

**ACM International Collegiate Programming Contest**  
**2009 East Central Regional Contest**  
**McMaster University**  
**University of Cincinnati**  
**University of Michigan – Ann Arbor**  
**Youngstown State University**  
**October 31, 2009**

**Sponsored by IBM**

Rules:

1. There are **eight** questions to be completed in **five hours**.
2. All questions require you to read the test data from standard input and write results to standard output. You cannot use files for input or output. Additional input and output specifications can be found in the General Information Sheet.
3. The allowed programming languages are C, C++ and Java.
4. All programs will be re-compiled prior to testing with the judges' data.
5. Non-standard libraries cannot be used in your solutions. The Standard Template Library (STL) and C++ string libraries are allowed. The standard Java API is available, except for those packages that are deemed dangerous by contestant officials (e.g., that might generate a security violation).
6. The input to all problems will consist of multiple test cases unless otherwise noted.
7. Programming style is not considered in this contest. You are free to code in whatever style you prefer. Documentation is not required.
8. All communication with the judges will be handled by the PC<sup>2</sup> environment.
9. Judges' decisions are to be considered final. No cheating will be tolerated.

## Problem A: Arithmetically Challenged

Challenge 24 is a popular mathematics game used in many grade schools. In each game, contestants are given a card with four positive integers  $i_1, i_2, i_3, i_4$  on it, and the first one who can use all of these numbers and any combination of the four basic arithmetic operations to get 24 wins. Each of the numbers  $i_1, i_2, i_3, i_4$  must be used exactly once. Division can be used only if the divisor evenly divides the dividend (i.e., you can perform  $6/2$  but not  $6/4$ ). For example, if the card contains the numbers 7, 2, 5 and 1, possible solutions are  $(7-2)*5-1$  or  $(7+1)*(5-2)$ . Hmmm . . . this sounds like a source of a good programming problem.

Write a program that determines the longest consecutive sequence of integers that can be obtained by different ways of arithmetically combining the four integers. For example, with 7, 2, 5 and 1 the longest consecutive sequence is -18 to 26 (yes, we're allowing final results to be negative). The “+” and “-” operators must be used as binary operators, not as unary signs.

### Input

Each test case will consist of a single line containing the four, not necessarily distinct, positive integers, none of which will exceed 100. A line containing four 0's will terminate input.

### Output

For each test case, output the case number and the longest consecutive sequence of obtainable values, in the format shown in the sample output. If there is more than one longest consecutive sequence, use the one with the largest first value.

### Sample Input

```
7 2 5 1
8 15 38 3
0 0 0 0
```

### Sample Output

```
Case 1: -18 to 26
Case 2: 150 to 153
```

## Problem B: Cover Up

“The Price is Right” is a popular game show where contestants play various games to win fabulous prizes. One of the games played on the show is called “Cover Up” whose object is to guess a 5-digit number (actually, the price of a new car). In the actual game, contestants are given two numbers to choose from for the first digit, three numbers to choose from for the second digit, and so on. A contestant selects one number for each digit and then is told which ones are correct; if at least one is correct, the player is allowed to guess again for all incorrect digits. The contestant keeps guessing as long as they keep getting at least one new digit correct. The game ends when either all the digits are correct (a win for the contestant) or after a turn when no new digit is guessed correctly (a loss).

Typically this game is not sheer luck. For example, suppose you had the following five possibilities for the last digit: 1, 3, 5, 8 and 9. Many car prices end with either a 5 or a 9, so you might have, say, a 70% chance that one of these two numbers is correct; this breaks down to a 35% chance for either the 5 or the 9 and a 10% chance for each of the other three digits. Now say you pick the 5 and it’s wrong, but some other guess you made was right so you still get to play. With this additional information the probabilities for the remaining 4 numbers change: the probability for the 9 is now close to around 54%, while each of the other three numbers now has a little over a 15% chance. (We’ll let you figure out how we got these values). We’ll call the 5 and the 9 in the original group the *likely* candidates, and typically there are likely candidates in other columns as well. For example, if the two numbers for the first (high order) digit are 1 and 9, the contestant can be 100% sure that the 1 is the correct digit (there aren’t too many \$90,000 cars to be given away).

For this problem, you are to determine the probability of winning the game if an optimal strategy for picking the numbers (based on probabilities such as those described above) is used.

### Input

Each test case will consist of two lines. The first will be  $n$ , the number of digits in the number to be guessed. The maximum value of  $n$  will be 5. The second line will contain  $n$  triplets of numbers of the form  $m\ l\ p$  where  $m$  is the number of choices for a digit,  $l$  is the number of likely candidates, and  $p$  is the probability that one of the likely candidates is correct. In all cases  $0 \leq l < m \leq 10$  and  $0.0 \leq p \leq 1.0$ . Whenever  $l = 0$  (i.e., when there are no likely candidates)  $p$  will always be 0.0. A line containing a single 0 will terminate the input.

### Output

Output for each test case is the probability of winning using optimal strategy. All probabilities should be rounded to the nearest thousandth, and trailing 0’s should not be output. (A 100% chance of winning should be output as 1.)

**Sample Input**

```
2
3 1 0.8 2 0 0.0
2
3 2 0.8 2 0 0.0
2
3 2 0.82 2 1 0.57
3
4 1 1.0 3 0 0.0 10 1 1.0
0
```

**Sample Output**

```
0.85
0.6
0.644
1
```

## Problem C: Decompressing in a GIF

One well known method to compress image files is the Graphics Interchange Format (GIF) encoding, created by CompuServe in 1987. Here's a simplified version applied to strings of alphabetic characters. Essential for this compression is a dictionary which assigns numeric encodings (we'll use base 10 numbers for this problem) to different strings of characters. The dictionary is initialized with mappings for characters or substrings which may appear in the string. For example, if we expect to encounter all 26 letters of the alphabet, the dictionary will initially store the encodings  $(A, 00)$ ,  $(B, 01)$ ,  $(C, 02)$ ,  $\dots$ ,  $(Z, 25)$ . If we are compressing DNA data, the dictionary will initially store only 4 entries:  $(A, 0)$ ,  $(T, 1)$ ,  $(G, 2)$  and  $(C, 3)$ . Note that the length of each initial encoding is the same for all entries (2 digits in the first example, and 1 digit in the second).

The compression algorithm proceeds as follows:

1. Find the longest prefix of the uncompressed portion of the string which is in the dictionary, and replace it with its numeric encoding.
2. If the end of the string has not been reached, add a new mapping  $(s, n)$  to the dictionary, where  $s$  = the prefix just compressed plus the next character after it in the string, and  $n$  = the smallest number not yet used in the dictionary.

For example, assume we started with the string **ABABBAABB** and a dictionary with just two entries,  $(A, 0)$  and  $(B, 1)$ . The table below shows the steps in compressing the string.

String	Longest Prefix	Replaced With	New Dictionary Entry
ABABBAABB	A	0	$(AB, 2)$
OBABBAABB	B	1	$(BA, 3)$
01ABBAABB	AB	2	$(ABB, 4)$
012BAABB	BA	3	$(BAA, 5)$
0123ABB	ABB	4	—

The final compressed string is **01234**.

There is only one other rule: the replacement strings used are always the size of the longest encoding in the dictionary at the time the replacement occurs. Thus, with the dictionary above, if the string to compress is long enough that an entry of the form  $(s, 10)$  is added to the dictionary, then from this point on all numerical replacement strings used in the compressed string must be expanded to 2 digits long (i.e., A will now be encoded as 00, B as 01, AB as 02, etc.); if an entry  $(s', 100)$  is added to the dictionary, all replacements from this point forward will increase to 3 digits long, and so on. Thus, the longer string **ABABBAABBAABAABAB** will be encoded as **01234027301**, not **0123402731**. Try it!

OK, now that you are experts at compressing, it's time to relax and decompress!

### Input

Each test case will consist of two lines. The first line will contain a string of digits to decompress. The second line will contain the initial dictionary used in the compression. This line will start with a positive integer  $n$  indicating the number of entries in the dictionary ( $1 \leq n \leq 100$ ), followed by  $n$  alphabetic strings. The first of these will be paired with 0 in the dictionary (or 00 if  $n > 10$ ), the second with 1, and so on. The last test case will be followed by a line containing a single 0.

## Output

For each test case, output a single line containing the case number (using the format shown below) followed by the decompressed string. All input strings will have been legally compressed.

## Sample Input

```
01234
2 A B
01234027301
2 A B
02151120182729
26 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
21104
3 BA A C
01
2 JA VA
0
```

## Sample Output

```
Case 1: ABABBAABB
Case 2: ABABBAABBAABAABAB
Case 3: CPLUSPLUS
Case 4: CAABAAA
Case 5: JAVA
```

## Problem D: Flipper

Little Bobby Roberts (son of Big Bob, of Problem G) plays this solitaire memory game called Flipper. He starts with  $n$  cards, numbered 1 through  $n$ , and lays them out in a row with the cards in order left-to-right. (Card 1 is on the far left; card  $n$  is on the far right.) Some cards are face up and some are face down. Bobby then performs  $n - 1$  flips — either right flips or left flips. In a right flip he takes the pile to the far right and flips it over onto the card to its immediate left. For example, if the rightmost pile has cards A, B, C (from top to bottom) and card D is to the immediate left, then flipping the pile over onto card D would result in a pile of 4 cards: C, B, A, D (from top to bottom). A left flip is analogous.

The very last flip performed will result in one pile of cards — some face up, some face down. For example, suppose Bobby deals out 5 cards (numbered 1 through 5) with cards 1 through 3 initially face up and cards 4 and 5 initially face down. If Bobby performs 2 right flips, then 2 left flips, the pile will be (from top to bottom) a face down 2, a face up 1, a face up 4, a face down 5, and a face up 3.

Now Bobby is very sharp and you can ask him what card is in any position and he can tell you!!! You will write a program that matches Bobby's amazing feat.

### Input

Each test case will consist of 4 lines. The first line will be a positive integer  $n$  ( $2 \leq n \leq 100$ ) which is the number of cards laid out. The second line will be a string of  $n$  characters. A character U indicates the corresponding card is dealt face up and a character D indicates the card is face down. The third line is a string of  $n - 1$  characters indicating the order of the flips Bobby performs. Each character is either R, indicating a right flip, or L, indicating a left flip. The fourth line is of the form  $m q_1 q_2 \dots q_m$ , where  $m$  is a positive integer and  $1 \leq q_i \leq n$ . Each  $q_i$  is a query on a position of a card in the pile (1 being the top card,  $n$  being the bottom card). A line containing 0 indicates end of input.

### Output

Each test case should generate  $m + 1$  lines of output. The first line is of the form

Pile  $t$

where  $t$  is the number of the test case (starting at 1). Each of the next  $m$  lines should be of the form

Card  $q_i$  is a face up  $k$ .

or

Card  $q_i$  is a face down  $k$ .

accordingly, for  $i = 1, \dots, m$ .

For instance, in the above example with 5 cards, if  $q_i = 3$ , then the answer would be

Card 3 is a face up 4.

### Sample Input

```
5
UUUDD
RRLL
5 1 2 3 4 5
10
UUDDUUDDUU
LLRRRRLRL
4 3 7 6 1
0
```

### Sample Output

```
Pile 1
Card 1 is a face down 2.
Card 2 is a face up 1.
Card 3 is a face up 4.
Card 4 is a face down 5.
Card 5 is a face up 3.
Pile 2
Card 3 is a face down 1.
Card 7 is a face down 9.
Card 6 is a face up 7.
Card 1 is a face down 5.
```

## Problem E: The Flood

Global warming has us all thinking of rising oceans — well, maybe only those of us who live near the ocean. The small island nation of Gonnasinka has employed you to answer some questions for them. In particular they want to know how high the water has to get before their island becomes two islands (or more).

Given a grid of integers giving the altitudes of the island, how high must the ocean rise before the land splits into pieces?

### Input

Each test case begins with a line containing two positive integers  $n$ ,  $m$  giving the dimensions of the island, then  $n$  lines each containing  $m$  positive integers. The integers indicate the original altitude of the grid elements. Grid elements are considered to be adjacent only if they share a horizontal or vertical edge. Values of zero (0) along the perimeter, and all zero cells connected to these, are ocean at its initial level. Cells of 0 not connected to the perimeter (that is, surrounded by higher land) are simply sea level elevations. Furthermore, assume the ocean initially surrounds the given grid. Neither  $n$  nor  $m$  will exceed 100 and heights will never exceed 1000. A line with 0 0 follows the last test case.

### Output

For each test case output one of the two following lines.

Case  $n$ : Island splits when ocean rises  $f$  feet.

or

Case  $n$ : Island never splits.

Our convention here is if your answer is, say, 5 feet, you more accurately mean “5 feet plus a little more.” That is, at least a little water will be flowing over the originally 5 foot high portion of land.

### Sample Input

```
5 5
3 4 3 0 0
3 5 5 4 3
2 5 4 4 3
1 3 0 0 0
1 2 1 0 0
5 5
5 5 5 5 7
4 1 1 1 4
4 1 2 1 3
7 1 0 0 4
7 3 4 4 4
0 0
```

### Sample Output

```
Case 1: Island never splits.
Case 2: Island splits when ocean rises 3 feet.
```

## Problem F: Here's a Product Which Will Make You Tensor

Most people are familiar with how to multiply two matrices together. However, an alternate form of multiplication known as tensor multiplication exists as well, and works more like you would expect matrix multiplication should. Let  $A$  be a  $p \times q$  matrix and  $B$  be an  $n \times m$  matrix, where neither  $A$  nor  $B$  is a  $1 \times 1$  matrix. Then the tensor product  $A \otimes B$  is a  $pn \times qm$  matrix formed by replacing each element  $a_{ij}$  in  $A$  with the matrix  $(a_{ij}) \cdot B$ . Two examples are shown below, which also demonstrate that, like normal matrix multiplication, tensor multiplication is non-commutative:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 1 & 1 & 2 & 2 & 2 \\ 1 & 1 & 2 & 2 & 2 & 4 \\ 3 & 3 & 3 & 4 & 4 & 4 \\ 3 & 3 & 3 & 4 & 4 & 4 \\ 3 & 3 & 6 & 4 & 4 & 8 \end{bmatrix}, \quad \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 2 \end{bmatrix} \otimes \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 & 1 & 2 \\ 3 & 4 & 3 & 4 & 3 & 4 \\ 1 & 2 & 1 & 2 & 2 & 4 \\ 3 & 4 & 3 & 4 & 6 & 8 \end{bmatrix}$$

Note that there is no restriction that the number of columns in the first matrix must equal the number of rows in the second, as there is with normal matrix multiplication. The object of this problem is to determine the number of ways (if any) a given matrix can be formed as a result of a tensor multiplication.

### Input

The first line of input for a test case will contain two positive integers  $r$  and  $c$  indicating the number of rows and columns in the matrix. After this will follow  $r$  lines each containing  $c$  positive integers. The values of  $r$  and  $c$  will be  $\leq 500$ , each entry in the matrix will be no greater than 65,536, and the last test case is followed by a line containing 0 0.

### Output

For each test case, output the number of different ways the matrix could be the tensor product of two positive integer matrices, neither of which is a  $1 \times 1$  matrix.

### Sample Input

```
6 6
1 1 1 2 2 2
1 1 1 2 2 2
1 1 2 2 2 4
3 3 3 4 4 4
3 3 3 4 4 4
3 3 6 4 4 8
2 2
3 6
4 9
2 4
15 18 30 36
20 24 40 48
0 0
```

### Sample Output

```
1
0
4
```

## Problem G: Trip the Lights Fantastic

Bob Roberts (father of Little Bobby of problem D) works at the Traffic Commission for a medium size town. Bob is in charge of monitoring the traffic lights in the city and dispatching repair crews when necessary. Needless to say, Bob has a lot of free time, so to while away the hours he tries to figure out the quickest way to take short trips between various points in the city. Bob has at his disposal a lot of information: the layout of streets in the city and the location and cycle times for all of the traffic lights. To simplify the solution process, he makes the following assumptions:

1. All cars travel at the same top speed, and, if sitting at a red light, take 5 seconds to react and get up to speed. (That is, Bob assumes the car is essentially standing still for 5 seconds, then proceeds at top speed. Bob also assumes the light will not have turned back to red in the 5 seconds it takes to get going.)
2. Each car approaches a light at full speed and either passes through the light if it is green or yellow, or comes to an immediate stop if it is red. Cars are allowed to pass through a light if they hit it just as it is turning to green. Cars must stop if they reach the light just as it is turning to red.
3. The time to make turns through a light is ignored. It is possible to travel between any two lights, although perhaps not directly.

Even given these assumptions, Bob has difficulty coming up with minimum time paths. Let's see if you can help him.

### Input

The first line of each test case will contain four positive integers  $n$ ,  $m$ ,  $s$ , and  $e$ , where  $n$  ( $2 \leq n \leq 100$ ) is the number of traffic lights (numbered 0 through  $n - 1$ ),  $m$  is the number of roads between the traffic lights, and  $s$  and  $e$  ( $s \neq e$ ) are the starting and ending lights for the desired trip. There will then follow  $n$  lines of the form  $g\ y\ r$  indicating the number of seconds that each light is green, then yellow, then red. ( $1 \leq g, y, r \leq 100$ .) The first of these lines refers to light 0, the second to light 1, and so on. Following these  $n$  lines will be  $m$  lines, each describing one road. These lines will have the form  $l1\ l2\ t$ , where  $l1$  and  $l2$  are the two lights being connected by the road and  $t$  is the time (in seconds,  $t \leq 500$ ) to travel the length of the road at full speed — you should add 5 to this value to obtain the travel time when driving the road beginning at a standstill. No value enter All roads are two way. At time 0, all lights are just starting their green period and your car is considered to be at a standstill at traffic light  $s$ . Since it takes 5 seconds to get going, you may assume that  $g + y$  is never less than or equal to 5. The last test case is followed by a line containing 0 0 0 0 indicating end-of-input.

### Output

For each test case, output a single line containing the minimum time to travel from the start light to the end light. Output your results in the form `mm:ss` indicating the number of minutes and seconds the trip takes. If the number of seconds is less than 10 then preface it with a 0 (i.e., output `4:05`, not `4:5`). Likewise, if the number of minutes is less than 10, print just one digit (as in `4:05`).

**Sample Input**

```
3 3 0 2
3 4 5
3 3 3
2 4 4
0 1 1
1 2 2
0 2 12
3 3 0 2
3 4 5
3 4 3
2 4 4
0 1 1
1 2 2
0 2 12
0 0 0 0
```

**Sample Output**

```
0:16
0:08
```

## Problem H: Windows

Emma is not very tidy with the desktop of her computer. She has the habit of opening windows on the screen and then not closing the application that created them. The result, of course, is a very cluttered desktop with some windows just peeking out from behind others and some completely hidden. Given that Emma doesn't log off for days, this is a formidable mess. Your job is to determine which window (if any) gets selected when Emma clicks on a certain position of the screen.

Emma's screen has a resolution of  $10^6$  by  $10^6$ . When each window opens its position is given by the upper-left-hand corner, its width, and its height. (Assume position  $(0,0)$  is the location of the pixel in the upper-left-hand corner of her desktop. So, the lower-right-hand pixel has location  $(999999, 999999)$ .)

### Input

Input for each test case is a sequence of desktop descriptions. Each description consists of a line containing a positive integer  $n$ , the number of windows, followed by  $n$  lines,  $n \leq 100$ , describing windows in the order in which Emma opened them, followed by a line containing a positive integer  $m$ , the number of queries, followed by  $m$  lines, each describing a query. Each of the  $n$  window description lines contains four integers  $r$ ,  $c$ ,  $w$ , and  $h$ , where  $(r, c)$  is the row and column of the upper left pixel of the window,  $0 \leq r, c \leq 999999$ , and  $w$  and  $h$  are the width and height of the window, in pixels,  $1 \leq w, h$ . All windows will lie entirely on the desktop (i.e., no cropping). Each of the  $m$  query description lines contains two integers  $cr$  and  $cc$ , the row and column number of the location (which will be on the desktop). The last test case is followed by a line containing 0.

### Output

Using the format shown in the sample, for each test case, print the desktop number, beginning with 1, followed by  $m$  lines, one per query. The  $i$ -th line should say either "window  $k$ ", where  $k$  is the number of the window clicked on, or "background" if the query hit none of the windows. We assume that windows are numbered consecutively in the order in which Emma opened them, beginning with 1. Note that querying a window does not bring that window to the foreground on the screen.

### Sample Input

```
3
1 2 3 3
2 3 2 2
3 4 2 2
4
3 5
1 2
4 2
3 3
2
5 10 2 10
100 100 100 100
2
5 13
100 101
0
```

### Sample Output

```
Desktop 1:  
window 3  
window 1  
background  
window 2  
Desktop 2:  
background  
window 2
```