UW Madison's

2005 ACM-ICPC Team Test #1

October 23, 1:00-6:00pm, 1350 CS

Overview:

This test consists of eight problems, which will be referred to by the following names (respective of order):

bridges, bulbs, bus, solitaire, reactions, pages, parties, lollies

Please note that some problems also include a problem number (or letter). Please ignore these labelings and use the names above.

As this is a team competition, you are allowed one computer for each team. Also, you may only use 25 pages of printed documentation in addition to the online STL and Java documentation.

Input/Output:

Your programs should take input from standard in (i.e. the keyboard) and output to standard out (i.e. the terminal). As is standard practice for the ICPC, you may assume that the input strictly follows the description in the problems. If a problem description does not indicate how the input will be terminated, you may assume it will be with the EOF character. It is your responsibility to ensure that your output **precisely** matches that described in the problems, or else risk your program being rejected with a "Presentation Error".

Problem Submission:

To submit a program, send e-mail to mwa+icpc@cs.wisc.edu and attach your source code. The subject line should contain only the problem name, **prob,** that you are submitting, and the attached source code should be named **prob.**{c|cpp|java}. For example, if you are using C++ and submitting the problem "bridges", the file should be named "bridges.cpp". You will receive the results of your submission as soon as possible via e-mail. You may submit from any e-mail account of a team member, but please try to be consistent and use only one throughout the contest.

Clarifications:

As in the ICPC, you may submit clarification requests as well. They should be sent to mwa+icpc@cs.wisc.edu, with a subject of "Clarification-**prob**", where **prob** is the name of the problem you wish to be clarified. Replace **prob** with "general" if there is an issue with the contest as a whole. Accepted clarification requests will be answered to all those taking the test via e-mail. Experience of previous years learns that most clarification requests are rejected, and receive a simple response such as "Read the problem description".

Printing:

You may print to the printer at any time during the test.

GOOD LUCK!

The 2003 27th Annual acm International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem A

Building Bridges

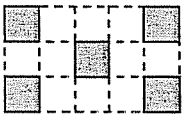
[redacted]

The City Council of New Altonville plans to build a system of bridges connecting all of its downtown buildings together so people can walk from one building to another without going outside. You must write a program to help determine an optimal bridge configuration.

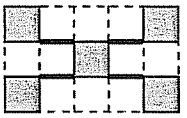
New Altonville is laid out as a grid of squares. Each building occupies a connected set of one or more squares. Two occupied squares whose corners touch are considered to be a single building and do not need a bridge. Bridges may be built only on the grid lines that form the edges of the squares. Each bridge must be built in a straight line and must connect exactly two buildings.

For a given set of buildings, you must find the minimum number of bridges needed to connect all the buildings. If this is impossible, find a solution that minimizes the number of disconnected groups of buildings. Among possible solutions with the same number of bridges, choose the one that minimizes the sum of the lengths of the bridges, measured in multiples of the grid size. Two bridges may cross, but in this case they are considered to be on separate levels and do not provide a connection from one bridge to the other.

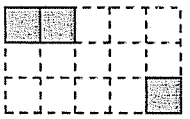
The figure below illustrates four possible city configurations. City 1 consists of five buildings that can be connected by four bridges with a total length of 4. In City 2, no bridges are possible, since no buildings share a common grid line. In City 3, no bridges are needed because there is only one building. In City 4, the best solution uses a single bridge of length 1 to connect two buildings, leaving two disconnected groups (one containing two buildings and one containing a single building).



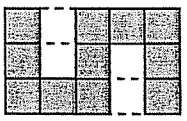
City 1



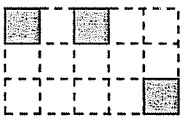
City 1
with bridges



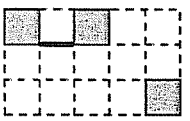
City 2
No bridges are possible



City 3
No bridges are needed



City 4



City 4
with bridges

Input

The input data set describes several rectangular cities. Each city description begins with a line containing two integers r and c, representing the size of the city on the north-south and east-west axes measured in grid lengths ($1 \le r \le 50$ and $1 \le c \le 50$). These numbers are followed by exactly r lines, each consisting of c hash ("#") and dot (".") characters. Each character corresponds to one square of the grid. A hash character corresponds to a square that is occupied by a building, and a dot character corresponds to a square that is not occupied by a building.

The input data for the last city will be followed by a line containing two zeros.

Output

For each city description, print two or three lines of output as shown below. The first line consists of the city number. If the city has fewer than two buildings, the second line is the sentence "No bridges are needed." If the city has two or more buildings but none of them can be connected by bridges, the second line is the sentence "No bridges are possible." Otherwise, the second line is "*N* bridges of total length *L*" where *N* is the number of bridges and *L* is the sum of the lengths of the bridges of the best solution. (If *N* is 1, use the word "bridge" rather than "bridges.") If the solution leaves two or more disconnected groups of buildings, print a third line containing the number of disconnected groups.

Print a blank line between cases. Use the output format shown in the example.

Sample Input	Output for the Sample Input
3 5 #...# ..#.. #...# 3 5 ##... ....# 3 5 #.### #.#.# ###.# 3 5 #.#.. ....# 0 0	City 1 4 bridges of total length 4 City 2 No bridges are possible. 2 disconnected groups City 3 No bridges are needed. City 4 1 bridge of total length 1 2 disconnected groups

The 2003 27th Annual acm International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem B

Light Bulbs

Hollywood's newest theater, the Atheneum of Culture and Movies, has a huge computer-operated marquee composed of thousands of light bulbs. Each row of bulbs is operated by a set of switches that are electronically controlled by a computer program. Unfortunately, the electrician installed the wrong kind of switches, and tonight is the ACM's opening night. You must write a program to make the switches perform correctly.

A row of the marquee contains n light bulbs controlled by n switches. Bulbs and switches are numbered from 1 to n, left to right. Each bulb can either be ON or OFF. Each input case will contain the initial state and the desired final state for a single row of bulbs.

The original lighting plan was to have each switch control a single bulb. However the electrician's error caused each switch to control two or three consecutive bulbs, as shown in Figure 1. The leftmost switch ($i = 1$) toggles the states of the two leftmost bulbs (1 and 2); the rightmost switch ($i = n$) toggles the states of the two rightmost bulbs ($n - 1$ and n). Each remaining switch ($1 < i < n$) toggles the states of the three bulbs with indices $i - 1$, i, and $i + 1$. (In the special case where there is a single bulb and a single switch, the switch simply toggles the state of that bulb.) Thus, if bulb 1 is ON and bulb 2 is OFF, flipping switch 1 will turn bulb 1 OFF and bulb 2 ON. The minimum cost of changing a row of bulbs from an initial configuration to a final configuration is the minimum number of switches that must be flipped to achieve the change.

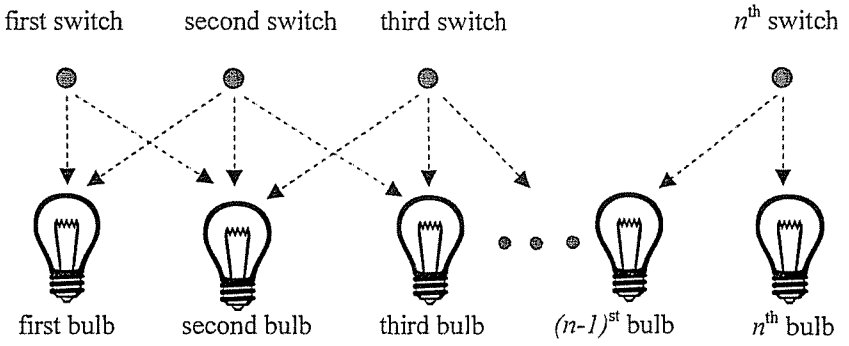


Figure 1

You can represent the state of a row of bulbs in binary, where 0 means the bulb is OFF and 1 means the bulb is ON. For instance, 01100 represents a row of five bulbs in which the second and third bulbs are both ON. You could transform this state into 10000 by flipping switches 1, 4, and 5, but it would be less costly to simply flip switch 2.

You must write a program that determines the switches that must be flipped to change a row of light bulbs from its initial state to its desired final state with minimal cost. Some combinations of initial and final states may not be feasible. For compactness of representation, decimal integers are used instead of binary for the bulb configurations. Thus, 01100 and 10000 are represented by the decimal integers 12 and 16.

Input

The input file contains several test cases. Each test case consists of one line. The line contains two non-negative decimal integers, at least one of which is positive and each of which contains at most 100 digits. The first integer represents the initial state of the row of bulbs and the second integer represents the final state of the row. The binary equivalent of these integers represents the initial and final states of the bulbs, where 1 means ON and 0 means OFF.

To avoid problems with leading zeros, assume that the first bulb in either the initial or the final configuration (or both) is ON. There are no leading or trailing blanks in the input lines, no leading zeros in the two decimal integers, and the initial and final states are separated by a single blank.

The last test case is followed by a line containing two zeros.

Output

For each test case, print a line containing the case number and a decimal integer representing a minimum-cost set of switches that need to be flipped to convert the row of bulbs from initial state to final state. In the binary equivalent of this integer, the rightmost (least significant) bit represents the n^{th} switch, 1 indicates that a switch has been flipped, and 0 indicates that the switch has not been flipped. If there is no solution, print "impossible". If there is more than one solution, print the one with the smallest decimal equivalent.

Print a blank line between cases. Use the output format shown in the example.

Sample Input	Output for the Sample Input
12 16 1 1 3 0 30 5 7038312 7427958190 4253404109 657546225 0 0	Case Number 1: 8 Case Number 2: 0 Case Number 3: 1 Case Number 4: 10 Case Number 5: 2805591535 Case Number 6: impossible

The 2003 27th Annual acm International Collegiate

Programming Contest World Finals

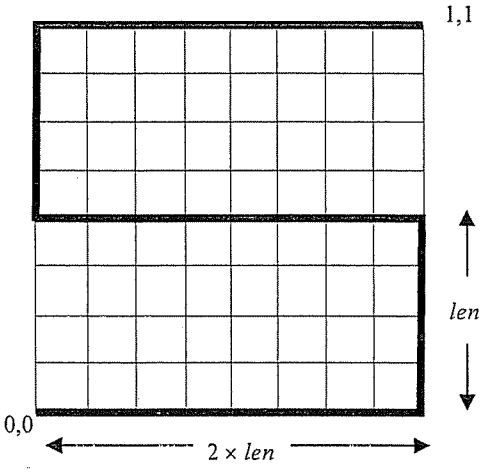
sponsored by IBM

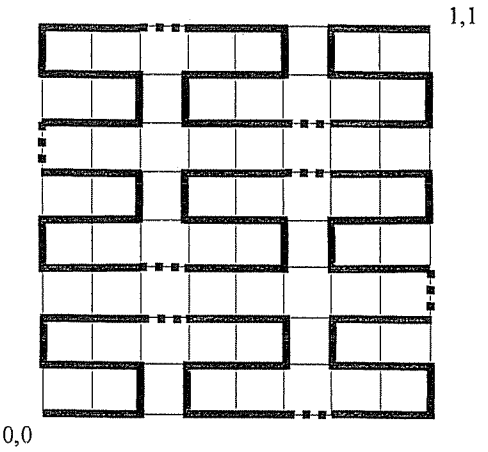
Problem C

Riding the Bus

The latest research in reconfigurable multiprocessor chips focuses on the use of a single bus that winds around the chip. Processor components, which can be anywhere on the chip, are attached to *connecting points* on the bus so that they can communicate with each other.

Some research involves bus layout that uses recursively-defined "SZ" curves, also known as "S-shaped Peano curves." Two examples of these curves are shown below. Each curve is drawn on the unit square. The order-1 curve, shown on the left, approximates the letter "S" and consists of line segments connecting the points (0,0), (1,0), (1,0.5), (0,0.5), (0,1), and (1,1) in order. Each horizontal line in an "S" or "Z" curve is twice as long as each vertical line. For the order-1 curve, the length of a vertical line, *len*, is 0.5.





The order-2 curve, shown on the right, contains 9 smaller copies of the order-1 curve (4 of which are reversed left to right to yield "Z" curves). These copies are connected by line segments of length *len*, shown as dotted lines. Since the width and height of the order-2 curve is 8 × *len*, and the curve is drawn on the unit square, *len* = 0.125 for the order-2 curve.

The order-3 curve contains 9 smaller copies of the order-2 curve (with 4 reversed left to right), connected by line segments, as described for the order-2 curve. Higher order curves are drawn in a similar manner. The *connecting points* to which processor components attach are evenly spaced every *len* units along the bus. The first connecting point is at (0,0) and the last is at (1,1). There are 9^k connecting points along the order-k curve, and the total bus length is $(9^k-1) \times len$ units.

You must write a program to determine the total distance that signals must travel between two processor components. Each component's coordinates are given as an x, y pair, $0 \le x \le 1$ and $0 \le y \le 1$, where x is the distance from the left side of the chip, and y is the distance from the lower edge of the chip. Each component is attached to the closest connecting point by a straight line. If multiple connecting points are equidistant from a component, the one with the smallest x coordinate and smallest y coordinate is used. The total distance a signal must travel between two components is the sum of the length of the lines connecting the components to the bus, and the length of the bus between the two connecting points. For example, the distance between components located at (0.5, 0.25) and (1.0, 0.875) on a chip using the order-1 curve is 3.8750 units.

Input

The input contains several cases. For each case, the input consists of an integer that gives the order of the SZ curve used as the bus (no larger than 8), and then four real numbers x_1, y_1, x_2, y_2 that give the coordinates of the processor components to be connected. While each processor component should actually be in a unique location not on the bus, your program must correctly handle all possible locations.

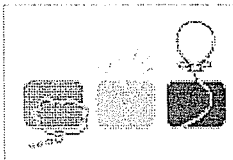
The last case in the input is followed by a single zero.

Output

For each case, display the case number (starting with 1 for the first case) and the distance between the processor components when they are connected as described. Display the distance with 4 digits to the right of the decimal point.

Use the same format as that shown in the sample output shown below. Leave a blank line between the output lines for consecutive cases.

Sample Input	Output for the Sample Input
1 0.5 .25 1 .875 1 0 0 1 1 2 .3 .3 .7 .7 2 0 0 1 1 0	Case 1. Distance is 3.8750 Case 2. Distance is 4.0000 Case 3. Distance is 8.1414 Case 4. Distance is 10.0000



3155 – Two–Stacks Solitaire

Latin America – South America – 2004/2005

PDF | PostScript | Submit | | Ranking

Card games for one player are called Patience in Britain and Solitaire in the United States. One very difficult (some say maddening) solitaire game is called Two–Stacks and has the following rules:

Tableau

The tableau consists of a stock pile, two intermediate piles and one foundation pile.

Cards

A game may use up to four complete decks, or parts of those four decks. A complete deck contains 52 cards; since all cards in a deck can be ordered, in this description we will forget about faces and suits and we will use the numbers from 1 to 52 to represent the cards in a deck.

Dealing

Once the cards to be used in the game have been chosen, they are dealt face up in the tableau, one on top of the other, forming the stock pile.

Moving cards

Cards may be moved one at a time. A card can be moved from the stock pile to one of the intermediate piles, or from one intermediate pile to the foundation pile. In a pile (stock or intermediate) only the topmost card can be moved, although all the others are visible.

Object

The goal is to have all cards used in the game in non--decreasing order, from bottom to top, constituting the foundation pile.

As you may have noticed by now, even when a solution exists, chances of winning a Two–Stacks solitaire game are very low. But your grandmother has just learned the game and loved it. She has asked you to help her to learn playing the game, by writing a program that would coach her through her first tries, showing which movements to make.

Input

The input consists of several test cases. The first line of a test case contains a single integer N ($1 \leq N \leq 208$), representing the number of cards in the game. The second line of a test case contains a sequence of N integers (between 1 and 52), separated by single blank spaces, representing the cards. The cards will be dealt in the order they appear in the input, so that the topmost card in the stock pile is the Nth card in the line. Notice that each number from 1 to 52 will appear at most 4 times in each test case. The end of input is indicated by a test case with $N = 0$.

Output

For each test case in the input your program must produce an answer. The first line of an answer must contain a test case identifier, in the form '#i' where i starts from 1 and is incremented for every test case. Then, if it is possible to win the game, print a sequence of movements to win the game. Each movement must be described in a separate line, in the form 'push x' or 'pop x' where 'x' is 1 or 2; 'push x' moves the topmost card from the stock pile to intermediate pile x, and 'pop x' moves the topmost card from intermediate pile x to the foundation pile. If more than one solution exists, print any one. If it is not possible to win the game, print a line with the word 'impossible'.

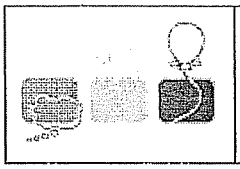
Sample Input

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

Sample Output

```
#1
push 1
push 2
push 1
pop 1
pop 1
push 1
pop 2
pop 1
#2
impossible
#3
push 1
push 2
push 2
push 2
pop 1
pop 2
pop 2
pop 2
```

South America 2004–2005



2454 – Chemical reactions

Europe – Northeastern – 2001/2002

Bill teaches chemistry in the school and has prepared a number of tests for his students. Each test has a chemical formula and a number of possible reaction outcomes that his students are to choose one correct from. However, Bill wants to make sure that he has not made any typos while entering his tests into a computer and that his students won't easily throw away wrong answers simply by counting a number of chemical elements on the left and on the right side of the equation, which should be always equal in a valid reaction.

You are to write a program that will help Bill. The program shall read the description of the test for the students that consists of the given left side of the equation and a number of possible right sides, and determines if the number of chemical elements on each right side of the equation is equal to the number of chemical elements on the given left side of the equation.

To help you, poor computer folks, that are unaware of the complex world of chemistry, Bill has formalized your task. Each side of the equation is represented by a string of characters without spaces, and consists of one or more chemical sequences separated by a '+' (plus) characters. Each sequence has an optional preceding integer multiplier that applies to the whole sequence and a number of elements. Each element is optionally followed by an integer multiplier that applies to it. An element in this equation can be either distinct chemical element or a whole sequence that is placed in round parenthesis. Every distinct chemical element is represented by either one capital letter or a capital letter that is followed by a small letter.

Even more formally, using notation that is similar to BNF, we can write:

- <formula> ::= [<number>] <sequence> { '+' [<number>] <sequence> }
- <sequence> ::= <element> [<number>] { <element> [<number>] }
- <element> ::= <chem> | '(' <sequence> ')'
- <chem> ::= <uppercase_letter> [<lowercase_letter>]
- <uppercase_letter> ::= 'A'..'Z'
- <lowercase_letter> ::= 'a'..'z'
- <number> ::= '1'..'9' { '0'..'9' }

Every distinct chemical element is said to occur in the given formula for some total number X, if X is the sum of all separate occurrences of this chemical element multiplied to all numbers that apply to it. For example, in the following chemical formula:

```
C2H5OH+3O2+3(SiO2)
```

- `C` occurs for a total of 2 times.
- `H` occurs for a total of 6 times (5 + 1).
- `O` occurs for a total of 13 times (1 + 3*2 + 3*2).
- `Si` occurs for a total of 3 times.

All multipliers in the formula are integer numbers that are at least 2 if explicitly specified and are 1 by default. Each chemical formula is at most 100 characters long, and every distinct chemical element is guaranteed to occur for a total of no more than 10000 times in each formula.

Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

The first line of the input file represents a chemical formula that is to be tested as the left side of the equation. The second line of the input file contains a single integer number N (1 ≤ N ≤ 10), which is the number of right sides of the equation that are to be tested. Each one of the following N lines represents one such formula.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

You are to write to the output file N lines – one line per each possible answer of the chemical test for Bill's students that is given in the input file. For each right–hand side formula that is encountered in the input file, write to the output file:

```
<left_formula>==<right_formula>
```

if the total number of occurrences of each distinct chemical element on the left–hand side equals to the total number of occurrences of this chemical element on the right–hand side. Otherwise write:

```
<left_formula>!=<right_formula>
```

Here `<left_formula>` must be replaced exactly (character by character) with the original left–hand side formula as it is given in the first line of the input file, and `<right_formula>` must be replaced exactly with each right–hand side formula as they are given in the input file. Do not place any spaces in the lines you write to the output file.

Sample Input

```
1

C2H5OH+3O2+3(SiO2)
7
2CO2+3H2O+3SiO2
2C+6H+13O+3Si
99C2H5OH+3SiO2
3SiO4+C2H5OH
C2H5OH+3O2+3(SiO2)+Ge
3(Si(O)2)+2CO+3H2O+O2
2CO+3H2O+3O2+3Si
```

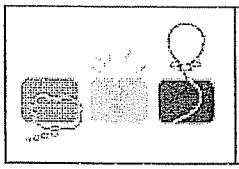
Sample Output

```
C2H5OH+3O2+3(SiO2)==2CO2+3H2O+3SiO2
C2H5OH+3O2+3(SiO2)==2C+6H+13O+3Si
C2H5OH+3O2+3(SiO2)!=99C2H5OH+3SiO2
C2H5OH+3O2+3(SiO2)==3SiO4+C2H5OH
C2H5OH+3O2+3(SiO2)!=C2H5OH+3O2+3(SiO2)+Ge
C2H5OH+3O2+3(SiO2)==3(Si(O)2)+2CO+3H2O+O2
C2H5OH+3O2+3(SiO2)!=2CO+3H2O+3O2+3Si
```

Note

The sample input and output do not contain digit '0' because it looks the same as the symbol for the chemical element oxygen. The actual tests may contain any allowed characters.

Northeastern 2001–2002



2484 – Book Pages

Oceania – South Pacific – 2002/2003

For the purposes of this problem, we will assume that every page in an Acmonian book is numbered sequentially, and that the first page is numbered 1.

How many digits would you need to use to number the pages of a 10 page book? Pages 1 to 9 would require 1 digit each (total 9), and page 10 would require 2 digits. This makes 11 digits. Similarly, a book of 34 pages would require 59 digits.

Can we work backwards? If you are told that a book requires 13 digits to number its pages, can you work out how many pages the book has? I hope so, because that is all you have to do for this problem. Each line in the input file represents the number of digits used in numbering a book. Your answer will be the number of pages the book has. If the number supplied cannot possibly be valid, your answer should be `Impossible!'. Beware that Acmonian books can be quite large, and the number of digits required for a given Acmonian book can reach 2000000000.

Input

Each line in the input file contains a single integer, between 1 and 2000000000, representing a number of digits used in numbering the pages of a book. A single # on a line indicates the end of input.

Output

Output for each input number must be on a single line. If the input value is valid, output the number of pages in the book. Otherwise, output `Impossible!'.

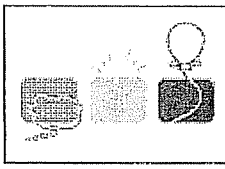
Sample Input

```
11
13
59
60
1999999998
#
```

Sample Output

```
10
11
34
Impossible!
234567900
```

South Pacific 2002–2003



2486 – Company Parties

Oceania – South Pacific – 2002/2003

The Acme company has an hierarchical organization, i.e., a tree–like structure with the *CEO* at the root and each other employee a child node of his/her manager. In addition to his/her position in the organization, each employee has a unique employee id (a string with no particular meaning) and a sociability measure (an integer number).

The *CEO* of the Acme company wants to organize a party for their employees. To make the party agreeable the *CEO* wants to make invitations such that:

- the *CEO* attends the party,
- an employee can be invited only if his/her direct manager is absent,
- the sum of the sociability measures of all who attend is a maximum.

Write a program that will determine the maximum sociability sum under these conditions for a given company structure.

Input

The input text consists of a number of company structures. The first line of a set is a title giving the company name. The company name may contain any printable non–space characters; and embedded spaces are also permitted. A single `#' on a line indicates the end of input.

The *name* line is followed by one line consisting of a single integer n, $1 \leq n \leq 100000$, that indicates the number of employees in this company. This line is followed by n further lines, one line for each employee.

Each *employee* line consists of three items separated by single spaces:

- an *employeeid*, which is a sequence of 1 to 10 letters and/or digits,
- a *sociabilitymeasure*, which is an integer number between 0 and 100,
- a *managerid*, which is the employee id of the current employee's direct manager, or the character `–' for the *CEO*.

The *employeeid* is a unique identifier within the company. The order in which the employees appear is arbitrary, i.e., not related to their employee or manager ids.

Output

There is a single output line for each company. Each output line consists of the company title, followed by a colon and a space, and finally the maximum attainable sociability measure under the above conditions.

Sample Input

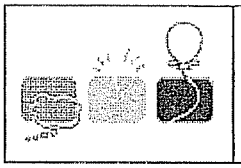
```
ACME 1
1
ID0 10 -
ACME II
2
```

```
ID0 10 -
ID1 21 ID0
ACME, INC.
8
ID4 20 ID3
ID5 1 ID4
ID6 1 ID3
ID7 10 ID6
ID0 10 -
ID1 21 ID0
ID2 10 ID1
ID3 11 ID0
#
```

Sample Output

```
ACME 1: 10
ACME II: 10
ACME, INC.: 50
```

South Pacific 2002–2003



2487 – Lollies

Oceania – South Pacific – 2002/2003

Every day on his way home, little Billy passes by his great aunt Clara Mitchum's house. Generally he stops in for a chat with the great ACM (as he lovingly refers to her) and sometimes he asks for some lollies. When he does, she generally gives him some, but then adds *now don't be asking for any more for another N days* where N is some positive integer. If $N = 1$ that means he can ask for some on the next day, but for example if it is April 6 and $N = 4$ then he must wait until April 10 or later before asking for more lollies.

One day Billy happened to catch sight of the great ACM's calendar, and noted that each day was marked with two integers. He also noted that the first of these referred to the number of lollies the great ACM would give him on a particular day, and the second to the delay that would then be required before making another request. He copied down as much of the information as he could, and has passed it to you to analyse. His objective, of course, is to get as many lollies as he can.

Your task is to write a program which will report the total number of lollies that can be obtained by Billy, and provide a schedule for obtaining that amount. In the event that there are two or more ways to obtain the maximum number of lollies, Billy will choose the one where his first collection is as late as possible, and among all collections with that first date, his second collection is as late as possible, and so on.

Input

The input text consists of a number of sets of unrelated problems. The first line of a set is a problem title consisting of a string of 1 to 20 letters. A single `#' on a line indicates the end of input.

The *title* line is followed by a sequence of *day* lines. Each problem set contains between 1 and 100 days, including the limits. In the given order, the first *day* line corresponds to day number 1, the second line to day number 2, the n–th line to day number n. Each *day* line consists of two integers separated by a single space:

- an integer L, which is the number of lollies available on that day ($1 \leq L \leq 100$),
- an integer N, which is the associated delay ($1 \leq N \leq 100$).

Conventionally, a delay N pointing to a day beyond the end of the current problem refers to a day with zero lollies and zero further delays ($L = 0$, $N = 0$).

Output

Each report must follow the following format (use single spaces for spacing):

```
In problem title total amount lollies can be obtained:
On day day number collect day amount lollies.
On day day number collect day amount lollies.
...
```

In this notation, *problem_title* represents the actual problem title, *total_amount*, *day_amount*, and *day_number* are numbers with self–described meaning, and *lollies* stands for either `lolly' or `lollies', as required by the context (the singular and plural forms must be used appropriately). Days must be given in increasing sequence numbers. Each group report should be separated from the next by a blank line.

Sample Input

```
January
1 1
2 2
3 3
February
10 3
7 1
5 2
1 1
March
2 3
1 1
3 7
2 7
#
```

Sample Output

```
In January 4 lollies can be obtained:
On day 1 collect 1 lolly.
On day 3 collect 3 lollies.

In February 12 lollies can be obtained:
On day 2 collect 7 lollies.
On day 3 collect 5 lollies.

In March 4 lollies can be obtained:
On day 2 collect 1 lolly.
On day 3 collect 3 lollies.
```
