Problems Index Sat Oct 08 05:29:45 AM EDT 2011

BOSPRE 2011 PROBLEMS
------ ---- --------

The problems are in approximate order of difficulty, easiest first.

```
    problems/sniffer
        Find your way.
        Boston Preliminary 2011

    problems/vlsicompact
        Squeeze and squish makes the dish.
        Boston Preliminary 2011

    problems/relativeneighbor
        I’m closer than he is.
        Boston Preliminary 2011

    problems/boolecipher
        Binary obscuration.
        Boston Preliminary 2011

    problems/delaunay
        Small circles are good circles.
        Boston Preliminary 2011

    problems/boolebreak
        So they thought they could fool you!
        Boston Preliminary 2011

    problems/opttriangulation
        For the triangular sophisticate.
        Boston Preliminary 2011

    problems/abduction
        Does your mind really work like this?
        Boston Preliminary 2011
```

```
Sniffer
-------

Sniffer has the task of marking the trail through the
Hungry Woods so her pack can travel it quickly at night.
Fortunately the path does not have any forks, so no
searching is required, but sometimes its hard to see
which way the path goes.

The path and forest are represented by an array of char-
acters which is a map of the forest area.  The edges of
this map array are marked with ‘+’s at the corners, ‘-’s
at top and bottom, and ‘|’s at the edges.  Within the
map ‘#’ represents impassable forest and ‘ ’ (single
space) represents the path or open passable forest.  The
path starts at the upper left corner (array row 2,
column 2) but may exit anywhere next to a map edge.
Sniffer and her pack members can only move on the path
by going left, right, up, or down; they CANNOT go dia-
gonally.  At any point on the path there is at most one
way to continue onward.  Note that the path may run
along the edge of the map.  The path ends when there is
no way to move forward, and Sniffer knows that the path
will not deadend inside the forest (i.e., surrounded by
‘#’s).

Sniffer marks the path by changing the ‘ ’ space charac-
ters that are on the path from ‘ ’ to ‘:’.


Input
-----

For each of several test cases, first a line containing
the test case name, then R lines each containing C char-
acters, which encode the array.  Each of the characters
in the R lines is ‘+’, ‘-’, ‘|’, ‘#’, or ‘ ’ and repre-
sents one element of the array.  R is the number of rows
in the array and C the number of columns, and each row
line has exactly C characters.  The R array lines are
followed by a line containing just ‘.’ that ends the
test case.

6 <= R,C <= 50.  No line, including the test case name
line, is longer than 80 characters.

The input ends with an end of file.


Output
------

For each test case, an exact copy of the input for the
test case, but with array characters on the path changed
from ‘ ’ to ‘:’.
```

```
Sample Input
------ -----

-- SAMPLE 1 --
+----------+
|  # #### #|
|# ##    ##|
|# ## ##   |
|#    # ###|
|######    |
+----------+
.
-- SAMPLE 2 --
+------------------------------+
| #  #########    #      # #  #|
| # #        ###### #### ######|
| ### ###### ###### #  # ######|
|     ######        #  #      #|
|######     #########   #####  |
+------------------------------+
.
```

```
Sample Output
------ ------

-- SAMPLE 1 --
+----------+
|::# #### #|
|#:##::::##|
|#:##:##:::|
|#::::# ###|
|######    |
+----------+
.
-- SAMPLE 2 --
+------------------------------+
|:#  #########    #::::::# #  #|
|:# #::::::::######:####:######|
|:###:######:######:#  #:######|
|:::::######::::::::#  #::::::#|
|######     #########   #####::|
+------------------------------+
.
```

File: sniffer.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Mon Oct 10 00:47:29 EDT 2011

RCS Info (may not be true date or author):

    $Author: walton $
    $Date: 2011/10/10 04:47:38 $
    $RCSfile: sniffer.txt,v $
    $Revision: 1.5 $

```
VLSI Compaction
---- ----------

When laying out a VLSI circuit, the following problem
arises after the circuit has been initially laid out.
The problem is to squish the circuit into a minimum
area.  It is too difficult to do this in more than one
dimension, but not hard to do it in a single dimension.

Abstractly the problem is as follows.  Given a set of
N horizontal coordinate values x[i] (actually the hori-
zontal coordinates of VLSI transistors and larger `com-
ponents'), and a set of constraints of the form

        0 <= d[i,j] <= x[j] - x[i]   where j > i

find positions x[i] such that x[i] - x[1] is minimized.
Here the value index i in x[i] ranges from 1 through N
and there is one constraint for every i and j for
which j > i, though d[i,j] = 0 will be true for many
of these.


Input
-----

For each of several test cases, first a line containing
just the test case name.  This is followed by one or
more lines containing the following non-negative integer
numbers:

        N d[1,2] d[1,3] d[1,4] ... d[1,N]
                 d[2,3] d[2,4] ... d[2,N]
                        d[3,4] ... d[3,N]
                               ..........
                               ... d[N-1,N]

The integers are in the given order, but any kind of
whitespace (spaces, tabs, line breaks) may occur between
any two consecutive integers.

2 <= N <= 100

Input ends with an end of file.


Output
------

For each test case two lines.  The first line is an
exact copy of the test case name input line.  The second
line contains the integers

        x[1] x[2] ... x[N]

in the given order, such that x[i] - x[1] is minimized
for i = 2, 3, ..., N.

x[1] = 0 is required (otherwise the numbers would be
under-determined).


Sample Input
------ -----

-- SAMPLE 1 --
5 3 0 8 2 3 0 3 2 0 8
-- SAMPLE 2 --
8 0 0 4 5 0 3 9
    2 3 1 2 1 0
      7 0 8 1 3
        2 0 5 2
          1 2 3
            1 0
              0
```

```
Sample Output
------ ------

-- SAMPLE 1 --
0 3 6 8 16
-- SAMPLE 2 --
0 0 2 9 11 12 14 14


File:       vlsicompact.txt
Author:     Bob Walton <walton@seas.harvard.edu>
Date:       Mon Oct 10 21:33:29 EDT 2011

The authors have placed this file in the public domain;
they make no warranty and accept no liability for this
file.

RCS Info (may not be true date or author):

    $Author: walton $
    $Date: 2011/10/11 01:34:33 $

    $RCSfile: vlsicompact.txt,v $
    $Revision: 1.7 $
```

Relative Neighbor Graphs

Given a set of points in a plane, the associated relative neighbor graph has an edge between two points P1 and P2 if and only if there is NO point P3 such that

```
    d(P3,P1) < d(P1,P2)  and d(P3,P2) < d(P1,P2)
```

where d(Px,Py) is the distance between Px and Py.

You have been asked to compute the relative neighbor graph of a set of points.

Input

For each test case, first a line containing just the test case name, and then lines containing the numbers

```
        N x[1] y[1] x[2] y[2] ... x[N] y[N]
```

where there are N points and (x[i],y[i]) is the i'th point for 1 <= i <= N. On these lines numbers may be separated by any whitespace, including spaces, tabs, and line feeds.

3 <= N <= 100. The x,y coordinates may be any floating point numbers.

Input ends with an end of file.

Output

For each test case, first a line that is an exact copy of the test case name input line. Then one line for each edge in the relative neighbor graph, this line having the format

```
        i j
```

to specify that there is an edge from (x[i],y[i]) to (x[j],y[j]). Here 1 <= i,j <= N. Do NOT output any edge more than once.

The output may be printed as a graph or displayed in an X-window by the commands:

```
        print_graph
        display_graph
```

provided the input and output of your program has been stored in the files

```
        relativeneighbor.in
        relativeneighbor.out
```

and the test case name lines in these files do not have a digit as their first non-whitespace character. To see the sample output instead use the commands

```
        print_graph sample.in sample.test
        display_graph sample.in sample.test
```

(here sample.test is the output for sample.in).

```
Sample Input
------ -----

-- SAMPLE 1 --
3 1 4 3 2 5 8
-- SAMPLE 2 --
7 -1.01 0 -1.01 5 1.01 2.01 3.04 3.02
  5.05 2.003 8.21 0 8.22 5.03


Sample Output
------ ------

-- SAMPLE 1 --
1 2
1 3
-- SAMPLE 2 --
1 3
2 3
3 4
4 5
5 6
5 7


File:      relativeneighbor.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Sun Oct  2 03:59:50 EDT 2011

The authors have placed this file in the public domain;
they make no warranty and accept no liability for this
file.

RCS Info (may not be true date or author):

    $Author: walton $
    $Date: 2011/10/02 08:05:34 $
    $RCSfile: relativeneighbor.txt,v $
    $Revision: 1.6 $
```

Boole Ciphers
----- -------

A Boole Cipher consists of a list of all the characters that can be used in a message in the form of a binary tree which has the syntax:

```
    tree ::= leaf | [left-child  right-child]
    left-child ::= tree
    right-child ::= tree
    leaf ::= character other than ‘[’ or ‘]’
```

For example,

```
        [[[eH][r ]][h[Ti]]]
```

represents the binary tree

```
                      _________*________
                     /                  \
                 ___*___              ___*___
                /       \            /       \
               *         *          h         *
              / \       / \                  / \
             e   H     r                    T   i
```

An encrypted message is a sequence of 0’s and 1’s that is interpreted by the following process:

(1) Start at the tree root.
(2) On reading a 0, move to the left child of the current tree node.
(3) On reading a 1, move to the right child of the current tree node.
(4) On reaching a leaf in (2) or (3), output the label character on the leaf and move back to the tree root.

In other words, each character is represented by the label of the path from the root to the character, where the path label is a sequence of 0’s and 1’s, with 0 meaning ‘move to the left child’ and 1 meaning ‘move to the right child’.

For example, the message ‘Hi There’ is encrypted using the above Boole Cipher as ‘0011110111101000001000O’.

You are given Boole Ciphers and messages encrypted with these ciphers and are being asked to decrypt the messages.

Input

For each of several test cases, first a line containing the test case name, second a line containing a Boole Cipher, and third a line containing a message encrypted using that cipher. The input ends with an end of file.

WARNING: The input lines can be up to 1000 characters long!

The characters ‘[’ and ‘]’ do not appear in messages or as leaves in ciphers, but any other ASCII character that prints a mark, and the single space character, can appear. No character appears more than once as a cipher leaf. At least two characters will appear in each cipher.

Output

For each test case, first an exact copy of the test case name line, second an exact copy of the encrypted test case message line copied from the input, and third the decrypted test case message on a line by itself.

Sample Input
------ -----

```
-- SAMPLE 1 --
[[[eH][r ]][h[Ti]]]
00111101111010000010000
-- SAMPLE 2 --
[[[t[h?]][[ s]a]]W]
10010011000010101000000001001100000011
-- SAMPLE 3 --
[o[ H]]
11010110
-- SAMPLE 4 --
[[h[[[e ][st]]a]][[oW][?g]]]
1010001101011010011111000100001010110
-- SAMPLE 5 --
[[eB][[[io]g][[sl][[ a][t[!r]]]]]]
011000111101100111001011110111011001111111100111110
```

Sample Output
------ ------

```
-- SAMPLE 1 --
00111101111010000010000
Hi There
-- SAMPLE 2 --
10010011000010101000000001001100000011
Whats that?
-- SAMPLE 3 --
11010110
Ho Ho
-- SAMPLE 4 --
1010001101011010011111000100001010110
What goes?
-- SAMPLE 5 --
011000111101100111001011110111011001111111100111110
Bits galore!
```

File: boolecipher.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Mon Oct 10 21:36:00 EDT 2011

RCS Info (may not be true date or author):

```
    $Author: walton $
    $Date: 2011/10/11 01:36:51 $
    $RCSfile: boolecipher.txt,v $
    $Revision: 1.9 $
```

```
Delaunay Triangulation
-------- -------------

You have been asked to find the Delaunay Triangulation
of a set S of points in the plane.

The Delaunay Triangulation of a set S of points in the
plane is a triangulation of the convex hull of S such
that the circumcircle of each triangle has no points of
S in its interior.  As long as there is no circle with 4
or more points of S on its boundary and no points of S
in its interior, the Delaunay Triangulation of S is
unique, and the edges of the triangulation are just the
edges of triangles with vertices in S which have no
points of S in the interior of their circumcircle.

The Delaunay Triangulation of S is coveted because among
all the possible triangulations of S it is the one that
maximizes the minimum angle between edges of the triang-
ulation.


Input
-----

For each of several test cases, first a line containing
nothing but the name of the test case, and then lines
containing the numbers

    N x[1] y[1] x[2] y[2] ... x[N] y[N]

where (x[i],y[i]) is the i'th point of S for 1 <= i <=
N.  3 <= N <= 100.  The xy coordinates are floating
point.

To simplify things, the input will be such that the
Delaunay triangulation is unique; that is, no 4 points
of S will be on the same circle if that circle contains
no points of S in its interior.

Input ends with an end of file.


Output
------

For each test case, first a line that is an exact copy
of the test case name input line.  Then one line for
each edge of the Delaunay Triangulation of S, this line
having the format

        i j

to specify that there is an edge from (x[i],y[i]) to
(x[j],y[j]).  Here 1 <= i,j <= N.  Do NOT output any
edge more than once.

The output may be printed as a graph or displayed in an
X-window by the commands:

        print_graph
        display_graph

provided the input and output of your program has been
stored in the files

        delaunay.in
        delaunay.out

and the test case name lines in these files do not have
a digit as their first non-whitespace character.  To see
the sample output instead use the commands

        print_graph sample.in sample.test
        display_graph sample.in sample.test

(here sample.test is the output for sample.in).
```

```
Note: The relative neighbor graph computed in the
Relative Neighbor Graphs problem is a subgraph of the
Delaunay Triangulation.


Sample Input
------ -----

-- SAMPLE 1 --
3 1 4 3 2 5 8
-- SAMPLE 2 --
7 -1.01 0 -1.01 5 1.01 2.01 3.04 3.02
  5.05 2.003 8.21 0 8.22 5.03


Sample Output
------ ------

-- SAMPLE 1 --
1 2
2 3
1 3
-- SAMPLE 2 --
1 2
2 3
1 3
3 5
1 5
5 6
1 6
3 4
2 4
4 7
2 7
4 5
5 7
6 7
```


File: delaunay.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Mon Oct 3 05:59:33 EDT 2011


```
RCS Info (may not be true date or author):

    $Author: walton $
    $Date: 2011/10/03 10:53:48 $
    $RCSfile: delaunay.txt,v $
    $Revision: 1.5 $
```

Breaking Boole Ciphers
-------- ----- -------

The enemy is using Boole Ciphers (see the Boole Cipher problem). Your spies have intercepted some messages in both encrypted and unencrypted form, and you have been asked to find the Boole Ciphers used to encrypt these messages.

Input

For each of several test cases, three lines. First a line containing the test case name, second a line containing the encrypted message, and third a line containing the unencrypted message. The input ends with an end of file.

The characters '[', ']', and '@' do not appear in unencrypted messages, but any other ASCII character that prints a mark, and the single space character, can appear.

WARNING: The input lines can be up to 1000 characters long!

Output

For each test case, three lines. First, an exact copy of the test case name line, second a line containing the Boole Cipher used to encrypt the message, and third a line copied from the input containing of the encrypted message.

The Boole Cipher may be under-determined. You are to output the cipher which gives the smallest cipher tree depth (i.e., length of longest path from the root), and among these the shortest encoding for the first character of the message, and among these the shortest encoding for the second character, etc. No character may label two leaves of the cipher.

The cipher tree depth MUST NOT be greater than 8. If no cipher with depth <= 8 can be found, output 'FAILED' in place of the cipher.

Also some subtrees of the cipher tree may be undetermined, and these are represented by the single character '@'.

The output file is formatted so it can be input to the boolecipher problem solution to reproduce the input file (if FAILED cases are excluded).

```
Sample Input
------ -----

-- SAMPLE 1 --
0011110111101000001000
Hi There
-- SAMPLE 2 --
1001001100001010100000000100110000011
Whats that?
-- SAMPLE 3 --
1101101101
Ho Ho
-- SAMPLE 4 --
10100011010110100111110001000011111111
What goes?
-- SAMPLE 5 --
011000111101100111001011110111011001111111100111111
Bits galore!


Sample Output
------ ------

-- SAMPLE 1 --
[[[eH][r ]][h[Ti]]]
0011110111101000001000
-- SAMPLE 2 --
[[[t[h?]][[ s]a]]W]
1001001100001010100000000100110000011
-- SAMPLE 3 --
[[@o][ H]]
1101101101
-- SAMPLE 4 --
FAILED!
10100011010110100111110001000011111111
-- SAMPLE 5 --
FAILED!
011000111101100111001011110111011001111111100111111
```


```
File:      boolebreak.txt
Author:    Bob Walton <walton@seas.harvard.edu>
Date:      Tue Oct  4 04:57:27 EDT 2011
```



```
RCS Info (may not be true date or author):

    $Author: walton $
    $Date: 2011/10/04 08:58:30 $
    $RCSfile: boolebreak.txt,v $
    $Revision: 1.7 $
```

Optimal Triangulation

A triangulation of a polygon is a division of the area of the polygon into disjoint triangles whose vertices are vertices of the polygon. Given an assignment of values to triangles, an optimal triangulation is a triangulation for which the sum of the values of the triangles is maximal.

You have been asked to find optimal triangulations of convex polygons. The triangle value function is represented in fully parenthesized prefix operator notation using the components:

a, b, c	The sizes in degrees of the angles of the triangle.
A, B, C	The lengths of the sides of the triangle. Side A is opposite angle a, side B opposite angle b, side C opposite angle c.
+	Sum of arguments.
*	Product of arguments.
-	If one argument, the negative of that, and if two arguments, the first minus the second. Illegal for more than two arguments.
^	Maximum of arguments. (circumflex)
v	Minimum of arguments. (letter v)

There is no whitespace in a function representation; for example, (+abc) denotes the sum of angles of a triangle (which always equals 180). A '(' is always followed by an operator, i.e., by '+', '*', '-', '^', or 'v'. An operator is always preceded by a '('. Operators always have at least 2 arguments, except '-' which can have one argument. Functions can return negative values.

The value function is required to be symmetric under permutation of the angles of a triangle, with sides being changed in corresponding fashion. For example,

```
(v(*aBC)(*bCA)(*cAB))
```

is symmetric, but (+ab) is not.

You are to represent a triangulation as a list of vertex triples, one for each triangle, giving the vertices of the triangle. If there are N polygon vertices, there will be N-2 triangles.

Input

For each of several test cases, a line containing just the test case name, followed by a line containing the triangle valuation function, followed by a lines containing the following numbers

```
N x[1] y[1] x[2] y[2] ... x[N] y[N]
```

where N is the number of vertices of the convex polygon and the vertices in counter-clockwise order are (x[1],y[1]), (x[2],y[2]), ..., (x[N],y[N]). These numbers are separated by spaces and new lines. The x and y coordinates are floating point numbers in the range [-1000,1000]. The polygons are guaranteed to be convex, without any 3 vertices being on a straight line.

3 <= N <= 100. Lines will have no more than 80 characters. Input ends with an end of file.

```
Output
------

For each test case, a line that is an exact copy of the
test case name input line, followed by N-2 lines each
with the format

        i j k

specifying the triangle whose vertices are

        (x[i],y[i])     (x[j],y[j])     (x[k],y[k])

These triangles give the desired triangulation whose
sum of triangle values is maximal.  The input will be
such that this triangulation is unique.



Sample Input
------ -----

-- SAMPLE 1 --
(^abc)
4 0 0 1 0 1 2 0 1
-- SAMPLE 2 --
(vabc)
4 0 0 1 0 1 2 0 1

Sample Output
------ ------

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


```
File:       opttriangulation.txt
Author:     Bob Walton <walton@seas.harvard.edu>
Date:       Tue Oct 11 08:08:21 EDT 2011
```



```
RCS Info (may not be true date or author):

    $Author: walton $
    $Date: 2011/10/11 12:08:34 $
    $RCSfile: opttriangulation.txt,v $
    $Revision: 1.7 $
```

Logical Abduction
------- ---------

Abduction is the process of finding hypotheses that explain observed facts. Suppose p, q, r, s, t, etc represent propositions that are either true or false. Let ‘p&q=>r’ be an ‘implication’ that means ‘p and q together imply r’. Suppose you know

```
        p&q=>r
        p&s=>r
        q&s=>p
```

and you want to explain

```
        r
```

The ‘hypotheses’ of an implication are the propositions appearing before the ‘=>’, and the conclusion is the proposition appearing after the ‘=>’. E.g., in ‘p&q=>r’ p and q are the hypotheses and r is the conclusion.

An ‘abduction’ is a set of implications that are used to derive the propositions to be explained. The hypotheses of the abduction are the hypotheses of used implications that are NOT the conclusions of any used implication, and also any propositions to be explained that are NOT conclusions of any used implication.

For example, if just

```
        p&q=>r
```

is used in an abduction of r, then p and q are the hypotheses of the abduction. If

```
        p&s=>r
        q&s=>p
```

are used instead, q and s are the hypotheses of the abduction. If NO implications are used, r is the sole hypothesis of the abduction.

In order to decide which abduction is best we assign a cost for assuming each proposition and try to minimize the total cost of all the hypotheses of the abduction. For each possible abduction costs are assigned according to the following rules:

(R1) The propositions to be explained are assigned a cost directly. These costs are strictly positive.

For example, we will write ‘r[10]’ to indicate that r is assigned the cost 10.

(R2) For an implication like ‘p&q=>r’, the hypotheses of the implication are assigned a weight, typically a small positive fraction. If the implication is used in the abduction, the cost of each hypothesis is assigned to be the cost of the conclusion times the weight of the hypothesis. Note that an impli- cation may not be used in the abduction if its conclusion has not been assigned a cost.

For example, the above implications might be written:

```
   p[0.5]&q[0.6]=>r
   p[0.4]&s[0.2]=>r
   q[0.3]&s[0.7]=>p
```

to indicate, for example, that if the first implication is used, p is assigned a cost equal to 0.5*cost-of-r. If r costs 10 and the first implication is used in an abduction, this implica- tion assigns 0.5*10 = 5 to p and 0.6*10 = 6 to q.

```
(R3) If a proposition is assigned more than one cost,
     the minimum cost is used for that proposition in
     ALL calculations.

     Thus if the abduction to explain r[10] uses the
     implications

        p[0.4]&s[0.2]=>r
        q[0.3]&s[0.7]=>p

     the costs are r = 10, p = 0.4*10 = 4, s = 0.2*10 =
     2, q = 0.3*4 = 1.2, s = 0.7*4 = 2.8, and as s has
     been assigned two costs, 2 and 2.8, the MINIMUM
     s = 2 is used.

     If a cost is reduced according to this rule, all
     costs calculated from this cost are correspondingly
     reduced.  Thus if the abduction also used a third
     implication

        p[0.3]&q[0.8]=>r

     that assigned p = 3, then the last implication

        q[0.3]&s[0.7]=>p

     would have to be revisited to assign q = 0.3*3 =
     0.9 and s = 0.7*3 = 2.1, and in the case of q
     this would reduce its previous cost of 1.2 to the
     new minimum 0.9.

(R4) After assigning proposition costs according to the
     above rules, the cost of the abduction is the sum
     of the costs of its HYPOTHESES plus 0.1 times the
     number of implications used in the abduction.

     Thus if just the first implication above is used in
     the abduction the cost is

       5 (i.e., cost of q) + 6 (i.e., cost of p)
       + 0.1 * 1 (number of implications) = 11.1

     and if instead the second two implications are used
     the cost is

       1.2 (i.e., cost of q) + 2 (i.e., cost of s)
       + 0.1 * 2 (number of implications) = 3.4

     Its also possible to use NO implications, in which
     case the cost is 10, i.e., the cost of directly
     assuming r.

     Lastly it is possible to use all three implica-
     tions, in which case the cost is

       1.2 (i.e., cost of q) + 2 (i.e., cost of s)
       + 0.1 * 3 (number of implications) = 3.5

     Notice that the hypothesis cost would be the same
     as the hypothesis cost of using just the last two
     implications, that is, addition of the first
     implication to the last two does not change the
     hypothesis cost, but we have added 0.1 times the
     number of implications as a penalty for such
     superfluous implications.

     If a minimum cost abduction is sought, the second
     two implications would be used.

You are being asked to find minimum cost abductions.
```

Input

Propositions are denoted by single LOWER CASE letters, and numbers are non-negative floating point numbers. In the following P denotes any proposition letter and # any number.

The input consists of any number of test cases. Each test case begins with a single line containing the test case name. This is followed by any number of lines of the formats:

```
        P[#]
        P[#]=>P
        P[#]&...&P[#]=>P
```

and these are followed by a line containing just ‘.’.

A line of the first above format defines a proposition to be explained for which # is its cost. A line of the second format defines an implication with a single hypothesis, and a line of the third format defines an implication with two or more hypotheses. In these last two cases the #’s are the weights of the implication hypotheses.

There are no space or tab characters in the input outside the test case name lines.

There are at most 100 implications in a test case. No proposition can be in more than one ‘P[#]’ line specifying a proposition to be explained, so there can be at most 26 such lines.

Input ends with an end of file.

Output

For each test case first an exact copy of the test case name line and then lines with similar formats to those used in the input, terminated by a line containing just ‘.’. The output describes the minimum cost abduction for the given test case input. There is one line for every abduction hypothesis, and one line for every implication in the abduction.

The output line for each abduction hypothesis has the format ‘P[#]’ which means that P has minimum cost #.

The output line for each implication in the abduction has the format ‘P[#]=>P[#]’ or ‘P[#]&...&P[#]=>P[#]’, where the #’s have the following meanings. The # for the conclusion is the minimum cost assigned to the conclusion. The # for each hypotheses is the cost assigned to the hypothesis by the implication, i.e., the hypothesis weight times the cost of the conclusion. This last may NOT be the minimum cost assigned to the hypothesis by all implications.

The numbers output must must have exactly 2 decimal places. There may be no spaces in any output line other than the test case name lines.

```
Sample Input
------ -----

-- SAMPLE 1 --
r[10]
p[0.5]&q[0.6]=>r
p[0.4]&s[0.2]=>r
q[0.3]&s[0.7]=>p
.
-- SAMPLE 2 --
b[10]
c[20]
n[0.3]=>p
q[1.4]&i[0.2]=>n
r[0.5]&n[0.2]=>b
s[1.3]&n[0.7]&j[0.2]=>p
p[0.9]=>c
.

Sample Output
------ ------

-- SAMPLE 1 --
q[1.20]
s[2.00]
p[4.00]&s[2.00]=>r[10.00]
q[1.20]&s[2.80]=>p[4.00]
.
-- SAMPLE 2 --
n[2.00]
r[5.00]
n[5.40]=>p[18.00]
r[5.00]&n[2.00]=>b[10.00]
p[18.00]=>c[20.00]
.
```

File: abduction.txt
Author: Bob Walton <walton@seas.harvard.edu>
Date: Mon Oct 10 21:44:43 EDT 2011

RCS Info (may not be true date or author):

```
    $Author: walton $
    $Date: 2011/10/11 01:45:32 $
    $RCSfile: abduction.txt,v $
    $Revision: 1.11 $
```