Algorithmique effective – DM 6
Ce n'est plus de la géométrie !

1 Repas de desserts

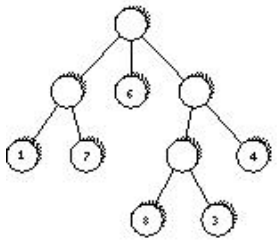
Les deux exercices suivants sont à faire en DM pour le mardi 21 mars 2006.

1.1 Une impressionante stratégie

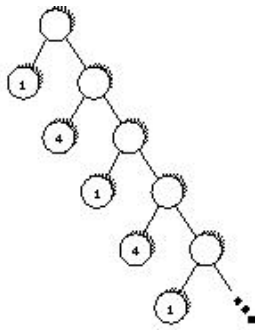
Exercice 1 Single-Player Games. UVa 664.
Playing games is the most fun if other people take part. But other players are not always available if you need them, which led to the invention of single-player games. One of the most well-known examples is the infamous "Solitaire" packaged with Zindozs, probably responsible for more wasted hours in offices around the world than any other game.

The goal of a single-player game is usually to make "moves" until one reaches a final state of the game, which results in a win or loss, or a score assigned to that final state. Most players try to optimize the result of the game by employing good strategies. In this problem we are interested in what happens if one plays randomly. After all, these games are mostly used to waste time, and playing randomly achieves this goal as well as any other strategy.

Games can very compactly represented as (possibly infinite) trees. Every node of the tree represents a possible game state. The root of the tree corresponds to the starting position of the game. For an inner node, its children are the game states to which one can move in a single move. The leaf nodes are the final states, and every one of them is assigned a number, which is the score one receives when ending up at that leaf.




```
a = ((1 7) 6 ((8 3) 4))
```




```
a = (1 b)
b = (4 a)
```

Trees are defined using the following grammar.

$$
\begin{array}{rcl}
\mathit{Definition} & ::= & \mathit{Identifier}\text{“}=\text{”}\mathit{RealTree} \\
\mathit{RealTree} & ::= & \text{“(”}\mathit{Tree}^{+}\text{“)”} \\
\mathit{Tree} & ::= & \mathit{Identifier} \mid \mathit{Integer} \mid \text{“(”}\mathit{Tree}^{+}\text{“)”} \\
\mathit{Identifier} & ::= & \mathtt{a} \mid \mathtt{b} \mid \ldots \mid \mathtt{z} \\
\mathit{Integer} & \in & \{\ldots, -3, -2, -1, 0, 1, 2, 3, \ldots, \}
\end{array}
$$

By using a Definition, the RealTree on the right-hand side of the equation is assigned to the Identifier on the left. A RealTree consists of a root node and one or more children, given as a sequence enclosed in brackets. And a Tree is either

- the tree represented by a given Identifier, or
- a leaf node, represented by a single Integer, or
- an inner node, represented by a sequence of one or more Trees (its children), enclosed in brackets.

Your goal is to compute the expected score, if one plays randomly, i.e. at each inner node selects one of the children uniformly at random. This expected score is well-defined even for the infinite trees definable in our framework as long as the probability that the game ends (playing randomly) is 1.

Input

The input file contains several gametree descriptions. Each description starts with a line containing the number n of identifiers used in the description. The identifiers used will be the first n lowercase letters of the alphabet. The following n lines contain the definitions of these identifiers (in the order `a`, `b`, ...). Each definition may contain arbitrary whitespaces (but of course there will be no spaces within a single integer). The right hand side of a definition will contain only identifiers from the first n lowercase letters. The input ends with a test case starting with $n = 0$. This test case should not be processed.

Output

For each gametree description in the input, first output the number of the game. Then, for all n identifiers in the order `a`, `b`, ..., output the following. If an identifier represents a gametree for which the probability of finishing the game is 1, print the expected score (when playing randomly). This value should be exact to three digits to the right of the decimal point.

If the game described by the variable does not end with probability 1, print `Expected score for id undefined` instead. Output a blank line after each test case.

Sample Input

```
1
a = ((1 7) 6 ((8 3) 4))
2
a = (1 b)
b = (4 a)
1
a = (a a a)
0
```

Sample Output

```
Game 1
Expected score for a = 4.917

Game 2
Expected score for a = 2.000
Expected score for b = 3.000

Game 3
Expected score for a undefined
```

1.2 Développement durable

Exercice 2 Turn the Lights Off. UVa 10309.
Since we all rely on mother earth, it's our duty to save her. Therefore, you're now asked to save energy by switching lights off.

A friend of yours has the following problem in his job. There's a grid of size 10×10, where each square has a light bulb and a light switch attached to it. Unfortunately, these lights don't work as they are supposed to. Whenever a switch is pressed, not only its own bulb is switched, but also the ones left, right, above and under it. Of course if a bulb is on the edge of the grid, there are fewer bulbs switched.

When a light switches it means it's now on if it was off before and it's now off if it was on before. Look at the following examples, which show only a small part of the whole grid. They show what happens if the middle switch is pressed. `O` stands for a light that's on, `#` stands for a light that's off.

```
###       #O#
###  ->   OOO
###       #O#

###       #O#
OOO  ->   ###
###       #O#
```

Your friend loves to save energy and asks you to write a program that finds out if it is possible to turn all the lights off and if possible then how many times he has to press switches in order to turn all the lights off.

Input
There are several test cases in the input. Each test case is preceded by a single word that gives a name for the test case. After that name there follow 10 lines, each of which contains a 10-letter string consisting of `#` and `O`. The end of the input is reached when the name string is `end`.

Output
For every test case, print one line that consists of the test case name, a single space character and the minimum number of times your friend has to press a switch. If it is not possible to switch off all the lights or requires more than 100 presses then the case name should be followed by a space and then a -1.

Sample Input

```
all_off
##########
##########
##########
##########
##########
##########
##########
##########
##########
##########
```

```
all_on
OOOOOOOOOO
OOOOOOOOOO
OOOOOOOOOO
OOOOOOOOOO
OOOOOOOOOO
OOOOOOOOOO
OOOOOOOOOO
OOOOOOOOOO
OOOOOOOOOO
OOOOOOOOOO
simple
#O########
OOO#######
#O########
####OO####
###O##O###
####OO####
##########
########O#
#######OOO
########O#
end
```

Sample Output

```
all_off 0
all_on 44
simple 4
```