Programming Assignment 3 Report

Team: Cheng Song, Yangxiao Wang

1. Recurrence relation of minCostVC method:

minPath(row i, Matrix M) is a method to calculate the shortest path from a cell [i,j] from row i to the last row in the matrix.

C[i, j] is the cost of the cell [i, j]

The total shortest path from row 1 to row N is

minPath(1, M) = min(C[1,1], C[1,2], C[1,3], … ,C[1,length] + minPath(2, M))

assume the choose cell from row 1 is cell[1, a]

if(row>=2)

if(a is first col)

minPath(row,M) =min(min(C[row, a], C[row, a+1]) + minPath(row+1,M))

else if(a is last col)

minPath(row,M) =min(min(C[row, a-1], C[row, a]) + minPath(row+1,M))

else

minPath(row, M) = min(min(C[row,a-1], C[row, a], C[row, a+1]) + minPath(row+1,M))

run-time: Since method need to compare number of col of matrix times shortest path, and each shortest path need 3* number of rows of matrix. T = row of matrix * col of matrix

which is O(m*n)

2. Recurrence relation of stringAlignment method:

   x and y are 2 input strings

   penalty(char a , char b) to define the score of match, mismatch and insert penalty

   For building the table:

   T(i.j) = max{ T(i-1, j-1) + penalty(x.charAt(i-1), y.charAt(j-1)),

   T(i-1, j) + insertion penalty,

   T(i, j-1) + insertion penalty }

   Then use the filled table to trace the best solution, in the cell constructor, we defined each cell's parent, which is the cell with max value from T(i-1, j-1), T(i, j-1) and T(i-1, j). So it is easily to track the solution for us.

   run-time: since the table size is (length of the longer string) * (length of the shorter string)

   then this algorithm runs in O(mn) time