Contents

1 Hidden Enchantment

While researching the history of the One Ring, Gandalf is puzzled by a mysterious passage in an ancient book. Parts of the passage seems to possess similarities to the words for an enchantment for invisibility, but with many missing and extra letters. To compare the two letter sequences, Gandalf decides to make a rectangular drawing constructed as follows:

The mysterious book passage is placed at the top of the drawing, running from left to right. The words for the enchantment are placed at the left of the drawing, running from top to bottom. When letters from the two sequences match, a letter "x" is placed in the drawing at the point where the letters intersect when lines (running either top-to-bottom or left-to-right) are drawn through the two letters. If three or more letters in a row (in the two sequences) match, a letter "Q" is drawn for each of the matches instead (note these matches form diagonals heading to the bottom right). A period "." is drawn for every mismatch. For instance, the two sequences "ring" and "grin" would produce the following drawing, with an "x" for the matching "g", and "Q" for the matching "rin":

```
  ring
  ----
g|...x
r|Q...
i|.Q..
n|..Q.
```

The comparison is case-sensitive, so "r" matches "r" but not "R". Once the drawing is complete, Gandalf can discover which sections of the book's mysterious passage match words in the enchantment by looking for long diagonal runs of "Q"s in the drawing.

Input Format

The input is simply the two sequences of letters, each on a separate line. The book passage comes first, followed by the enchantment. You may assume that the longest length for either sequence is at most 100 letters.

Output Format

The drawing representing the comparison is as follows. The first row contains the book passage, beginning at the 3rd column (passage is preceded by 2 spaces). The second row is a row of dashes the same length as the passage, also starting at the 3rd column. The third and succeeding rows contain the enchantment and matches. The enchantment is displayed vertically in the first column, starting in the 3rd row. The second column is a row of vertical bars, the same length as the enchantment. The remaining columns are the matches / mismatches represented by "x", "Q", and "."s.

Examples

Input 1:

```
ThisIsTheOneRing
OneRingToFindThem
```

Output 1:

```
  ThisIsTheOneRing
  ----------------
O|.........Q......
n|..........Q...x.
e|........x..Q....
R|............Q...
i|..x..........Q..
n|..........x...Q.
g|...............Q
T|x.....x.........
o|................
F|................
i|..x..........x..
n|..........x...x.
d|................
T|x.....Q.........
h|.x.....Q........
e|........Q..x....
m|................
```

Input 2:

```
OneRingToRuleThemAll,OneRingToFindThem
OneRingToBringThemAll,AndInTheDarknessBindThem
```

Output 2:

```
  OneRingToRuleThemAll,OneRingToFindThem
  --------------------------------------
O|Q....................Q................
n|.Q...x................Q...x.....x.....
e|..Q.........x..x.......Q............x.
R|...Q.....x..............Q.............
i|....Q....................Q.....x......
n|.x...Q................x...Q.....x.....
g|......Q....................Q..........
T|.......Q.....x..............Q.....x...
o|........Q....................Q........
B|......................................
r|......................................
i|....Q....................Q.....x......
n|.x...Q................x...Q.....x.....
g|......Q....................Q..........
T|.......Q.....Q..............Q.....Q...
h|..............Q....................Q..
e|..x.........x..Q.......x............Q.
m|................Q....................Q
A|.................Q....................
l|...........x......Qx..................
l|...........x......xQ..................
,|....................Q.................
A|.................x....................
n|.x...x................x...x.....x.....
d|.................................x....
I|......................................
n|.x...x................x...x.....x.....
T|.......x.....Q..............x.....Q...
h|..............Q....................Q..
e|..x.........x..Q.......x............Q.
D|......................................
a|......................................
r|......................................
k|......................................
n|.x...x................x...x.....x.....
e|..x.........x..x.......x............x.
s|......................................
s|......................................
B|......................................
i|....x....................x.....Q......
n|.x...x................x...x.....Q.....
d|.................................Q....
T|.......x.....Q..............x.....Q...
h|..............Q....................Q..
e|..x.........x..Q.......x............Q.
m|................Q....................Q
```

2 Family Name Distance

While writing the story of his adventures in Rivendell, Bilbo decided to make draw up a family tree for the different hobbit families in the Shire. Since the actual relationships were long lost to history, Bilbo decided to calculate the "distance" between two families as a function of their family names. After much consideration, Bilbo chose to use the following formula:

Two identical letters have a distance of 0. The distance between any two mismatched letters is 50 plus the number of letters between them in the alphabet. For instance "B" and "B" has as distance of 0, while "B" and "D" has a distance of $50 + 2$. The maximum distance between two letters is thus 75, between "A" and "Z". The distance between any letter and blank space is defined as 4. The distance between two names is calculated as the sum of the distances for the letters in each name, but weighted so that the score of each letter is reduced by 10 percent for each position away from the beginning of the name. The formula is thus:

```
Distance = (distance between 1st letters)
           + 9/10 * (distance between 2nd letters)
           + 9/10 * 9/10 * (distance between 3rd letters)
           + 9/10 * 9/10 * 9/10 * (distance between 4th letters)
           + ...
```

For instance, the distance between "aa" and "ga" is $50 + 6$ (distance between 1st letters) + $9/10 * 0$ (distance between 2nd letters) = 56. The distance between "aa" and "ddd" is $50 + 3$ (distance between 1st letters) + $9/10 * (50 + 3)$ (distance between 2nd letters) + $9/10 * 9/10 * 4$ (distance between 3rd letters) = 103.94.

Once Bilbo calculates the distance between every pair of family names, he can then build the family tree to reflect the relationships between families, with less "distant" families closer together on the family tree.

Input Format

The number of family names (as an integer), followed by each family name (as a string) on a separate line, in alphabetical order. Either all or none of the names will be capitalized (so you will only be computing difference between pairs of upper or lower case letters, never between upper and lower case letter). You may assume there are a maximum of 50 hobbit families in the Shire.

Output Format

The number of family names (as an integer), followed by individual *distance descriptions*. A distance description a single line containing the distance (as a floating point value with 6 significant digits), followed by two family names (as strings), all separated by spaces. The two family names in each distance description are sorted in alphabetical order. The list of distance descriptions are lexicographically sorted by family names, with ties broken using the second family name. Do not output duplicate distance descriptions. For n families there should be exactly $n(n-1)/2$ distance descriptions.

Examples

Input 1:

```
6
Baggins
Brandybuck
Digger
Digswell
Gamgee
Took
```

Output 1:

```
6
260.110 Baggins Brandybuck
173.642 Baggins Digger
258.433 Baggins Digswell
172.754 Baggins Gamgee
219.058 Baggins Took
266.443 Brandybuck Digger
340.538 Brandybuck Digswell
289.179 Brandybuck Gamgee
218.474 Brandybuck Took
131.053 Digger Digswell
187.761 Digger Gamgee
207.732 Digger Took
244.412 Digswell Gamgee
214.687 Digswell Took
207.072 Gamgee Took
```

Input 2:

```
6
a
aa
b
cd
ddd
ga
```

Output 2:

```
6
3.60000 a aa
51.0000 a b
55.6000 a cd
59.8400 a ddd
59.6000 a ga
54.6000 aa b
99.7000 aa cd
103.940 aa ddd
56.0000 aa ga
54.6000 b cd
58.8400 b ddd
58.6000 b ga
54.2400 cd ddd
101.700 cd ga
103.940 ddd ga
```

3 Hobbit Family Tree

After deciding on how to calculate the distance between hobbit families, Bilbo continues to draw up the family tree for the Shire. Assuming Bilbo knows "distance" between two families as a function of their family names, Bilbo wants to build the family tree to reflect the relationships between families, with less "distant" families closer together on the family tree, and the height of the branches reflecting the distances between the families.

To build the family tree from family name distances, Bilbo decides to use the following method. At each stage, he'll join the two families (and/or subtrees) with the smallest distance, and set the height of the branch for the join to be 1/2 the distance between the families. The subtree J resulting from a join is treated as a single hobbit family, represented by the (alphabetically) first family. Thus the distance between another family X and the subtree J is simply the distance between X and the first family in J. The entire process is repeated until the last two subtrees are joined, with the root of the entire family tree being 1/2 the distance between the last two family names.

Your job is to calculate the order families need to be joined in order to build the hobbit family tree. Start by finding the closest pair of families, join them, and repeat. If multiple pairs of families have the same distance, chose the pair which is first in alphabetical order.

Input Format

The number of family names (as an integer), followed by individual distance descriptions. A distance description a single line containing the distance (as a floating point value with 6 significant digits), followed by two family names (as strings), all separated by spaces. The two family names in each distance description are sorted in alphabetical order.

The list of distance descriptions are sorted by family names, with ties broken the second family name. There are no duplicate distance descriptions. For n families there should be exactly $n(n-1)/2$ distance descriptions. You may assume there are a maximum of 50 hobbit families in the Shire.

Output Format

The ordered list of joins used to produce each family tree, starting from the closest pair of family names. Each join is similar to a distance description, except the distance is halved to represent the height of the tree branch joining the two names.

If the order list of joins is correct, the included program will (optionally) draw the family tree. This tree is for display purposes only and will not used in the judging process (though it may be useful for debugging).

Examples

Input 1:

```
6
260.110 Baggins Brandybuck
173.642 Baggins Digger
258.433 Baggins Digswell
172.754 Baggins Gamgee
219.058 Baggins Took
266.443 Brandybuck Digger
340.538 Brandybuck Digswell
289.179 Brandybuck Gamgee
218.474 Brandybuck Took
131.053 Digger Digswell
187.761 Digger Gamgee
207.732 Digger Took
244.412 Digswell Gamgee
214.687 Digswell Took
207.072 Gamgee Took
```

Output 1:

```
65.5265 Digger Digswell
86.3770 Baggins Gamgee
86.8210 Baggins Digger
109.237 Brandybuck Took
130.055 Baggins Brandybuck

               Maximum Tree Height = 130.055
     100%         75%            50%            25%            0%
     987654321098765432109876543210987654321098765432109876543210

      ----------------------------------------------------------- Baggins
     |                   ||
     |                   | -------------------------------------- Gamgee
     |                   |
     |                    --------------------------------------- Digger
-----|                             |
     |                              ----------------------------- Digswell
     |
      ----------------------------------------------------------- Brandybuck
               |
                ------------------------------------------------- Took
```

Input 2:

```
6
3.60000 a aa
51.0000 a b
55.6000 a cd
59.8400 a ddd
59.6000 a ga
54.6000 aa b
99.7000 aa cd
103.940 aa ddd
56.0000 aa ga
54.6000 b cd
58.8400 b ddd
58.6000 b ga
54.2400 cd ddd
101.700 cd ga
103.940 ddd ga
```

Output 2:

```
1.80000 a aa
25.5000 a b
27.1200 cd ddd
27.8000 a cd
29.8000 a ga

               Maximum Tree Height = 29.8000
     100%         75%            50%            25%            0%
     987654321098765432109876543210987654321098765432109876543210

      ----------------------------------------------------------- a
     |    |   |                                               |
     |    |   |                                                -- aa
     |    |   |
     |    |    -------------------------------------------------- b
-----|    |
     |     ------------------------------------------------------ cd
     |     |
     |      ----------------------------------------------------- ddd
     |
      ----------------------------------------------------------- ga
```

4 Middle Earth Base Math

One base does NOT rule them all.

In Middle Earth, Humans use base 10 math while Elves use base 8, Dwarves use base 5 and Wizards use base 2. These differences can cause great confusion between the races.

On their trek to Mordor, the Fellowship begin discussing the merits of each other's respective math systems. Aragorn maintains base 10 is perfect for humans because he can count in multiples of 10 using his fingers. Gandalf claims using base 2 math allows wizards to better appreciate the relationship between abstract numbers and physical objects, improving their mystical abilities. Legolas believes base 8 makes sense for elves because of their ability to blend wizardry with the natural world. Gimli prefers base 5 for dwarfs because it suits their practical, ordered personalities.

To make sense of all the different bases used for math in the Middle Earth, Samwise and Merry decide to write a program to provide examples of calculations using the different math bases, taking as input the race, a pair of numbers, and operation to be performed.

To reduce work you are provided a skeleton program that will read the information from the input, process it by calling a single procedure, then output the result. You will need to implement the procedure and return results to the skeleton program so it can be printed. The skeleton program stores numbers as their base 10 versions to simplify reading & writing (for instance, 101 in Wizard base 2 is still stored as 101, not as 5), but you are not required to use this representation.

Input Format

Your calculator will take as input a race, denoted 'h', 'e', 'd', or 'w', for human, elf, dwarf, or wizard, respectively, two numbers in the base for the race, and an arithmetic operator (either add '+' or multiply '*'). All input numbers are guaranteed to have four or fewer digits, and be in the correct base.

Output Format

Your calculator should return the result of the operation on the two numbers in the input, in the same base (of the race input). Additionally, if the operation is multiplication and there are multiple steps involved, the calculator should return the partial results at each step in an array.

Examples

Input 1:

```
d
14
13
+
```

Output 1:

```
  14
+ 13
====
  32
```

Input 2:

```
d
14
13
*
```

Output 2:

```
   14
*  13
-----
  102
  14
=====
  242
```

Input 3:

```
h
1234
4321
*
```

Output 3:

```
     1234
*    4321
---------
     1234
    2468
   3702
  4936
=========
  5332114
```

Input 4:

```
e
1234
4321
*
```

Output 4:

```
     1234
*    4321
---------
     1234
    2470
   3724
  5160
=========
  5600534
```

Input 5:

```
d
1234
4321
*
```

Output 5:

```
      1234
*     4321
----------
      1234
     3023
    4312
  11101
==========
  12114214
```

Input 6:

```
w
1010
101
*
```

Output 6:

```
    1010
*    101
--------
    1010
      0
  1010
========
  110010
```

5 Saruman's Secret Message

Saruman occasionally sends top-secret messages to his orc commanders using a horde of bats. Because each bat can only carry a limited amount of weight, Saruman breaks up each message into many smaller fragments, with each bat carrying a single message fragment. To reduce the possibility of detection in case one or more bats are captured, Saruman sends many copies of each message, with copies fragmented in different ways.

To decode the message, the recipient must combine the different message fragments, using the overlapping regions as hints. For instance, the fragment "abcde" can be combined with the fragment "cdefgh" to form "abcdefgh". Though there may be many ways to combine the message fragments, the shortest combination is generally taken to be the most correct result. For instance, given the fragments "abaca" and "acada", two possible combinations are "abacada" and "abacacada". To simplify decoding the message, Saruman always begins each message with the marker "ZZZ".

If parts of the message are lost, it is possible that the message fragments cannot be combined into a single contiguous message. You may assume this did not happen with your message. Even without losing any fragments, assembling a single message can be difficult (or impossible) in the general case. For simplicity you may assume that the shortest combination (with maximum overlap) yields the correct message, and that there is only one such combination (i.e., at each step only one message fragment will have the maximum overlap).

A few days ago Saruman sent you and your fellow orcs to track down and "take care" of some troublesome hobbits near your tower. You can tell you are getting close and are looking forward to having a little "snack" upon completion of your task, when bats begin arriving with many message fragments. You sigh and begin the laborious task of putting the coded message together, wondering in the back of your mind whether wizards will ever discover some spell to magically assemble such messages.

Input Format

The number of message fragments (as an integer), followed by each message fragment (as a string) on a separate line. You may assume that all message fragments are exactly 12 letters, and that you will receive at most 100 message fragments. If there are multiple ways to overlap fragments, choose the combination which yields the shortest result (i.e., maximum overlap). You may assume the fragment with the maximum overlap is unique. The beginning of each message always contains the marker "ZZZ".

Output Format

The text of the complete message, without the beginning "ZZZ" marker.

Examples

Input 1:

```
17
em.down!.Do.
l.they.are.f
ZZZHunt.them
o.not.stop.u
.until.they.
t.them.down!
Hunt.them.do
n!.Do.not.st
y.are.found.
p.until.they
.them.down!.
they.are.fou
.not.stop.un
.Do.not.stop
down!.Do.not
t.stop.until
.stop.until.
```

Output 1:

```
Hunt.them.down!.Do.not.stop.until.they.are.found.
```

Input 2:

```
35
oilt..Kill.t
lflings.carr
m.to.me.aliv
ings.carries
ZZZOne.of.th
arries.somet
t.value..Bri
lue..Bring.t
.Halflings.c
ng.of.great.
thing.of.gre
es.something
ll.the.other
.of.great.va
ing.them.to.
One.of.the.H
ngs.carries.
them.to.me.a
.Bring.them.
nd.unspoilt.
hing.of.grea
t..Kill.the.
ue..Bring.th
e.Halflings.
e.alive.and.
of.the.Halfl
ies.somethin
ve.and.unspo
ng.them.to.m
me.alive.and
unspoilt..Ki
em.to.me.ali
.of.the.Half
.the.others.
value..Bring
```

Output 2:

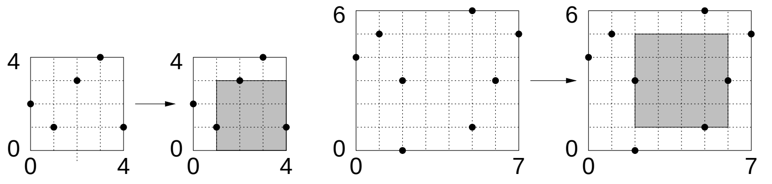
```
One.of.the.Halflings.carries.something.
of.great.value..Bring.them.to.me.alive.
and.unspoilt..Kill.the.others.
```

6 Pippin's Garden

After returning to the shire from his adventures, Pippin decides to build a garden in his yard. Being an orderly Hobbit, Pippin's yard is rectangular but he wants a square garden. Pippin wants to place the largest square garden in his yard, but must avoid the trees that already grow there. Your job is to write a program to help Pippin select the best spot for his garden.

The input consists of the width and height of Pippin's yard, along with the (x, y) coordinates of the trees in his yard. All quantities are integers. Your program should compute the largest square that is entirely contained within his yard such that none of the trees lies in the interior of this square. Trees are allowed to lie on the boundary of the square. By *largest* we mean that square that has the longest side length.

Your program will read the inputs and then compute and output the largest square, by giving its left, right, bottom, and top coordinates. Note that there may generally be many choices for the largest square. Your program should output any one of them. Two examples are shown in the figures below. (In the second case there are two possible choices for the largest square.)



output: 1 4 0 3

output: 2 6 1 5
(also allowed: 2 6 2 6)

Input format

The first line of the input contains the width and height of Pippin's yard. Both numbers are positive integers. The next line contains the number of trees, n. This is followed by n lines, each containing the (x, y) coordinates of the trees. You may assume that every tree lies either within or on the boundary of the yard. You may assume that the width and height of his yard are each at most 100 units, and that there are at most 100 trees.

Output format

Output on one line the 4 coordinates for the left, right, bottom, and top (i.e., smallest x, largest x, smallest y, largest y) of the final square.

Examples

Input 1:	Output 1:
4 4	1 4 0 3
5	
1 1	
0 2	
2 3	
3 4	
4 1	

Input 2:	Output 2:
7 6	2 6 1 5
8	
0 4	
2 0	
1 5	
2 3	
5 1	
6 3	
7 5	
5 6	

Note that since multiple answers are possible, .out files with correct answers to the problem for the "try" command may contain multiple answers. In this case comparisons using the "diff" command will always report an error since the program output will not match the .out file. In such cases simply examine the output of "diff" to determine whether your output matched one correct answer in the .out file.

7 The Game of Rings

Before going on his quest for the ring, Frodo and Bilbo played games in the shire to sharpen their wits. One such game was called *Rings*. Bilbo and Frodo alternate moves, with Bilbo going first (in deference to his age). Initially the board consists of three piles of rings, which we denote (a, b, c), meaning that there are a rings in the first pile, b in the second pile, and c in the third. The board always consists of three numbers that are ≥ 0. During a move a player can do any one of the following:

(1) Remove 1 ring from pile 1.

(2) Remove either 1 or 2 rings from pile 2.

(3) Remove either 1 or 2 or 3 rings from pile 3.

(4) Remove 1 ring from each of the nonempty piles.

Rings removed are discarded from the game. The first player who cannot make a move loses. That is, if it is a player's turn to move and the board is empty $(0, 0, 0)$, then he loses.

Write a program that will, given numbers (a, b, c), determine whether Bilbo or Frodo has a winning strategy if the game is started with the board (a, b, c). That is, your program should determine which player wins, assuming both players play as well as is possible. (Observe that because ties are not possible, and no randomness is involved, one of the two players always has a winning strategy for any given board.)

For example, suppose that the initial board is $(1, 4, 0)$. In this case Bilbo has a wining strategy. He uses rule (4) to remove 1 ring from piles 1 and 2, resulting in the board $(0, 3, 0)$. Frodo's only choice is to apply rule (2) and select either 1 or 2 rings from pile 2, resulting in the possible boards $(0, 2, 0)$ and $(0, 1, 0)$. In either case, Bilbo applies rule (2) and removes the remaining rings from the second pile, leaving Frodo with the board $(0, 0, 0)$. Since Frodo cannot move he loses.

Note that since 7 moves are possible each time, with n rings in each pile the number of possible combination of moves is 7^n. With 100 rings in each pile, the number of possible combinations to consider is more than 10^{84}, well beyond the capability of modern computers.[1] As a result you will need to consider solutions that do not require explicitly examining all possible combinations of moves.

Input format

The input consists of a sequence of triples of numbers separated by blanks. Each line represents a starting board. The program stops when the triple "0 0 0" is input. You may assume that no pile contains more than 100 rings.

Output format

For each input line, echo the input and then print "Frodo wins" or "Bilbo wins", depending on which player has a winning strategy.

Examples

Input 1:	Output 1:
`1 4 0`	`1 4 0 Bilbo wins`
`0 2 1`	`0 2 1 Frodo wins`
`5 7 8`	`5 7 8 Bilbo wins`
`10 4 9`	`10 4 9 Bilbo wins`
`0 0 0`	

Input 2:	Output 2:
`0 0 9`	`0 0 9 Bilbo wins`
`0 9 0`	`0 9 0 Frodo wins`
`9 0 0`	`9 0 0 Bilbo wins`
`8 17 6`	`8 17 6 Bilbo wins`
`0 0 0`	

[1] The world's fastest computer is currently the NEC Earth Simulator, a 640-node 8-way vector supercomputer. The Earth Simulator can perform 35 trillion calculations per second, but would take 10^{63} years to consider every possible combination of moves for a game with 100 rings.

8 Grokking Names of Middle Earth (GNOME)

Frodo has been wandering in magical lands for so long that he is finding it difficult to keep track of all the places he has been, creatures he has met, and magical objects he has encountered. You decide to help him by building a database that stores the names of places, creatures, magical objects, and other items that Frodo may want to record. You decide to name this database GNOME, for Grokking[2] Names of Middle Earth. Since GNOME must run on an ancient computer with limited external storage and even more limited random-access memory, you need to encode the names in a space-efficient format that is easy to decode. You decide on the following scheme for storing names:

Database Format Names are stored in the database in lexicographic order (defined below). The database entry for each name consists of two parts. The first part, called the *differential offset*, encodes the prefix of the name that is identical to the name occurring just before it in the database. The second part, called the *name suffix*, stores the part of the name after the prefix encoded by the differential offset. The differential offset for the first entry in the database is always 0, and the name suffix is the entire name of that entry. For every other entry in the database, let p be the length of the *longest common prefix* (defined below) of the name that entry represents and the name of the previous entry in the database. We refer to p as the *offset* for this entry. For example, if the third name is `factoid` and the fourth name is `factotum`, the longest common prefix for the fourth database entry is `facto`, and its length, p, is 5. The name suffix of an entry (other than the first) is the string obtained by removing longest common prefix from the name. In our example, the name suffix is `tum`, obtained by removing the prefix `facto` from `factotum`. The differential offset of an entry (other than the first) is the difference between its offset and the offset of the previous entry in the database. (The offset of the first entry is defined to be 0.) In our example, if the offset of the third entry is 3, then the differential offset of the fourth entry is $5 - 3 = 2$. Note that the differential offset is negative for entries with an offset smaller than that of the preceding database entry.

Two's Complement Representation The representation of differential offsets in GNOME is based on the *two's complement* representation of numbers, which we now describe. In this representation, a byte can represent numbers in the range $[-2^7, 2^7 - 1] = [-128, 127]$. Nonnegative numbers are represented by their usual binary forms. For example, 5 is represented as 0000 0101 (since $5 = 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1$). The two's complement binary representation of a negative number (e.g., -5) is obtained by starting with the binary representation of the magnitude (0000 0101), complementing each bit (1111 1010), and adding one to the result, ignoring any overflow bits (1111 1010 + 1 = 1111 1011). To determine the number denoted by a byte in this representation, we begin by examining the most significant bit (MSB). If it is 0, the byte is the binary representation of a nonnegative number. For example, 00110101 has MSB 0, and encodes the number 53. If the MSB is 1, the byte represents a negative number whose magnitude is obtained by complementing each bit and adding one to the result. For example, given 10110010, we note that the MSB is 1, complement each bit to obtain 01001101, and add 1 to get 01001110, or 78 as the magnitude of the negative number. Thus, 10110010 is the two's complement representation of -78. This scheme extends to two or more bytes in a natural manner. For example, 1111 1101 0100 0110 has MSB 1. Therefore it represents a negative number. To obtain the magnitude of this number, we complement each bit and add 1 to obtain 0000 0010 1011 1001 + 1 = 0000 0010 1011 1010, which is 698 in decimal. Thus 1111 1101 0100 0110 represents -698.

Encoding the Offset There are two cases to consider in determining the representation of the differential offset in GNOME:

Case 1: If the differential offset is in the range $[-2^7 + 1, 2^7 - 1] = [-127, 127]$, we encode it by the byte corresponding to its two's complement representation, as described above. (Note that we do not use the representation for -128, which is 10000000.)

Case 2: If the differential offset is in the range $[-2^{-15}+1, -2^7] \cup [2^7, 2^{15}-1]$, i.e, $[-32767, -128] \cup [128, 32767]$, we encode it using three bytes. The first byte is the marker byte 1000 0000. The following two bytes are the two's complement representation of the differential offset. For example, a differential offset of -698 is represented in GNOME by the following three byte sequence: 1000 0000 1111 1101 0100 0110.

[2]To grok something is to understand it well. The origin of the term is Robert Heinlein's book *Stranger in a Strange Land.*

Summary A GNOME database is a sequence of entries. Each entry in the database represents one name, and the entries are sorted in lexicographic order of the names. Each entry consists of a differential offset, coded using one or three bytes as described above, and the name suffix, which is obtained from the name by removing the prefix it shares with the previous entry.

Many moons later, this database format would be rediscovered by a gnu called *Loca Tes...*

Input Format

The first line of the input always contains a single character (followed by the newline character that terminates the line), called the *command character*, which is either `e` or `d`.

If the command character is `e`, your program is required to encode the names that follow into the GNOME database. (Your program must output the resulting database, as described in the Output section.) In this case, the second line contains a single integer indicating the number of names that follow. The rest of the input consists of a list of names, one per line. Frodo has agreed to use names consisting of at most 500 characters. Further, he will limit his choice of characters to the alphanumeric characters (a through z, A through Z, and 0 through 9), `/` (slash), and `-` (dash). Frodo has also indicated that he will never need to store more than 1000 names. (We will not give you any input that Frodo wouldn't.) You may assume that the input will not contain any blank lines.

If the command character is `d`, your program is required to decode the pretty-printed version of a GNOME database given by the rest of the input. (Your program must output the decoded list of names, as described in the Output section.) In this case, the second line contains a single integer indicating the number of database entries that follow. The rest of the input consists of a list of database entries, one per line, in database order (sorted by name). Each line consists of a pretty-printed representation of the differential-offset-encoding bytes, followed by a single space character, followed by the name suffix. The encoding bytes are pretty-printed as sequence of `0` and `1` characters, with a space between every consecutive pair of nybbles. (A nybble is half a byte, or 4 bits.)

Output Format

If the command character in the input is `e`, the first line of output produced by your program must be the single character `d` (followed by the line-terminating newline character). The second line of the output is the number of entries in the database. The following lines of output must be a pretty-printed version of the GNOME database built from the input names. The output format for this database is identical to the pretty-printed database format described in the Input section.

If the command character in the input is `d`, the first line of output produced by your program must be the single character `e` (followed by the line-terminating newline character). The second line of the output is the number of entries in the database. The rest of the output must list the names encoded by the database, one per line, in database order.

Examples

Consider the first input depicted below. The first column of Figure 1 depicts the result of sorting the names in lexicographic order. The second column of the figure indicates the offset for each entry except the first; it is the length of the longest common prefix of that entry and the previous one. The third column lists the differential offset for each entry, obtained as the difference between the offsets of that entry and the previous entry. The last column lists the encoded differential offset for each entry. In this example, all differential offsets are in the range $[-127, 127]$ and are therefore coded using only one byte. The second input-output pair depicted below illustrates the use of three bytes to code differential offsets that are too large. Note that that the strings of the form "`...[200 a's in all]...`" are not part of the input or output, but are used to suggest the presence of a large number of `a`s in the input string. The third input-output pair illustrates the program operating in decoding mode.

Name	Offset	Differential	Coded
Beruthiel	0	0	0000 0000
Beruthiel/Cat	9	9	0000 1001
Ghan-Buri-Ghan	0	-9	1111 0111
Gil-Galad	1	1	0000 0001
Gondar	1	0	0000 0000
Gondar/Aragon	6	5	0000 0101
Gondar/Boromir	7	1	0000 0001
Gondar/Denethor	7	0	0000 0000
Gondar/Denethor/brother/Faramir	15	8	0000 1000
Gondar/Denethor/castle/Minast-Irith	16	1	0000 0001
Orthanc	0	-16	1111 0000
Orthanc/ruler/Saruman	7	7	0000 0111
Shalom	0	-7	1111 1001

Figure 1: Worked example

Note that the output of the program is, by design, valid input to the program. If we use $P(I)$ to denote the output of the program on input I, then $P(P(P(P(I)))) = P(P(I))$. You may use this property to test your code.

Input 1:

```
e
13
Gondar/Denethor
Gondar
Orthanc/ruler/Saruman
Ghan-Buri-Ghan
Gondar/Denethor/brother/Faramir
Gil-Galad
Gondar/Denethor/castle/Minast-Irith
Beruthiel/Cat
Shalom
Gondar/Aragon
Orthanc
Gondar/Boromir
Beruthiel
```

Output 1:

```
d
13
0000 0000 Beruthiel
0000 1001 /Cat
1111 0111 Ghan-Buri-Ghan
0000 0001 il-Galad
0000 0000 ondar
0000 0101 /Aragon
0000 0001 Boromir
0000 0000 Denethor
0000 1000 /brother/Faramir
0000 0001 castle/Minast-Irith
1111 0000 Orthanc
0000 0111 /ruler/Saruman
1111 1001 Shalom
```

Input 2:

```
e
4
baa
aaa...[200 a's in all]...aaa
b
aaa...[200 a's in all]...aaabcd
```

Output 2:

```
d
4
0000 0000 aaa...[200 a's in all]...aaa
1000 0000 0000 0000 1100 1000 bcd
1000 0000 1111 1111 0011 1000 b
0000 0001 aa
```

Input 3:

```
d
4
0000 0000 benito
0000 0011 jamin
1111 1101 cucaracha
0000 0011 koo
```

Output 3:

```
e
4
benito
benjamin
cucaracha
cuckoo
```

Appendix: Definitions

Lexicographic Order on Strings To compare two strings lexicographically, we compare each character in one string with the corresponding character of the other until we find a pair of characters that are unequal, or we run out of characters in one or both strings. If a pair of unequal characters is fount, the string with the smaller character is lexicographically smaller than the other. If we run out of characters in only one string, that string is lexicographically smaller than the other. Otherwise (we run out of characters in both strings), the two strings are lexicographically equal (and thus identical). We order characters as follows: `-, /, 0, 1, ..., 9, A, B, C, ..., Y, Z, a, b, c, ..., y, z`. (Frodo has agreed to use only these characters in names.) For example, `foobar` is smaller than `fooboo` (first case) and `foo` is smaller than `foobar` (second case).

Longest Common Prefix We now define the longest common prefix of two strings, which we used in our description of the GNOME database format. Given two strings $s_1 = a_1 a_2 \dots a_m$ and $s_2 = b_1 b_2 \dots b_n$, let p be the largest integer in $[0, \min(m, n)]$ such that $a_i = b_i$ for all $0 < i < p$. (Note that p may be 0, indicating that the two strings differ in their first characters.) If $p > 0$, the string $a_1 a_2 \dots a_p$ is the longest common prefix of s_1 and s_2. If p is 0, the longest common prefix of the two strings is the empty string, ϵ.

0 Keyword Search

Elrond is reading histories of Middle-Earth from his castle in Rivendell, trying to find clues on how to destroy Sauron's ring. To help him find the relevant passages, you attempt to write a program to scan through text, returning all occurrences of a keyword and 12 letters of surrounding text. Assume your search is case-sensitive (i.e., upper/lower cases have to match) and that the text can be read in as a single string.

Input Format

The input is simply two sequences of letters, each on a separate line. The keyword comes first, followed by the full text. You may assume that the longest length for either keyword or text is at most 250 letters.

Output Format

The output is a series of matches of the keyword in the passage. For each match, return 12 letters before the occurrence of the keyword, the keyword, and 12 letters after the keyword. If the keyword is close to the beginning of end the text passage, you may need to return fewer than 12 letters around the keyword.

Examples

Input 1:

```
elf
I.will.be.dead.before.I.see.the.ring.in.the.hands.of.an.elf!.Never.trust.an.elf!
```

Output 1:

```
hands.of.an.elf!.Never.trus
er.trust.an.elf!
```

Input 2:

```
Sauron
This.is.the.one.ring,.forged.by.the.Dark.Lord,.Sauron.in.the.fires.of.Mount.Doom,
.taken.by.Isildur.from.the.hand.of.Sauron.himself.
```

Output 2:

```
.Dark.Lord,.Sauron.in.the.fire
the.hand.of.Sauron.himself.
```

Input 3:

```
ring
You.are.a.ring-bearer,.Frodo..To.bear.a.ring.of.power.is.to.be.alone.
```

Output 3:

```
You.are.a.ring-bearer,.Fro
..To.bear.a.ring.of.power.is
```