CSDS233 Homework 2 – Report

Ben Scholar – bbs27

February 2021

1 Summary of the Algorithm

Beginning at the top right element of the matrix, move down one row if the current element is less than the target element and move one column left if the current element is less than the target element. Returning true if the target element is found and false if you are outside the bounds of the matrix.

```
public static boolean inMatrix(final int[][] matrix, final int x) {
    int r = 0;
    int c = matrix[0].length - 1;

    while(r < matrix.length && c >= 0) {
        final int currentElem = matrix[r][c];
        if(currentElem == x) {
            return true;
        } else if(x > currentElem) {
            r++
        } else {
            c--;
        }
    }


    return false;
}
```

2 Theoretical Analysis

Theoretically, the worst case scenario for this algorithm is that the desired element is in the bottom left corner of the matrix (assuming we start from the top right of the matrix), which would require $2N - 1$ iterations if N is the square matrix's dimensions. This means that the worst case run time scenario is $T(N) = O(N)$, meaning the run time is linear with respect to N

The easiest way to visualize this, is to imagine going straight down from the starting location to the bottom right corner of the matrix, then going left to the bottom left corner. We traverse N elements along each 'edge' of the matrix, subtracting one for the corner element. You will ahve the same number of iterations from the top right to bottom left corners regardless of the path taken (This might be a proven theorem?). See the following example:

Example:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \tag{1}$$

There will be 5 iterations to find 7 in the above matrix, because $2 \cdot (3) - 1 = 5$ (from the equation $I(N) = 2N + 1$.
Steps:
Assume our target element is 7 (the bottom left element) and we start at 3 (top right corner).
1. Since $3 < 7$, move down one row. Current element is now 6.
2. Since $6 < 7$, move down one row. Current element is now 9 (in the bottom right).
3. Now, since $9 > 7$, move left one column. Current element is now 8.
4. Since $8 > 7$, move left one column. Current element is now 7.
5. Finally, return true since $7 == 7$.

Thus, in the worse case, we will have 5 iterations in a 3 by 3 matrix. Every other element in the matrix can be accessed in fewer iterations than this.

3 Empirical Analysis

The empirical analysis of the algorithm confirms that $T(N) = O(N)$. Using a test class (Test.java in the source), every matrix for $2 \leq N \leq 501$ was input into the algorithm 100 times, and the average run time was calculated. Each matrix was run a large amount of times to reduce the amount of noise within the data. The data is then written to a csv file (runtimes.csv), and analyzed using a python script(plot.py). The analysis revealed a line of best fit of $T(N) = 55.6 + 3.1 \cdot N$ with $R^2 = 0.995$ (Figure 2). Which implies that the runtime is truly linear. It can be reasonably inferred from the plot (Figure 1) that the runtime is linear as well. In addition, the residual plot did not indicate any systematic errors within the data.

Notes: The analysis was run twice in each invocation of the test program, but data was only recorded on the second iteration in order to allow the JVM (Java Virtual Machine) to "spin up," as initially there were problems with the JVM class loader making the initial iterations much slower than later iterations and other similar issues. Allowing for a "warm up" seems to be a best practice for bench marking within Java, due to the nature of the JIT compiler and class loader.

4 Comments

This algorithm seems to be as efficient as possible for the constraints we are given. I can't think of a way to improve it unless the constraints were changed (e.g. the elements are in order).

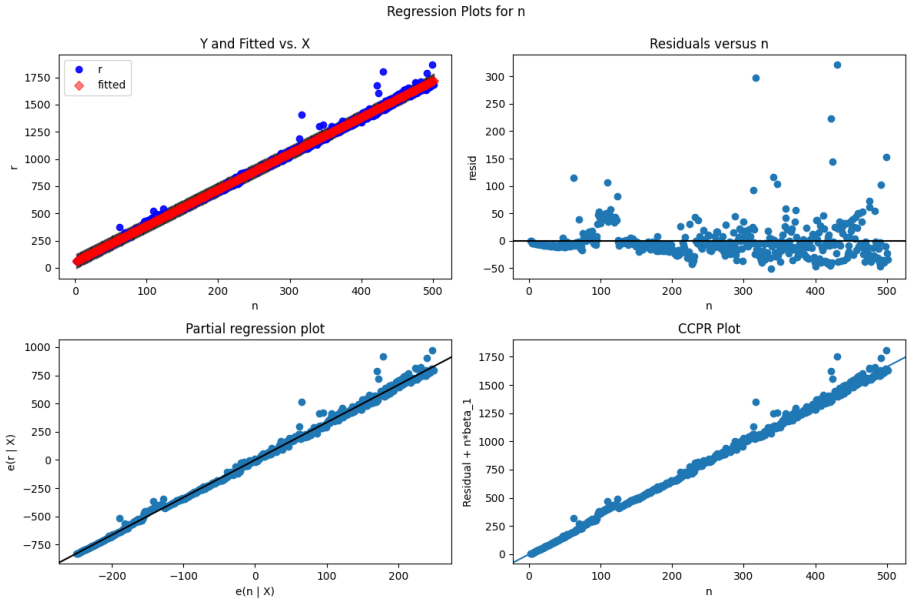

Figure 1: N vs. Runtime Plots

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      r   R-squared:                       0.995
Model:                            OLS   Adj. R-squared:                  0.995
Method:                 Least Squares   F-statistic:                 9.621e+04
Date:                Fri, 26 Feb 2021   Prob (F-statistic):               0.00
Time:                        13:17:09   Log-Likelihood:                -2479.8
No. Observations:                 500   AIC:                             4964.
Df Residuals:                     498   BIC:                             4972.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     55.8012      3.105     17.972      0.000      49.701      61.901
n              3.3211      0.011    310.175      0.000       3.300       3.342
==============================================================================
Omnibus:                      483.859   Durbin-Watson:                   1.793
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            19738.646
Skew:                           4.210   Prob(JB):                         0.00
Kurtosis:                      32.607   Cond. No.                         583.
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Figure 2: Stats