Thank you for you interesting in becoming a developer for Solium Capital. We like to make sure you write code as part of the interview process, because that’s what you’ll be doing if we hire you. Please take a look at the three problems below and create a solution to solve ONE of them, and mail it to [nathan.smith@solium.com](mailto:nathan.smith@solium.com).

We respect that people have other commitments, but in general we like to see solutions within 5 business days.

**Please read and understand the following before reading the problems as this section describes general requirements.** If you have any questions, don’t hesitate to send an email to nathan.smith@solium.com.

For your solution, you may use the language of your choice, so long as it can be run on Windows or Linux. Since our application is written in Java, there is a slight bias toward it, but we also welcome more creative submissions. This is your chance to prove that you can be one of our developers by impressing us. You must package all of the source code and documentation within a single file archive, preferably gzipped tarballs or zip files.

The correctness of your submission will be determined using an automated test suite, so your program **must** read input from **stdin** and write output to **stdout**. Command-line arguments or GUI applications will not be compatible; any submission where these are the only means of I/O will not be considered.

You may not use any enhanced functionality of any language (i.e. do not use the AWT libraries to determine if a point lies within a polygon.) We encourage the use of tools such as Maven for build Java programs and JUnit for testing them, but you may not use external libraries for any other purpose (i.e. OpenGL for geometry, or a command line argument parsing library – we like to see how **you** solve the problem)

Finally, please note that while we provide sample input and output, we expect your solution to be general and work for **whatever legal input** we choose to give it.

Good Luck.

## Problem #1: Cylindrical Matrix

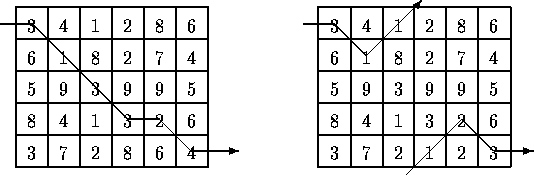
Problem

Given a matrix of integers, you are to write a program that computes a path of minimal weight. A path starts anywhere in column 0 (the first column) and consists of a sequence of steps terminating in column x (the last column). A step consists of traveling from column i to column i+1 in an adjacent (horizontal or diagonal) row. The first and last rows (rows 1 and m) of a matrix are considered adjacent, i.e., the matrix ``wraps'' so that it represents a horizontal cylinder. Legal steps are illustrated below.

  
   
The weight of a path is the sum of the integers in each of the n cells of the matrix that are visited.

The weights may be negative.

For invalid input, just exiting with a non-0 status is fine

For example, two slightly different matrices are shown below (the only difference is the numbers in the bottom row).   


The minimal path is illustrated for each matrix. Note that the path for the matrix on the right takes advantage of the adjacency property of the first and last rows.

**Input**: the input consists of a sequence of matrix specifications. The first line of each specification will indicate which criteria will be used to find the path. ‘S’ for the standard question, ‘B1’ for bonus 1 and ‘B2’ for bonus 2. The next line will consist of the row and column dimensions in that order on a line followed by integers where m is the row dimension and n is the column dimension for the standard version. For each bonus the input will consist of the row and column dimensions followed by the start and end points in the format **row,column** The integers appear in the input in row major order, i.e., the first n integers constitute the first row of the matrix, the second n integers constitute the second row and so on. The integers on a line will be separated from other integers by one or more spaces. For each specification the number of rows will be between 1 and x inclusive; the number of columns will be between 1 and y inclusive. No path's weight will exceed integer values representable using 30 bits.

**Output.** Two lines should be output for each matrix specification in the input file, the first line represents a minimal-weight path, and the second line is the cost of a minimal path. The path consists of a sequence of (*x,y*) pairs of integers (separated by one or more spaces) representing the rows that constitute the minimal path. If there is more than one path of minimal weight, the path that is *lexicographically* smallest. For example, if the contents of four equal paths are:

*(1,1) (1,2)*

And

*(2,1) (2,2)*

And

*(1,1) (2,2)*

And

*(2,1) (1,2)*

Then the first set of points should be output.

**BONUS 1:** Rather than going from west column to the east column, extend the solution to go from any arbitrary source coordinate to any arbitrary destination coordinate, still via the shortest path. Path may proceed in any of the eight cardinal directions (N, E, W, S, NE, SE, SW, NW) Output is exactly the same as standard question.

**BONUS 2:** Extend Bonus 1 so that that the extreme west and east columns are adjacent, turning the cylinder into a torus. Again path may proceed in any of the eight cardinal directions. Output is exactly the same as bonus 1.

**Sample Standard Version Input and Output**

**Input:**

S

5 6

3 4 1 2 8 6

6 1 8 2 7 4

5 9 3 9 9 5

8 4 1 3 2 6

3 7 2 8 6 4

**Output:**

(1,1) (2,2) (3,3) (4,4) (4,5) (5,6)  
16

**Sample Bonus 1 Standard Version**

**Input:**

B1

5 6 1,1 5,6  
3 4 1 2 8 6  
6 1 8 2 7 4  
5 9 3 9 9 5  
8 4 1 3 2 6  
3 7 2 8 6 4

**Output:**

(1,1) (2,2) (3,3) (4,4) (4,5) (5,6)  
16

## Programming Problem #2: Points within a Square

### Problem

Given a set of (x,y) coordinates on a Euclidean geometric plane, and a set of four (x,y) coordinates that define a square, determine if all the points are contained within the defined square. Points that are found on the sides of the square or the corners of the square count as points contained within the square. The square is not necessarily parallel to the axes.

The use of OpenGL or any other Vector or Polygon libraries is strictly ***prohibited***.

**Input:** A set of comma separated (x,y) coordinates, which are separated by spaces.

The first four coordinates define the corners of a square (in no particular order) and the remainder of the comma separated coordinates will be the points.

For example:

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

(*Defines a square with corners (0,0)/(0,5)/(5,0)/(5,5) and three points (1,1)/(2,2)/(3,3)*)

**Output:** “True” if all the points are contained within the square or “False” if at least one point is not contained within the square.

### Example Input / Output:

**Input**:

0,0 0,1 1,1 1,0 0.5,0.5

**Output**:

True

**Input**:

0,0 0,5 5,0 5,5 1,1 2,2 3,3 9,9  
**Output**:

False

**Input**:

0,0 0,5 5,0 5,5 1,1 2,2 -1,3  
**Output**:

False

**BONUS 1:** Complete the main question, but for an arbitrary convex polygon. In this case the input will start with the number of vertices followed by the coordinates of the vertices, and the remainder will be the points to check (output is exactly the same as standard question).

For example:

**Input**:

4 0,0 0,5 5,0 5,5 1,1 2,2 3,3

**Output**:

True

**BONUS 2:** Complete bonus 1, but for any simple polygon (no holes, no crossing lines, but may be concave) To differentiate from Bonus 1, add the string “B2” to the input (output is exactly the same as bonus 1):

For example:

**Input**:

B2 5 2,0 4,0 6,7 3,5 0,7 3,6

**Output**:

False

## PROGRAMMING PROBLEM #3: Poker Chip Distribution

Problem: Write a program that calculates the optimum distribution of poker chips that maximizes the number of chips that each player receives while also making sure everyone receives the SAME breakdown of poker chips.

Input: Input will consist of 3 lines. The first line will be a comma separated list of chip quantities and denominations in the form qty/$denomination. The second line is the number of players. The third line is the buy in.

Output: The breakdown of how many of each kind of chip each player should receive. Each chip breakdown should be on a separate line in the format $denomination – number of chips.

For example:

**Input**:

50/$2.00,50/$1.00,100/$0.50,100/$0.25,100/$0.10,100/$0.05

10

$10.00

**Output**:

$2.00 - 0

$1.00 - 1

$0.50 - 10

$0.25 - 10

$0.10 - 10

$0.05 - 10

Note: The breakdown that is calculated must add up to equal the buy in amount.

**BONUS 1**: Calculate the breakdown if you require that each person receive at least one chip of each denomination. To indicate Bonus1 calculations, add a line to the beginning of the input just containing “B1”. Output is exactly as for the standard question.

**Example Input:**

B1

50/$2.00,50/$1.00,100/$0.50,100/$0.25,100/$0.10,100/$0.05

10

$10.00

**BONUS 2:** Based on purely the quantity of each type (colour) of chip inputted have your program calculate the optimal denomination to assign to each colour (from normal currency values $0.01, $0.05, $0.10, $0.25, $0.50, $1.00, $2.00, $5.00, $10.00, $20.00, $50.00, $100.00, $1000.00) and the quantity of each chip that players should receive to maximize the number of chips while adding up to the buy-in amount. Input to this question will have “B2” on the first line. The second line will be in the format “quantity/colour”. The third line will be the number of players, and the fourth line will be the buy-in. For the output, put the quantity of each colour on its own line in the format “colour - $denomination – quantity”

Example:

**Input:**

B2

50/Red,50/Blue,100/Black,100/Green,100/Yellow,100/Taupe

10

$10.00

**Output:**

Blue - $1.00 - 4

Red - $0.50 - 4

Taupe - $0.25 - 10

Yellow - $0.10 - 9

Green - $0.05 - 10

Black - $0.01 - 10