Z5158334
Yuanzhi Shang

Question 2

For this question, we drive the algorithm base on the thinking of "what is the optimal solution when we arrive point a, b which is $OPT(a, b)$" We then consider the optimal solution of the point which just before we arrive a, b. If the value of point (a, b) is bigger than the value of its previous point i, j then $OPT(a, b) = OPT(i, j) + 1$. If not, $OPT(a, b) = OPT(i, j)$.
Because it can only move to the right or down, according to the requirement of the question, we have two situations.

1. ***The number in the first column and row of the map***
   In the first column, the point can only arrive via the point above on it. In the first row. The point can only arrive via the point on its left.
   Pseudo code:

```
/* represent the map by a 2D array */
map[R][C]
/* optimal solution for every point on map */
opt[R][C]

/* init the base equal to 0*/
/* because the situation from low to high has not happended */
opt[0][0] = 0

/* we can track the path via any data structure */
/* record_previous_point represent a pseudo funciton that maintain
 * a linklist structure. record_previous_point(a,b) record point b a
s
 * the previous point of a
*/

/* the first column */
int j = 0
int i = 0
for(i=1;i < R;i++){
    /* compare to point from top */
    if (map[i][j] > map[i-1][j]){
        opt[i][j] = opt[i-1][j] + 1
    }
    else {
        opt[i][j] = opt[i-1][j]
    }

    record_previous_point(map[i][j],map[i-1][j])
}
```

```
/* the first row */
i = 0
for (j=1;j < C;j++){
    /* compare the point from left */
    if(map[i][j] > map[i][j-1]{
        opt[i][j] = opt[i][j-1] + 1
    }
    else{
        opt[i][j] = opt[i][j-1]
    }
    record_previous_point(map[i][j],map[i][j-1])
}
```

2. *The other points*

   The way that could arrive the other points is either from its left or top. Thus, we just need to compare the optimal solution from the point on its left or top respectively and then choose the smaller one.

   Pseudo code:

```
/* the other point */
for(i=1;i < R;i ++) {
    for(j=1;j < C;j ++){
        /* compare to point from top */
        if (map[i][j] > map[i-1][j]){
            top = opt[i][j-1] + 1
        }
        else {
            top = opt[i][j-1]
        }
        /* compare the point from left */
        if (map[i][j] > map[i][j-1] ){
            left = opt[i-1][j] + 1
        }
        else{
            left = opt[i-1][j]
        }
        /* compare which one is optimal(from left or from top) */
        opt[i][j] = top <= left ? top : left
        /* record */
        if top <= left {
            /* the point from top would be the previous point */
            record_previous_point(map[i][j],map[i-1][j-1])
        }
        else{
            record_previous_point(map[i][j],map[i][j-1])
```

```
            }
        }
    }
```

Then, start from the point from lower right corner. We check its previous point recursively and record the path. In such algorithm, we can find a path which has the minimum number of moves from lower elevation to higher elevation.