BITS N BYTES

CSA OPEN DAY Final Round

March 4, 2011

Rules

- This is the final round of BITS AND BYTES.
- This is a coding round.
- Only C and C++ are allowed.
- There are 7 problems of varying difficulty.
- Teams have to code the solutions to these problems.
- The deadline for submission of the solutions is 12pm 5th March.
- Each team will be provided with a username, with which they can login into the systems in the common lab.
- Finalists can use the computing resources provided only during the working hours.
- Details of submission of solutions will be made available during the event.
- You can use only the gcc,g++ compiler to compile your program.
- Programs with any compile error will not be evaluated.
- Each problem has a very specific output format, any program not consistent with the specified output format will not be evaluated.

1 Balancing Chemical Equations

We have all done this manually in our chemistry classes. Some of us also regret doing some silly mistakes in exams. Lets make sure that from now on there is no scope for any silly mistakes when we balance a chemical equation. Yes, we will write a program that will do this automatically for us. For example:

$$\_\_CH_4 + \_\_O_2 = \_\_CO_2 + \_\_H_2$$

Will be balanced as :

$$1CH_4 + 2O_2 = 1CO_2 + 2H_2$$

1.1 Input Format

- Input is from file.
- One chemical equation in one file.
- An equation will be input in the form of a sequence of lines.
- One line for each molecule(eg., H2O or CO2).
- Each line has the following form:

$$sign\ n\ atom_1\ count_1\ atom_2\ count_2 \ldots atom_n\ count_n$$

- There is a space after each component.
- Where *sign* is +1 if the molecule occurs on left side of the equation, -1 if it occurs on the right side of the equation.
- n is the number of atoms in the molecule. n is a positive integer.
- $atom_i$ is the i^{th} atom in the molecule. $atom_i$ consists of two characters, first of which must be alphabetic, and the second of which may be alphabetic or blank.(Like C for Carbon or MG for Magnesium). Please note there is a blank space after C.
- $count_i$ is the number of atoms of $atom_i$. $count_i$ is a positive integer.
- Like the molecule CO_2 will be written as 2 C O 2
- To mark the end of input there is a delimiting line of the form:

00 00

1.2 Example

(Unbalanced equations)

$$Mg + Mn_2O_3 = MgO + Mn$$

will be input as:
+1 1 MG 1
+1 2 MN 2 O 3
−1 2 MG 1 O 1
−1 1 MN 1

If there are m input lines the problem is to find m positive integers values $C_1, \ldots, C_m$, such that for every distinct two-character atom name appearing in the input,

$$\sum sign_i \cdot C_i \cdot count_{i,j} = 0 \tag{1}$$

where i is the line number and j is the atom number in the same line.

Let N be one of the atoms. So your program has to get the result of adding up the number of atoms of N with a +1 *sign* and subtracting the number of

atoms with -1 *sign* so as to get a total of zero. The result of balancing the equation shown in the example above is:
$C_1 = 3$
$C_2 = 1$
$C_3 = 3$
$C_4 = 2$
that is :

$$3Mg + 1Mn_2O_3 = 3MgO + 2Mn$$

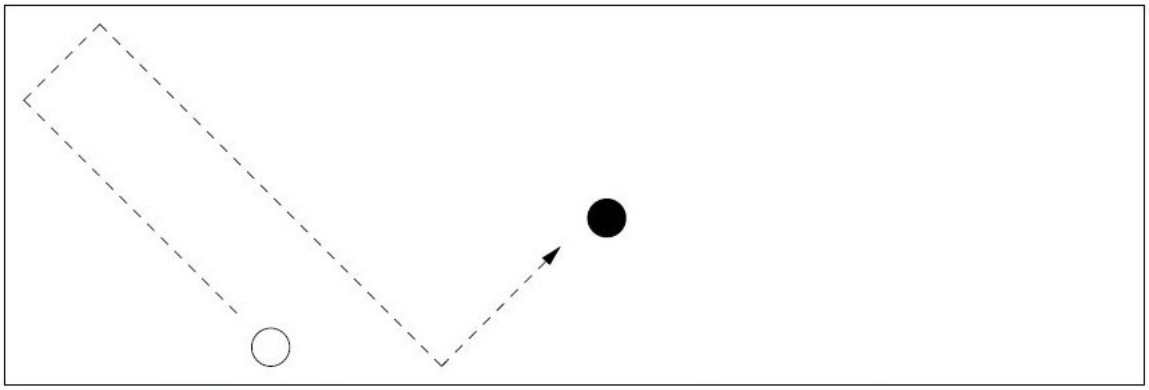
1.3 Output Format

- Output has to be in a file.
- Each C_i value must be written on the i^{th} line.
- So the output for the above example must be in a file where each C_i is separated by a new line, like this:
  3
  1
  3
  2

ASSUME THAT C_i ARE BOUNDED ABOVE BY 10.

2 Billiards

If you are already done with Chemistry lets move on to some Physics.
Assume a flat frictionless rectangular table. Two balls are kept on that table. One of the cue balls is hit. Your program has to determine whether the balls collide for the first time after the first ball has rebounded from the cushions enclosing the table exactly three times. The following figure illustrates the situation. The dashed arrow indicates the path of the center of the cue(white) ball, which banks(rebounds) three times and .strikes the black ball



The balls collide with each other when their centers are 2.0 units apart. The cue ball collides with a cushion if its center comes to within 1.0 unit of the cushion. And of course laws of reflection apply here, that is the angle at which the cue ball collides with a cushion is equal to that at which it leaves the cushion.

2.1 Input Format

- Input is from a file.
- Each of the following numbers will be separated by newline in the sequence specified.
-
  - Floating-point numbers x_c and y_c, the initial position of the center of the cue ball. The point (0,0) is what appears in the diagram as the lower-left corner.
  - Floating point numbers x_s and y_s, the position of the center of the stationary ball.
  - Floating point numbers d_x and d_y, the initial direction in which the cue ball moves. That is, the cue ball initially moves so that at time t it is at:
    $$(x_c + t \cdot d_x, y_c + t \cdot d_y)$$
    At least one of d_x and d_y must be non-zero.
  - Floating point numbers l_x and l_y, the dimensions of the table in the x and y directions, respectively.

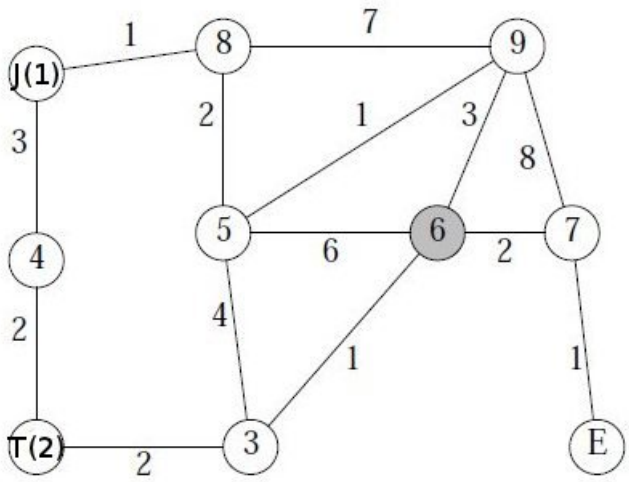
- Example input:
  14.0
  2.0
  32.5
  10.75
  -2
  2
  60
  20

2.2 Output Format

- Output must be in a file.
- if the ball collides with the stationary ball for the first time exactly after 3 bounces, then output 1.
- For any other case output 0.
- So the output in the file must be a single integer, 0 or 1 .

3 Catch Me if You Can

Now its time for some tom and jerry game.
Tom and Jerry find themselves in a maze of passages that connect numerous rooms, one of which is the maze exit. As always Tom wants to catch Jerry, and Jerry wants to escape the maze through the exit as fast as possible. But jerry runs twice as fast as Tom and fortunately for Tom, he can sense exactly where Jerry is at any time in the maze. If Tom can arrive at the exit or in any of the rooms in Jerry's path before Jerry does(strictly before, not at the same time), then he can catch Jerry. Let us assume that Jerry will always use the shortest path to exit, even if Tom is waiting on it(He is not aware of it).
Thus in the following maze Tom(starting at T) can catch Jerry in the shaded room, which he reaches in 6 time units and the Jerry in 7.
The numbers on the connecting passages indicate distances. Tom travels at 0.5 units/hour, and Jerry at 1 units/hour.



You have to write a program which reads in a maze and outputs whether Tom catches Jerry or not.

3.1 Input Format

The input file consists of the following in sequential order, separated by a new line.

- A positive integer $3 \leq N \leq 1024$ indicating number of rooms. The rooms are numbered from 0 to $N - 1$. Room 0 is the exit. Initially Jerry is in room 1 and Tom is in room 2.
- A sequence of edges. Each edge consists of two room numbers followed by an integer distance.(direction does not matter)

Example:
10
2 3 2
2 4 2

3 5 4
3 6 1
4 1 3
5 6 6
5 8 2
and so on...

3.2 Output Format

Output File must consist of 1 if Tom catches Jerry else 0.

3.3 Hint

Tom knows each passage of the maze very well, and he knows that jerry is going to take the shortest path.

4 Flip Flop

This is a famous puzzle, consisting of a five-by-five square array of push buttons, some of which are lit, and asks you to find a set of moves that turn them all off. A move consists of pushing one button, which has the effect of flipping it and each of its north, south,east and west neighbors(if any: one or two of these neighbors will be missing on the edges).Flipping means turning it on if it is off and vice versa.
You have to write a program which that reads an initial configuration of lights and computes a set of moves to turn them all OFF.
Also note that sequence of operations does not matter because flipping operations commute and performing the same flip twice is an identity operation. Therefore all we need to know is which set of lights to press, not the order.

4.1 Input Format

- The input file consists of five rows of '1's and '0 's
- There is no space between two characters in the same row.
- A 1 indicates the light is on, and 0 indicates off.
- Example input:
  01000
  11100
  01000
  00000
  00000

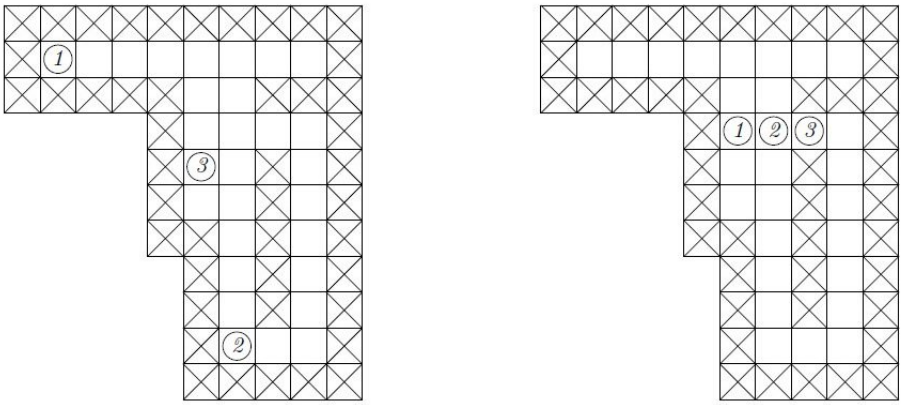
4.2 Output Format

The output file consists of five rows of '1's and '0 's. With 1 indicating that the button has to be pushed and 0 indicating button need not be pushed. Example output for the above input
00000
01000
00000
00000
00000

5 Move the Marbles

This is again a puzzle and involves moving a set of marbles around a restricted set of paths so as to achieve a certain configuration. A typical puzzle looks like this:

The problem is to move the marbles around the grid so as to convert the initial configuration on the left into the position on the right. Well the constraint is that if a marble starts to move it cannot stop until it hits a wall or another marble. So a move will consist of moving a marble in one of four possible directions(north, south, west or east) as far it will go without entering a square that is occupied. A square may be occupied by another marble or a section of wall(Shown as a square with an '**X** '). The marble must move as far as possible each time, until it collides. For example the above puzzle may be solved with the sequences of moves:



2 north, 1 east, 1 south, 3 east, 2 south, 3 south, 3 east, 3 north, 3 west.

5.1 Input Format

The input file consists of the following sequentially:

- First line consists of 3 integers: height of grid, width of grid, number of marbles
- Next the initial configuration of the grid is given using two characters ('**B**', '**X**'). Each character represents one cell.
- A hyphen('**B**') indicates an unoccupied cell, and an '**X** 'indicates a wall.
- Each line of the input file represents one row of the grid.
- Next comes the initial position of the Marbles and lastly the final position.
- Assume a coordinate system which runs from top left, at which $x = y = 0$, with x running from left to right and y from top to bottom.

- Let there be M marbles, W be the width of the grid, H be the height of the grid.
- Initial position of the marbles is represented by a pair of integers $0 \leq x_i < W$ and $0 \leq y_i < H$, $1 \leq i \leq M$ for all marbles.
- The initial configuration is followed by a new line consisting of final position of marbles in a similar fashion.
- You can assume that there are enough wall squares to prevent any marbles from moving off the grid.
- End of the input file will be specified by a demarcating new line consisting of 4 consecutive 0s.
- Example: Input file for the puzzle shown in the figure will be as follows
  10 11 3
  XXXXXXXXXX
  XBBBBBBBBX
  XXXXXBBXXX
  BBBBXBBBBX
  BBBBXBBXBX
  BBBBXBBXBX
  BBBBXXBXBX
  BBBBBXBXBX
  BBBBBXBXBX
  BBBBBXBBBX
  BBBBXXXXXX
  1 1 6 9 5 4 5 3 6 3 7 3

5.2 Output Format

Your program must output the minimum number of steps required to solve the problem in the a file.

6 Jigsaw Puzzle

We have all solved jigsaw puzzles manually, lets automate it this time.
This is a simplified version of jigsaw puzzles, you have to construct a rectangle from given pieces. The pieces are themselves made from unit rectangles glued together along their edges.

6.1 Input Format

- The first line of the file consists of 3 positive integers: W, H, P.W stands for width, H stands for height, P stands for number of pieces.
- Next comes a sequence of P piece descriptions.
  - The piece description begins with 2 positive integers W_i and H_i, the width and height of the piece.
  - The next H_i lines describe the piece.
  - Each of H_i lines, contain W_i '1's and '0's.
  - '1' represents a solid unit rectangle, and '0' represents spaces.
  - The first and last lines each contain at least one '1 ', as do first and last columns.
- Example:
  4 4 4
  2 2
  1 0
  1 1
  3 3
  1 1 1
  0 1 1
  0 0 1
  1 2
  1
  1
  4 2
  1 1 1 1
  0 0 0 1

Your Job is to align these pieces without overlapping, rotating, or flipping and must fill the space entirely to form a rectangle of dimension W by H.

6.2 Output Format

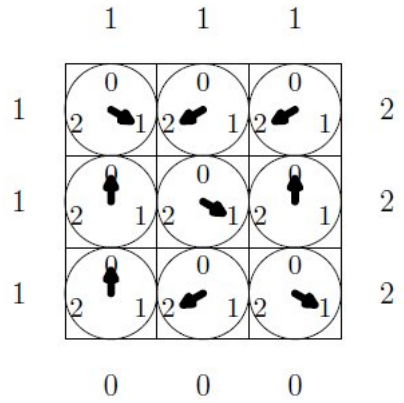
Your program should solve the jigsaw puzzle and output the final configuration of all the pieces.

- For the purpose of output you must label each of the pieces 'a'through 'z'in the order they are specified.

- For example the above example input must be labeled as follows:
  Piece a:
  a 0
  a a
  Piece b:
  b b b
  0 b b
  0 0 b
  Piece c
  c
  c
  Piece d
  d d d d
  0 0 0 d
- This labeling need not be output in the file, only the final solution needs to be printed.
- The output file must consist a solution of the puzzle in the format specified in the example.
- Example: Output for the above example input is:
  dddd
  bbbd
  abbc
  aabc
- If no solution exists then output a single number 0.

7 Egyptian Door Locks

In this problem you get to unlock the ancient Egyptian buildings. Various buried cities in Egypt have been found to use an interesting lock in storage buildings. The lock consists of a square array of three-positional dials, with a sequence of carved digits(0-2) around the periphery.



The lock is unlocked when the dials point to the number, that is equal to the sum of the numbers of its four neighbors modulo 3. Where neighbors can be either other dials or the fixed/carved numbers around the edge. The above figure illustrates this.
You have been contacted by some archaeologists, who are desperate to unlock all the rooms. You have to help then out by writing a calculator that computes the appropriate dial settings to open the lock given the fixed/carved numbers.

7.1 Input Format

- One input file will contain one lock specification.
- The specification of the lock begins with the number of rows and columns of the dials.
- So the first line contains an integer: N. N= number of rows and columns.
- Next line contains a sequence of $4N$ integers, indicating the carved numbers left-right across the top, then top-to-bottom down the right, then left-to-right across the bottom, then top-to-bottom down the left.
- All the numbers are separated by blank space.
- Examples
  - Lock 1 :The specification of the above lock:
    3
    1 1 1 2 2 2 0 0 0 1 1 1
  - Lock 2 :
    3
    0 1 2 1 2 1 2 0 1 2 1 0

```
Lock 2:
    0 1 2
-------------
2 | 0 2 2 | 1
1 | 2 2 0 | 2
0 | 2 1 0 | 1
-------------
    2 0 1
```

7.2 Output Format

The output must consist of the dial settings as displayed in the format below:

- Example 1:
  1 2 2
  0 1 0
  0 2 1
- Example 2:
  0 2 2
  2 2 0
  2 1 0