20th Annual Rowan University Programming Contest

hosted by the
Computer Science Department

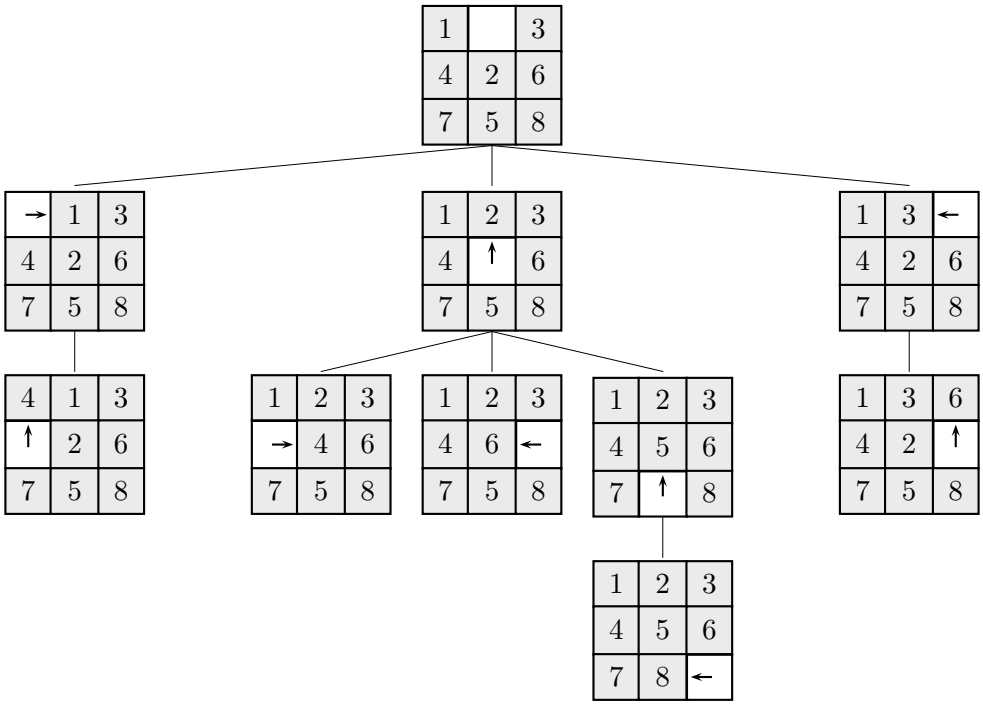
Friday, 28 April 2006

Contest Problem

1 Background

Since the earliest days of computing, computers have been programmed to play games, solve problems, and do other activities which require intelligence when done by human beings.

Many algorithms employed in such programs involve *tree search*, in which the computer plays out the next several moves, generating many possible outcomes, and then chooses the path which leads to the best result. For a standard 8-tile slide puzzle, a portion of the tree may look like this:



The root of this tree, at the top, represents our starting configuration. The 3 diagrams in the middle row represent the results, respectively, of moving the 1 right, the 2 up, and the 3 left. The next 5 diagrams represent the possible moves following those, except for moves which return to the start configuration. The diagram on the lower left represents moving the 1 right and then the 4 up. The diagram on the lower right represents moving the 3 left and then the 6 up. The fourth row diagrams have been skipped, except for the one that represents solving the puzzle.

In some cases, the complete tree is so big that listing all possible outcomes is intractable. In such cases, the program generates only a portion of the tree (the next four moves, for example), and then evaluates those intermediate arrangements of pieces. This evaluation is done by a function called a *heuristic*. When the best possible outcome several moves away has been identified, the program makes those moves, and then looks a few more moves into the future to find its next goal.

Design of heuristics is a continuing area of research. This problem will consider one heuristic for a slide puzzle. The overall notion is that the farther from 'home' each tile is, the farther the puzzle is from being solved, and the higher the value of the heuristic.

The heuristic for this problem involves something called 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.

In this present case, the heuristic will be to find the Manhattan Distance between where each tile is and where it goes. Add up the distances for each tile. Do not include the blank space.

Your task in this programming contest is to read the size (in rows and columns) of a slide puzzle, read the positions of the tiles, compute the heuristic value of that configuration and all the configurations which can be reached by moving only one tile.

2 Input/Output Specification

2.1 Input

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

1. A single integer, $\mathcal{N}$, where $\mathcal{N} \geq 1$, which specifies the number of configurations to be checked.
2. $\mathcal{N}$ data sets, each of which is in this format:
   (a) A single integer, $\mathcal{R}$, where $1 \leq \mathcal{R} \leq 5$, which specifies the number of rows in this puzzle.
   (b) A single integer, $\mathcal{C}$, where $1 \leq \mathcal{C} \leq 5$, which specifies the number of columns in this puzzle.
   (c) $\mathcal{R}$ rows of $\mathcal{C}$ numbers, separated by spaces, indicating the location of each tile. The value `0` (zero) will stand in for the blank square.

You need not do error-checking on the input. There will be nothing on the first line except a single integer. For each data set, there will be nothing on the first line except two integers, and nothing on the following lines but numbers and spaces.

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.

All puzzles are solved when they read from 1 to $\mathcal{R} \times \mathcal{C} - 1$ in order left-to-right, top-to-bottom, with the blank space at the bottom right. These five puzzles are solved:

```
 1  2  3          1  2  3  4          1  2  3          1  2  3  4  5
 4  5  6          5  6  7  8          4  5  6          6  7  8  9 10          1
 7  8  0          9 10 11 12          7  8  9         11 12 13 14 15          2
                 13 14 15  0         10 11  0         16 17 18 19 20          0
                                                      21 22 23 24  0
```

2.2 Output

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

1. The text '`Configuration C:`', where **C** is the number of the configuration being reported on.
2. The current configuration printed out as a grid.
3. The current configuration's heuristic value.
4. Output for each of four directions a tile can be slid, in the order Up, Down, Left, Right. The output should include:
   (a) The words '`slide tile`', followed by the direction the tile is being slid.
   (b) Either the word 'impossible', if no tile can be slid in the specified direction, or the puzzle configuration which results from sliding the tile in that direction.
   (c) If a move is possible, the heuristic value of the configuration which results.

Your output should match the sample output as closely as possible as regards formatting, but need not match exactly as regards use of upper/lower case.

3 Sample Data

3.1 Sample Input #1

```
2                    this file has 2 data sets
3 3                  board #1 is 3 by 3
1 2 3 }
4 5 6 }              the 8 tiles of
7 0 8 }              board #1
4 4                  board #2 is 4 by 4
 1  2  3  4 }
 5  6  7  8 }         the 15 tiles of
 9 10 11 12 }         board #2
13 14  0 15 }
```

(This input file is available on the web at: <http://www.rowan.edu/hspc/2006/sample1.txt>.)

3.2 Sample Output #1

```
Configuration 1:
    current configuration:
         1  2  3
         4  5  6
         7  0  8
    value: 1

    slide tile up: impossible

    slide tile down:
         1  2  3
         4  0  6
         7  5  8
    value: 2

    slide tile left:
         1  2  3
         4  5  6
         7  8  0
    value: 0

    slide tile right:
         1  2  3
         4  5  6
         0  7  8
    value: 2

Configuration 2:
    current configuration:
         1  2  3  4
         5  6  7  8
         9 10 11 12
        13 14  0 15
    value: 1

    slide tile up: impossible

    slide tile down:
         1  2  3  4
         5  6  7  8
         9 10  0 12
        13 14 11 15
    value: 2

    slide tile left:
         1  2  3  4
         5  6  7  8
         9 10 11 12
        13 14 15  0
    value: 0

    slide tile right:
         1  2  3  4
         5  6  7  8
         9 10 11 12
        13  0 14 15
    value: 2
```

3.3 Sample Input #2

```
3                                  this file has 3 data sets
5 5                                board #1 is 5 by 5
 0  1  2  3  4  }
 6  7  8  9  5  }
11 12 13 14 10  } the 24 tiles of
16 17 18 19 15  } board #1
21 22 23 24 20  }
2 2                                board #2 is 2 by 2
1  0  }
3  2  }                           the 3 tiles of board #2
1 5                                board #3 is 1 by 5
1 2 0 3 4 }                        the 4 tiles of board #2
```

(This input file is available on the web at: <http://www.rowan.edu/hspc/2006/sample2.txt>.)

3.4 Sample Output #2

(Appears in next column.)

```
Configuration 1:
    current configuration:
         0  1  2  3  4
         6  7  8  9  5
        11 12 13 14 10
        16 17 18 19 15
        21 22 23 24 20
    value: 8

    slide tile up:
         6  1  2  3  4
         0  7  8  9  5
        11 12 13 14 10
        16 17 18 19 15
        21 22 23 24 20
    value: 9

    slide tile down: impossible

    slide tile left:
         1  0  2  3  4
         6  7  8  9  5
        11 12 13 14 10
        16 17 18 19 15
        21 22 23 24 20
    value: 7

    slide tile right: impossible

Configuration 2:
    current configuration:
         1  0
         3  2
    value: 1

    slide tile up:
         1  2
         3  0
    value: 0

    slide tile down: impossible

    slide tile left: impossible

    slide tile right:
         0  1
         3  2
    value: 2

Configuration 3:
    current configuration:
        1 2 0 3 4
    value: 2

    slide tile up: impossible

    slide tile down: impossible

    slide tile left:
        1 2 3 0 4
    value: 1

    slide tile right:
        1 0 2 3 4
    value: 3
```

3.5 Sample Input #3

```
2                 this file has 2 data sets
4 4               board #1 is 4 by 4
11 10  6  4  }
 3  0 15  5  } the 15 tiles of
13  8  2  9  } board #1
 7 14 12  1  }
2 3               board #2 is 2 by 3
1  2  3  }
0  4  5  } the 5 tiles of board #3
```

(This input file is available on the web at: <http://www.rowan.edu/hspc/2006/sample3.txt>.)

3.6 Sample Output #3

```
Configuration 1:
    current configuration:
        11 10  6  4
         3  0 15  5
        13  8  2  9
         7 14 12  1
    value: 38

    slide tile up:
        11 10  6  4
         3  8 15  5
        13  0  2  9
         7 14 12  1
    value: 37

    slide tile down:
        11  0  6  4
         3 10 15  5
        13  8  2  9
         7 14 12  1
    value: 37

    slide tile left:
        11 10  6  4
         3 15  0  5
        13  8  2  9
         7 14 12  1
    value: 39

    slide tile right:
        11 10  6  4
         0  3 15  5
        13  8  2  9
         7 14 12  1
    value: 37

Configuration 2:
    current configuration:
         1  2  3
         0  4  5
    value: 2

    slide tile up: impossible

    slide tile down:
         0  2  3
         1  4  5
    value: 3

    slide tile left:
         1  2  3
         4  0  5
    value: 1

    slide tile right: impossible
```

4 Test Data

Run your program on this input and make screenshots showing the output. Your program will also be tested on data known only to the judges.

```
9
1 2
0 1
2 1
1
0
2 2
2 3
1 0
3 3
0 8 7
6 5 4
3 2 1
4 4
11 15  6 12
 3  8  9  5
 7  0 13  2
 4 14  1 10
5 5
 6  1  2  3  4
11 12  7  8  5
16 17  0  9 10
21 18 13 14 15
22 23 24 19 20
4 4
 3  5  0  6
10 11 14  8
13  7  4 15
 9  1  2 12
3 3
5 4 2
0 8 3
1 7 6
4 4
 4 13  8  6
 5  0  3 14
 1 11 15 10
 2 12  7  9
```

(This input file is available on-line at: <http://www.rowan.edu/hspc/2006/testdata.txt>.)

5 Notes (not required to solve the problem)

An important element of problem solving, beyond the scope of today's contest, has to do with undoing progress in order to advance. For example, this puzzle configuration:

1	2	3
4	5	7
	8	6

has a value (per our heuristic) of 4. Both of the configurations which can be reached from this one,

1	2	3
↓	5	7
4	8	6

and

1	2	3
4	5	7
8	←	6

have heuristic values of 5, which is higher. But in order to get the puzzle closer to its solved state, we have to make one of those two moves. After making that move, we can't just go to the 'next move with lowest value', because we'll be back to where we started, and we'll get stuck in an infinite loop moving the same tile back and forth.

One standard algorithm for solving puzzles of this type is called A* (pronounced 'A-star'), which tracks previous configurations to avoid infinite cycles, and which allows a program to undo progress so that it will eventually solve the puzzle. Working out the ten steps to solve the puzzle above, and the heuristic value of each configuration on the path may prove a worthwhile exercise. (Hint: the heuristic gets as high as six, drops to four, and then goes back up to five before finally dropping to zero as the puzzle is solved.)

An alternative version of the 8-puzzle has this as its solved configuration:

1	2	3
8		4
7	6	5

This variation, which is **not** the considered solved for today's problem, can be used for puzzles with pictures of circular items (such as Christmas wreaths) on the tiles.

A heuristic variation for this puzzle is to set each tile's value to the Manhattan Distance from its correct position *plus* a value from a *sequence function* described by Nils Nilsson in *Principles of Artificial Intelligence* (Morgan Kaufmann, 1980). The sequence function takes into account that this puzzle's tiles are numerically sorted along the edge of the puzzle. If they're in the right order, but out of place, the puzzle is easier to solve than if they're totally scrambled.

To find the value of the sequence function, you go clockwise around the outside of the puzzle, and check each tile to see if the tile 'ahead' of it is the right one (either one value higher, or 1 if the current tile is 8). If it's the right one, you add zero to the heuristic. If it's not the right one, you add 2. If the center is blank, you add zero; else you add one.

As with all puzzles, the better the heuristic, the smarter the program will be, and the more quickly it can find the optimal solution path.

Id: hspc-2006-prob.tex,v 1.9 2006/04/28 21:21:45 kilroy Exp kilroy