IEEEXtreme 4.0
24-Hour Programming Competition 2010,
Programming Problems

October 23, 2010

Abstract

This document presents an entire problem collection for the IEEEXtreme 4.0 24-hour programming competition.

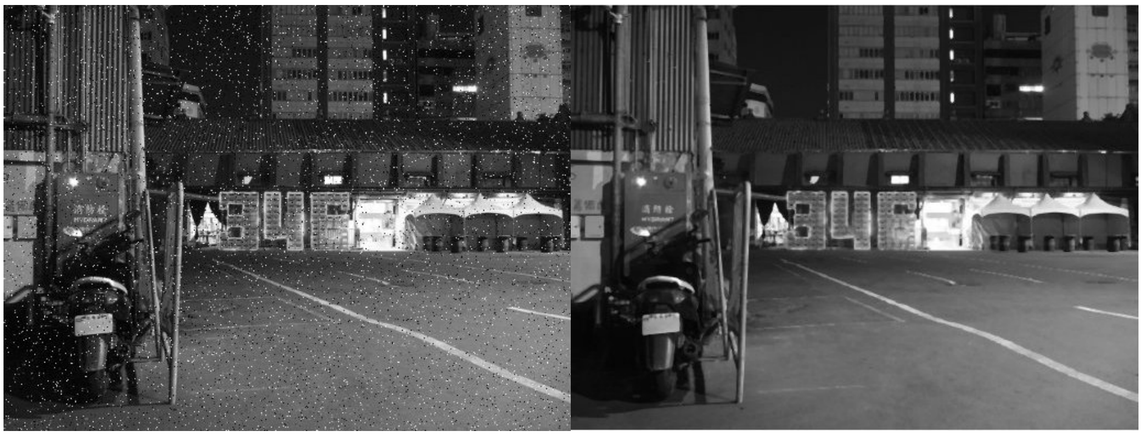
Contents

1 Median Filter (C)

1.1 Introduction

Milly Vamedian is a musician that loves photography. However his skills as a photographer are lacking and all of his images come out noisy and distorted. Milly is trying to implement a filter that will help solve the noise issues he sees in all his photographs. After some careful research Milly has decided that a median filter is what he needs to use to fix his pictures.

The median filter is one of the familiar noise reduction filters in image processing. This filter exhibits its high performance for salt and pepper noise, as shown in the following Figures 1a and 1b. In this problem, you will program an ordinary 3×3 median filter, whose input and output image format is ASCII portable gray map (PGM).



(a) Input image with salt and pepper noise (b) Output of median filter

Figure 1: Median filter

1.2 Task

Design a median filter for Milly where each pixel value of an input image f is given by $f(x, y)$, where x and y are the x and y coordinates of the pixel, respectively. It should be noted that x, y, and $f(x, y)$ are integer values. The output of the median filter, $g(x, y)$, is given by the following:

$$g(x,y) = Med\{f(x+i, y+j); \quad i = -1, 0, 1; \quad j = -1, 0, 1\}, \tag{1}$$

where the operator Med returns the median value of given nine pixels. For example, in the case of

$$Med\{100, 110, 120, 95, 105, 115, 92, 102, 112\}, \tag{2}$$

at first the values are sorted as {92, 95, 100, 102, 105, 110, 112, 115, 120}, and then the fifth value (median value) of the sorted data, which is 105 in this case, is returned as the result of the operator Med. The output image can be obtained as a result of calculation for all pixels of $g(x, y)$ using Equation (1).

Assuming the image width and height are w and h, respectively, only the pixel values for $0 \leq x < w$ and $0 \leq y < h$ are valid. To obtain $g(0, 0)$, invalid pixels such as $f(-1, -1)$ are required. In this case, please use:

$$f(0, y) \text{ instead of} f(-1, y), \tag{3}$$
$$f(w-1, y) \text{ instead of} f(w, y), \tag{4}$$
$$f(x, 0) \text{ instead of} f(x, -1), \tag{5}$$
$$f(x, h-1) \text{ instead of} f(x, h). \tag{6}$$

1.3 Input

The program gets its input image from the standard input stream. Input images are given as ASCII PGM format. A sample of this format is shown as follows.

Sample of ASCII PGM format:

```
P2
640 480
255
23 39 46 44 28 17 21 33 40 40 39 30 17 11 13 13 13
14 14 12 20 39 41 39 27 19 31 40 41 38 29 19 23 21
```

The magic code (P2) in the first line indicates that this file is ASCII PGM. The values in the second line indicate the width (640) and height (480) of this image. The value in the third line indicates the maximum value (255) of each pixel. You can assume the maximum value is 255, which means $0 \leq f(x, y) \leq 255$. After these three lines, the pixel values are stored in the raster scan order, which is the order of

$$\begin{aligned} f(0,0), f(0,1), ..., f(0,h-1), f(1,0), f(1,1), ..., f(1,h-1), ..., \\ f(w-1,0), f(w-1,1), ..., f(w-1,h-1). \end{aligned} \tag{7}$$

The pixel values are separated by white spaces or line breaks. The pixel values can be parsed easily by using `scanf`.

1.4 Output

The program writes its output image to the standard output stream as the above-mentioned ASCII PGM format.

Sample Input

```
P2
12 9
255
37 64 67 33 14 255 60 58 31 22 73 98
46 70 77 40 23 41 66 65 27 36 86 120
47 68 84 47 0 40 51 48 0 35 47 57
100 91 94 44 39 52 58 61 69 81 67 41
90 78 66 69 104 99 109 189 177 103 118 84
90 23 43 68 63 69 82 112 105 75 85 94
23 43 57 68 79 93 106 108 116 0 255 138
1 22 49 91 110 0 110 117 128 137 144 143
31 31 54 83 97 102 106 106 115 123 133 134
```

Sample Output

```
P2
12 9
255
46 64 64 33 33 60 60 58 31 31 73 98
47 67 67 40 40 41 58 51 35 35 57 86
68 77 70 44 40 41 52 58 48 47 57 57
90 84 69 66 47 52 58 61 69 69 67 57
90 90 68 66 68 69 82 105 103 85 84 84
78 57 66 68 69 93 106 109 108 105 94 94
23 43 49 68 69 82 106 110 112 116 137 138
23 31 54 79 91 102 106 110 116 128 137 138
31 31 54 83 97 102 106 110 117 128 134 134
```

2 Static Timing Analysis (D)

2.1 Introduction

Static Timing Analysis (STA) is a method of estimating the expected timing of a digital circuit without simulation. In Figure 2, a 2-bit adder circuit diagram as a sample digital circuit is shown. The purpose of STA is to find a path which gives maximum delay among all paths from inputs to outputs. Assuming that the inputs A0, A1, B0, and B1 are stored in flip-flops, and that the outputs S0, S1, and C are will be also stored to flip-flops, the maximum delay from inputs to outputs gives a performance metric of the circuit. If the maximum delay is 5,000 ps, the circuit can work at 200 MHz clock frequency.

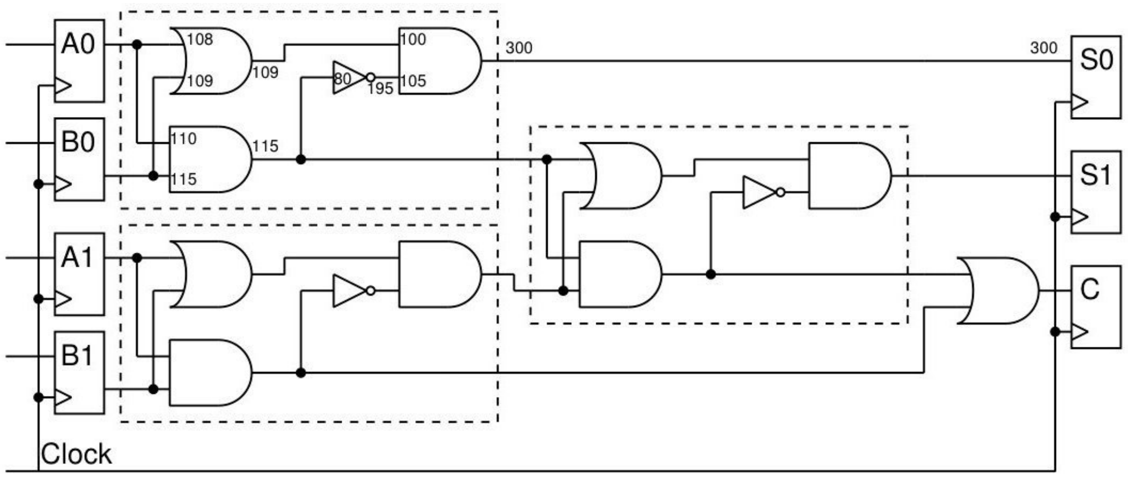


Figure 2: 2-Bit Adder

2.2 Task

For each output, the program calculates its maximum path delay from inputs. In this problem, the program needs not to show the path, which gives the maximum delay.

2.3 Input

The program gets its input data, which is a circuit diagram data, from the standard input stream. Circuit diagrams are given by a specific text format as shown below. Empty lines should be skipped by the program. Each lines conforms the following instance definition.

Instance Definition

cell_name instance_name input0 input0_delay input1 input1_delay ...

The first element cell name indicates one of primitives such as flip-flop, OR gate, AND gate, NOT gate and so forth. Basically the cell name is not important in this problem. But the program should know which instance is output. So instances of OUT cell may be referred as outputs by the program. The second element instance name indicates the unique name of each instance.

The next pair of input0 and input0 delay indicates the input from another instance which is connected to this instance, and its delay time, respectively. The input from another instance (input0) is specified by instance name, since each instance has only one output in this problem. The number of input-and-delay pairs is same as the number of inputs of the cell.

Note, that how to calculate the maximum delay to the output S0 is exemplified in Figure 2.

2.4 Output

The program writes its output to the standard output stream.

Sample Input

```
IN A0
IN A1
IN B0
IN B1
OR OR_0 A0 108 B0 109
AND AND_00 A0 110 B0 115
NOT NOT_0 AND_00 80
AND AND_01 OR_0 100 NOT_0 105
OR OR_1 A1 103 B1 108
AND AND_10 A1 120 B1 103
NOT NOT_1 AND_10 83
AND AND_11 OR_1 100 NOT_1 98
OR OR_2 AND_00 102 AND_11 109
AND AND_20 AND_00 101 AND_11 109
NOT NOT_2 AND_20 82
AND AND_21 OR_2 99 NOT_2 97
OR OR_3 AND_20 98 AND_10 97
OUT S0 AND_01 0
OUT S1 AND_21 0
OUT C OR_3 0
```

Sample Output

```
S0: 300
S1: 589
C: 508
```

3 Root Finder (E)

3.1 Introduction

In a planet where all people are living side by side, drinking from the same river and eating the same food there was an attempt to unify their languages and make the planet stronger and more globalized. An old wise man, proposed to the rulers of the planet a language of 10's of thousands of words that can describe everything in life. His proposal stood among others because all the words he generated were driven from limited number of roots, suffixes, and prefixes.

Word	Suffix	Prefix	Root	Meaning
ktb	–	–	ktb	writing
ktbo	–	o	ktb	they wrote
yaktbo	ya	o	ktb	they are writing
yaktb	ya	–	ktb	he is writing

Table 1: Unified Language

3.2 Task

You need to design a program (Morphological Analyzer) that can find the root of each given word. You have to use the training data provided in this problem to train your software on the possibility of suffixes and prefixes.

Tip 1: Calculate the probability for a combination of letters to be prefix or suffix then determine the possible root.

Tip 2: Hardcode the training data in your code.

3.3 Input

Each line is a new word that you need to find its root.

3.4 Output

The root for each input world in the same sequence.

Training Data	Sample Input	Sample Output
`ktbo,ktb`	`falo`	`fal`
`ktbn,ktb`	`faln`	`fal`
`syktb,ktb`	`syfal`	`fal`
`syktbn,ktb`	`syfaln`	`fal`
`yktb,ktb`	`yfal`	`fal`
`faln,fal`	`ktbo`	`ktb`
`syfal,fal`	`ktbn`	`ktb`
`syfaln,fal`	`syktb`	`ktb`
`yfal,fal`	`syktbn`	`ktb`
`falo,fal`	`yktb`	`ktb`
`syrqsn,rqs`	`rqso`	`rqs`
`yrqs,rqs`	`rqsn`	`rqs`
`rqsn,rqs`	`syrqs`	`rqs`
`syrqs,rqs`	`syrqsn`	`rqs`
`rqso,rqs`	`yrqs`	`rqs`

4 Recommend Cloths (F)

4.1 Introduction

You couldn't finish your IT degree and your family is poor. The only job offered is a fashion consultant. People come to you requesting your help to find a shirt that looks good with blue pants, or what can they buy for pink socks? As a genius drop out, you could detect a pattern in customer's preferences. After you made your studies and analysis, you found out that everything a fashion consultant needs to know is customer's eyes color before he can recommend any cloths as indicated in Table 2.

4.2 Task

Design a program that can recommend a shirt color if the consultant knows the eyes and pants color. Similarly recommend a pants color if the consultant knows the eyes and shirt color. Moreover, the system can identify the best eye color customer for a combination of cloth (pants and shirts).

4.3 Rules

The list below describes only two colors of pants and selected number of shirts that are usually available in the market. Your software should rely on the list below, but not limited to it.

Eye Color	Pants Color	Shirt Color
green	blue	black
green	black	white
brown	blue	pink
brown	black	green
blue	blue	blue
blue	black	gray
gray	blue	yellow
gray	black	red

Table 2: Cloths Patterns

4.4 Input/Output

Sample Input 1

```
eyes:gray pants:black what do you recommend for me?
```

Sample Output 1

```
if your eyes are gray and you have black pants I recommend you buy red shirt
```

Sample Input 2

```
eyes:gray shirt:red what do you recommend for me?
```

Sample Output 2

```
if your eyes are gray and you have red shirt I recommend you buy black pants
```

Sample Input 3

```
shirt:red pants:black Do you think they are going to fit me?
```

Sample Output 3

```
black pants and red shirts goes perfect on gray eyes person
```

Sample Input 4

```
eyes:gray shirt:pink what do you recommend for me?
```

Sample Output 4

```
sorry I don’t have anything for you, sir
```

5 Longest Palindromes (I)

5.1 Task

Given a string with no more than 20,000 characters long, you are called to find the longest palindrome in it while ignoring punctuation, whitespace, numbers, and case. The longest palindrome is guaranteed to be at most 2,000 characters long before whitespace and punctuation are removed.

5.2 Input

The input will be surrounded by quotation marks `"..."`.

5.3 Output

Only count the number of characters ignoring whitespaces and punctuation.

Sample Input 1

```
"Are we not drawn onward, we few, drawn onward to new era?"
```

Sample Output 1

```
43
```

6 Hidden Words (J)

6.1 Task

Write a program that reads from the input stream, one line at a time. Each line consists of a number of words separated by a blank character or a dot. Your program must check each line and display it on the screen if it contains one of the following words, either written correctly or written with their letters in a random order.

Words to look for: sit, won, to, not

Each line consists of at most 80 characters, and each word of at most 15 characters.

6.2 Input

M lines that are composed from N characters, where $1 \leq N \leq 80, \quad 1 \leq M$.

6.3 Output

K lines, which are the ones in which your program found hidden words, where $K \leq M$.

Sample Input 1

```
page now lying before not be found with any integers signal.
This has been already too much an object oriented The value cannot be found.
There must be an error.
The upper regions of the globe have not the.
continue to search however its nothing there.
Oh this is a sample of a text.
To the earth and moon are forever not to its dreams.
to its flowers, to its golden goals.
perfect and limitless programs exist.
it not hang eternally between their loops and programming.
```

Sample Output 1

```
page now lying before not be found with any integers signal.
The upper regions of the globe have not the.
continue to search however its nothing there.
To the earth and moon are forever not to its dreams.
to its flowers, to its golden goals.
it not hang eternally between their loops and programming.
```

7 Gomoku (K)

7.1 Introduction

In this problem, we think about the game where two players set black and white stones on the lattice points of the game board as shown in Figure 3. The first move must begin with a black stone. The game will finish when each of the players sets five stones in a row whose direction is vertical, horizontal, or diagonal.

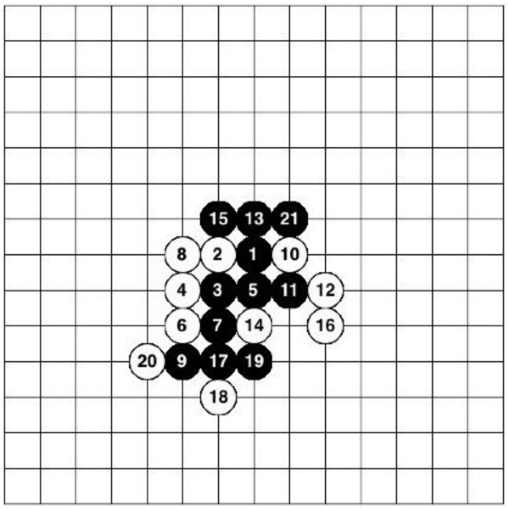


Figure 3: **Gomoku Example.** This image was copied from Wikipedia (En). The original description was: Moves 1-21 of a sample game of gomoku. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

7.2 Task

Please write a program that judges whether the black player can set five stones in a row in the next two turns or not when a current state is given, where "turn" includes a motion of black and a motion of white. Here, both players set stones aiming their own win. The board is regular tetragon and its size is given from standard input with a current state by the following input format.

7.3 Input

The first line shows the size of the board. The following lines show the list of the coordinates of the stones set by players alternately. Here, the number of coordinates is certainly even.

```
5    # The size of the board
1,0  # First coordinate of the black
0,1  # First coordinate of the white
1,1  # Coordinate of the black in the 2nd turn
0,2  # Coordinate of the white in the 2nd turn
2,2  # Coordinate of the black in the 3rd turn
0,3  # Coordinate of the white in the 3rd turn
```

7.4 Output

If the black wins by setting five stones in a row in the next two turns, the output is 1, and in the other cases the output must be 0.

Sample Input

```
5
1,0
0,1
1,1
0,2
2,2
0,3
```

Sample Output

```
0
```

8 Underground (M)

8.1 Introduction

The mayor of Extreme City has decided that a new underground line is necessary to accommodate the astonishing progress and exponential growth of his megalopolis. Extreme City has been very carefully planned, in a Manhattan-like style, and it is deployed forming a rectangular grid of east-west avenues and north-south streets. Furthermore, for ease of maintenance, the underground tunnels lie exactly below the avenues and streets, and change direction only at the intersection of a street and an avenue. There are a number of lines already in place, and they are working to their limits. Therefore, it is out of question for the new line to share any of the existing tunnels or even make any part of the new tunnel deeper below an existing one. Indeed, safety restrictions mandate that there can be at most one tunnel between any two adjacent intersections. Still, new tunnels can cross the old tunnels, perpendicularly, below them, at any of the intersections between streets and avenues.

The sequence of stations for the new line has been chosen, based on estimated traffic, and also on political convenience, we suspect. The problem now is to find the path the tunnel should follow for connecting each pair of adjacent stations in the new line, subject to the constraint that no part of the new tunnel can be built between any two intersections for which a tunnel already exists. Yes, the mayor wants the shortest possible such tunnel.

8.2 Task

Your task is to create a program that will analyze the current network of tunnels in Extreme City and, given the locations of two adjacent stations on the new line, computes the shortest tunnel that can be built between those two stations, such that no part of this tunnel coincides with a part of an existing tunnel.

8.3 Input

The first line of the input contains two integers, separated by spaces, representing the number of streets, S, and the number of avenues, A, in Extreme City. The second line contains one integer, representing the number of existing tunnels, T. Exactly T lines follow, each line describing one of the existing tunnels. Each line contains an odd number of integers: the first number represents the number of intersections used to define the corresponding tunnel, N; N pairs of numbers follow (in the same line) $X_1Y_1, X_2X_2, ..., X_nY_n$, each X_iY_i representing the intersection of the avenue X_i with the street Y_i. This sequence means that the tunnel starts below the intersection X_1Y_1, ends below the intersection X_nY_n, and changes direction at each of the intervening points $X_2Y_2, ..., X_{n-1}Y_{n-1}$, and goes on a straight line from $X_{i-1}Y_{i-1}$ to X_iY_i, for $i \in [1..n]$. In Extreme City, streets are numbered from south to north, starting at zero, and avenues are numbered from west to east, also starting at zero. One more line exists after the T lines just described. This last line contains two pairs of numbers SX SY FX FY, the former pair representing the intersection where lies the station of where the new tunnel starts and likewise for latter pair, now referring the station where the new tunnel ends.

8.4 Output

The output file contains one line, describing the shortest tunnel as specified, using the same format as the one used for describing each tunnel in the input file. Note that the length of the tunnel is measured by the number of intersections that the tunnel touches, not by the number of intersections in which it changes direction. Still, the output must display only these intersections where the tunnel changes direction (including the starting point and the ending point).

In case more than one tunnel can be built with the same minimal length, your program should prefer the one that moves northwards earlier, then eastwards, then southwards. More precisely: take two tunnels, T and U, with same minimal length, going from the starting intersection to the finishing intersection, and consider the first intersection where those tunnels diverge. Let dT be the direction taken by T and dU the direction taken by U at this diverging intersection. The possible values for the dT and dU are north, east, south and west. Your program should choose T if dT comes before dU in the sequence of directions as listed in the previous sentence; otherwise, it should choose U.

In case it is impossible to build the tunnel, the output file should contain a single line, carrying the message "impossible" (without the quotes).

Constraints

Number of streets, $S : 2 \leq S \leq 100$.

Number of avenues, $A : 2 \leq A \leq 100$.

Number of existing tunnels: $0 \leq T \leq 100$.

Number of pair of integers describing each tunnel, $2 \leq N \leq 100$.

Each tunnels is well formed: for all $i \in [1..n-1]$ it is true that
$(X_i == X_{i+1}$ and $Y_i! = Y_{i+1})$ or $(X_i! = X_{i+1}$ and $Y_i == Y_{i+1})$.

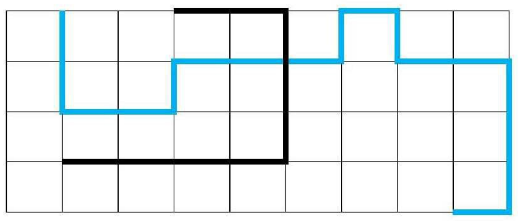
Each given pair of given coordinates XY is within bounds: $0 \leq X < A, 0 \leq Y < S$.

Sample Input 1

```
10 5
2
4 1 1 5 1 5 4 3 4
11 1 4 1 2 3 2 3 3 6 3 6 4 7 4 7 3 9 3 9 0 8 0
0 0 9 3
```

Sample Output 1

```
7 0 0 3 0 3 2 8 2 8 4 9 4 9 3
```


(a) Given tunnels

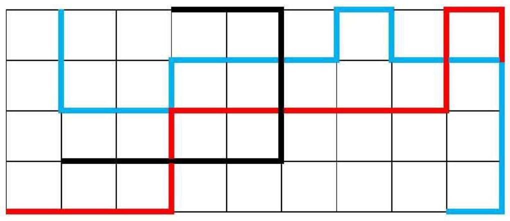

(b) New tunnel

Figure 4: Solution corresponding to Sample Input 1, in red

Note on Sample Output 1

A tunnel with the same length could be built going east from (0, 0) to (4, 0), then north to (4, 2) and continuing like the given solution. However, we prefer the solution as computed, because it goes north at the intersection (3, 0), where as the other goes east.

Sample Input 2

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

Sample Output 2

```
impossible
```

Note

The requirement that the new tunnel cannot be run between two adjacent intersections which are already connected by a part of a tunnel does not apply to the current network. More precisely, in the set of tunnels described in the input file, there may be tunnels parts of which lie below the same part of a street or avenue.

9 Paged Memory Management (O)

9.1 Introduction

Operating systems use many techniques to allow large programs to be executed using the RAM, which can be smaller than the total size of the program in main memory. One such technique is Paging. In this technique, the main memory is conceptually divided into equal sized “pages”. The RAM is also divided into blocks of the same size. The program execution can begin with a few pages of the program loaded in the RAM. Additional “pages” of the program are fetched as needed. A page table keeps information about the pages in RAM. If a page is not in RAM a “page fault” has occurred and the page will be fetched into the RAM from main memory before the execution can continue. When all the available RAM locations are full, the next “page” required for execution replaces a page in the RAM, based on one of many schemes. We will use the Least Frequently Used (LFU) scheme to select the page that is to be replaced.

All addresses in the program are virtual, that is, they do not refer to actual RAM locations. The virtual address (VA) will be translated to the physical address (PA). In our scheme, the address is a 32 bit integer in which the most significant 20 bits comprise the page number and the last 12 bits are the offset within the page. This gives a page size of 4096 bytes. And 2^{20} number of pages for a program.

Here is an example, a virtual address 51400 in the program, will be used to calculate the page number as 51400/4096 (the page size) giving page number 12 and the remainder becomes the offset of 2248. Assuming the page is in RAM, the page table provides the base address for the page, say 86016. Thus the physical address is $86016 + 2248 = 88264$.

If the page is not in the RAM, a page fault has occurred and a page from memory has to be replaced. The page table has a column that stores the count of how many times a page has been used, and the page that has been used least frequently is replaced. For example, page 29 that is at address 28672 is selected and the page table is modified to reflect the change.

9.2 Task

Please write a program that simulates paged memory management using the above description. The program will read a starting set of pages in RAM starting from a base address that is also provided. The second input is a list of virtual addresses that are to be executed. The program will calculate the physical address and the page to be replaced if there is a page fault.

You may assume in this task that only 5 rows of the page table are available to you. This is done to provide sample data and create page faults in this task.

9.3 Input

There will be two inputs from the standard input stream. One input is “Pagefile”, that provides the starting address for RAM, followed by the number of pairs specifying page numbers of pages that are in the RAM and the starting usage frequency count for the page. The second input “VirtualAddressfile” gives the number of virtual addresses followed by a list of virtual addresses that are to be executed.

9.4 Output

Output will be a line for each virtual address, giving the virtual address, physical address and the replaced page (if applicable) else -1.

Sample Input

Note, page table size is only 5.

```
Pagefile: 75000 5 4 6 12 2 20 8 70 5 22 1
VirtualAddressfile: 5 16400 17800 51500 27300 54310
```

Sample Output

```
[Virtual Address: 16400 PhysicalAddress: 75016 Replaced Page: -1]
[Virtual Address: 17800 PhysicalAddress: 76416 Replaced Page: -1]
[Virtual Address: 51500 PhysicalAddress: 81444 Replaced Page: -1]
[Virtual Address: 27300 PhysicalAddress: 94108 Replaced Page: 22]
[Virtual Address: 54310 PhysicalAddress: 92446 Replaced Page: 6]
```

10 Digraph Cipher (P)

10.1 Introduction

Exchanging information securely has been a challenge and a fascination since the time human beings started communicating. A digraph cipher is an encryption mechanism that replaces each pair of letters in the plaintext (message to be encrypted) by another pair of letters. In order to encrypt and transmit a message, the sender and receiver must agree on a keyword. Before encryption, the letters of the alphabet are written in a 5×5 square (The Keyword is "XTREM" for all Encryptions), beginning with the keyword, and combining the elements "I" and "J" into a single cell.

Next, the message is broken into pairs of letters called *digraphs.* So that each pair contains two unique letters, an "x" is added between any repeated letters that fall in the same pair or at the end to make a single final letter a digraph. All digraphs fall into one of three categories: both letters are in the same row; both letters are in the same column; or both letters are in different rows and columns.

For encryption, letters in the same row are replaced by the letter to the immediate right of each one in the matrix. If a letter is at the end of a row, it is replaced by the letter at the beginning. Letters in the same column are replaced by the letter immediately below each one. If a letter is at the bottom of a row, it is replaced by the letter at the top. If the letters are in neither the same row nor the same column, the first letter is replaced by the letter in its row that is at the intersection with the column of the second letter. The second letter is replaced by the letter in its row that is at the intersection with the column of the first letter. The message is easily deciphered by the recipient using the same keyword.

10.2 Task

Please, write a program that takes a keyword and a message and returns the message in its encrypted format.

Encryption Square

X	T	R	E	M
A	B	C	D	F
G	H	I/J	K	L
N	O	P	Q	S
U	V	W	Y	Z

10.3 Input

The program gets two unique inputs: the keyword and the message.

Valid Assumptions

1. Keyword contains only alphabets and message text contain only alphabets and spaces.
2. The message is only one line of text.
3. DO NOT include white spaces when comparing.
4. Add X's for double letters before adding X's at the end to create an incomplete digraph, e.g. littles $\rightarrow$ li tx tl e $\rightarrow$ li tx tl ex
5. If I/J is selected as a digraph solution for a message, then the output should only be "I" (not IJ, J, I/J), e.g. lp $\rightarrow$ is

10.4 Output

The program outputs the given message in an encrypted form.

Sample Input

```
Keyword: Xtreme
Message: This is fun
```

Sample Output

```
Message: Bolp lp azua
```

11 Heckle or Jeckle (Q)

11.1 Introduction

Heckle and Jeckle were captured in Hula Hula Land. They were both poisoned and left with a box containing a number of jars filled with magical antidote. The antidote will work only on one of them (Heckle or Jeckle) when the box is empty. They are to take turns in drinking the jars. Heckle goes first. In each turn, Heckle or Jeckle must drink some jars. For the magical antidote to work, the number of jars they drink in one turn must be a power of six, i.e. 1, 6, 36, 216, and so on. The last one to drink the antidote will survive.

11.2 Task

Please, write a program that predicts the name of the survivor for a certain number N of jars in the box. Assume that both, Heckle and Jeckle, drink the antidote optimally to survive.

11.3 Input

The program should read the input file from the standard input. The input file consists of many test cases. Each test case consists of one integer, N, on a separate line, where

$$1 \leq N \leq 1,000,000,000.$$

The last line is a 0 which is not a test case.

11.4 Output

The output for each case should be the case number, starting with 1, followed by a dash "-", and the name of the survivor, either Heckle or Jeckle.

Sample Input

```
1
2
3
9
0
```

Sample Output

```
1-Heckle
2-Jeckle
3-Heckle
4-Jeckle
```

12 Paintball (R)

12.1 Introduction

You are challenging your friends in paintball. You claim that you can shoot a rectangular target with different shots, and be able to paint it whole with the minimum number of shots. The target is a white rectangle of size width by height. The center of your shots should be on a horizontal line (parallel to the width of the rectangle) drawn through the middle of the rectangle so that it divides the rectangle into two congruent rectangles. This line extends infinitely out of the rectangle.

You will shoot alternating red and green shots on the rectangle so that the entire rectangle is covered. The shots will paint a perfect disk on the rectangle, and the center of every disk/shot is placed on the line, not necessarily within the rectangle bounds.

You will shoot sequentially from left to right, i.e. the center of each next shot must lie strictly to the right of the center of the last previously placed shot. Each shot is placed on top of all previously placed shots, i.e. when a shot is fired it covers any parts of previous shots that overlap. To challenge yourself even more, you have decided to alternate shot colors, starting with either red or green shots.

12.2 Task

Please, write a program that calculates the minimum number of shots you need to make, if possible, to paint a white rectangular target. Note that each shot can only be used at most once.

12.3 Input

The program should read the input file from the standard input. It will be tested on one or more test cases in the file. Each test case is made of three lines. The first line contains the width and height of the white target rectangle (both between 1 and 10000, inclusive). The second line contains the set of red shots you have and the third line is the set of green shots (each between 1 and 50 shots, inclusive). The elements on every line are separated by one space and they represent the diameter of the disk you will make when you make this shot. The diameter is between 1 and 10000 inclusive. The end of the test cases is -1 -1.

12.4 Output

Each line in the output should be the smallest number of shots that you can make as described above such that every point in the rectangle is covered by at least one disk. If this is not possible to cover the target, the answer is -1. Figure 5 illustrates the solution of the first test case.

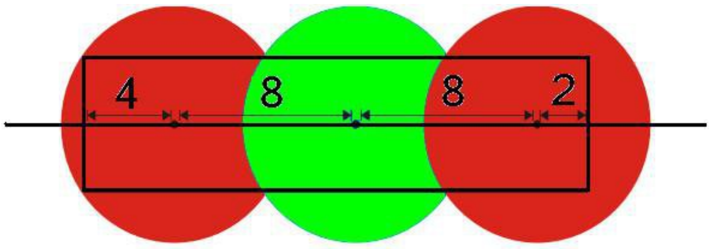


Figure 5: Paintball Shots

Sample Input

```
22 6
10 10
10 8
30 5
4 10 7 8 10
5 6 11 7 5
16 4
6 5 7
5
4 4
5
6
-1 -1
```

Sample Output

```
3
4
-1
1
```

13 I-Game (S)

13.1 Introduction

Electronic devices are getting smaller and smaller, so the only way to push its buttons is by using a special mini-screwdriver. Since it is an IEEE screwdriver, its head is shaped as the letter I, which means that touching a button (marked with an "X") on the grid may actually hit six others buttons (marked with an "$*$"). In the initial state of the game all lights are off. Touching a button will flip the light state of the pressed buttons. Touching buttons on the edge of the grid reduces the number of extra buttons affected.

$*$	$*$	$*$
	X	
$*$	$*$	$*$

13.2 Task

Please, write a program to simulate the I-game on a 10×10 grid: given a list of points the I-screwdriver touched as an input you should give the lights state as an output.

13.3 Input

A list of $0 \leq x, y \leq 9$ coordinates, each one in a single line:

X Y, where X—column index, Y—row index.

Note: Grid starts at 0 0 and ends at 9 9.

13.4 Output

The resulting light pattern as a 10×10 square listed as "X" for on and "." for off.

Sample Input 1

```
6 7
0 0
6 7
```

Sample Output 1

```
X . . . . . . . . . .
X X . . . . . . . . .
. . . . . . . . . . .
. . . . . . . . . . .
. . . . . . . . . . .
. . . . . . . . . . .
. . . . . . . . . . .
. . . . . . . . . . .
. . . . . . . . . . .
. . . . . . . . . . .
```

Sample Input 2

```
5 5
1 7
6 6
```

Sample Output 2

```
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . X X X . . .
. . . . . . X X . .
X X X . X X . . . .
. X . . . X X X . .
X X X . . . . . . .
. . . . . . . . . .
```

14 I-Game, Part II (T)

14.1 Introduction

Remember I-Game? Well, now we would like you to do the reverse: given the output compute a minimal list of moves which can generate it.

14.2 Task

Please, write a program to simulate the inverse of the I-game on a 10×10 grid (index starts at 0 0): given an output of the I-game, provide a minimal length possible input for it.

14.3 Input

The desired light pattern as a 10×10 square listed as "X" for on and "." for off.

14.4 Output

A list of $0 \leq x, y \leq 9$ coordinates, each one in a single line, which generate the desired light pattern:

X Y, where X—column index, Y—row index.

Note: Grid starts at 0 0 and ends at 9 9.

Output should be ordered in an incremental index based on the left index lowest to highest in case left index is the same then right index is should be used.

Sample Input 1

```
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . X X X
. . . . . . . . X .
. . . . . . . X X X
. . . . . . . . . .
```

Sample Output 1

```
8 7
```

Sample Input 2

```
. . . X X X . . . .
. . . X . X . . . .
. . . X . X . . . .
. . . X X X . . . .
. . . . . . . . . .
. . . . . . . . . .
. . . . . . . . . .
. X . . X . . . . .
. . X X . . . . . .
. X . . X . . . . .
```

Sample Output 2

```
2 8
3 8
4 1
4 2
```

15 Service Periods (U)

15.1 Introduction

Nowadays, even the dispatchers and organizers of bus, subway, and ferry systems are using computers and software to make their daily lives easier. A lot of things go into creating and managing a transit system: you've got vehicles, stops, routes, and schedules to keep track of. Fortunately, our system already has all of the software we need to get OUR job done, but Google has come up with a new 'standard' for representing a transit system that we need to accommodate so that we will show up in Google Maps. They're introducing something that we never even thought we'd need: *Service Periods.* A Service Period is an unbroken stretch of dates during which our system is operational. Keep this in mind: each route is only active during scheduled dates. We need to examine all of our routes (and all of their schedules) to figure out what we should tell Google our Service Periods are.

15.2 Task

Please, write a program that takes a collection of Routes and Schedules, and outputs all of the service periods that encapsulate that collection. The list of service periods that is output should be sorted by starting date.

15.3 Input

The program gets its input from the standard input stream. The first line of input contains a non-negative integer $n, 0 \leq n \leq 100,$ the number of routes in the system. Next will be n route entries, one per line. Each route entry is made up of a name (alpha-numeric, no whitespace) and an ID number separated by whitespace. Following the route entries will be exactly one blank line. A non-negative integer $m, 0 \leq m \leq 1000$ will be on the line following the blank line, it describes the number of schedules our system contains. Finally, the input will contain m schedule entries, one per line. A schedule entry consists of three pieces of information: unique schedule ID, the ID of the route that the schedule belongs to, and a range of dates that the schedule is active. Each of the schedule entries' pieces of information are separated by whitespace. The dates are given in ISO-8601 format: YYYY-MM-DD and are separated by two colons "::".

15.4 Output

The program writes its output to the standard output stream. The first line of output will describe the number of service periods for our transit system. Following the first line will be a list of service periods, one per line. Each service period will be described by a line containing the start and end dates of the service period in ISO-8601 format, separated by two colons "::".

Sample Input

```
3

uptown 1

downtown 2

crosstown 3


5

1 1 2010-10-01::2010-10-30

2 1 2010-12-01::2011-01-31

3 2 2010-10-15::2010-11-15

4 3 2010-09-01::2010-10-01

5 3 2010-12-10::2010-12-25
```

Sample Output

```
2

2010-09-01::2010-11-15

2010-12-01::2011-01-31
```

16 Navigation (V)

16.1 Introduction

GPS devices are becoming so ubiquitous that you, a college student in engineering, probably have two: one in your car and one on your phone. GPS can be especially useful when combined with route planning software to give you "turn-by-turn" instructions on how to get to your destination. It becomes even more powerful when you combine those instructions with knowledge about the traffic situation in the area. For some reason, the turn-by-turn software on your phone has stopped working. Luckily, you've got your Laptop and a USB cable with you. Also, to your good fortune, you happen to be in a city where the roads are laid out in a uniform grid that consists of streets which intersect at 90 degree angles oriented with the cardinal directions. Nobody can escape traffic though; certain segments of road will have heavier traffic than others. Your job is to figure out the fastest route to take you from where you are to where you need to go.

Keep in mind that roads which travel East/West are called "streets" and are numbered from South to North starting at 1. Also, roads which travel North/South are called "avenues" and are numbered from West to East starting at 1. If there is no traffic on a segment of road, you can assume it will take one minute to cross the segment.

16.2 Task

Please, write a program that finds the fastest route from one location to another and outputs turn-by-turn instructions.

16.3 Input

The program gets its input from the standard input stream. The first line of input contains a perfect square $n, 0 < n < 1000$: the number of intersections in the city. This implies that the city is $\sqrt{n} - 1$ blocks from west-to-east and $\sqrt{n} - 1$ blocks from north-to-south.

The next line will contain a non-negative integer m. That line is followed by m lines, each containing two pairs of integers and real number in the following format: $(x_1, y_1), (x_2, y_2)t$. Each line describes the traffic congestion for a segment of road in the city. So it'll take t minutes to traverse the the segment between the intersection of street x_1, avenue y_1, and street x_2, avenue y_2. Keep in mind that the physical distance between the two points on each line will always be one block.

The last line of input contains two pairs of integers listed as tuples in the following format: $(x_1, y_1), (x_2, y_2)$. The first pair is your current location, while the second is your intended destination. In each pair, the first number indicates a street number and the second number indicates an avenue number.

16.4 Output

The program writes its output to the standard output stream. Output a list of instructions, followed by the total time the trip will take. Remember to take care to properly pluralize blocks and minutes.

Sample Input

```
9
3
(1, 1), (2, 1) 5
(1, 2), (1, 3) 10
(2, 2), (3, 2) 2
(1, 1), (3, 3)
```

Sample Output

```
Start at the intersection of 1st Street and 1st Ave.
Drive East 1 block and turn left on 2nd Ave.
Drive North 1 block and turn right on 2nd St.
Drive East 1 block and turn left on 3rd Ave.
Drive North 1 block and arrive at your destination.
Total trip time: 4 minutes.
```

17 Beautiful People (W)

17.1 Introduction

During televised sports events, be it football, basketball, tennis or whatever, we, spectators at home, are often entertained during breaks and stoppages with shots of beautiful people in the arena: smiling girls with tight colorful t-shirts, brave young men with funny hats, faces painted with their team's colors, the elegant wives of players, always displaying a mix of serenity and anxiety, an occasional VIP hiding behind sun glasses, sometimes evil-looking, angry hooligans shouting at the referee.

Broadcasting an important match is a complicated task and it requires a lot of skill from the director, who has to manage dozens of cameras at the same time. In fact, some of these cameras are handled automatically, which is a great, because the director can concentrate on capturing the sporting action in the field and not be distracted with the mundane activity going on in the seating blocks.

One of those automatic cameras is the so-called *beautiful-people camera.* This camera is hung by steel cables just above the center of the field and scans the seating area always in a clockwise manner, showing "beautiful people". The system receives the seating coordinates of spectators worth showing, on the fly, as they are collected by a team of human scouts, together with the amount of time that they are to be displayed. For example, the triplet `<12, 34, 5>` means the camera should focus for exactly 5 seconds on the person seating (or standing) at row 12, column 34. The camera is controlled remotely, by software.

17.2 Task

Your task is to create a program to automatically control the beautiful-people camera for a football stadium. Your program must write a sequence of high level commands that will be sent to the camera interface, in real time, in order to turn the camera incrementally or to have it stay still focusing on someone, thus showing the beautiful people that the scouts have spotted.

These are the rules:

1. The seating area, encompassing the stadium, has a fixed number of rows and all rows have the same fixed number of seats. The seating area forms a rectangular grid, where each seat is represented by the row number and the column number. The first row (closer to the field) is row 0. The first column, arbitrarily located, is column 0. The numbering of seats along a rows starts at column 0, and increases clockwise. As the stadium is closed, the last column of seats is located immediately to the left of first column.
2. Initially, the camera points to the first row of the first column.
3. Every second, at the tick of the second, starting at second 1, the software first checks for new triplets from its input line, each triplet referencing a beautiful person that should be shown for some seconds, then processes triplets it has received so far, and sends the appropriate commands to the camera interface.
4. The camera interface supports the following commands, where, in all cases, the argument is a positive integer:
    (a) up x: turn the camera upwards on the current column by x rows, instantaneously.
    (b) down x: turn the camera downwards on the current column, by x rows, instantaneously.
    (c) right x: turn the camera clockwise (i.e. to the right), on the current row, by x columns, instantaneously.

(d) wait x: lock the camera in the current position for x seconds.

5. The rules for turning the camera when it unlocks after showing a beautiful person and there are pending requests are the following:

   (a) If there is a pending request for showing a person in the current column above the current position, not counting the persons that have already been shown on the current camera cycle, turn the camera up to the closest such person and show that person.
   (b) Otherwise, if there is a pending request for showing a person in the current column below the current row, not counting the persons that have already been shown on the current camera cycle, turn the camera down to the closest such person and show that person.
   (c) Otherwise, turn the camera right to the next column for which there is a pending request for showing a person and then apply Rule 6, below.

6. The rules for handling the camera after it has turned right are the following are the following:

   (a) If there is a pending request for showing the person in the current position, show that person.
   (b) Otherwise, if there is a pending request for showing a person in the current column above the current position, turn the camera up to the closest such person and show that person.
   (c) Otherwise, turn the camera down to the closest person below the current position and show that person.

7. The rules for turning the camera at the start of the operations, provided there are requests at the tick of second 1, are the following:

   (a) If there is a pending request for showing the person in the initial position, show that person.
   (b) Otherwise, if there is a pending request for showing a person in the first column above the first row, turn the camera up to the closest such person and show that person.
   (c) Otherwise, turn the camera right to the next column for which there is a pending request for showing a person sitting in that column and apply Rule 6, above.

8. All pending requests for showing the same person are handled together as a single request for the maximum duration of those requests.

9. If there are no pending requests, but the end of the request stream has not been reached, keep the camera locked in the current position for one second.

10. Continue until the end of the request stream has been reached and there are no more pending requests.

Note that Rule 5 implies that after the camera has moved away from a person it will only return that person in the future after it has gone around the stadium at least once. This is fine if there are other persons waiting to be shown in other columns, but it is awkward in the uncommon situation where there are no requests for persons other than persons in the current column and all these persons have been shown in the present cycle. In this case, the camera does a full circle and returns to the same column. *This is a feature, not a bug.*

17.3 Input

The first line of the input contains two positive integers, representing the number of rows, NR, and the number of columns NC, in the stadium. An indeterminate number of lines follow. Each of these lines contains a non-negative integer T, followed by T triplets of integers `<R, C, S>`, each triplet representing a request for showing the VIP seating in row R, column C, for duration of S seconds. The first line in this group of lines represents the requests that are available as the processing starts, at the tick of second 1. In general, the $(i+1)$th line in the input represents the requests that have arrived "just before" the tick of second i. Thus, the requests from the $(i+1)$th line are available at the tick of second i, and thereafter (until they are processed), but not before.

17.4 Output

The output contains the sequence of commands that have been sent to the camera interface for processing the request, one command per line. Each command is written as a string and a positive integer, separated by a space.

Constraints

Number of rows in the stadium, $NR : 1 < NR \leq 100$.

Number of columns in the stadium, $NC : 1 < NC \leq 1000$.

Number of triplets in each line, $T : 0 \leq T \leq 10$.

Row number, $R : 0 <= R < NR$.

Column number, $C : 0 \leq C < NC$.

Duration of display, in seconds, $S : S > 0$.

The number of lines in the input is not greater than 1001.

Sample Input 1

```
10 15
2 2 9 2 0 2 2
0
1 0 2 2
0
0
0
0
0
1 0 2 5
```

Sample Output 1

```
right 2
wait 2
right 7
up 2
wait 2
right 8
down 2
wait 2
wait 1
wait 1
right 15
wait 5
```

Sample Input 2

```
15 20
2 4 6 1 0 0 1
0
2 4 6 2 2 2 3
3 5 8 2 4 8 2 3 8 2
0
```

Sample Output 2

```
wait 1
right 6
up 4
wait 1
right 16
down 2
wait 3
right 4
up 2
wait 2
right 2
wait 2
up 1
wait 2
down 2
wait 2
```

Sample Input 3

```
15 20
2 3 3 1 4 3 1
0
1 3 3 1
2 5 7 1 9 7 1
2 8 7 1 9 7 2
```

Sample Output 3

```
right 3
up 3
wait 1
up 1
wait 1
right 20
down 1
wait 1
right 4
up 2
wait 1
up 3
wait 1
up 1
wait 2
```