|  |
| --- |
| buetlogo.JPG |
| Onsite Practice Contest |
| ACM ICPC, CSE, BUET |
|  |
| **Problems: A – F (9 pages including Cover)** |
| **21-Aug-14** |



**Problem A**

**Gopher**

****

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

There 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.

**Input**

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 meters; all times are in seconds; all velocities are in meters per second.

**Output**

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 | **Sample Output**  1 |

**Problem B**

**Compressed Words**

Steve has come up with a way to compress text, though it may not actually compress the text. Steve considers only individual words, and uses the following rules to define a "compressed word":

1. a single, lower-case letter is a compressed word.
2. *(e1 e2 ... et n)* where *t* and *n* are non-negative integers and *ei* is a compressed word.

You should observe that a compressed word of one character is the same as an uncompressed word. To uncompress the compressed word *(e1 e2 ... et n)* we uncompress each *ei*, concatenate those uncompressed words into a new word, and repeatedly concatenate that word *n* times. For example:

* x would be uncompressed as x,
* (t 3) would be uncompressed as ttt,
* (a (b c 2) 3) would be uncompressed as abcbcabcbcabcbc.

Write a program to uncompress a compressed word.

**Input**

Your program will be tested on one or more test cases. Each test case is made of one correctly formed compressed word on a separate line. A $ character identifies the end of line. The last line of the input, which is not part of the test cases, contains a $ by itself (possibly with leading and/or trailing white spaces). Every compressed word in the input is correct according to the rules specified above. Note that a compressed word may contain leading, trailing, and/or embedded spaces. Such spaces should be ignored. Letters and numbers are separated from each other by at least one space character.

**Output**

For each test case (i.e., each compressed word), write the uncompressed word on a separate line. There should be no spaces (other than newlines) in the output.

|  |  |
| --- | --- |
| **Sample Input**  x$  (t 3)$  ( a ( b c 2 ) 3) $  $ | **Sample Output**  x  ttt  abcbcabcbcabcbc |

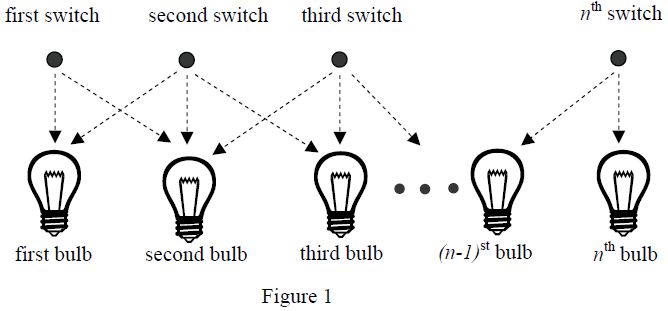
**Problem C**

**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.



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**  12 16  1 1  3 0  30 5  7038312 7427958190  4253404109 657546225  0 0 | **Sample Output**  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 D**

**Combining Images**

As the exchange of images over computer networks becomes more common, the problem of image compression takes on increasing importance. Image compression algorithms are used to represent images using a relatively small number of bits.

One image compression algorithm is based on an encoding called a “Quad Tree.” An image has a Quad Tree encoding if it is a square array of binary pixels (the value of each pixel is 0 or 1, called the “color” of the pixel), and the number of pixels on the side of the square is a power of two.

If an image is homogeneous (all its pixels are of the same color), the Quad Tree encoding of the image is 1 followed by the color of the pixels. For example, the Quad Tree encoding of an image that contains pixels of color 1 only is 11, regardless of the size of the image.

If an image is heterogeneous (it contains pixels of both colors), the Quad Tree encoding of the image is 0 followed by the Quad Tree encodings of its upper-left quadrant, its upper-right quadrant, its lower-left quadrant, and its lower-right quadrant, in order.

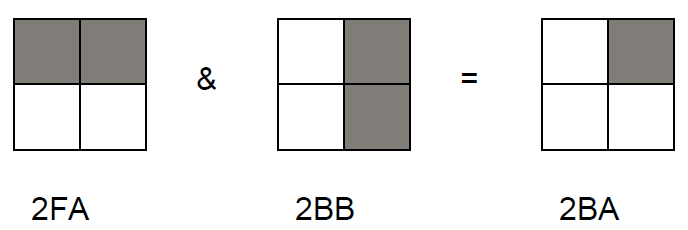
The Quad Tree encoding of an image is a string of binary digits. For easier printing, a Quad Tree encoding can be converted to a Hex Quad Tree encoding by the following steps:

1. Prepend a 1 digit as a delimiter on the left of the Quad Tree encoding.
2. Prepend 0 digits on the left as necessary until the number of digits is a multiple of four.
3. Convert each sequence of four binary digits into a hexadecimal digit, using the digits 0 to 9 and capital A through F to represent binary patterns from 0000 to 1111.

For example, the Hex Quad Tree encoding of an image that contains pixels of color 1 only is 7, which corresponds to the binary string 0111.

You must write a program that reads the Hex Quad Tree encoding of two images, computes a new image that is the intersection of those two images, and prints its Hex Quad Tree encoding. Assume that both input images are square and contain the same number of pixels (although the lengths of their encodings may differ). If two images A and B have the same size and shape, their intersection (written as A & B) also has the same size and shape. By definition, a pixel of A & B is equal to 1 if and only if the corresponding pixels of image A and image B are both equal to 1.

The following figure illustrates two input images and their intersection, together with the Hex Quad Tree encodings of each image. In the illustration, shaded squares represent pixels of color 1.



**Input**

The input data set contains a sequence of test cases, each of which is represented by two lines of input. In each test case, the first input line contains the Hex Quad Tree encoding of the first image and the second line contains the Hex Quad Tree encoding of the second image. For each input image, the number of hexadecimal digits in its Hex Quad Tree encoding will not exceed 100.The last test case is followed by two input lines, each containing a single zero.

**Output**

For each test case, print “Image” followed by its sequence number. On the next line, print the Hex Quad Tree encoding of the intersection of the two images for that test case. Separate the output for consecutive test cases with a blank line.

|  |  |
| --- | --- |
| **Sample Input**  2FA  2BB  2FB  2EF  7  2FA  0  0 | **Sample Output**  Image 1:  2BA  Image 2:  2EB  Image 3:  2FA |

**Problem E**

**Hotter Colder**

The children's game Hotter Colder is played as follows. Player A leaves the room while player B hides an object somewhere in the room. Player A re-enters at position (0,0) and then visits various other positions about the room. When player A visits a new position, player B announces "Hotter" if this position is closer to the object than the previous position; player B announces "Colder" if it is farther and "Same" if it is the same distance.

**Input**

Input consists of up to 50 lines, each containing an x, y coordinate pair followed by "Hotter", "Colder", or "Same". Each pair represents a position within the room, which may be assumed to be a square with opposite corners at (0,0) and (10,10).

**Output**

For each line of input print a line giving the total area of the region in which the object may have been placed, to 2 decimal places. If there is no such region, output 0.00.

|  |  |
| --- | --- |
| **Sample Input**  10.0 10.0 Colder  10.0 0.0 Hotter  0.0 0.0 Colder  10.0 10.0 Hotter | **Sample Output**  50.00  37.50  12.50  0.00 |

**Problem F**

**Matryoshka Dolls**

****

You most likely have seen the Russian Dolls which stack inside each other. Each doll has a different weight and storage ability. Your task is to nest as many dolls as possible.

**Input**

Input consists of several testcases, until the end of file. First line of a testcase contains an integer *n* denoting the number of dolls in the testcase. Each of next *n* lines contains a pair of integers separated by one or more space characters, specifying the weight and storage ability of a doll. The weight of the doll is in grams. The storage ability, also in grams, is the doll's overall storage capacity, including its own weight. That is, a doll weighing 400g with strength of 900g could carry 500g of dolls inside it. There are at most 6000 dolls. The maximum weight of any doll is 100000g and the maximum storage capacity is 20000000g.

**Output**

For each testcase, print the case number, followed by a single integer indicating the maximum number of dolls that can be nested without exceeding the storage ability of any one. See the sample input and output for exact formatting.

|  |  |
| --- | --- |
| **Sample Input**  4  300 1000  1000 1200  200 600  100 101 | **Sample Output**  Case 1: 3 |