CS 61B Data Structures and Programming Methodology

July 31, 2008

David Sun

Announcements

- Midterm II results available on glookup.
  - Mean 37.3 / 50
  - Std Dev: 9
- Midterm II papers will be returned to you during discussion sections.
- Same re-grading policies as last time.
  - Write a note and tell us what you think was incorrectly graded. Hand it to me or one of the t.a.s. Check you solution against the solution online to make sure you won't loose more points than you'd gain.
  - Best to get it to us before the end of next week.

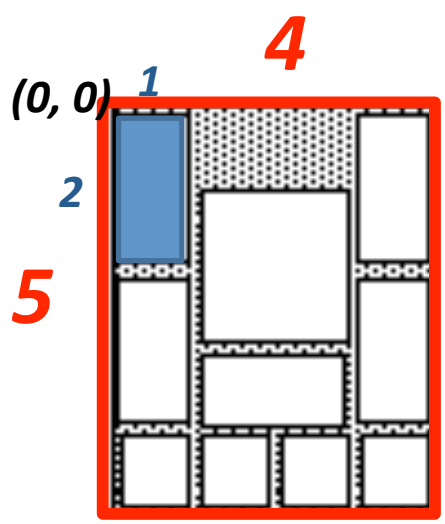
Project 3

- Sliding block puzzle:
  - A tray stores of a number of rectangular blocks, the game is to slide the pieces without lifting any out of the tray, until achieving a certain configuration.
- Write a program that solves a sliding block puzzle in as little *execution time* as possible.
  - Need not find the shortest possible sequence of move.

Initial Configuration

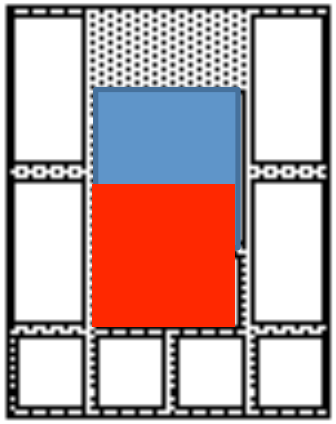
- Input: a file containing an *encoding* of the *initial* tray configuration.

```
5 4
2 1 0 0
2 1 0 3
2 1 2 0
2 1 2 3
2 2 1 1
1 2 3 1
1 1 4 0
1 1 4 1
1 1 4 2
1 1 4 3
```

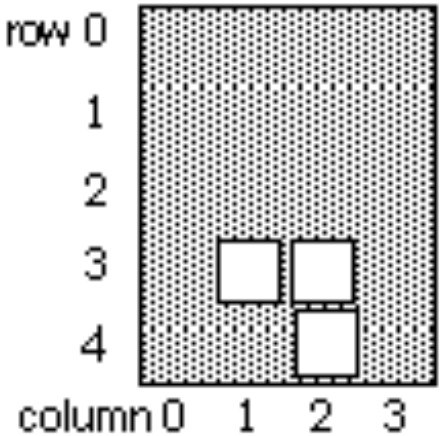


Goal Configuration

- Input: a file that specifies an *encoding* of the final or *goal* configuration.
  - This file does not contain entries for all blocks in the tray.
  - Blocks may appear in any order in this file.



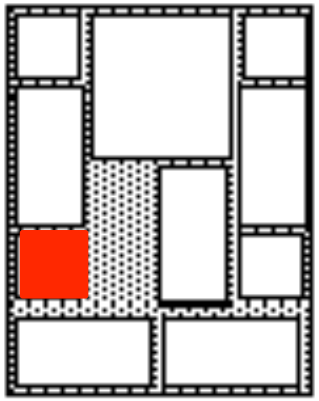
2 2 3 1



1 1 3 1
1 1 4 2
1 1 3 2

Solution

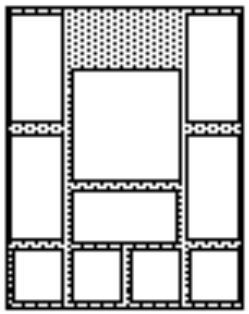
- A sequence of moves of blocks that turns the initial configuration to the in the goal.
- Legal moves:
  - Slide block horizontally or vertically into adjacent empty space.



  - Blocks may only be moved an integer number of spaces.
  - Either the row or the column will be the same in the start position as in the end position for each move.

Format of Solution

- *Sequence* of block movements that lead to a solution.
- Each such line will contain four integers: the starting row and column of the upper left corner of the moving block, followed by the upper left corner's updated coordinates.



Slide blocks until the 2x2 block is in the middle of the bottom row.

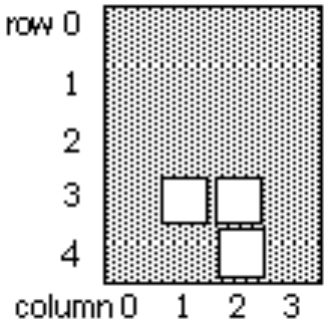


Figure 1

1 1 0 1 move the 2x2 block up
3 1 2 1 move the 1x2 block up
4 1 3 1 move a 1x1 block up
4 2 3 2 move another 1x1 block up
4 0 4 2 move the leftmost 1x1 block two spaces over

The Game

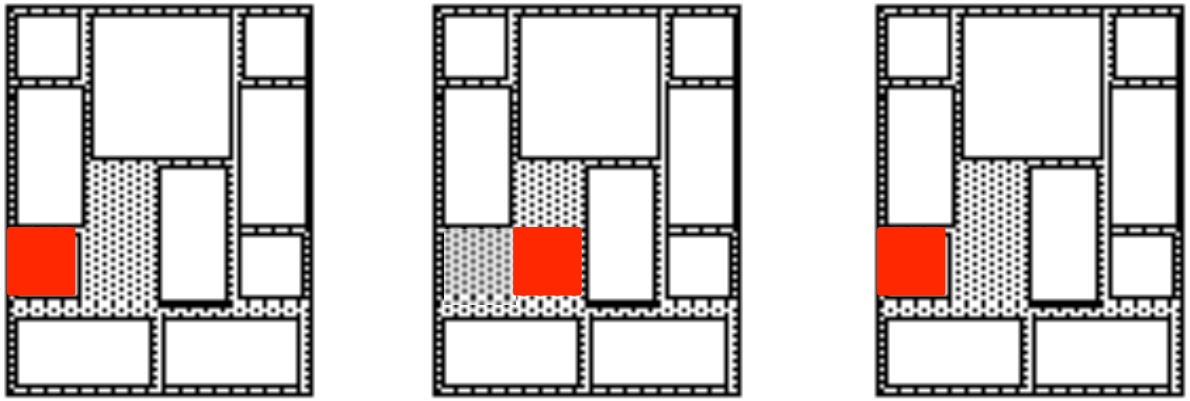
- Your program will search the tree of possible move sequences to find a solution to the puzzle.
- Top level algorithm:
  0. *Initialization – read in the init and goal configurations.*
  1. *Generate moves from the current configuration*
  2. *Pick a move*
  3. *Update the current configuration*
  4. *Continue if the current configuration does not match the end configuration*

Key Design Designs

- Should I traverse the possible configurations breadth first or depth first?
  - Either way, you'll likely hit some resource limit.
- DFS:
  - If you find a solution you don't know whether it is the best solution or not.
  - You can run into a very deep branch in the search tree before hitting the solution or the stack may over flow
  - Can try bounding the search at some fixed depth, and do iterative deepening if solution doesn't exist.
- BFS
  - First solution is the most optimal solution, i.e. minimal number of steps
  - If there are many empty spaces in a configuration, the set of possible moves from any given configuration can be huge. Your search tree will be very wide, and the queue will run out of memory before you hit a solution.

Preventing Cycles

- How do I prevent cycles?
  - Some way of remembering the search path I've taken or the configurations I've seen.
  - Maybe use a hash tables – O(1) lookup time if designed well.
  - Not just look up time though, how long does it take to compute the hash code?



Key Data Structure

- Tray: representation of the current configuration of the board.
  - How do I represent the blocks?
  - How do I represent the empty spaces?
  - Should I represent the empty spaces explicitly?
  - Should I represent the blocks explicitly?
- Tray representation will affect:
  - *How you read in the configuration files*
  - *How you find the moves to generate from the current configuration*
  - *How you update a configurations.*
  - *How you hash the configurations.*
  - *How you compare configurations.*
  - *How long you can grow the stack or queue before the it explodes.*

One Design

```
class Tray {
  //unsorted, added in the order specified in the
  //init config file
  ArrayList<Blocks> config;
  . . .
  class Blocks {
      int row;
      int col;
      int length;
      int width;
      . . .
  }
```

What are some of the issues with this design?

Implementation Advice

- Think carefully about how the tray should be represented but do not dwell on it for too long.
- Get the brute-force (exhaustive search) working on the easy puzzles before start tuning.
- Don't waste your time on finding an *optimal* solution. Test your algorithm on the input and output files and see if it works reasonably well!
- Time the modules in your code to find where the bottle necks are!
  - May yield surprises…
  - There are many timing facilities in Java that let's you do this.

Main Modules?

Testing

- Checker program that checks whether a given sequence of moves solves a given puzzle.
  - Takes an initial configuration and a goal configuration in the same format as those for Solver.java.
  - Takes a sequence of moves, in the format to be produced by Solver.java, as standard input.
  - Its output indicates whether the moves solve the puzzle, and if not, why not.
- 3 sets of testing files:
  - Easy: 50 points if you solve almost all of them.
  - Medium: get no points for solving these.
  - Hard: 2 points for each hard puzzle you solve under 2 minutes.

Debug Info

- An optional first argument will be a string whose first two characters are "–o" and whose remaining characters specify information about what debugging output the program should produce.
- You may choose the format of this information.
- You may choose any debugging information to output
  - Doesn’t need to a whole lot
  - Demonstrates to us that you used some form of debugging output to help you verifying the result.

Useful Debugging Info?

HW for next Monday

- Earn up to 8 points,

  8 Completion of 3 of the tasks listed below, with evidence that they work correctly.

  7 Completion of 2 of the tasks listed below, with evidence that it works correctly, plus significant progress on one of the other tasks.

  6 Completion of 2 of the tasks listed below

  5 Completion of 1 of the tasks listed below, with evidence that it works correctly, plus significant progress on one of the other tasks.

  4 Completion of 1 of the tasks listed below

  2 No working code, but with one of the tasks below almost implemented.

  0 No apparent progress on the project.

- The tasks:
  - input of blocks from a file into a tray data structure
  - comparison of a tray with a goal configuration
  - generating moves from a given configuration
  - making a move
  - implementation of a good hashing scheme