# CSCI-1200 Data Structures — Fall 2016
# Homework 6 — Paint by Pairs Recursion

In this homework we will solve a grid-based black & white image puzzle game by Conceptis Puzzles that has appeared in Games magazine. This puzzle game is named "Paint by Pairs" or "B&W Link-a-Pix". You can play online versions of this game here:

http://www.gamesmagazine-online.com/gameslinks/lap_puzzle.html
http://www.conceptispuzzles.com/index.aspx?uri=puzzle/link-a-pix

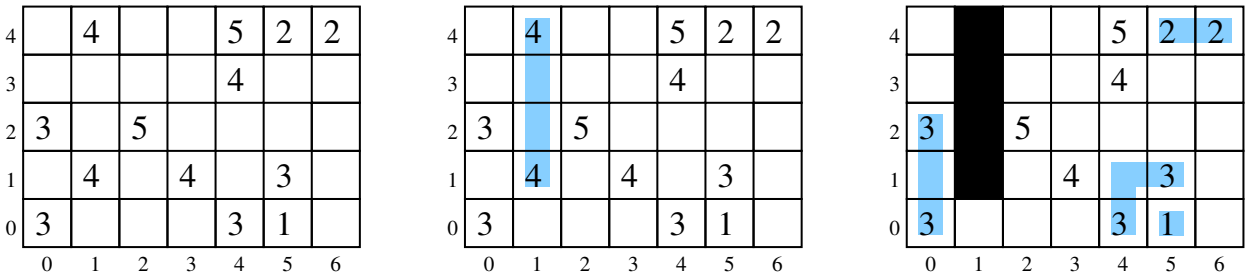
This is a popular game with lots of material available online. You may not search for, study, or use any code related to the solvers for this puzzle game. *Please carefully read the entire assignment and study the examples before beginning your implementation.*

## Paint by Pairs Puzzles - How to Play

Your program will accept one or two command line arguments. The first argument is the name of a paint by pairs puzzle board file similar to the file shown on the right.

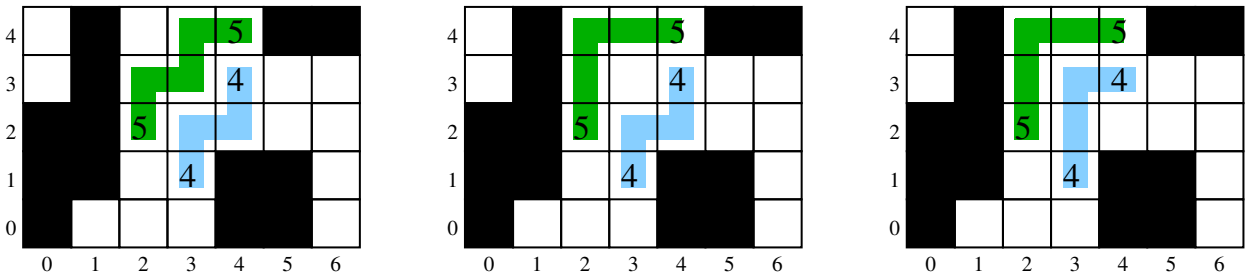
```
width 7
height 5
1 4 4
4 4 5
5 4 2
6 4 2
4 3 4
0 2 3
2 2 5
1 1 4
3 1 4
5 1 3
0 0 3
4 0 3
5 0 1
```

This sample file begins with the dimensions of the board. Next, we assign numbers to some of the cells in the board. Each assignment begins with the $x$ and $y$ coordinates of the cell. *Note: cell (0,0) is the lower left hand corner of our image.* The assignment is an integer length >= 1, which indicates this cell must be the endpoint of a painting stroke of the specified length. These cell assignments lines may appear in any order in the input file. The same sample puzzle board is shown in the left image below.



So let's start to play the game. Starting with the '4' in the top row, let's find a stroke of length 4 connecting this cell to another cell labeled with '4' in the grid. We find just one such stroke, highlighted in blue in the middle image above. Note that strokes must step one cell at a time vertically or horizontally (they may not move diagonally in a single step).

Once the stroke is confirmed, we color those squares black. Because strokes may not cross, those squares are not available for use by any other stroke. In the right image we add 4 more strokes. Note that a cell labeled '1' is a single pixel stroke and is the only stroke endpoint that doesn't need to find another cell with matching length value to pair up with.



As shown in the images above, at this point we have multiple options. If we draw the jagged length 4 blue stroke, we have two choices for which green length 5 stroke we draw. If instead we choose the 'L' shaped blue stroke of length 4, we have only one choice for the final green length 5 stroke. Thus, this puzzle has 3 unique final solutions.

Your task for the core of this homework is to find one of these solutions (if it exists). And for full credit your program should find all solutions (if requested).

### Command Line Arguments and Output Formatting

Your program should accept one or two command line arguments. The first is the name of the input puzzle file. If the second argument is not provided, your program should find a solution to the puzzle (any solution) and print that solution to `std::cout`. If provided, the second argument will be the string `find_all_solutions`, and your program should find all solutions.

A sample full credit output to the puzzle above is shown on the right. First, the output prints the original puzzle board. For numbers greater than or equal to 10, we switch to capital letters. Then we print the string "`Solution:`", followed by the path of coordinates of each stroke, each on its own line. The strokes may appear in any order (this sample output has them going from short to long, but that is not required). Similarly, the path sequence for a stroke may start at either endpoint. After the strokes, we display a framed ASCII art version of the final image using 'X' for the painted grid cells.

If the second argument `find_all_solutions` is specified, your program should output all valid, unique solutions (in any order) and then also print at the bottom the number of solutions found, e.g., "`Found 3 solution(s)`". If the puzzle has no solutions, your program should print "`No solutions`". Note that we define solution uniqueness by the details of the strokes. It is possible that two different sets of strokes can produce the same final B&W image. We count this as two different solutions.

```
+-------+
| 4  522|
|    4  |
|3 5    |
| 4 4 3 |
|3   31 |
+-------+

Solution:
(5,0)
(5,4)(6,4)
(0,0)(0,1)(0,2)
(4,0)(4,1)(5,1)
(1,1)(1,2)(1,3)(1,4)
(2,2)(2,3)(2,4)(3,4)(4,4)
(3,1)(3,2)(3,3)(4,3)
+-------+
| XXXXXX|
| XXXX  |
|XXXX   |
|XX XXX |
|X   XX |
+-------+
```

### Additional Requirements: Recursion, Order Notation, & Contest Submission

We provide a small amount of starter code to read the input puzzle file into a simple board object. You may modify any or all of this provided code.

You must use recursion in a non-trivial way in your solution to this homework. As always, we recommend you work on this program in logical steps. Partial credit will be awarded for each component of the assignment. *IMPORTANT NOTE: This problem is computationally expensive, even for medium-sized puzzles! Be sure to create your own simple test cases as you debug your program.*

Once you have finished your implementation, analyze the performance of your algorithm using order notation. What important variables control the complexity of a particular problem? The dimensions of the board ($w$ and $h$)? The number of strokes ($s$)? The total number of cells painted black ($p$) or the total number of unpainted cells ($u$)? The average length of each stroke ($l$)? In your `README.txt` file write a concise paragraph ($< 200$ words) justifying your answer. Also include a simple table summarizing the running time and number of solutions found by your program on each of the provided examples. If you recognize the image in the picture, include that in your table too. *Note: It's ok if your program can't solve the biggest puzzles in a reasonable amount of time.*

**IMPORTANT:** All students are required to submit their program to the Homework 6 contest (see below). Extra credit will be awarded for programs that have a strong performance in the contest.

You must do this assignment on your own, as described in the "Collaboration Policy & Academic Integrity" handout. If you did discuss this assignment, problem solving techniques, or error messages, etc. with anyone, please list their names in your `README.txt` file.

### Homework 6 Paint by Pairs Contest Rules

- Contest submissions are a separate homework submission. Contest submissions are due Sunday Oct 30th at 11:59pm. You may not use late days for the contest. (The regular homework deadline is Thursday Oct 27th at 11:59pm and late days are allowed for the regular homework submissions.)
- You may submit the same code for both the regular homework submission and the contest. Or you may make a small or significant change for the contest.
- Contest submissions *do not* need to use recursion.
- Contest submissions must follow the output specifications and match the formatting of the examples posted on the course webpage.
- We will compile your code with optimizations enabled (`g++ -O3 *.cpp`) and run all submitted entries on the homework server. Programs that do not compile, or do not complete the basic tests in a reasonable amount of time with correct output, will not receive extra credit.
- Programs must be single-threaded and single-process.
- We will run your program by *redirecting* `std::cout` to a file and measure performance with the UNIX `time` command. For example:

```
time paint_by_pairs.out puzzle1.txt > out_puzzle1.txt
time paint_by_pairs.out puzzle1.txt find_all_solutions > out_puzzle1_all.txt
```

- You may want to use a *C++ code profiler* to measure the efficiency of your program and identify the portions of your code that consume most of the running time. A profiler can confirm your suspicions about what is slow, uncover unexpected problems, and focus your optimization efforts on the most inefficient portions of the code.
- We will be testing with and without the optional command line argument `find_all_solutions` and will highlight the most correct and the fastest programs.
- You may submit up to two interesting new test cases for possible inclusion in the contest. Name these tests `smithj_1.txt` and `smithj_2.txt` (where `smithj` is your RCS username). Extra credit will be awarded for interesting test cases that are used in the contest. Caution: Don't make the test cases so difficult that your program cannot solve them in a reasonable amount of time!
- In your `README_contest.txt` file, describe the optimizations you implemented for the contest, describe your new test cases, and summarize the performance of your program on all test cases.
- Extra credit will be awarded based on overall performance in the contest.