28th Annual Rowan University Programming Contest

hosted by the
Computer Science Department

Friday, 11 April 2014

Contest Problem

Rowan University

1 Introduction

2048 is an game in which the objective is to slide number tiles on a grid to combine them and eventually create a tile with the number 2048. The game was created by Gabriele Cirulli as a programming exercise, basing on an earlier game called *1024*, which itself has roots in an earlier game called *Threes!*. As Cirulli's version is the one created to learn programming, it's the one chosen as the basis for this year's contest.

When one of the arrow keys – up, down, left, or right – is pressed, all the tiles which can move in that direction move as far as possible. A tile can move in a given direction if either (a) the space next to it is unoccupied, or (b) there is a tile with the same number is next to it in that direction, and the tile next to it cannot move. In the second case, the tiles merge and a new tile is created with their sum.

After each a move, a new tile is added to the grid, either a 2 or a 4, in a random empty location.

If two tiles with '1024' merge, that creates '2048' and wins the game.

If a tile is added and no move is possible, the game is lost.

2 Playing Hints

There are several hints about how to play, such as:

1. Choose the move that will create the most merges, to keep as much of the board empty as possible.
2. Choose the move that will result in the highest possible value after the move is finished.
3. Choose the move that keeps the largest tile in one of the corners.
4. Choose the move which gives the 'smoothest' result (see below for explanation of smoothness).

Your challenge for this contest is to write a program which reads in the numbers on a grid for a number game like *2048*, and evaluates all 4 of the possible moves on the basis of the hints described above.

3 Details

3.1 Tile Merging

When several tiles are moving in the same row or column, the one farthest in the direction of movement will merge first. However, newly merged tiles do not then continue to merge.

If a row displays ' 0 2 2 2 ', and the user moves left, the result is ' 4 2 0 0 '. All three tiles moved left one. The leftmost tile stopped moving and the next tile merged with it. Then the last tile continued into the space available.

If a row is ' 2 4 2 2 ', and the user moves right, the result is ' 0 2 4 4 '; the 2s on the right merge to create a 4, but that newly-created 4 does *not* merge with the 4 next to it.

If a row displays ' 2 2 4 4 ', and the user moves right, the result is ' 0 0 4 8 '. The leftmost 4 merged with the rightmost 4, leaving a space available for the rightmost 2. Both 2s moved over one space, and then the rightmost 2 stopped, and the leftmost 2 merged with it.

If a row contains ' 4 4 2 0 ', and the user moves right, the result is ' 0 0 8 2 '. The 2 moves until it hits the edge of the grid, and then the rightmost 4 stops at the 2, and the other 4 merges with it.

If a row contains ' 4 0 2 2 ', and the user moves right, the result is ' 0 0 4 4 '. The 2s combine, leaving a space, and the 4 moves over next to the new 4. The newly-formed 4 does not combine with the four that moves next to it.

Similarly, if a row contains '8 8 8 8', and the user moves left, the result is ' 16 16 0 0 '. The first pair combine into a 16, and then the second pair combine into a 16. The newly-formed 16s do *not* merge.

3.2 Distance

A standard method of measuring distance on a grid is the *Manhattan Distance*, so named because the roadways in Manhattan are laid out in a rectangular grid. If you need to go two blocks north and three blocks west, then the Manhattan Distance between you and your destination is five blocks. It doesn't matter if you go north first, or west first, or zig-zag, because in the end your total distance will be the same. Both elements of a Manhattan Distance are always positive.

For example, in this grid:

```
0   0   0   2
0   0   2   8
0   4 128  32
4   8  16  64
```

The distance to the corner of 64 is zero. 32 and 16 both have a distance of one from the nearest corner. 128 has a distance of two.

Note that if there is a tie for the tile with the largest value, you report the distance of the one *closest* to a corner.

3.3 Smoothness

Smoothness has to do with whether adjacent tiles have similar values. The idea is that a grid like this:

```
0   0   0   4
0   0   2   8
0   4   4  16
4   8  16  32
```

is less likely to get stuck than one like this:

```
0   0   0  16
0   0   8   4
0   4  16   2
4  16   8  32
```

In the second grid, the isolated '2' will be difficult to combine with a new 2 that is added, because there are tiles in between it and the open area of the grid. Also, getting the tiles around the 32 up to a value of 32 will be much harder than in the previous grid, where the tiles next to 32 are both 16s.

To measure smoothness of a grid, we take every pair of adjacent nonzero tiles and count how many powers of two separate them. The total smoothness is the sum of all those differences. (Tiles in the corner have only two adjacent squares, and tiles on the side have only three.) The lower the smoothness number, the better.

So for example, a 4 next to a 16 is a difference of 2, because 16 is 4×2^2. A 32 next to a 2 gives a difference of 4, because 32 is 2×2^4. (The difference is always positive.)

The first grid above has a smoothness of 14, and the second grid has a smoothness of 22.

3.4 Complications

For the purposes of our programming problem, we will allow grids of different sizes, and we will allow different stopping points.

To avoid excess complication, all tile values will be powers of two.

4 Input

4.1 Input Specification

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

1. An integer, $\mathcal{D}$, where $1 \leq \mathcal{D} \leq 50$, 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, $\mathcal{R}$, where $1 \leq \mathcal{R} \leq 8$, which is the number of rows in this grid.
   (b) One line with one integer, $\mathcal{C}$, where $1 \leq \mathcal{C} \leq 8$, which is the number of columns in this grid.
   (c) One line with one integer, $\mathcal{W}$, where $2 \leq \mathcal{W} \leq 4096$, which is the value that wins the game.
   (d) $\mathcal{R}$ lines, each line having $\mathcal{C}$ numbers, which are the values in the grid

4.2 Sample Input #1

Data in file	Item #	Meaning in plain English
`1`	`1`	*this file has 1 data set*
`4`	`2a`	*Data Set 1 has 4 rows*
`4`	`2b`	*Data Set 1 has 4 columns*
`2048`	`2c`	*Data Set 1 is won with the value 2048*

`0 0 0 0`
`0 0 0 0`
`0 2 0 0`
`0 0 0 2`
	`2d`	*the position of the tiles on grid #1*

(This input will be on the website as **sample1.txt**.)

4.3 Sample Input #2

Data in file	Item #	Meaning in plain English
`2`	`1`	*this file has 2 data sets*
`4`	`2a`	*Data Set 1 has 4 rows*
`4`	`2b`	*Data Set 1 has 4 columns*
`2048`	`2c`	*Data Set 1 is won with the value 2048*

`0 0 0 2`
`0 0 2 8`
`0 4 128 32`
`4 8 16 64`
	`2d`	*the position of the tiles on grid #1*

`2`	`2a`	*Data Set 2 has 2 rows*
`6`	`2b`	*Data Set 2 has 6 columns*
`128`	`2c`	*Data Set 2 is won with the value 128*

`0 2 2 0 4 8`
`0 2 0 4 4 8`
	`2d`	*the position of the tiles on grid #2*

(This input will be on the website as **sample2.txt**. The first data set corresponds to the grid in §3.2.)

4.4 Sample Input #3

Data in file	Item #	Meaning in plain English
3	1	*this file has 3 data sets*
1	2a	*Data Set 1 has 1 row*
8	2b	*Data Set 1 has 8 columns*
8	2c	*Data Set 1 is won with the value 8*
2 2 2 2 2 8 2 2	2d	*the position of the tiles on grid #1*
2	2a	*Data Set 2 has 2 rows*
8	2b	*Data Set 2 has 8 columns*
16	2c	*Data Set 2 is won with the value 16*
2 2 4 4 8 8 2 2 4 4 4 4 4 4 4 4	2d	*the position of the tiles on grid #2*
3	2a	*Data Set 3 has 3 rows*
3	2b	*Data Set 3 has 3 columns*
16	2c	*Data Set 3 is won with the value 16*

2 0 4
0 2 4
2 4 8
	2d	*the position of the tiles on grid #3*

(This input will be on the website as **sample3.txt**.)

Notes:

The number of rows and columns will always be positive integers between 1 and 8.

The grid will always have at least 2 spaces. (No grid will be 1×1.)

The numbers in the grid will always be non-negative integers which are powers of 2, or zero.

The win value will always be a power of 2.

There will be nothing but integers on the input.

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. There will be no blank lines.

5 Output

5.1 Output Specification

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

1. The text '`Analyzing D data sets`', where D is the number of data sets in the input.
2. For each data set:
   (a) The text '`Data Set S:`', where S is the number of the data set being reported on.
   (b) The text '`Grid Size:  R x C`', where R is the number of rows and C is the number of columns.
   (c) The text '`Win value:  W`', where W is the value needed to win for this grid.
   (d) One of three possibilities:
      i. If the game has a tile with a value of equal to the grid's win value, report that the game is won and do not print anything else for that data set.
      ii. If the game is not won, then for each of the possible moves – Left, Right, Up, Down – print either a note that the move is illegal (because nothing will move), or the resulting grid, followed by four lines of output, one for each of the hints described in section 2. In the event this move wins the game, print a message saying so.
      iii. If there are no legal moves remaining – because there are no empty squares and no tiles able to merge – print that the game is lost.

5.2 Sample Output 1

```
Analyzing 1 data set(s)
Data Set 1:
   Grid Size: 4 x 4
   Win Value: 2048

   Left:                                           Up:
      0 0 0 0                                          0 2 0 2
      0 0 0 0                                          0 0 0 0
      2 0 0 0                                          0 0 0 0
      2 0 0 0                                          0 0 0 0
      merges: 0                                        merges: 0
      largest value: 2                                 largest value: 2
      distance to corner of largest: 0                 distance to corner of largest: 0
      smoothness: 0                                    smoothness: 0
  Right:                                           Down:
      0 0 0 0                                          0 0 0 0
      0 0 0 0                                          0 0 0 0
      0 0 0 2                                          0 0 0 0
      0 0 0 2                                          0 2 0 2
      merges: 0                                        merges: 0
      largest value: 2                                 largest value: 2
      distance to corner of largest: 0                 distance to corner of largest: 0
      smoothness: 0                                    smoothness: 0
```

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

Your output **does not** have to duplicate the sample output as regards spacing or use of upper/lower case. Your output should be neat, but need not exactly match the sample.

5.3 Sample Output 2

```
Analyzing 2 data set(s)
Data Set 1:
  Grid Size: 4 x 4
  Win Value: 2048

  Left:
        2    0    0    0
        2    8    0    0
        4  128   32    0
        4    8   16   64
    merges: 0
    largest value: 128
    distance to corner of largest: 2
    smoothness: 23
  Right:
    not a legal move
  Up:
        4    4    2    2
        0    8  128    8
        0    0   16   32
        0    0    0   64
    merges: 0
    largest value: 128
    distance to corner of largest: 2
    smoothness: 25
  Down:
    not a legal move
Data Set 2:
  Grid Size: 2 x 6
  Win Value: 128

  Left:
        4    4    8    0    0    0
        2    8    8    0    0    0
    merges: 2
    largest value: 8
    distance to corner of largest: 1
    smoothness: 5
  Right:
        0    0    0    4    4    8
        0    0    0    2    8    8
    merges: 2
    largest value: 8
    distance to corner of largest: 0
    smoothness: 5
  Up:
        0    4    2    4    8   16
        0    0    0    0    0    0
    merges: 3
    largest value: 16
    distance to corner of largest: 0
    smoothness: 4
  Down:
        0    0    0    0    0    0
        0    4    2    4    8   16
    merges: 3
    largest value: 16
    distance to corner of largest: 0
    smoothness: 4
```

(These correspond to Sample Inputs 2 and 3.)

5.4 Sample Output 3

```
Analyzing 3 data set(s)
Data Set 1:
  Grid Size: 1 x 8
  Win Value: 8

    The game is won!
Data Set 2:
  Grid Size: 2 x 8
  Win Value: 16

  Left:
        4    8   16    4    0    0    0    0
        8    8    8    8    0    0    0    0
    merges: 8
    largest value: 16
    distance to corner of largest: 2
    smoothness: 7
    This move wins the game!
  Right:
        0    0    0    0    4    8   16    4
        0    0    0    0    8    8    8    8
    merges: 8
    largest value: 16
    distance to corner of largest: 1
    smoothness: 7
    This move wins the game!
  Up:
        2    2    8    8    8    8    2    2
        4    4    0    0    4    4    4    4
    merges: 2
    largest value: 8
    distance to corner of largest: 2
    smoothness: 10
  Down:
        2    2    0    0    8    8    2    2
        4    4    8    8    4    4    4    4
    merges: 2
    largest value: 8
    distance to corner of largest: 2
    smoothness: 10
Data Set 3:
  Grid Size: 3 x 3
  Win Value: 16

  Left:
        2    4    0
        2    4    0
        2    4    8
    merges: 0
    largest value: 8
    distance to corner of largest: 0
    smoothness: 4
  Right:
        0    2    4
        0    2    4
        2    4    8
    merges: 0
    largest value: 8
    distance to corner of largest: 0
    smoothness: 6
  Up:
        4    2    8
        0    4    8
        0    0    0
    merges: 2
    largest value: 8
    distance to corner of largest: 0
    smoothness: 5
  Down:
        0    0    0
        0    2    8
        4    4    8
    merges: 2
    largest value: 8
    distance to corner of largest: 0
    smoothness: 4
```

6 Test Data

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

6.1 Test Input #1

```
8
1
2
4
2 2
5
5
64
0 2 0 4 4
32 0 16 0 16
4 4 0 4 4
2 2 4 4 8
16 32 0 32 16
3
6
32
2 0 2 0 2 0
0 4 0 4 0 4
2 2 2 2 2 2
1
8
8
2 2 2 2 2 8 2 2
2
8
16
2 2 4 4 8 8 2 2
4 4 4 4 4 4 4 4
3
1
4
2
0
2
3
8
16
2 2 2 2 2 2 2 2
4 4 4 0 0 4 4 4
8 0 0 0 0 0 0 8
8
8
512
2 2 4 4 8 8 16 16
2 4 4 8 8 16 16 32
4 4 8 8 16 16 32 32
4 8 8 16 16 32 32 64
8 8 16 16 32 32 64 64
8 16 16 32 32 64 64 128
16 16 32 32 64 64 128 128
16 32 32 64 64 128 128 256
```

6.2 Test Input #2

```
7
2
2
4
2 2
2 2
5
5
128
0 2 0 4 4
32 0 16 0 16
4 4 0 4 4
2 2 4 4 8
16 32 0 32 16
3
6
16
2 0 2 0 2 0
4 4 0 4 0 4
2 2 2 2 2 2
3
8
16
2 2 2 2 2 8 2 2
2 2 4 4 8 8 2 2
4 4 4 4 4 4 4 4
3
8
32
2 8 4 4 4 4 16 4
4 4 8 8 8 16 4 8
2 8 8 8 8 16 4 8
4
8
32
2 2 2 2 2 2 2 2
4 4 4 0 0 4 4 4
8 8 0 0 0 0 8 8
16 0 0 0 0 0 0 16
8
8
512
4 4 8 8 16 16 32 32
4 8 8 16 16 32 32 64
8 8 16 16 32 32 64 64
8 16 16 32 32 64 64 128
16 16 32 32 64 64 128 128
16 32 32 64 64 128 128 256
32 32 64 64 128 128 256 256
32 64 64 128 128 256 256 0
```

All sample and test data sets are available at <http://elvis.rowan.edu/rupc/2014>