```
'''
    Team Member #1: Jason Conci
    Team Member #2: N/A
    Zagmail: jconci@zagmail.gonzaga.edu
    Project 5:
        Given two command line strings, we
            - compute the minimum edit distance & display this matrix,
            - compute and display the alignment implied in the MED
             computation.
        The matrix is printed once when empty, and once after computed.
        The alignment is printed once computed, in a vertical fashion.
        Due: 2018-10-5_18:00:00

    --INSTRUCTIONS FOR USE--
    Run at command line (python 2.7._):
        python project5.py source_word target_word
    -- Any extraneous arguments will be ignored --
'''
# import sys so we can use argv
import sys
# Global variables for SUB, DEL, and INS operations
SUB = "SUB"
DEL = "DEL"
INS = "INS"


'''
Defining a cell class so we can store the score & backpointers
'''
class Cell:
    def __init__(self,  score=0, xcoord = 0, ycoord = 0):
        self.xcoord = xcoord
        self.ycoord = ycoord
        self.score = score
        self.directions = []
    # function to print the value of a cell
    def print_cell(self):
        print " %2d " % self.score,
    # Function to print all attributes of a Cell
    def print_cell_all(self):
        print " %2d " % self.score, self.xcoord, self.ycoord,
         self.directions
    # Function for equating two Cells (not used)
    def __eq__(self, other):
        return self.score == other.score and self.xcoord == other.xcoord
         and self.ycoord == other.ycoord

'''
```

```
Function fo printing out the matrix in the format specified in the book,
because the data structure I use is [0][0] is top left, not bottom left
'''
def print_matrix(matrix, source_string, target_string):
    target_string = " " + target_string
    for i,row in enumerate(matrix[::-1]):
        # Print out an appropriate target letter for each row
        if i != len(target_string):
            print(target_string[len(target_string)-1-i]),
        for each in row:
            each.print_cell(),
        print
    # Print out a row of appropriate source letters
    print "     ",
    for i in source_string:
        print "   " + i,
    print '\n'


'''
Returns the cost of substitution, given two characters
'''
def calc_sub_cost(char_one, char_two):
    return 0 if char_one == char_two else 2


'''
Function for determining the lowest cost direction, and placing info
 into
a Cell in the matrix, at index [i][j]. This information includes the
 score,
which is the lowest cost of the directions, and all directions that
 score
could have come from (SUB, DEL, or INS)
'''
def input_score_and_pointers(matrix, i, j, str_one, str_two):
    ins_cost = matrix[i-1][j].score + 1
    del_cost = matrix[i][j-1].score + 1
    sub_cost = matrix[i-1][j-1].score + calc_sub_cost(str_one[j-1],
     str_two[i-1])

    matrix[i][j].score = min(ins_cost, del_cost, sub_cost)

    ''' I know there's gotta be a better way, but I don't want to find
     it right now '''

    # if they're all the same:
    if ins_cost == del_cost == sub_cost:
```

```
            matrix[i][j].directions = [INS, DEL, SUB]

        # if two are the same, other greater:
        elif ins_cost == del_cost < sub_cost:
            matrix[i][j].directions = [INS, DEL]
        elif ins_cost == sub_cost < del_cost:
            matrix[i][j].directions = [INS, SUB]
        elif del_cost == sub_cost < ins_cost:
            matrix[i][j].directions = [DEL, INS]

        # if one is lower than all others:
        elif ins_cost < del_cost and ins_cost < sub_cost:
            matrix[i][j].directions = [INS]
        elif del_cost < ins_cost and del_cost < sub_cost:
            matrix[i][j].directions = [DEL]
        elif sub_cost < ins_cost and sub_cost < del_cost:
            matrix[i][j].directions = [SUB]

    return matrix

def create_matrix(str_one, str_two):
    n = len(str_one)
    m = len(str_two)

    matrix = [[Cell(0, i, j) for i in range(n+1)] for j in range(m+1)]
    for i in range(n+1):
        matrix[0][i].score = i
        matrix[0][i].directions = ["DEL"]
    for j in range(m+1):
        matrix[j][0].score = j
        matrix[j][0].directions = ["INS"]
    print "EMPTY MATRIX"
    print_matrix(matrix, str_one, str_two)

    for i in range(1, m+1):
        for j in range(1, n+1):
            matrix = input_score_and_pointers(matrix, i, j, str_one,
             str_two)
    print "COMPLETED MATRIX"
    print_matrix(matrix, str_one, str_two)
    return matrix


def get_alignment_backtrace(matrix, source_string, target_string):
    # Getting the lengths of strings
    n = len(source_string)
    m = len(target_string)
    # These lists are where we will place our operations
```

```
source_string_ls = []
target_string_ls = []
op_ls = []

print "Minimum edit distance score:", matrix[m][n].score
print "Computing Alignment..."

# While we're not at our root node,
while n > 0 or m > 0:
    # Gather what directions we're allowed to move
    children = matrix[m][n].directions
    # If SUB is a direction we can move, and our indexes allow,
    if SUB in children and m > 0 and n > 0:
        # Only give "SUB" direction if characters aren't equal
        if source_string[n-1] != target_string[m-1]:
            op_ls.insert(0, SUB)
        else:
            op_ls.insert(0, ' ')
        # Insert both characters, and move left one, down one
        source_string_ls.insert(0, source_string[n-1])
        target_string_ls.insert(0, target_string[m-1])
        n -=1
        m -= 1
    # Otherwise, if DEL is a direction we can move,
    elif DEL in children and n > 0:
        # insert DEL instruction, add source letter to source,
        # add # to target, move left
        op_ls.insert(0,DEL)
        source_string_ls.insert(0, source_string[n-1])
        target_string_ls.insert(0, "%")
        n -= 1
    # Otherwise, if INS is a direction we can move,
    elif INS in children and m > 0:
        # insert INS instruction, add target letter to target,
        # add # to source, and move down
        op_ls.insert(0, INS)
        source_string_ls.insert(0, "%")
        target_string_ls.insert(0, target_string[m-1])
        m -= 1

# Once we've reached our root node, we print out the strings
 vertically
# in the format: Src - Trg OPR
print
print "To get from " + source_string + " to " + target_string +
 " ..."
for i in range(len(target_string_ls)):
    print source_string_ls[i],
```

```
        print " - ",
        print target_string_ls[i],
        print op_ls[i]

    return (source_string_ls, target_string_ls, op_ls)

'''
 Driver for the program. Gathers strargs, creates matrix based on
  strargs,
 and creates the alignment for the created matrix. All of this is
  printed
 to screen at various points throughout the program.
'''
def main(strargs):
    str_one = str(strargs[0])
    str_two = str(strargs[1])
    matrix = create_matrix(str_one, str_two)
    alignment_tuple = get_alignment_backtrace(matrix, str_one, str_two)



# allows us to easily run at command line, "python project5.py arg1 arg2
 ..."
if __name__ == "__main__":
    if len(sys.argv) < 3:
        exit("Cannot compute MED with insufficient arguments")
    main(sys.argv[1:])
```