Members: gaorui

Vankin’s Mile problem can be solved by two method, one is violence law, the other is dynamaic programm. The violence law needs to get every maximum value for each grid as start grid, so it needs higher algorithm complexity. We solve this problem with top-down dynamic programming algorithm. The time complexity for dynamic programming is: O(nlogn), and the space complexity is: O(n^2).

**Algorithm Description:**

Firstly, we define a n\*n matrix named “maximum” to record each grid maximum value, when start with each grid. If the 3 \* 3 square grids is like follows:

|  |  |  |
| --- | --- | --- |
| Number 1 grid | Number 2 grid | Number 3 grid |
| Number 4 grid | Number 5 grid | Number 6 grid |
| Number 7 grid | Number 8 grid | Number 9 grid |

The row number is from 0 to 2, and the column number is from 0 to 2.

The basic problem for dynamic programming is start with number 9 grid. That is to say maximum[n-1][n-1] = grid[n-1][n-1]. And then, we solve other start grid with top-down dynamic programming algorithm. If it start with number 6 grid, the number 6 grid maximum score is: max(maximum[2][2], grid[1][2]), because maximum[2][2] is our basic problem we know, and grid[1][2] we also know, so we can get the number 6 maximum score. We can solve other grids one by one from up to down with this method.

**The pseudocode is as follows:**

|  |
| --- |
| input: matrix\_dimension, grid\_square, dynamic\_programming\_record\_matrix, max\_num call function: find\_largest\_sum(matrix\_dimension, matrix\_dimension - 1, matrix\_dimension - 1, grid\_square, dynamic\_programming\_record\_matrix) the final result is: max\_num  find\_largest\_sum(matrix\_dimension, x, y, grid\_square, dynamic\_programming\_record\_matrix): if dynamic\_programming\_record\_matrix[x][y] != -99999 then  return dynamic\_programming\_record\_matrix[x][y] else: if y = 0 then  max\_top = 0 else: max\_top = find\_largest\_sum(matrix\_dimension, x, y-1, grid\_square, dynamic\_programming\_record\_matrix)  if x = 0 then  max\_left = 0 else: max\_left = find\_largest\_sum(matrix\_dimension, x-1, y, grid\_square, dynamic\_programming\_record\_matrix)  current = grid\_square[x][y] biggest = find\_max(max\_top + current, max\_left + current, current, matrix\_dimension, x, y) dynamic\_programming\_record\_matrix[x][y] = biggest return biggest  find\_max(up, left, current, dimension, x, y): temp = up if temp < left then temp = left if temp < current then temp = current if temp > max\_sum then if x == dimension - 1 or y == dimension - 1 then max\_sum = temp return temp |

**Algorithm complexity:**

Because in each recursive step, the function needs to call itself 2 times, and in each times we need loop n times to find maximum score for each grid. Thus, the algorithm complexity is: O(nlogn). Also, we use one n \* n matrix to record each gird maximum score named “maximum”, so the space complexity is: O(n^2).