# Algorithms and Data Structures: Homework #12

Due on May 4, 2020 at 23:00

**Henri Sota**

# Problem 12.3

Consider a puzzle that consists of a $n \times n$ grid where each field contains a value $x_{ij} \in \mathbb{N}$. Our player starts in the top left corner of the grid. The goal of the game is to reach the bottom right corner with the player.
*Rules of the game*: On each turn, you may move your player up, down, left or right. The distance by which the player moves in a chosen direction is given by the number of its current cell. You must stay within the board (you cannot go off the edge of the board).
*Example*: If your player is on a square with value 3, then you may move either three steps up, down, left or right (as long as you do not leave the board).

c) Implement an algorithm that returns the minimum number of moves required to solve this problem. If there is no solution, your algorithm should determine this fact.

```
int PuzzleBoard::solve() {
    int n = this->boardSize;

    // Run Dijkstra's algorithm on the puzzle board
    // Set the initial position at the upper left corner of the board
    this->fields[0][0].steps = 0;
    std::priority_queue<Field> Q;
    // Push first Field onto the queue
    Q.push(this->fields[0][0]);
    Field currentField;

    while (!Q.empty()) {
        // Pop from the queue one Field and save its properties
        currentField = Q.top();
        Q.pop();
        // Change the current xPos and yPos to the popped Field
        this->xPos = currentField.xPos;
        this->yPos = currentField.yPos;

        // Calculate new positions from jumping in all 4 directions starting
        // from current Field
        for (int i  = 0; i < 4; ++i) {
            // Check and make a move if possible to be performed in direction i
            if (makeMove(i)) {
                // Push Field at newly calculated xPos and yPos
                Q.push(fields[this->xPos][this->yPos]);
                this->xPos = currentField.xPos;
                this->yPos = currentField.yPos;
            }
        }
    }
    // Return -1 if no solution was found, otherwise return number of steps
    // taken to arrive at bottom right corner of the puzzle board
    return (this->fields[n - 1][n - 1].steps == INF) ? -1 : this->fields[n - 1][n - 1].
    steps;
}
```

Listing 1: Solve Procedure for Number Maze Implementation in C++

The rest of the implementation is inside "PuzzleBoard.cpp", inside the "numberMaze" directory. One test case has been provided as input and is found in the file "testcases.txt", inside the "numberMaze" directory. Input can given or the puzzle board can be randomly generated depending on the second input given to the program.