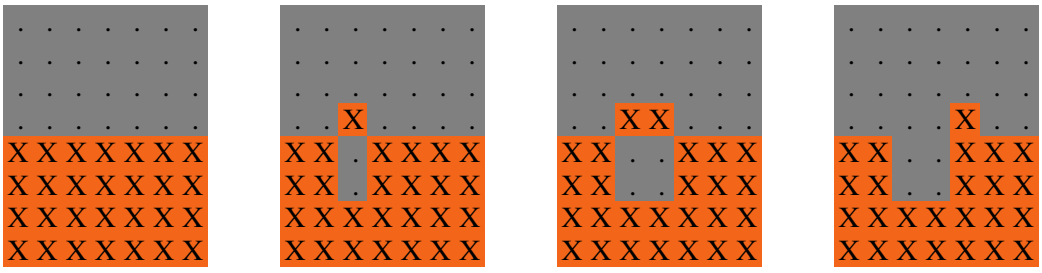
7. Searching Boards

7.1 Conway's Soldiers

The one player game, *Conway's Soldiers* (sometimes known as *Solitaire Army*), is similar to peg solitaire. For this exercise, Conway's board is a 7 (width) × 8 (height) board with tiles on it. The lower half of the board is entirely filled with tiles (pegs), and the upper half is completely empty. A tile can move by jumping another tile, either horizontally or vertically (but never diagonally) onto an empty square. The jumped tile is then removed from the board. A few possible moves are shown below:



The user enters the location of an empty square they'd like to get a tile into, and the program demonstrates the moves that enables the tile to reach there (or warns them it's impossible). To do this you will use a list of boards. The initial board is put into this list. Each board in the list is, in turn, read from the list and all possible moves from that board added into the list. The next board is taken, and all its resulting boards are added, and so on.

Each structure in the list will contain (amongst other things) a board and a record of its parent board, i.e. the board that it was created from.

Exercise 7.1 Write a program that:

- Inputs a target location for a tile to reach (x in argv[1], y in argv[2]).
- Demonstrates the correct solution (reverse order is fine) using plain text.

■

Use the algorithm described above and not anything else.

7.2 The 8-Tile Puzzle

The Chinese 8-Tile Puzzle is a 3×3 board, with 8 numbered tiles in it, and a hole into which neighbouring tiles can move: 

After the next move the board could look like: 

or 

The problem generally involves the board starting in a random state, and the user returning the board to the 'ordered' "12345678" state.

In this problem, a solution could be found in many different ways; the solution could be recursive, or you could implement a queue to perform a breadth-first seach, or something more complex allowing a depth-first search to measure 'how close' (in some sense) it is to the correct solution.

Exercise 7.2 Read in a board using `argv[1]`, e.g.:

```
$ 8tile "513276 48"
```

To do this you will use a list of boards. The initial board is put into this list. Each board in the list is, in turn, read from the list and all possible moves from that board added into the list. The next board is taken, and all its resulting boards are added, and so on. This is, essentially, a queue.

However, one problem with is that repeated boards may be put into the queue and 'cycles' occur. This soon creates an explosively large number of boards (several million). You can solve this by only adding a board into the queue if an identical one does not already exisit in the queue. A linear search is acceptable for this task of identifying duplicates. Each structure in the queue will contain (amongst other things) a board and a record of its parent board, i.e. the board that it was created from.

Be advised that a solution requiring as 'few' as 20 moves may take 10's of minutes to compute. If the search is successful, display the solution to the screen using plain-text.

Use the method described above and only this one. Use a static data structure to acheive this (arrays) and **not** a dynamic method such as linked-lists; a (large) $1D$ array of structures is acceptable. Because this array needs to be so large, it's best to declare it in `main()` using something like:

```
static boards[NUMBOARDS];
```

■

Exercise 7.3 Repeat Exercise 7.2, but use SDL for the ouput rather than plain-text. ■

Exercise 7.4 Repeat Exercise 7.2, but using a dynamic (linked-list), so that you never have to make any assumptions about the maximum numbers of boards stored. ■

Exercise 7.5 Repeat Exercise 7.4, but using a 5×5 board instead. ■