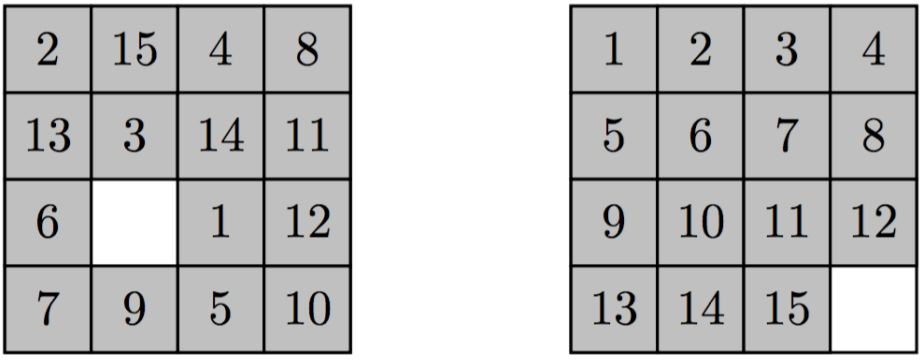

(a) A scrambled board (b) A solved board

Figure 1: Examples of the 16-puzzle.

The 16-puzzle consists of 15 tiles containing the numbers $1, 2, \ldots, 15$ in a 4×4 grid, with an empty space left by the missing 16th tile. The goal of the 16-puzzle is to rearrange the tiles into order by sliding tiles to occupy an empty space. Figure 1 shows a sample board along with the goal configuration where the tiles are in order.

A similar puzzle can be devised for any $n \times n$ board. On a board with N positions (including the empty space), the total number of possible configurations is $N!$, since every arrangement of tiles can be encoded by a permutation of $\{1, \ldots, N\}$ (where the empty space is treated as an invisible tile marked with N), although some configurations cannot be solved. The game graph for the N-puzzle contains vertices for each possible board, and an undirected edge connects every pair of boards which can be transformed into each other by one move. Since every move is reversible (that is, we can always move a tile back after the initial move), there is no need for directed edges. The game graph for the 4-puzzle contains only $4! = 24$ states, and is shown in Figure 2. The goal state is framed in green.

The 9-puzzle has $9! = 362880$ states, so it is possible to compute and store the entire game graph on a current machine. Graph algorithms can then be used to find solutions to each board. For example, a path from a given board b to the goal configuration g (in which all tiles are in order and the empty space is at the lower right) represents a sequence of valid moves which solve b. If g is not reachable from b, then b has no solution. In general, the game graph of a puzzle may have several different connected components, and there may not be a goal state in each component. The game graph for an N-puzzle always has two components, and there is only one goal state. Algorithms for finding connected components can be used to find all solvable configurations of a puzzle. For the N-puzzle, it is also possible to

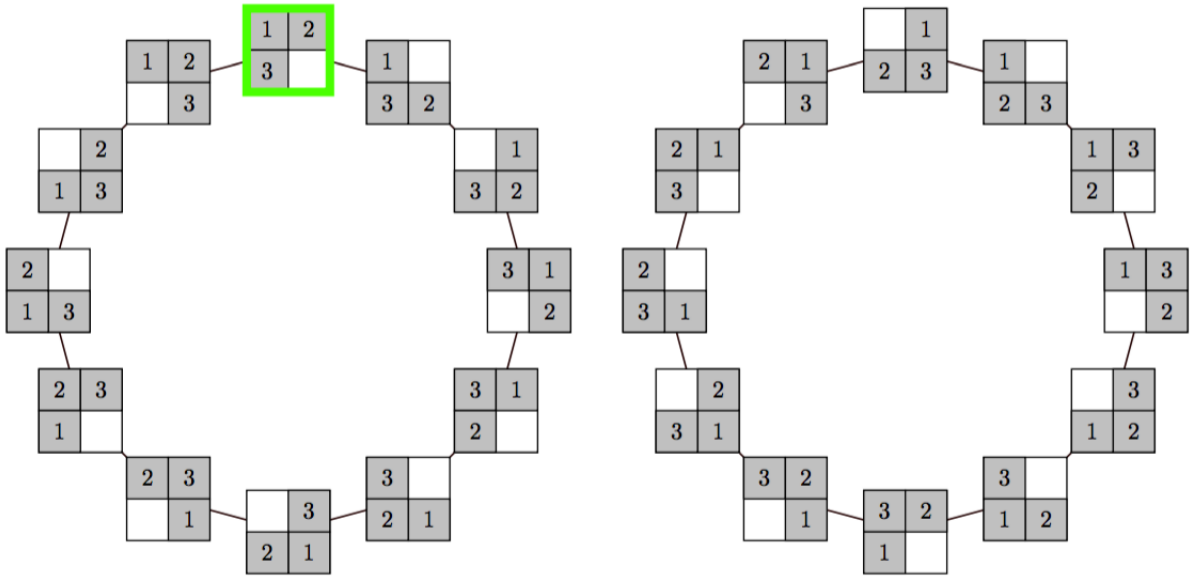

Figure 2: The entire game graph for the 4-puzzle, with the goal state framed in green.

determine whether a given board is solvable without traversing the game graph by using techniques from permutation theory (which is beyond the scope of this course).

Figure 3 shows the neighbourhood of the goal state of the 9-puzzle. Algorithm 27 gives pseudocode to build the game graph of an N puzzle.

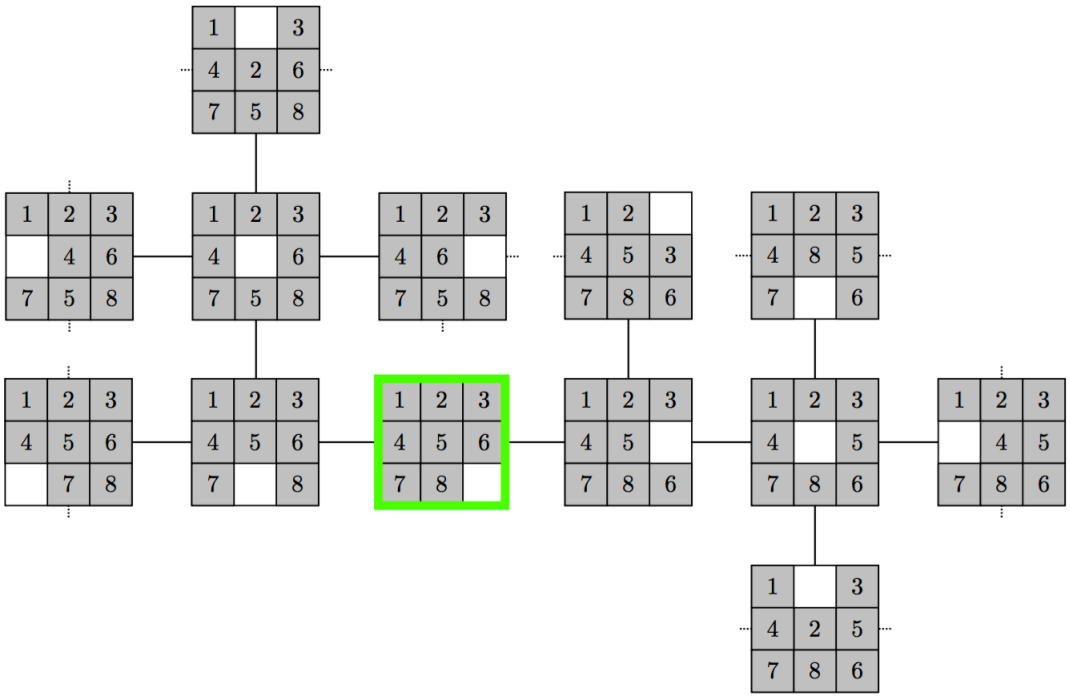

Figure 3: A subset of the game graph for the 9-puzzle, with the goal state framed in green.

Algorithm 27 Build the game graph of an N-puzzle

```
procedure BuildNPuzzleGraph(N)
    G ← New graph on N! vertices, each corresponding to a board.
    n ← √N (number of rows/columns of the board)
    for each vertex v of G do
        B ← The board corresponding to v
        i, j ← Coordinates of the empty space in B
        {Consider all of the neighbouring squares of the empty space}
        {and find all of the boards which can be created by moving them.}
        if i > 0 then
            B' ← The board created by moving tile [i − 1, j] down one row.
            u ← The vertex corresponding to board B'
            Add the edge uv to G
        end if
        if i < n − 1 then
            B' ← The board created by moving tile [i + 1, j] up one row.
            u ← The vertex corresponding to board B'
            Add the edge uv to G
        end if
        if j > 0 then
            B' ← The board created by moving tile [i, j − 1] to the right.
            u ← The vertex corresponding to board B'
            Add the edge uv to G
        end if
        if j < n − 1 then
            B' ← The board created by moving tile [i, j + 1] to the left.
            u ← The vertex corresponding to board B'
            Add the edge uv to G
        end if
    end for
end procedure
```