The 2003 27th Annual **acm** International Collegiate

Programming Contest World Finals

sponsored by IBM

Problem B

Light Bulbs

Input File: bulbs.in

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.

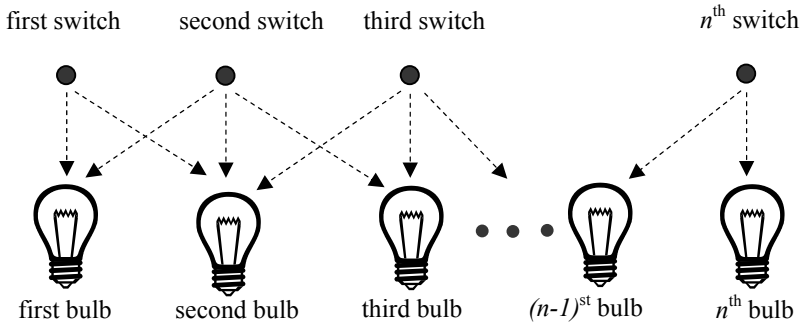


Figure 1

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

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
```

Kingdom of Magic

ngdom of Magic has had a network of bidirectional magic portals between cities since ancient times. Each portal magically connects a pair of cities and allows fast magical communication and travel between them. Cities that are connected by a magic portal are called *neighboring*.

Prince Albert and Princess Betty are living in the neighboring cities. Since their childhood Albert and Betty were always in touch with each other using magic communication Orbs, which work via a magic portal between the cities.

Albert and Betty are in love with each other. Their love is so great that they cannot live a minute without each other. They always carry the Orbs with them, so that they can talk to each other at any time. There is something strange about their love — they have never seen each other and they even fear to be in the same city at the same time. People say that the magic of the Orbs have affected them.

Traveling through the Kingdom is a complicated affair for Albert and Betty. They have to travel through magic portals, which is somewhat expensive even for royal families. They can simultaneously use a pair of the portals to move to a different pair of cities, or just one of them can use a portal, while the other one stays where he or she is. At any moment of their travel they have to be in a neighboring cities. They cannot simultaneously move through the same portal.

Write a program that helps Albert and Betty travel from one pair of the cities to another pair. It has to find the cheapest travel plan — with the minimal number of times they have to move though the magic portals. *When they move through the portals simultaneously it counts as two moves.*

Input

The input consists of the description of a series of test cases, terminated by the end of the file. The first line of each test case contains integer numbers n, m, a_1, b_1, a_2, b_2. Here n ($3 \leq n \leq 100$) is a number of cities in the Kingdom (cities are numbered from 1 to n); m ($2 \leq m \leq 1000$) is a number of magic portals; a_1, b_1 ($1 \leq a_1, b_1 \leq n$, $a_1 \neq b_1$) are the neighboring cities where Albert and Betty correspondingly start their travel from; a_2, b_2 ($1 \leq a_2, b_2 \leq n$, $a_2 \neq b_2$) are the neighboring cities where Albert and Betty correspondingly want to get to ($a_1 \neq a_2$ or $b_1 \neq b_2$).

The following m lines describe the portals. Each line contains two numbers p_{i1} and p_{i2} ($1 \leq p_{i1}, p_{i2} \leq n$, $p_{i1} \neq p_{i2}$) — cities that are connected by the portal. There is at most one portal connecting two cities.

Output

Print out one line per test case, each containing one number c, the minimal number of moves in the travel plan for that test case.

Sample input and output

4 5 1 2 2 1 1 2 2 3 3 4 4 1 1 3	3

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.
```

South Pacific 2002-2003

ACM International Collegiate Programming Contest 95/96

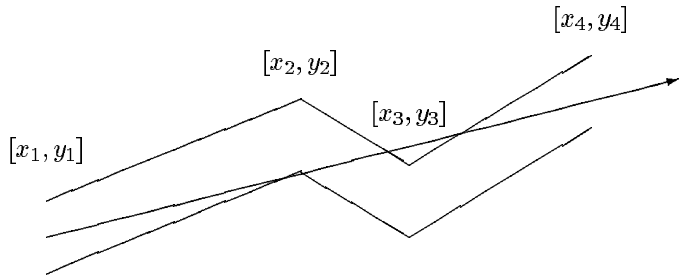
Sponsored by Microsoft

Central European Regional Contest

Problem D: Pipe

Input file: `pipe.in`
Output file: `pipe.out`
Program file: `pipe.pas` or `pipe.cpp`

The GX Light Pipeline Company started to prepare bent pipes for the new transgalactic light pipeline. During the design phase of the new pipe shape the company ran into the problem of determining how far the light can reach inside each component of the pipe. Note that the material which the pipe is made from is not transparent and not light reflecting.



Each pipe component consists of many straight pipes connected tightly together. For the programming purposes, the company developed the description of each component as a sequence of points $[x_1, y_1]$, $[x_2, y_2]$, ..., $[x_n, y_n]$, where $x_1 < x_2 < \ldots x_n$. These are the upper points of the pipe contour. The bottom points of the pipe contour consist of points with y-coordinate decreased by 1. To each upper point $[x_i, y_i]$ there is a corresponding bottom point $[x_i, y_i - 1]$ (see picture above). The company wants to find, for each pipe component, the point with maximal x-coordinate that the light will reach. The light is emitted by a segment source with endpoints $[x_1, y_1 - 1]$ and $[x_1, y_1]$ (endpoints are emitting light too). Assume that the light is not bent at the pipe bent points and the bent points do not stop the light beam.

Input

The input file contains several blocks each describing one pipe component. Each block starts with the number of bent points $2 \leq n \leq 20$ on separate line. Each of the next n lines contains a pair of real values x_i, y_i separated by space. The last block is denoted with $n = 0$.

Output

The output file contains lines corresponding to blocks in input file. To each block in the input file there is one line in the output file. Each such line contains either a real value, written with precision of two decimal places, or the message `Through all the pipe.`. The real value is the desired maximal x-coordinate of the point where the light can reach from the source for corresponding pipe component. If this value equals to x_n, then the message `Through all the pipe.` will appear in the output file.

Example

Input file

```
4
0 1
2 2
4 1
6 4
6
0 1
```

```
2 -0.6
5 -4.45
7 -5.57
12 -10.8
17 -16.55
0
```

Output file

```
4.67
Through all the pipe.
```

2589 - Twenty Questions

North America - North Central - 2002/2003

In the game of ``twenty questions'' I think of an item (like a fish) from a set of *N* items, and you get to ask me at most twenty questions that can only be answered ``yes'' or ``no'' to identify the item. For example, you might ask ``Is it living?'' If I answer ``yes'', then you might ask, ``Does it have fur?'' If I answer ``no'', then you might then ask, ``Does it have fins?'' This continues until you either guess the item (in which case you win), or youve asked twenty questions without identifying the item (in which case I win).

With just 20 questions you could identify any one of 524,288 items, assuming you can distinguish among them by asking 19 ``yes/no'' questions and then with your $20^{th}$ question ask, ``Is it X?'' Of course it might take fewer than 20 questions if you have a good idea about the identity of the item, but in this problem well assume all the questions are used.

Suppose you could ask questions that could be answered with more than just a ``yes'' or ``no''. For example, suppose you could ask, ``Does it weight less than, equal to, or greater than 10 pounds?'' This question has three possible answers. Then how many questions would you need to ask in order to distinguish among *N* items?

In this problem you will be told how many different answers can be given to each of your questions, and the number of items in the set of possible choices for the item. All you need to do is determine the maximum number of questions that must be asked to identify the item. This assumes that your questions are chosen in such a way as to divide the remaining candidates for the item into suitably sized groups.

Input

There will be multiple cases to consider. For each case there will be two integers in the input. The first integer, *K* , is the number of possible answers to each question (no larger than 10). The second integer, *N* , is the number of items in the set of possible choices (no larger than 2,147,483,647). A pair of zeroes will follow the last case.

Output

For each input case, display a single line that looks like this:

N `items,` *K* `answers per question,` *M* `questions`

where *N* and *K* are the values from the input, and *M* is the maximum number of questions required.

Sample Input

```
2 524288
3 524288
4 524288
3 9
10 1000
0 0
```

Sample Output

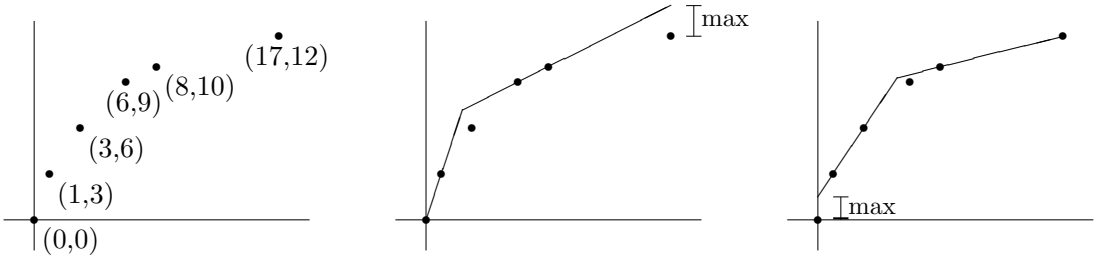
```
524288 items, 2 answers per question, 20 questions
524288 items, 3 answers per question, 13 questions
524288 items, 4 answers per question, 11 questions
9 items, 3 answers per question, 3 questions
1000 items, 10 answers per question, 4 questions
```

North Central 2002-2003

Problem F: Roofing It

Bill Eaves owns the *Shingle Minded* roofing company which is a sub-contracting firm specializing in putting roofs on buildings. Often, Bill will receive a design for a roof from some young hot-shot architect which, though it may have some aesthetic appeal, is totally impractical. In these cases, Bill will begin negotiations with the architect and the client to find a simpler design. Bill's only concern is that the roof be convex, to allow rain, snow and frisbees to roll off easily. The architect's main concern is that the maximum height between his original roof and the compromise roof be as small as possible. The client's main concern is to get out of this meeting as soon as possible and get back to watching TV.

The architect's plans for the roof come in the form of a series of n points (x_i, y_i) specifying the outline of the roof as seen from the side of the house. The roofs are always symmetrical, so the architect only shows the front side of the roof (from its start at the front of the house to its peak). Once Bill gets these plans and a decision is made on how many sections the convex roof should have, he must decide how to place the sections so as to 1) make sure that all the original (x_i, y_i) points lie on or below the new roof, and 2) to minimize the maximum vertical distance between any of the original (x_i, y_i) points and the new roof. All sections must lie on at least two of the initial points specified by the architect. An example is shown below. On the left are the initial points from the architect. The next two pictures show an approximation of the roof using only two sections. While both of these are convex, the second of the two is the one which minimizes the maximum distance.



Input

Input will consist of multiple test cases. Each case will begin with two positive integers n and k ($2 \le n \le 100, 1 \le k < n$) indicating the number of points used in the original roof plan and the number of sections used in the compromise roof. These will be followed by n lines each containing two floating points numbers x_i y_i, specifying the n points in the roof plan. These values will be given in increasing order of x, and the last point will be guaranteed to have the highest y value of any of the points. All values will be between 0.0 and 10000.0. The last case is followed by a line containing 0 0 which indicates end-of-input and should not be processed.

Output

For each test case, output the maximum distance between the best compromise roof and the initial points, rounded to the nearest thousandth.

Sample Input

```
6 2
0.0 0.0
1.0 3.0
3.0 6.0
6.0 9.0
8.0 10.0
17.0 12.0
0 0
```

Sample Output

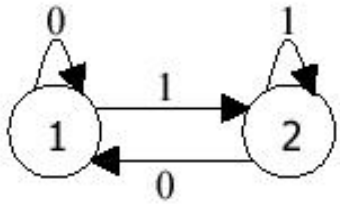
```
1.500
```

2596 - Acceptable Strings

North America - North Central - 2002/2003

A deterministic finite automaton has a finite set of states, with directed edges leading from one state to another. Each edge is labeled with a symbol. In this problem, we are only concerned about automata (the plural of automaton) that use the binary digits 0 and 1 as symbols. Each edge is thus labeled with 0 or 1. One state is identified as the start state, and one or more states are identified as final states.

A finite automaton is usually represented by a graph. For example, consider the finite automaton represented by the graph shown below; the states are shown as circles, and are named 1 and 2 for ease of identification. In this automaton, state 1 is the start state, and state 2 is the final state.



Each automaton in this problem accepts or rejects a string as follows. Beginning in the start state, for each symbol (0 or 1) in the input string (working from left to right in sequence), the automaton follows the one edge labeled with the input symbol from the current state to the next state. After making the transition associated with the last symbol in the input string, if the automaton is in a final state, then the input is accepted. Otherwise (that is, if the automaton is not in a final state), the input is rejected.

For the string 0101 and the automaton shown above, we start in state 1 (the start state). Since the first input symbol is 0, the edge labeled 0 from state 1 back to state 1 is followed, leaving us in state 1. The next input symbol, 1, causes a transition to state 2. The next symbol, 0, moves us back to state 1. The last input symbol, 1, causes the last transition, from state 1 to state 2. Since state 2 is a final state, the automaton accepts the string 0101. Note that the string 010 would have been rejected, since the automaton would have been in state 1 (which is not a final state) at the end of the input. This automaton happens to accept all binary strings that end with 1.

In this problem you will be given one or more automata and an integer N . For each of these, you are to find the number of binary strings having each length less than or equal to N that are accepted by the automaton. For example, for $N = 3$ with the automaton above, the output would specify 0 strings of length 0 (since state 1 is not a final state), 1 string of length 1 (1), 2 strings of length 2 (01 and 11), and 4 strings of length 3 (001, 011, 101, and 111).

Input

There will be multiple input cases. For each case the input begins with three integers N , S , and F . N (no larger than 10) specifies the maximum length of the strings that are sought. S (no larger than 100) specifies the number of states in the automaton. F (no larger than 10) specifies the number of final states. Following these three integers are S pairs of integers. Each pair specifies the labels on the edges from the states, in order, starting with state 1. The first integer in each pair specifies the state to which the edge labeled 0 connects; the second integer specifies the state to which the edge labeled 1 connects. Finally, the last F integers identify the final states. State 1 will always be the start state. The input for the last case is followed by three zeroes.

Output

For each case, display the case number (they are numbered sequentially, and start with 1). Then display the number of strings of each length (from 0 to N) accepted by the automaton, using a separate line for each length. The output must be identical in format to that shown in the examples below.

Sample Input

```
3 2 1
1 1
1 1
2

3 2 1
1 2
1 2
2

10 7 1
2 2
3 3
4 4
5 5
6 6
7 7
7 7
6

0 0 0
```

Sample Output

```
Case 1:
    Length = 0, 0 strings accepted.
    Length = 1, 0 strings accepted.
    Length = 2, 0 strings accepted.
    Length = 3, 0 strings accepted.
Case 2:
    Length = 0, 0 strings accepted.
    Length = 1, 1 string accepted.
    Length = 2, 2 strings accepted.
    Length = 3, 4 strings accepted.
Case 3:
    Length = 0, 0 strings accepted.
    Length = 1, 0 strings accepted.
    Length = 2, 0 strings accepted.
    Length = 3, 0 strings accepted.
    Length = 4, 0 strings accepted.
    Length = 5, 32 strings accepted.
    Length = 6, 0 strings accepted.
    Length = 7, 0 strings accepted.
    Length = 8, 0 strings accepted.
    Length = 9, 0 strings accepted.
    Length = 10, 0 strings accepted.
```

North Central 2002-2003

ACM International Collegiate Programming Contest 95/96

Sponsored by Microsoft

Central European Regional Contest

Problem C: John's trip

Input file: `trip.in`
Output file: `trip.out`
Program file: `trip.pas` or `trip.cpp`

Little Johnny has got a new car. He decided to drive around the town to visit his friends. Johnny wanted to visit all his friends, but there was many of them. In each street he had one friend. He started thinking how to make his trip as short as possible. Very soon he realized that the best way to do it was to travel through each street of town only once. Naturally, he wanted to finish his trip at the same place he started, at his parents' house.

The streets in Johnny's town were named by integer numbers from 1 to n, $n < 1995$. The junctions were independently named by integer numbers from 1 to m, $m \leq 44$. No junction connects more than 44 streets. All junctions in the town had different numbers. Each street was connecting exactly two junctions. No two streets in the town had the same number. He immediately started to plan his round trip. If there was more than one such round trip, he would have chosen the one which, when written down as a sequence of street numbers is lexicographically the smallest. But Johnny was not able to find even one such round trip.

Help Johnny and write a program which finds the desired shortest round trip. If the round trip does not exist the program should write a message. Assume that Johnny lives at the junction ending the street No. 1 with smaller number. All streets in the town are two way. There exists a way from each street to another street in the town. The streets in the town are very narrow and there is no possibility to turn back the car once he is in the street.

Input

Input file consists of several blocks. Each block describes one town. Each line in the block contains three integers x, y, z, where $x > 0$ and $y > 0$ are the numbers of junctions which are connected by the street number z. The end of the block is marked by the line containing $x = y = 0$. At the end of the input file there is an empty block, $x = y = 0$.

Output

The output file consists of 2 line blocks corresponding to the blocks of the input file. The first line of each block contains the sequence of street numbers (single members of the sequence are separated by space) describing Johnny's round trip. If the round trip cannot be found the corresponding output block contains the message `Round trip does not exist`. The second line of each block is empty.

Example

input file:

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

output file:

```
1 2 3 5 4 6

Round trip does not exist
```