COMP3121 21T2 Assignment 3 Q2

Written by Zheng Luo (z5206267)

In order to move from the top left corner $(1, R)$ to bottom right corner $(C, 1)$ with minimum amount of moves from lower elevation to higher elevation, both row position and column position need to be considered as variables, hence the solution will be constituted by using two while loops for specified column i and specified row j. Consequently, the corresponding choice is moving right or moving down at each position. Assuming `dp[row][column]` is defined as the minimum amount of path with lower to higher elevation at current position. Then the way to find minimum number of moves from lower elevation to higher elevation can be solved by using dp table and implemented in the function `minElevatedPath(int[R][C] elevation)` below.

There are 3 base cases, which have to completed before the calculation. The purple section $(1, R)$ is 0, since where is starting position, no height difference at that position. yellow section is assuming character is only moving toward right since starting position, and each position can be calculated by inheriting the dp value from left position and plus 1 if has altitude increase, plus 0 otherwise. Similar operation needs to be perform for green section.

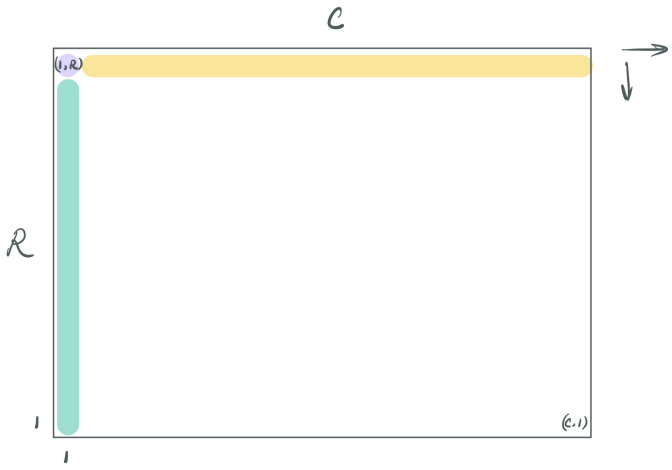

Figure 1: Illustration of Base Case.

For example, at position $(2, R-1)$, the function will compare the dp value at its top and its left, the smaller dp value means less ascending movements are required, hence the dp value at $(2, R-1)$ will become $dp[2][R-1] = min(dp[1][R-1], dp[2][R]) + 1$. The plus 1 will exist if that was a ascending movement, plus 0 otherwise.

Hence in general, repeat the same process until dp table is full, the minimum result can be found at `dp[column][0]` since where is the destination.

The following pseudo code can illustrate the process in a more detailed manner.

```
int minElevatedPath(int[R][C] elevation) {
    // dp has block shifted compared to elevation[R][C],
    // hence all elevation coordination related need to +1.
    int[][] dp = new int[R][C];
    // Base case: starting point (0, row) has no elevation change yet.
    dp[0][R] = 0;

    // Other base cases: calculate the min path elevation
    // for first row on top,
    // and first column on left as preconditions.
    for (int i = 1; i < C; i++) {
        dp[i][R] = dp[i-1][R] +
            elevationCal(elevation[i-1+1][R+1], elevation[i+1][R+1]);
    }
    for (int j = R-1; j > 0; j--) {
        dp[C][j] = dp[C][j-1] +
            elevationCal(elevation[C+1][j-1+1], elevation[C+1][j+1]);
    }
    // Going right.
    for (int i = 1; i < C; i++) {
        // Going down.
        for (int j = R-1; j > 0; j--) {
            dp[i][j] = Math.min(dp[i - 1][j] +
                                elevationCal(elevation[i+1][j+1],
                                             elevation[i+1+1][j+1]),
                                dp[i][j + 1] +
                                elevationCal(elevation[i+1][j+1],
                                             elevation[i+1][j-1+1]));
        }
    }
    return dp[C][0];
}


// Calculate the height difference between two positions,
// return 1 if A < B, 0 otherwise.
int elevationCal(int heightA, int heightB) {
    if (heightA < heightB) {return 1;} else {return 0;}
}
```

In the end, the minimum actual path can be found by selecting the smaller dp value for either top or left block in ending block, then move to the selected block, repeat the process until reach the starting block. The overall time complexity for this solution is $O(n^2)$ by using double while loop.