

Final Assignment

Each group is to submit one solution to eConestoga.

Your group will be expected to present your solution to the class during the live class of the last week.

Your mark on this assignment will depend on a combination of the quality, functionality, and adhesion to coding standards of your code. The novelty and/or elegance of your solution will also be taken into consideration.

If you are absent without excuse from this class will result in a mark of zero for the presentation portion of the mark (20%)

I may ask anyone in the group to explain any portion of the code. The clarity of your response WILL affect your personal grade on this assignment.

Group 1 Problem - Time and motion



Movement has long been used to measure time. For example, the ball clock is a simple device which keeps track of the passing minutes by moving ball-bearings. Each minute, a rotating arm removes a ball bearing from the queue at the bottom, raises it to the top of the clock and deposits it on a track leading to indicators displaying minutes, five-minutes and hours. These indicators display the time between 1:00 and 12:59, but without 'a.m.' or 'p.m.' indicators. Thus 2 balls in the minute indicator, 6 balls in the five-minute indicator and 5 balls in the hour indicator displays the time 5:32.

Unfortunately, most commercially available ball clocks do not incorporate a date indication, although this would be simple to do with the addition of further carry and indicator tracks. However, all is not lost! As the balls migrate through the mechanism of the clock, they change their relative ordering in a predictable way. Careful study of these orderings will therefore yield the time elapsed since the clock had some specific ordering. The length of time which can be measured is limited because the orderings of the balls eventually begin to repeat. Your program must compute the time before repetition, which varies according to the total number of balls present.

Operation of the Ball Clock

Every minute, the least recently used ball is removed from the queue of balls at the bottom of the clock, elevated, then deposited on the minute indicator track, which is able to hold four balls. When a fifth ball rolls on to the minute indicator track, its weight causes the track to tilt. The four balls already on the track run back down to join the queue of balls waiting at the bottom in reverse order of their original addition to the minutes track. The fifth ball, which caused the tilt, rolls on down to the five-minute indicator track. This track holds eleven balls. The twelfth ball carried over from the minutes causes the five-minute track to tilt, returning the eleven balls to the queue, again in reverse order of their addition. The twelfth ball rolls down to the hour indicator. The hour indicator also holds eleven balls, but has one extra fixed ball which is always present so that counting the balls in the hour indicator will yield an hour in the range one to twelve. The twelfth ball carried over from the five-minute indicator causes the hour indicator to tilt, returning the eleven free balls to the queue, in reverse order, before the twelfth ball itself also returns to the queue.

Input

The input defines a succession of ball clocks. Each clock operates as described above. The clocks differ only in the number of balls present in the queue at one o'clock when all the clocks start. This number is given for each clock, one per line and does not include the fixed ball on the hour indicator. Valid numbers are in the range 27 to 127. A zero signifies the end of input.

Output For each clock described in the input, your program should report the number of balls given in the input and the number of days (24-hour periods) which elapse before the clock returns to its initial ordering.

Sample Input

```
30
45
0
```

Output for the Sample Input

```
30 balls cycle after 15 days.
45 balls cycle after 378 days.
```

Group 2 Problem - Comma Sprinkler

As practice will tell you, the English rules for comma placement are complex, frustrating, and often ambiguous. Many people, even the English, will, in practice, ignore them, and, apply custom rules, or, no rules, at all.

Doctor Comma Sprinkler solved this issue by developing a set of rules that sprinkles commas in a sentence with no ambiguity and little simplicity. In this problem you will help Dr. Sprinkler by producing an algorithm to automatically apply her rules.

Dr. Sprinkler's rules for adding commas to an existing piece of text are as follows:

1. If a word anywhere in the text is preceded by a comma, find all occurrences of that word in the text, and put a comma before each of those occurrences, except in the case where such an occurrence is the first word of a sentence or already preceded by a comma.
2. If a word anywhere in the text is succeeded by a comma, find all occurrences of that word in the text, and put a comma after each of those occurrences, except in the case where such an occurrence is the last word of a sentence or already succeeded by a comma.
3. Apply rules 1 and 2 repeatedly until no new commas can be added using either of them.

As an example, consider the text

please sit spot. sit spot, sit. spot here now here.

Because there is a comma after spot in the second sentence, a comma should be added after spot in the third sentence as well (but not the first sentence, since it is the last word of that sentence). Also, because there is a comma before the word sit in the second sentence, one should be added before that word in the first sentence (but no comma is added before the word sit beginning the second sentence because it is the first word of that sentence). Finally, notice that once a comma is added after spot in the third sentence, there exists a comma before the first occurrence of the word here. Therefore, a comma is also added before the other occurrence of the word here. There are no more commas to be added so the final result is:

please, sit spot. sit spot, sit. spot, here now, here.

Input

The input contains one line of text, containing at least 2 characters and at most 1000000 characters. Each character is either a lowercase letter, a comma, a period, or a space. We define a word to be a maximal sequence of letters within the text. The text adheres to the following constraints:

- The text begins with a word.
- Between every two words in the text, there is either a single space, a comma followed by a space, or a period followed by a space (denoting the end of a sentence and the beginning of a new one).
- The last word of the text is followed by a period with no trailing space.

Output

Display the result after applying Dr. Sprinkler's algorithm to the original text.

Sample Input 1

please sit spot. sit spot, sit. spot here now here.

Sample Output 1

please, sit spot. sit spot, sit. spot, here now, here.

Sample Input 2

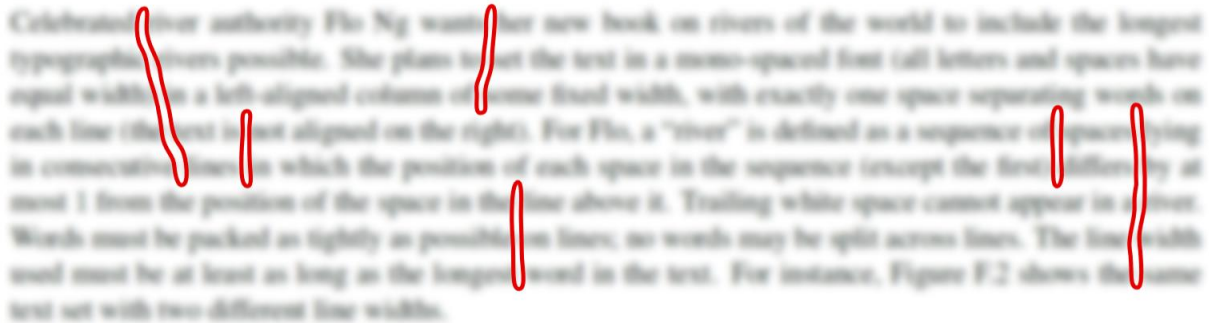
one, two. one tree. four tree. four four. five four. six five.

Sample Output 2

one, two. one, tree. four, tree. four, four. five, four. six five.

Group 3 Problem - Go with the Flow

In typesetting, a “river” is a string of spaces formed by gaps between words that extends down several lines of text. For instance, Figure F.1 shows several examples of rivers highlighted in red (text is intentionally blurred to make the rivers more visible).



Celebrated river authority Flo Ng wants her new book on rivers of the world to include the longest typographic rivers possible. She plans to set the text in a mono-spaced font (all letters and spaces have equal width) in a left-aligned column of some fixed width, with exactly one space separating words on each line (the text is not aligned on the right). For Flo, a “river” is defined as a sequence of spaces lying in consecutive lines in which the position of each space in the sequence (except the first) differs by at most 1 from the position of the space in the line above it. Trailing white space cannot appear in a river. Words must be packed as tightly as possible on lines; no words may be split across lines. The line width used must be at least as long as the longest word in the text. For instance, Figure F.2 shows the same text set with two different line widths.

Figure F.1: Examples of rivers in typeset text.

Celebrated river authority Flo Ng wants her new book on rivers of the world to include the longest typographic rivers possible. She plans to set the text in a mono-spaced font (all letters and spaces have equal width) in a left-aligned column of some fixed width, with exactly one space separating words on each line (the text is not aligned on the right). For Flo, a “river” is defined as a sequence of spaces lying in consecutive lines in which the position of each space in the sequence (except the first) differs by at most 1 from the position of the space in the line above it. Trailing white space cannot appear in a river. Words must be packed as tightly as possible on lines; no words may be split across lines. The line width used must be at least as long as the longest word in the text. For instance, Figure F.2 shows the same text set with two different line widths.

Line width 14 / River of length 4

```
The Yangtze is |
the third      |
longest river  |
in*Asia and   |
the*longest in|
the*world to  |
flow*entirely |
in one country|
```

Line width 15 / River of length 5

```
The Yangtze is |
the third      |
longest*river  |
in Asia*and the|
longest*in the |
world to*flow  |
entirely*in one|
country        |
```

Figure F.2: Longest rivers (*) for two different line widths.

Given a text, you have been tasked with determining the line width that produces the longest river of spaces for that text.

Input

The first line of input contains an integer n ($2 \leq n \leq 2500$) specifying the number of words in the text. The following lines of input contain the words of text. Each word consists only of lowercase and uppercase letters, and words on the same line are separated by a single space. No word exceeds 80 characters.

Output

Display the line width for which the input text contains the longest possible river, followed by the length of the longest river. If more than one line width yields this maximum, display the shortest such line width.

Sample Input 1

21 The Yangtze is the third longest river in Asia and the longest in the world to flow entirely in one country

Sample Output 1

15 5

Sample Input 2

25 When two or more rivers meet at a confluence other than the sea the resulting merged river takes the name of one of those rivers

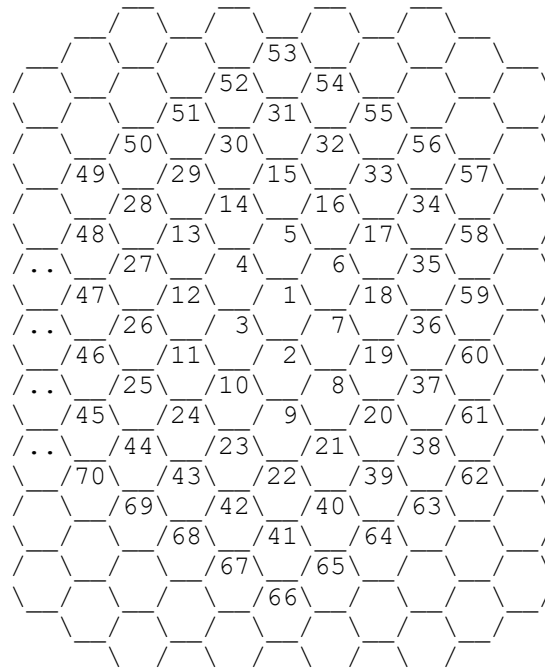
Sample Output 2

21 6

Group 4 Problem - Bee Breeding

Professor B. Heif is conducting experiments with a species of South American bees that he found during an expedition to the Brazilian rain forest. The honey produced by these bees is of superior quality compared to the honey from European and North American honey bees. Unfortunately, the bees do not breed well in captivity. Professor Heif thinks the reason is that the placement of the different maggots (for workers, queens, etc.) within the honeycomb depends on environmental conditions, which are different in his laboratory and the rain forest.

As a first step to validate his theory, Professor Heif wants to quantify the difference in maggot placement. For this he measures the distance between the cells of the comb into which the maggots are placed. To this end, the professor has labeled the cells by marking an arbitrary cell as number 1, and then labeling the remaining cells in a clockwise fashion, as shown in the following figure.



For example, two maggots in cells 19 and 30 are 5 cells apart. One of the shortest paths connecting the two cells is via the cells 19 - 7 - 6 - 5 - 15 - 30, so you must move five times to adjacent cells to get from 19 to 30.

Professor Heif needs your help to write a program that computes the distance, defined as the number of cells in a shortest path, between any pair of cells.

Input

The input consists of several lines, each containing two integers a and b ($a, b \leq 10000$), denoting numbers of cells. The integers are always positive, except in the last line where $a=b=0$ holds. This last line terminates the input and should not be processed.

Output

For each pair of numbers (a, b) in the input file, output the distance between the cells labeled a and b . The distance is the minimum number of moves to get from a to b .

Sample Input

```
19 30
0 0
```

Output for the Sample Input

```
The distance between cells 19 and 30 is 5.
```

Group 5 Problem - System Dependencies

Components of computer systems often have dependencies--other components that must be installed before they will function properly. These dependencies are frequently shared by multiple components. For example, both the TELNET client program and the FTP client program require that the TCP/IP networking software be installed before they can operate. If you install TCP/IP and the TELNET client program, and later decide to add the FTP client program, you do not need to reinstall TCP/IP.

For some components it would not be a problem if the components on which they depended were reinstalled; it would just waste some resources. But for others, like TCP/IP, some component configuration may be destroyed if the component was reinstalled.

It is useful to be able to remove components that are no longer needed. When this is done, components that only support the removed component may also be removed, freeing up disk space, memory, and other resources. But a supporting component, not explicitly installed, may be removed only if all components which depend on it are also removed. For example, removing the FTP client program and TCP/IP would mean the TELNET client program, which was not removed, would no longer operate. Likewise, removing TCP/IP by itself would cause the failure of both the TELNET and the FTP client programs. Also if we installed TCP/IP to support our own development, then installed the TELNET client (which depends on TCP/IP) and then still later removed the TELNET client, we would not want TCP/IP to be removed.

We want a program to automate the process of adding and removing components. To do this we will maintain a record of installed components and component dependencies. A component can be installed explicitly in response to a command (unless it is already installed), or implicitly if it is needed for some other component being installed. Likewise, a component, not explicitly installed, can be explicitly removed in response to a command (if it is not needed to support other components) or implicitly removed if it is no longer needed to support another component.

Input

The input will contain a sequence of commands (as described below), each on a separate line containing no more than eighty characters. Item names are case sensitive, and each is no longer than ten characters. The command names (**DEPEND**, **INSTALL**, **REMOVE** and **LIST**) always appear in uppercase starting in column one, and item names are separated from the command name and each other by one or more spaces. All appropriate **DEPEND** commands will appear before the occurrence of any **INSTALL** dependencies. The end of the input is marked by a line containing only the word **END**.

<i>Command Syntax</i>	<i>Interpretation/Response</i>
DEPEND item1 item2 [item3 ...]	item1 depends on item2 (and item3 ...)
INSTALL item1	install item1 and those on which it depends
REMOVE item1	remove item1 , and those on which it depends, if possible
LIST	list the names of all currently-installed components

Output

Echo each line of input. Follow each echoed **INSTALL** or **REMOVE** line with the actions taken in response, making certain that the actions are given in the proper order. Also identify exceptional conditions (see *Expected Output*, below, for examples of all cases). For the **LIST** command, display the names of the currently installed components. No output, except the echo, is produced for a **DEPEND** command or the line containing **END**. There will be at most one dependency list per item.

Sample Input

```
DEPEND    TELNET TCPIP NETCARD
DEPEND TCPIP NETCARD
DEPEND DNS TCPIP NETCARD
DEPEND BROWSER    TCPIP HTML
INSTALL NETCARD
INSTALL TELNET
INSTALL foo
REMOVE NETCARD
INSTALL BROWSER
INSTALL DNS
LIST
REMOVE TELNET
REMOVE NETCARD
REMOVE DNS
REMOVE NETCARD
INSTALL NETCARD
REMOVE TCPIP
REMOVE BROWSER
REMOVE TCPIP
LIST
END
```

Output for the Sample Output

```
DEPEND    TELNET TCPIP NETCARD
DEPEND TCPIP NETCARD
DEPEND DNS TCPIP NETCARD
DEPEND BROWSER    TCPIP HTML
INSTALL NETCARD
    Installing NETCARD
INSTALL TELNET
    Installing TCPIP
```

```
    Installing TELNET
INSTALL foo
    Installing foo
REMOVE NETCARD
    NETCARD is still needed.
INSTALL BROWSER
    Installing HTML
    Installing BROWSER
INSTALL DNS
    Installing DNS
LIST
    HTML
    BROWSER
    DNS
    NETCARD
    foo
    TCPIP
    TELNET
REMOVE TELNET
    Removing TELNET
REMOVE NETCARD
    NETCARD is still needed.
REMOVE DNS
    Removing DNS
REMOVE NETCARD
    NETCARD is still needed.
INSTALL NETCARD
    NETCARD is already installed.
REMOVE TCPIP
    TCPIP is still needed.
REMOVE BROWSER
    Removing BROWSER
    Removing HTML
    Removing TCPIP
REMOVE TCPIP
    TCPIP is not installed.
LIST
    NETCARD
    foo
END
```

Group 6 Problem – Firetruck

The Center City fire department collaborates with the transportation department to maintain maps of the city which reflects the current status of the city streets. On any given day, several streets are closed for repairs or construction. Firefighters need to be able to select routes from the firestations to fires that do not use closed streets.

Central City is divided into non-overlapping fire districts, each containing a single firestation. When a fire is reported, a central dispatcher alerts the firestation of the district where the fire is located and gives a list of possible routes from the firestation to the fire. You must write a program that the central dispatcher can use to generate routes from the district firestations to the fires.

Input and Output

The city has a separate map for each fire district. Streetcorners of each map are identified by positive integers less than 21, with the firestation always on corner #1. The input file contains several test cases representing different fires in different districts. The first line of a test case consists of a single integer which is the number of the streetcorner closest to the fire. The next several lines consist of pairs of positive integers separated by blanks which are the adjacent streetcorners of open streets. (For example, if the pair 4 7 is on a line in the file, then the street between streetcorners 4 and 7 is open. There are no other streetcorners between 4 and 7 on that section of the street.) The final line of each test case consists of a pair of 0's.

For each test case, your output must identify the case by number (case #1, case #2, etc). It must list each route on a separate line, with the streetcorners written in the order in which they appear on the route. And it must give the total number routes from firestation to the fire. *Include only routes which do not pass through any streetcorner more than once.* (For obvious reasons, the fire department doesn't want its trucks driving around in circles.) Output from separate cases must appear on separate lines. The following sample input and corresponding correct output represents two test cases.

Sample Input

```
6
1 2
1 3
3 4
3 5
4 6
5 6
2 3
2 4
0 0
4
2 3
3 4
5 1
```

```
1 6
7 8
8 9
2 5
5 7
3 1
1 8
4 6
6 9
0 0
```

Sample Output

CASE 1:

```
1 2 3 4 6
1 2 3 5 6
1 2 4 3 5 6
1 2 4 6
1 3 2 4 6
1 3 4 6
1 3 5 6
```

There are 7 routes from the firestation to streetcorner 6.

CASE 2:

```
1 3 2 5 7 8 9 6 4
1 3 4
1 5 2 3 4
1 5 7 8 9 6 4
1 6 4
1 6 9 8 7 5 2 3 4
1 8 7 5 2 3 4
1 8 9 6 4
```

There are 8 routes from the firestation to streetcorner 4.