Homework 7: Unique Paths LeetCode Challenge

Hayden C. Daly

September 28, 2020

1 Approach

Started with the naive approach of using recursion and backtracking but then realized the problem could be easily done with dynamic programming and memoization. Started with using a grid directly corresponding to the `x, y` values but realized this was wasted space complexity since it was doing horizontal checks on a row. Using the prior values, I was able to save all of the sums rather than backtracking values reducing the complexity significantly compared to the recursive approach. This change optimized the space complexity from O(m*n) to O(n). In an effort to further optimize the space complexity, I made the row the minimum of m, n to marginally save space on edge cases where one value is significantly larger than the other.

```cpp
int uniquePaths(int m, int n) {
    // Edge case
    if (m == 0 || n == 0) return 0;

    // Initialize array of all ones in O(n) -> Uses smaller size to save space complexity
    int row[min(m, n)], x = 1, y;
    std::fill_n(row, min(m, n), 1);

    // Iterature over grid in O(n*m)
    for (; x < max(m, n); ++x)
        for (y = 1; y < min(m, n); ++y)
            // Only need to do row because horizontal access
            row[y] += row[y-1];

    // Return last index of array
    return row[min(m, n)-1];
}
```

Listing 1: Submitted Algorithm

The algorithm implementation shown above resulted in a program with a time complexity of O(m*n) and a space complexity of O(n). Given the LeetCode submission percentiles, I believe my implementation is optimal in terms of time complexity but there exists room for improvement in terms of the space. I am not certain but I believe since the problem is not dependent upon data rather just the size of the matrix, there must exist a mathematical representation for the problem which would have a constant space complexity.

2 Proof of Completion

Figure 1: LeetCode Submission Proof