CS325 - Group Assignment 2
Group: 123gogogo
Group members:
Ziwen Tong
Yu Chien, Lin
Xiaoru Chen
2020-10-26

Description of our algorithm:

We solved this problem by using dynamic programming, we are remembering all the intermediate results during iteration to avoid repeated calculation, we are using space to win time.

STEP 1:

Read from the input file, get variable ***n*** and put the matrix into a Two-Dimensional array ***scor***e.

STEP 2:

Variables:
x[i, j]: represents the maximum value(score) we can get from the path starting from square [i, j].
score[i, j]: represents the numerical value at square [i, j].
maxScore: represents the maximum path value of all points that have been iterated as the starting point.

Iterate the matrix from bottom right.

Base case: the matrix size is 1 x 1, or if we start to iterate the matrix from bottom right corner, then x[i, j] = the bottom right corner value, which is score[i, j]

Condition #1: If the starting square is a bottom square (i = n - 1):
If the starting square is a bottom square (i = n - 1), then its x[i, j] can be determined by the sum of the maximum value(score) on its right, which is x[i, j+1], and score[i, j], which is itself. However, if x[i, j+1] is negative, that means the maximum value of that path will decrease the value of our x[n-1, j], we just simply choose to end the game at the current square. If x[i, j+1] is positive, that means the maximum value of that path will increase the value of our x[n-1, j].

Condition #2: If the starting square is on the most right column (j = n - 1):
If the starting square is on the most right column (j = n - 1), then its x[i, j] can be determined by the sum of the maximum value(score) on its bottom, which is x[i+1, j], and score[i, j], which is itself. However, if x[i+1, j] is negative, that means the maximum value of that path will decrease the value of our x[i, n-1], we just simply choose to end the game at the current square. If x[i+1, j] is positive, that means the maximum value of that path will increase the value of our x[i, n-1].

Condition #3: If we are not at bottom or the most right column:
When we are at square i, j, we can either go right or down. We already know the value x[i, j+1] and x[i+1, j], we can now decide which path yields a larger score.
Compare the value of x[i, j+1] and x[i+1, j]. If (x[i, j+1] > x[i+1, j], then the right direction has a larger path value, otherwise, the downward direction has a larger path value.
Then, we can get the maximum path value by adding the value of score[i, j](itself) to the maximum (larger) path value.

And then we update our maxScore if x[i, j] is larger than the previous largest path value.

STEP 3:

Write out ***maxScore*** into an output file.

Running time analysis:

We iterate through the matrix by using a nested-loop. We decrease i value from n-1 to 0, and decrease j value from n-1 to 0. And the time complexity of the max function is O(1). Therefore, the time complexity of our algorithms is:

$$T(n) = \left(O(n) \times O(1)\right) \times \left(O(n) \times O(1)\right) = O(n^2)$$

Correctness:

Our algorithm iteratively verifies all the maximum path value of each square that goes right and downwards.