32^{nd} Annual
Rowan University
Programming Contest

hosted by the
Computer Science Department

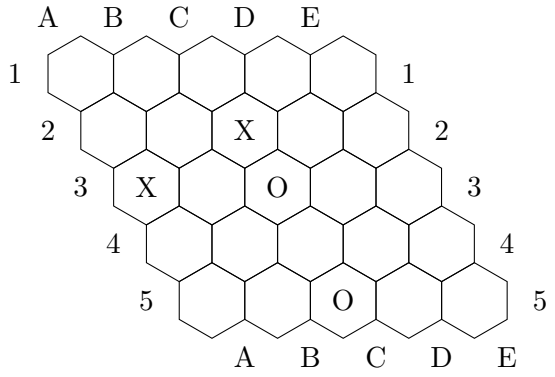
Friday, 27 April 2018

Contest Problem

1 Introduction

The game *Hex* involves a grid of hexagons in a rhomboid arrangement, with two players, for our purposes *X* and *O*, taking alternate turns placing markers in any open space, with the goal of creating a continuous path of markers connecting opposite sides. The most common size for the rhombus is 11 hexagons on a side, but other sizes are also played.

Here is a sample size 5 grid.



Player X tries to complete a path from left to right (which would connect the numbers indicating the rows) while the opponent, Player O, is trying to complete a path from top to bottom (which would connect the letters indicating the columns).

In the game as shown at left, player X has marked positions A3 and C2, while player O has marked positions C3 and C5.

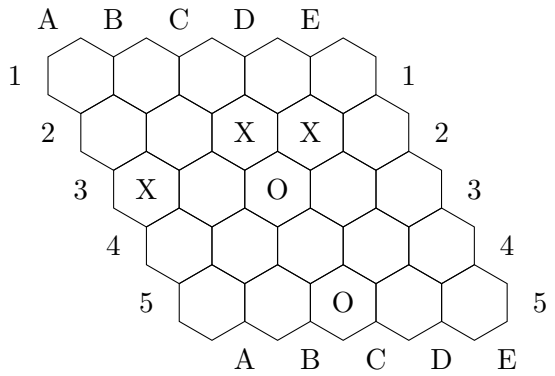
Things to notice about this position:

1. X can connect A3 and C2 in a single turn by going to either B2 or B3. If O goes to B2, X can still play B3, and vice-versa. But player O can only connect C3 and C5 in one turn by going to C4. But if X goes to C4, then O will require a longer sequence to connect them, either B4 and B5, or D3 and D4. (Note that O going to B5 makes C5 superfluous.)

2. X's best case for a victory is three moves, which can be achieved in six ways: B2-D2-E2, B3-D2-E2, B2-D2-E1, B3-D2-E1, B2-D1-E1, or B3-D1-E1.

3. O's best case for victory is also three moves, but instead of having six ways to do that, O has only two: C4-D2-D1 and C4-D2-E1.

We will follow Tic-Tac-Toe rules, so player X goes first. If X's next turn is D2, the board looks like this:

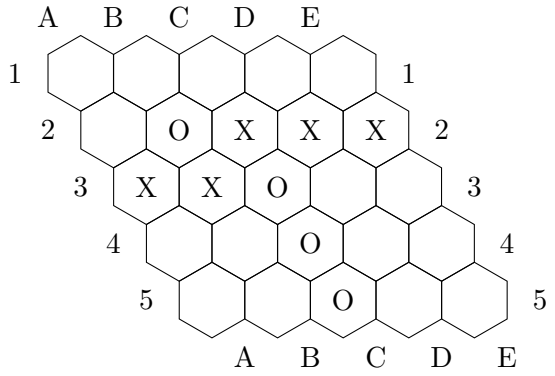


Now player O's best path to victory requires 4 moves, which can be done in three ways: B1-B2-B3-C4, C1-B2-B3-C4, and E1-E2-D3-C4.

More importantly, O can no longer stop X from winning in two turns. X needs only to connect in column B and E to win, and has two ways to make those connections.

Strategically, having more paths improves a player's chances of winning, because it is harder for the opponent to block.

Here is that same game, played to the end.



Player X has completed a path from left to right, winning the game.

The design of the game means it is impossible for a draw to happen. The only way to completely block a path from left to right would be to have a complete path from top to bottom, and vice-versa.

The challenge for this contest is to read in information about a grid in the game Hex, and print out information about the position.

2 Problem Requirements

This kind of game is known as *a zero-sum game of perfect information*, meaning that there is one winner and one loser, and that each player has full knowledge of the position at all times. (This is different from many card games, where a player does not know what cards the opponent is holding.)

When playing a game of this kind, a computer program will have a function known as a *heuristic*, which assigns a numerical value to the position. Generally, a high number means the position is good for the computer, and a low number means the position is bad for the computer.

Heuristics take into account different aspects of the board position in order to arrive at a result. A position in which the computer has won would obviously get a high value, and one in which the computer has lost would get a low value. A position in which the computer can win in one move is worth more if it is the computer's turn than if it is the opponent's turn.

For this problem, your program will run several tests on a board position in the game *Hex*.

1. Is the game over?

2. What's the minimum number of moves for each player to win?

2.1 Game over

The game is over if there is an unbroken sequence of X markers that connects the leftmost and rightmost columns, or if there is an unbroken sequence of O markers that connects the highest and lowest rows.

If the game is over, your program should print one of the messages 'X has won!' or 'O has won!'

2.2 Minimum number of moves to win

If the game is not over, your program should print a board displaying the shortest possible path for each player to win, along with the lowest number of moves needed to complete that path. In case there are several paths that give the same result, any is acceptable. (These results assume that the opponent gets no turns; this is just a count of the best theoretical possible outcome.)

3 Input

3.1 Input Specification

For text input, your program should accept input in the following format:

1. An integer, $\mathcal{D}$, where $1 \leq \mathcal{D} \leq 100$, which is the number of datasets in this file.
2. $\mathcal{D}$ data sets, each of which is in this format:
   (a) One line with one integer, $\mathcal{S}$, where $1 \leq \mathcal{S} \leq 21$, indicating the size of the rhombus. (That is, the number of rows and columns.)
   (b) One line with two integers, $\mathcal{X}$ and $\mathcal{O}$, where $0 \leq \mathcal{X}, \mathcal{O} \leq \mathcal{S}^2$, separated by a single space, which are the number of moves already made by players X and O, respectively.
   (c) One line with $\mathcal{X}$ entries, each with a letter followed by a number, listing all of the positions claimed by player X.
   (d) One line with $\mathcal{O}$ entries, each with a letter followed by a number, listing all of the positions claimed by player O.

Note: the number of spaces already claimed by the players may not be the same. A player may have been given an extra move or two at the start of the game, in order that a weaker player has a chance against a much stronger player. (Similar to the way an expert chess player might remove their queen at the start of a game against a beginner.)

3.2 Sample Input #1

Data in file	Item #	Meaning in plain English
2	1	*this file has 2 data sets*
5	2.a	*Data Set 1 has a 5x5 board*
2 2	2.b	*Data Set 1 has two spaces claimed by X and two by O*
C2 A3	2.c	*The two spaces claimed by X*
C5 C3	2.d	*The two spaces claimed by O*
5	2.a	*Data Set 2 has a 5x5 board*
3 2	2.b	*Data Set 2 has three spaces claimed by X and two by O*
C2 A3 D2	2.c	*The three spaces claimed by X*
C5 C3	2.d	*The two spaces claimed by O*

(This input, which corresponds to the board positions on page 2, is on the website as **sample1.txt**.)

A1 is always the top-left corner. All boards will always have the same number of rows and columns.

You may choose to have your program read the input from the keyboard, or ask the user for a filename and then read the file. Users of GUI-based programming environments may prefer to use text boxes into which the values can be entered, and buttons to begin their calculation. Any reasonable variation in the spirit of the problem is acceptable.

You need not do error-checking on the input. Each line will have exactly the number of items described with no stray characters or extra spaces. There will be no cases where X and O have claimed the same spot.

All sample and test data sets are available at <http://elvis.rowan.edu/rupc/2018>

4 Output

4.1 Output Specification

For each data set configuration, your program must generate output as follows:

1. The text '`Analyzing D data set(s)`', where D is the number of data sets in the input.
2. For each data set, print the following:
   (a) '`Data Set D`', where D is the number of the data set being reported on.
   (b) '`Board Position:`', followed by a text printout showing the board. Use 'X' and 'O' for hexagons claimed by a player, and '-' for unclaimed spots. There should be three spaces between each character, and each line should start indented two spaces from the line above.
   (c) Depending on the input, one of the following results:
      i. The text '`X has won!`' or '`O has won!`', if the input describes a position in which the game has already ended.
      ii. For each player, the text '`Best case for player:`' where *player* is one of 'X' or 'O', followed by a board position showing a completed path for that player, followed by the text '`Number of moves: N`', where N is the minimum number of moves needed to win.

4.2 Sample Output #1 (broken across two columns)

```
Analyzing 2 data set(s)
  Data set 1
  Board Position:
    -   -   -   -   -
      -   -   X   -   -
        X   -   O   -   -
          -   -   -   -   -
            -   -   O   -   -

    Best case for X:
    -   -   -   X   X
      -   X   X   -   -
        X   -   O   -   -
          -   -   -   -   -
            -   -   O   -   -
    Number of moves: 3

    Best case for O:
    -   -   -   O   -
      -   -   X   O   -
        X   -   O   -   -
          -   -   O   -   -
            -   -   O   -   -
    Number of moves: 3
```

```
  Data set 2
  Board Position:
    -   -   -   -   -
      -   -   X   X   -
        X   -   O   -   -
          -   -   -   -   -
            -   -   O   -   -

    Best case for X:
    -   -   -   -   X
      -   X   X   X   -
        X   -   O   -   -
          -   -   -   -   -
            -   -   O   -   -
    Number of moves: 2

    Best case for O:
    -   O   -   -   -
      -   O   X   X   -
        X   O   O   -   -
          -   -   O   -   -
            -   -   O   -   -
    Number of moves: 4
```

(This output corresponds to Sample Input 1 from page 4.)

Except for the board, your output does **not** have to duplicate the sample output exactly with respect to spacing or use of upper/lower case. The board should have three blanks between characters, each line indented two spaces more than the line above.

5 Sample #2

5.1 Sample Input #2

```
1
8
1 28
A1
B1 B2 B3 B4 B5 B6 B7 D2 D3 D4 D5 D6 D7 D8 F1 F2 F3 F4 F5 F6 F7 H2 H3 H4 H5 H6 H7 H8
```

5.2 Sample Output #2

```
Analyzing 1 data set(s)
  Data set 1
  Board Position:
    X   O   -   -   -   O   -   -
      -   O   -   O   -   O   -   O
        -   O   -   O   -   O   -   O
          -   O   -   O   -   O   -   O
            -   O   -   O   -   O   -   O
              -   O   -   O   -   O   -   O
                -   O   -   O   -   O   -   O
                  -   -   -   O   -   -   -   O

    Best case for X:
    X   O   -   X   X   O   -   X
      -   O   X   O   X   O   X   O
        -   O   X   O   X   O   X   O
          -   O   X   O   X   O   X   O
            -   O   X   O   X   O   X   O
              -   O   X   O   X   O   X   O
                -   O   X   O   X   O   X   O
                  X   X   -   O   X   X   -   O
    Number of moves: 25

    Best case for O:
    X   O   -   O   -   O   -   -
      -   O   -   O   -   O   -   O
        -   O   -   O   -   O   -   O
          -   O   -   O   -   O   -   O
            -   O   -   O   -   O   -   O
              -   O   -   O   -   O   -   O
                -   O   -   O   -   O   -   O
                  -   -   -   O   -   -   -   O
    Number of moves: 1
```

Notice that for X to win with this board layout, the player will have to make 25 moves in a row with no moves played by O. This would be unlikely in an actual game, of course. So would O having made 28 moves while X had made only 1.

6 Sample #3

6.1 Sample Input #3

```
3
6
0 0



5
5 4
C2 A3 B3 D2 E2
C5 C3 C4 B2
5
5 6
C2 A3 B5 D2 E5
C5 C3 C4 B2 B3 C1
```

Notes:

1. There are two blank lines in the first data set. Because X and Y each have zero spots already marked, there are two lines with zero spots indicated on them.
2. In the first data set, the length of the shortest path for both X and O can only be 6. There are many possible paths of length 6 which can win for either side. Your program must print out the correct length, and can display any of the possible correct paths.

6.2 Sample Output #3

```
Analyzing 3 data set(s)
  Data set 1
  Board Position:
    -   -   -   -   -   -
      -   -   -   -   -   -
        -   -   -   -   -   -
          -   -   -   -   -   -
            -   -   -   -   -   -
              -   -   -   -   -   -
    Best case for X:
    X   X   X   X   X   X
      -   -   -   -   -   -
        -   -   -   -   -   -
          -   -   -   -   -   -
            -   -   -   -   -   -
              -   -   -   -   -   -
    Number of moves:  6

    Best case for O:
    O   -   -   -   -   -
      O   -   -   -   -   -
        O   -   -   -   -   -
          O   -   -   -   -   -
            O   -   -   -   -   -
              O   -   -   -   -   -
    Number of moves:  6

  Data set 2
  Board Position:
    -   -   -   -   -
      -   O   X   X   X
        X   X   O   -   -
          -   -   O   -   -
            -   -   O   -   -
    X has won!

  Data set 3
  Board Position:
    -   -   O   -   -
      -   O   X   X   -
        X   O   O   -   -
          -   -   O   -   -
            -   X   O   -   X
    O has won!
```

7 Test Data

The contest website, <http://elvis.rowan.edu/rupc/2018>, has three test data files: **test1.txt**, **test2.txt**, and **test3.txt**.

Run your program on those files and print the results. **You must submit printed output to earn full points.** Your program will also be run on data known only to the judges.

8 Lee's Algorithm

One solution to finding optimal paths was developed by C. Y. Lee; it is similar to a 'flood fill' familiar from computer drawing programs.

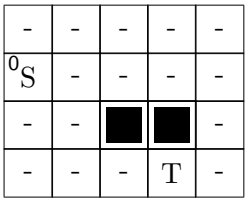
This example is rectangular, but the solution is general. (In this example, the blocked areas are marked with black squares.)

8.1 Illustration

1: Initialization

Mark the start point with zero.

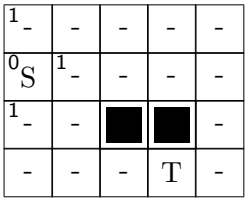
Set $i = 0$



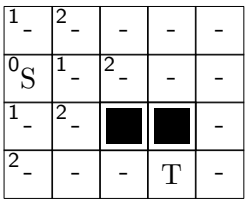
2) Wave expansion:

Until you reach the target, or have marked all points, (a) mark all neighbors of points with value i as having value *i+1*, and then (b) increment i. (If you cannot mark any more points, and have not reached the target, that means all paths are blocked.)

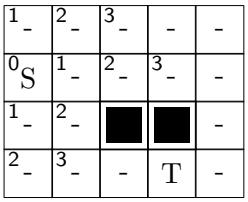
i is zero, unmarked neighbors of zeros get ones.



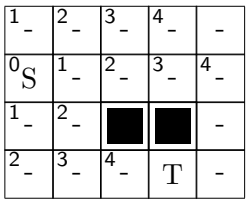
i is one, unmarked neighbors of ones get twos.



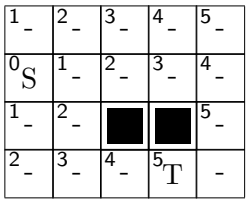
i is two, unmarked neighbors of twos get threes.



Continuing, unmarked neighbors of threes get fours:



... and unmarked neighbors of fours get fives:

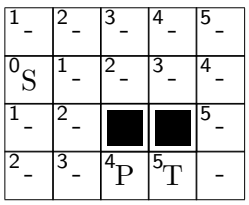


We have reached the target, and we know the minimum cost is going to be 5.

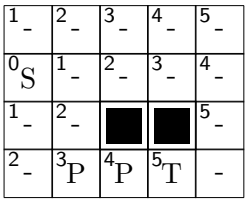
3) Backtrace

Working backwards from *Target*, until we reach *Start*, (a) look for a neighbor of the current point that has a lower value than the current point, and (b) add that node to the path and make it the new current point.

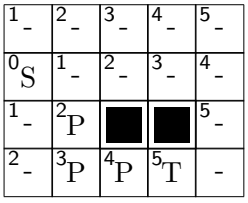
Target has value 5; there is only one neighbor with value 4, so it's added to the path (marked with 'P'):



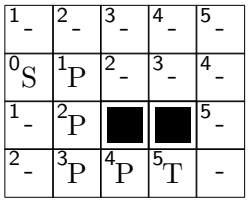
That node has only one neighbor of value 3:



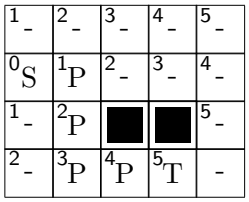
There are two choices for a 2, but which we choose does not matter:



Again there are two choices for a 1, but which we choose does not matter:



There is only one choice for zero, which is the start point.



And we have found one possible shortest path from *Start* to *Target*.

There are other possible paths, but we only need one.

8.2 Complications

1. For this problem, you do not have a single start or end point: your program only cares about the shortest path from one side of the grid to the other.

2. Some of your steps may have a cost of zero, because the player may have an X (or an O) already in the middle of the grid. This affects the wave expansion, where a position of value 2 may have a neighbor which also gets marked with 2, and also the backtrace, where you might have no neighbor with a lower value but do have a neighbor with the same value as the spot you're on.

3. Instead of having a maximum of four neighbors, each point in your grid can have up to six.

9 Notes (not needed to solve the problem)

Hex was invented by Piet Hein in 1942, and then independently re-invented by John Nash in 1948. It had several different names, but when Parker Brothers marketed a version called *Hex*, that name stuck. You can learn about Hex, work out puzzles, and read about strategy, at the Wiki page: <https://www.hexwiki.net/index.php/Main_Page>.

You can play online at several places, including <http://www.lutanho.net/play/hex.html> and <http://www.mattesmedjan.se/hexilla/>.