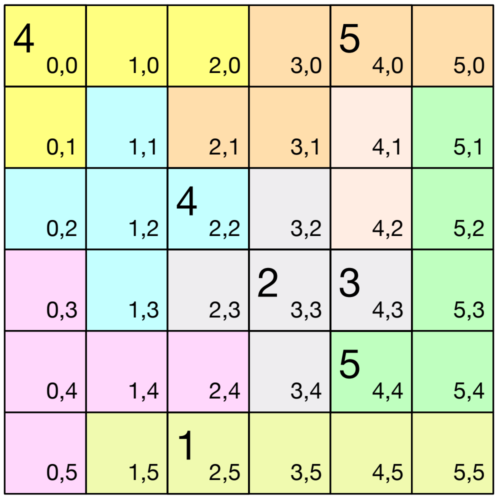
Backtracking 2:

Backtracking can also be used to solve many mathematical puzzles such as Cryptogram and Sudoku. There is a variant of Sudoku, that has not yet found a published solution using backtracking in the net. This means that you do not need to worry about classmates gaining an unfair advantages by downloading and adapting someone else's solution.

A Number Block puzzle consists of a rectangular grid of cells. It is divided into blocks each containing up to five cells. Each cell contains a digit from 1 to *n*, with *n* being the number of cells in the block. So, a single cell block contains only a cell with 1, a two-cell block contains one cell with 1 and one with 2, and so on. The same digit is not allowed to appear in a neighboring cell, not even diagonally.

Use the <u>recursive backtracking scheme</u> from the previous assignment to solve the number block puzzle given below:



4 0,0	1,0	2,0	3,0	5 4,0	5,0
0,1	1,1	2,1	3,1	4,1	5,1
0,2	1,2	4 2,2	3,2	4,2	5,2
0,3	1,3	2,3	2 3,3	3 4,3	5,3
0,4	1,4	2,4	3,4	5 4,4	5,4
0,5	1,5	1 2,5	3,5	4,5	5,5

Here, i have numbered each cell with row and column coordinates. I invite you to try the number puzzle yourself. This will convince you that there is only one valid solution.

Hint: Create a list of lists to form a two-dimensional matrix, initialized with zeroes. Add the numbers in the current cell, i.e. `matrix[0][0] = 4`. Then create a list of sets of 2-tuples to define the blocks. You need to implement:

- `valid_so_far`, which checks that (a) no two non-zero integers are next to each other, even diagonally (b) each area contains only numbers larger than 0 once and not larger than the number of cells in the area.
- `done`, which checks that there are no zero cells left
- `find_empty`, which returns the coordinates of the first cell that still contains a zero.
- `print_it`, which prints out the matrix.

For your convenience, here is the encoding of the original board (called matrix):

```
di = 6
```

```
dj = 6
configuration = [{(0,0), (1,0), (0,1), (2,0)},
                 {(2,1), (3,1), (3,0), (4,0), (5,0)},
                 {(0,2), (1,1), (1,2), (1,3), (2,2)},
                 {(4,1), (4,2)},
                 {(5,1), (5,2), (5,3), (4,4), (5,4)},
                 {(3,2), (2,3), (3,3), (4,3), (3,4)},
                 {(0,3), (0,4), (0,5), (1,4), (2,4)},
                 {(1,5), (2,5), (3,5), (4,5), (5,5)}
                 ]


matrix = [[0 for j in range(dj)] for i in range(di)]
matrix[0][0]=4
matrix[4][0]=5
matrix[2][2]=4
matrix[3][3]=2
matrix[4][3]=3
matrix[4][4]=5
matrix[2][5]=1
```