Problem 4: The Fence Builder

A fence builder has been given a strange task. Provided with *N* (between 3 and 100) pieces of straight fencing, each having an arbitrary length, the builder is to enclose as large a region as possible. The customer wants to know the area of the region that can be enclosed by the fence before it is built. There is only one constraint on the construction: each piece of fencing is connected only at its endpoints to exactly two other different pieces of fencing. That is, after completion, the fence will look like a (possibly irregular) polygon with *N* sides. The customer has guaranteed the builder that the fencing provided will allow for a region with a non-zero area to be enclosed.

Input

There will be multiple cases in the input. For each case, the input begins with the number of pieces of fencing (an integer, *N*). There then follow *N* positive, non-zero real numbers giving the lengths of the fence pieces. A single integer zero follows the last case in the input.

Output

For each case, display the case number (starting with 1) and the maximum area that can be enclosed by the provided fencing materials. Show three fractional digits in each answer. Use the format shown below in displaying the results.

Sample Input

```
3 2.0 2.0 2.0
4 1.0 1.0 1.0 1.0
4 5.0 5.0 3.0 11.0
0
```

Expected Output

```
Case 1: maximum area = 1.732
Case 2: maximum area = 1.000
Case 3: maximum area = 21.000
```

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 ($1 \le L \le 100$),
- an integer N, which is the associated delay ($1 \le N \le 100$).

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.
```

South Pacific 2002-2003

Problem D
Conduit Packing
Input File: conduit.in

Allied Conduit Manufacturing (ACM) makes metal conduit tubes with round cross-sections that enclose many different types of wires. The circular cross-section of a wire can have a diameter up to 20 millimeters (20000 micrometers). ACM needs a program to compute the minimum diameter of a conduit that can hold 4 wires with specified diameters.

Figure 4 shows examples of fitting four wires of different sizes into conduits of minimum diameters.

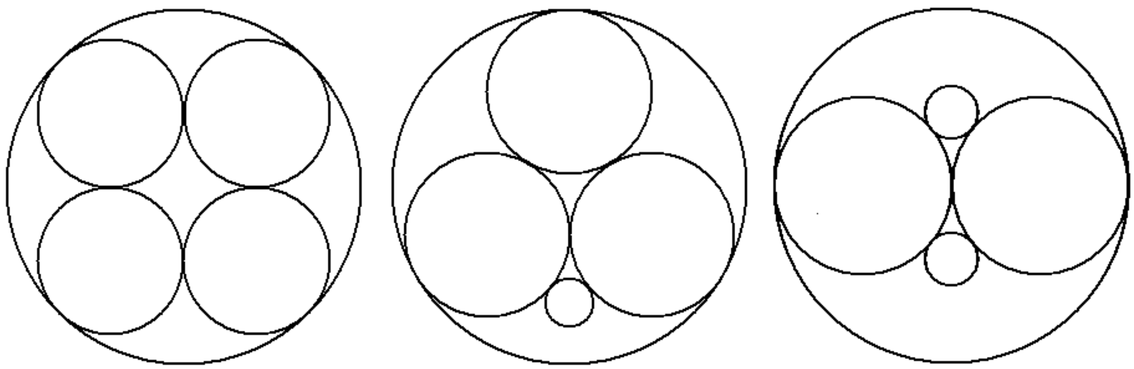


Figure 4: Fitting wires inside conduits

Your program must take the diameters of wires and determine the minimum inside diameter of the conduit that can hold the wires.

Input

The input file contains several test cases. Each test case consists of a line with four integers, d_1, d_2, d_3, and d_4, which are the diameters of the wires in micrometers. The integers satisfy $20000 \geq d_1 \geq d_2 \geq d_3 \geq d_4 > 0$. The last test case is followed by a line containing a single integer zero.

Output

For each test case, print the number of the test case (starting with 1) followed by the minimum conduit diameter in micrometers, rounded to the nearest integer. Follow the format of the sample output.

Sample Input	Output for the Sample Input

`10000 10000 10000 10000`
`10000 10000 10000 3000`
`12000 12000 3600 3600`
`0`
	`Case 1: 24142`
	`Case 2: 21547`
	`Case 3: 24000`

2 Fibonacci (`fib.{c,cc,java}`)

2.1 Description

In the Fibonacci integer sequence, $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 2$. For example, the first ten terms of the Fibonacci sequence are:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

An alternative formula[1] for the Fibonacci sequence is

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \underbrace{\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \cdots \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}}_{n \text{ times}}.$$

Given an integer n, your goal is to compute the last 4 digits of F_n.

2.2 Input

The input test file will contain multiple test cases. Each test case consists of a single line containing n (where $0 \leq n \leq$ 1,000,000,000). The end-of-file is denoted by a single line containing the number -1.

```
0
9
999999999
1000000000
-1
```

2.3 Output

For each test case, print the last four digits of F_n. If the last four digits of F_n are all zeros, print '0'; otherwise, omit any leading zeros (i.e., print F_n mod 10000).

```
0
34
626
6875
```

[1] As a reminder, matrix multiplication is associative, and the product of two 2×2 matrices is given by

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}.$$

Also, note that raising any 2×2 matrix to the 0th power gives the identity matrix:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}^0 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Problem A: Gopher II

The gopher family, having averted the canine threat, must face a new predator.

The are *n* gophers and *m* gopher holes, each at distinct (x, y) coordinates. A hawk arrives and if a gopher does not reach a hole in *s* seconds it is vulnerable to being eaten. A hole can save at most one gopher. All the gophers run at the same velocity *v*. The gopher family needs an escape strategy that minimizes the number of vulnerable gophers.

The input contains several cases. The first line of each case contains four positive integers less than 100: *n*, *m*, *s*, and *v*. The next *n* lines give the coordinates of the gophers; the following *m* lines give the coordinates of the gopher holes. All distances are in metres; all times are in seconds; all velocities are in metres per second.

Output consists of a single line for each case, giving the number of vulnerable gophers.

Sample Input

```
2 2 5 10
1.0 1.0
2.0 2.0
100.0 100.0
20.0 20.0
```

Output for Sample Input

```
1
```

2591 - The Tree Movers

North America - North Central - 2002/2003

Given two binary search trees, A and B, with nodes identified by (that is, having keys equal to) positive, non-zero integers, and the use of commands ``delete $K$ " and ``add K " (defined below), what is the smallest number of commands that can be used to transform tree A into tree B?

Recall that in a binary search tree, the keys of all nodes in the left subtree of a node with key K must be less than K . Similarly, the keys of all nodes in the right subtree of a node with key K must be greater than K . There are no duplicate nodes.

The ``delete $K$ " command will delete the tree (or subtree) with its root at the node with the key $K$ . Deleting the root of the entire tree leaves an empty tree. The ``add K " command will add a new node identified by the integer K . This node will naturally be a leaf node.

Since we seek to transform tree A into tree B, it follows that commands will be applied only to tree A; tree B is ``read only".

It is easy to see that it should never require more than $N + 1$ commands to achieve the transformation of A into B, since deletion of the root node of tree A followed by the addition of one node for each of the N nodes in B (in the proper order) will achieve the desired goal. Equally easy to determine is the minimum number of commands required: if A and B are identical, then zero commands are required.

Input

There will be multiple input cases. For each case, the input contains the description of tree A followed by the description of tree B. Each tree description consists of an integer N that specifies the number of nodes in the tree, following by the keys of the N nodes in an order such that N ``add" commands would create the tree. The last case is followed by the integer `-1'. No node will have a key larger than 10^9 , and N will be no larger than 100.

Output

For each case, display a single line containing the input case number (1, 2,...) and the number of commands required to transform tree A into tree B, formatted as shown in the examples below.

Sample Input

```
4 5 2 7 4 6 5 3 7 1 4 9
0 0
1 100 0
0 1 100
3 100 49 37 2 200 152
-1
```

Sample Output

```
Case 1: 5 commands.
Case 2: 0 commands.
Case 3: 1 command.
```

```
Case 4: 1 command.
Case 5: 3 commands.
```

North Central 2002-2003

	2997 - Intersecting Dates World Finals - Prague - 2003/2004

A research group is developing a computer program that will fetch historical stock market quotes from a service that charges a fixed fee for each day's quotes that it delivers. The group has examined the collection of previously-requested quotes and discovered a lot of duplication, resulting in wasted money. So the new program will maintain a list of all past quotes requested by members of the group. When additional quotes are required, only quotes for those dates not previously obtained will be fetched from the service, thus minimizing the cost.

You are to write a program that d etermines when new quotes are required. Input for the program consists of the date ranges for which quotes have been requested in the past and the date ranges for which quotes are required. The program will then determine the date ranges for which quotes must be fetched from the service.

Input

There will be multiple input cases. The input for each case begins with two non-negative integers *NX* and *NR*, ($0 \le NX, NR \le 100$). *NX* is the number of existing date ranges for quotes requested in the past. *NR* is the number of date ranges in the incoming requests for quotes. Following these are *NX* + *NR* pairs of dates. The first date in each pair will be less than or equal to the second date in the pair. The first *NX* pairs specify the date ranges of quotes which have been requested and obtained in the past, and the next *NR* pairs specify the date ranges for which quotes are required.

Two zeroes will follow the input data for the last case.

Each input date will be given in the form *YYYYMMDD*. *YYYY* is the year (1700 to 2100), *MM* is the month (01 to 12), and *DD* is the day (in the allowed range for the given month and year). Recall that months 04, 06, 09, and 11 have 30 days, months 01, 03, 05, 07, 08, 10, and 12 have 31 days, and month 02 has 28 days except in leap years, when it has 29 days. A year is a leap year if it is evenly divisible by 4 and is not a century year (a multiple of 100), or if it is divisible by 400.

Output

For each input case, display the case number (1, 2, ...) followed by a list of any date ranges for which quotes must be fetched from the service, one date range per output line. Use the American date format shown in the sample output below. Explicitly indicate (as shown) if no additional quotes must be fetched. If two date ranges are contiguous or overlap, then merge them into a single date range. If a date range consists of a single date, print it as a single date, not as a range consisting of two identical dates. Display the date ranges in chronological order, starting with the earliest date range.

Sample Input

```
1 1
19900101 19901231
19901201 20000131
0 3
19720101 19720131
19720201 19720228
19720301 19720301
1 1
```

```
20010101 20011231
20010515 20010901
0 0
```

Sample Output

```
Case 1:
    1/1/1991 to 1/31/2000

Case 2:
    1/1/1972 to 2/28/1972
    3/1/1972

Case 3:
    No additional quotes are required.
```

Prague 2003-2004

Tests-Setter: Rujia Liu
Special Thanks: Yao Guan

108 Maximum Sum

Background

A problem that is simple to solve in one dimension is often much more difficult to solve in more than one dimension. Consider satisfying a boolean expression in conjunctive normal form in which each conjunct consists of exactly 3 disjuncts. This problem (3-SAT) is NP-complete. The problem 2-SAT is solved quite efficiently, however. In contrast, some problems belong to the same complexity class regardless of the dimensionality of the problem.

The Problem

Given a 2-dimensional array of positive and negative integers, find the sub-rectangle with the largest sum. The sum of a rectangle is the sum of all the elements in that rectangle. In this problem the sub-rectangle with the largest sum is referred to as the *maximal sub-rectangle*. A sub-rectangle is any contiguous sub-array of size 1×1 or greater located within the whole array. As an example, the maximal sub-rectangle of the array:

$$\begin{array}{rrrr} 0 & -2 & -7 & 0 \\ 9 & 2 & -6 & 2 \\ -4 & 1 & -4 & 1 \\ -1 & 8 & 0 & -2 \end{array}$$

is in the lower-left-hand corner:

$$\begin{array}{rr} 9 & 2 \\ -4 & 1 \\ -1 & 8 \end{array}$$

and has the sum of 15.

Input and Output

The input consists of an $N \times N$ array of integers. The input begins with a single positive integer N on a line by itself indicating the size of the square two dimensional array. This is followed by N^2 integers separated by white-space (newlines and spaces). These N^2 integers make up the array in row-major order (i.e., all numbers on the first row, left-to-right, then all numbers on the second row, left-to-right, etc.). N may be as large as 100. The numbers in the array will be in the range $[-127, 127]$.

The output is the sum of the maximal sub-rectangle.

Sample Input

```
4
0 -2 -7  0 9  2 -6  2
-4  1 -4  1 -1
8  0 -2
```

Sample Output

```
15
```

2721 - Building Bridges

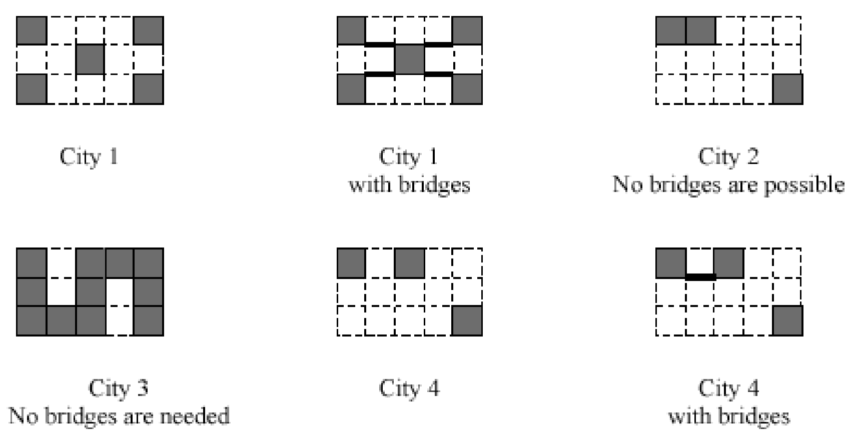
World Finals - Beverly Hills - 2002/2003

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).



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 ($1 \le r \le 100$ and $1 \le c \le 100$). 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

```
3 5
#...#
..#..
#...#
3 5
##...
.....
....#
3 5
#.###
#.#.#
###.#
3 5
#.#..
.....
....#
0 0
```

Sample Output

```
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
```

Beverly Hills 2002-2003