = 1 + 2 + 5 = 8. 10 is defective.
Example 2: divisors of 6 = {1 2 3}, Sum = 1 + 2 + 3 = 6. 6 is perfect.
Example 3: divisors of 12 = {1 2 3 4 6}, Sum = 1 + 2 + 3 + 4 + 6 = 16. 12 is excessive.
From Simon Singh's book: "Fermat's Last Theorem", Fourth Estate, London 1997, ISBN 1-85702-669-1.

Your program will read numbers from system in and determine if the number is perfect. It should print to system out the number and whether it is perfect or not.
You may notice that this problem has a strong resemblance to problem one. The main difference is, in this problem the numbers you will receive are quite high. The highest number you will receive is the 18th perfect number.

To help you on your way here are a few formulas that you may find useful.

Mersenne numbers are described by the following formula: $M_p = 2^n - 1$

The Lucas-Lehmer test for Mersenne Primes:

$$s_i = \begin{cases} 4 & \text{if } i = 0; \\ s_{i-1}^2 - 2 & \text{otherwise.} \end{cases}$$

$$s_{p-2} \equiv 0 \pmod{M_p}.$$

Perfect Numbers: $P = (2^n - 1) * 2^{(n-1)}$
P is a perfect number when n is a Mersenne prime.

Input:
12
10
6
45654854
33550336

Output:
12 is not perfect.
10 is not perfect.
6 is perfect.
45654884 is not perfect.
33550336 is perfect.

9. Automatic Editing

Text-processing tools like allow you to automatically perform a sequence of editing operations based on a script. For this problem we consider the specific case in which we want to perform a series of string replacements, within a single line of text, based on a fixed set of rules. Each rule specifies the string to find, and the string to replace it with, as shown below.

Rules:
Find Replace-by
1. ban -> bab
2. baba -> be
3. ana -> any
4. ba b -> hind the g

To perform the edits for a given line of text, start with the first rule. Replace the first occurrence of the find string within the text by the replace-by string, then try to perform the same replacement again on the new text. Continue until the find string no longer occurs within the text, and then move on to the next rule. Continue until all the rules have been considered. Note that (1) when searching for a find string, you always start searching at the beginning of the text, (2) once you have finished using a rule (because the find string no longer occurs) you never use that rule again, and (3) case is significant.

For example, suppose we start with the line:
banana boat

and apply these rules. The sequence of transformations is shown below, where occurrences of a find string are underlined and replacements are boldfaced. Note that rule 1 was used twice, then rule 2 was used once, then rule 3 was used zero times, and then rule 4 was used once.

Before	After
banana boat	**bab**ana boat
babana boat	ba**bab**a boat
bababa boat	**be**ba boat
beba boat	be**hind the g**oat

The input contains one or more test cases, followed by a line containing only 0 (zero) that signals the end of the file. Each test case begins with a line containing the number of rules, which will be between 1 and 10. Each rule is specified by a pair of lines, where the first line is the find string and the second line is the replace-by string. Following all the rules is a line containing the text to edit. For each test case, output a line containing the final edited text.

Both find and replace-by strings will be at most 80 characters long. Find strings will contain at least one character, but replace-by strings may be empty (indicated in the input file by an empty line). During the edit process the text may grow as large as 255 characters, but the final output text will be less than 80 characters long. The first test case in the sample input below corresponds to the example shown above.

Example input:

```
4
ban
bab
baba
be
ana
any
ba b
hind the g
banana boat
1
t
sh
toe or top
0
```

Example output:

```
behind the goat
shoe or shop
```

10. Transferable Voting

The Transferable Vote system for determining the winner of an election requires that a successful candidate obtain an absolute majority of the votes cast, even when there are more than two candidates. To achieve this goal, each voter completes his or her ballot by listing all the candidates in order of preference. Thus if there are five candidates for a single position, each voter's ballot must contain five choices, from first choice to fifth choice.

In this problem you are to determine the winner, if any, of elections using the Transferable Vote system. If there are c candidates for the single position, then each voter's ballot will include c distinct choices, corresponding to identifying the voter's first place, second place, ..., and nth place selections. Invalid ballots will be discarded, but their presence will be noted. A ballot is invalid if a voter's choices are not distinct (choosing the same candidate as first and second choice isn't permitted) or if any of the choices aren't legal (that is, in the range 1 to c).

For each election candidates will be assigned sequential numbers starting with 1. The maximum number of voters in this problem will be 100, and the maximum number of candidates will be 5.

Table A

Voter	1st Choice	2nd Choice	3rd Choice
1	1	2	4
2	1	3	2
3	3	2	1
4	3	2	1
5	1	2	3
6	2	3	1
7	3	2	1
8	3	1	1
9	3	2	1
10	1	2	3
11	1	3	2
12	2	3	1

Table B

1st Choice	2nd Choice	3rd Choice
1	3	2
3	2	1
3	2	1
1	2	3
3	1	
3	2	1
3	2	1
1	2	3
1	3	2
3	1	

Consider a very small election. In Table A are the votes from the 12 voters for the three candidates. Voters 1 and 8 cast invalid ballots; they will not be counted. This leaves 10 valid ballots, so a winning candidate will require at least 6 votes (the least integer value greater than half the number of valid ballots) to win. Candidates 1 and 3 each have 4 first choice votes, and candidate 2 has two. There is no majority, so the candidate with the fewest first choice votes, candidate 2, is eliminated. If there had been several candidates with the fewest first choice votes, any of them, selected at random, could be selected for elimination.

With candidate 2 eliminated, we advance the second and third choice candidates from those voters who voted for candidate 2 as their first choice. The result of this action is shown in Table B. Now candidate 3 has picked up 2 additional votes, giving a total of 6. This is sufficient for election. Note that if voter 12 had cast the ballot "2 1 3" then there would have been a tie between candidates 1 and 3.

Input

There will be one or more elections to consider. Each will begin with a line containing the number of candidates and the number of voters, c and n. Data for the last election will be followed by a line containing two zeroes.
Following the first line for each election will be n additional lines each containing the choices from a single ballot. Each line will contain only c non-negative integers separated by whitespace.

Output

For each election, print a line identifying the election number (they are numbered sequentially starting with 1). If there were any invalid ballots, print an additional line specifying the number of such. Finally, print a line indicating the winner of the election, if any, or indication of a tie; be certain to identify the candidates who are tied. Separate the output for each election by a single blank line.

Sample Input

```
3 12
1 2 4
1 3 2
3 2 1
3 2 1
1 2 3
2 3 1
3 2 1
3 1 1
3 2 1
1 2 3
1 3 2
2 3 1
3 12
1 2 4
1 3 2
3 2 1
3 2 1
1 2 3
2 3 1
3 2 1
3 1 1
3 2 1
1 2 3
1 3 2
2 1 3
4 15
4 3 1 2
4 1 2 3
3 1 4 2
1 3 2 4
4 1 2 3
3 4 2 1
2 4 3 1
3 2 1 4
3 1 4 2
1 4 2 3
3 4 1 2
3 2 1 4
4 1 3 2
3 2 1 4
4 2 1 4
0 0
```

Sample Output

```
Election #1
2 bad ballot(s)
Candidate 3 is elected.

Election #2
2 bad ballot(s)
The following candidates are tied: 1 3

Election #3
1 bad ballot(s)
Candidate 3 is elected
```