

```

class GeneSequencing:

    def __init__( self ):
        pass

# This is the method called by the GUI. _sequences_ is a list of the ten
sequences, _table_ is a
# handle to the GUI so it can be updated as you find results, _banded_ is a
boolean that tells
# you whether you should compute a banded alignment or full alignment, and
_align_length_ tells you
# how many base pairs to use in computing the alignment
    def align( self, sequences, table, banded, align_length ):
        self.banded = banded
        self.MaxCharactersToAlign = align_length
        results = []

        for i in range(len(sequences)):
            jresults = []
            for j in range(len(sequences)):
                if j < i:
                    s = {}
                else:
#####
#####
# your code should replace these three statements and populate the three
variables: score, alignment1 and alignment2

                # Comparing itself
                if i == j:
                    score = max(-3*align_length, -3*len(sequences[i]))
                    alignment1 = 'self'
                    alignment2 = 'self'
                # Comparing with artificial sequences
                elif (i == 0 and j != 1) or (i == 1 and j != 0):
                    score = float('inf')
                    alignment1 = 'No alignment possible'
                    alignment2 = 'No alignment possible'
                else:
                    sequence_i_length = len(sequences[i])
                    sequence_j_length = len(sequences[j])
                    # Initialize the arrays
                    if align_length > sequence_i_length:
                        if align_length > sequence_j_length:
                            matrix_distance = [[0 for column in
range(sequence_i_length + 1)] for row in range(sequence_j_length + 1)]
                            matrix_path = [['' for column in
range(sequence_i_length + 1)] for row in range(sequence_j_length + 1)]
                        else:
                            matrix_distance = [[0 for column in
range(align_length + 1)] for row in range(align_length + 1)]

```

```

        matrix_path = [['' for column in range(align_length +
1)] for row in range(align_length + 1)]

    # Filling out the first row
    for k in range(len(matrix_distance[0])):
        if banded:
            if k > 3:
                matrix_distance[0][k] = float('inf')
            else:
                matrix_distance[0][k] = k * INDEL
            matrix_path[0][k] = 'r'

    # Filling out the first column
    for k in range(len(matrix_distance)):
        if banded:
            if k > 3:
                matrix_distance[k][0] = float('inf')
            else:
                matrix_distance[k][0] = k*5
            matrix_path[k][0] = 't'
    matrix_path[0][0] = ''

    for count in range(1, len(matrix_distance)):
        if banded:
            if count - 3 > 0:
                start = count - 3
            else:
                start = 0
            if count + 4 < len(matrix_distance[0]):
                finish = count + 4
            else:
                finish = len(matrix_distance[0])
        else:
            start = 1
            finish = len(matrix_distance[0])

    # loop through and calculate each value and find the
    minimum cost, filling out the matrix
    # path matrix keeps track of which direction it came
    from

    # Would take at most O(mn) time and space
    for index in range(start, finish):
        diagonal = matrix_distance[count-1][index-1] +
self.get_difference(sequences[i], sequences[j], count-1, index-1)
        right = matrix_distance[count-1][index] + INDEL
        top = matrix_distance[count][index-1] + INDEL

        min_val = min(right, top, diagonal)

        matrix_distance[count][index] = min_val
        if min_val == diagonal:

```

```

        matrix_path[count][index] = 'd'
    elif min_val == top:
        matrix_path[count][index] = 't'
    elif min_val == right:
        matrix_path[count][index] = 'r'

    # Assign alignments using backtrace
    # Would take O(mn) time and space
    alignment1, alignment2 =
self.get_string_alignment(matrix_path, sequences[i], sequences[j])
    score = matrix_distance[-1][-1]
    #if i == 2 and j == 9:
    #    print(alignment1)
    #    print(alignment2)

#####
#####

        s = {'align_cost':score, 'seqi_first100':alignment1,
'seqj_first100':alignment2}
        table.item(i,j).setText('{}'.format(int(score) if score !=
math.inf else score))
        table.repaint()
        jresults.append(s)
        results.append(jresults)
    return results

def get_string_alignment(self, matrix_path, sequence_i, sequence_j):
    path = ''
    i_final = ''
    j_final = ''
    j,k = -1,-1

    # Should start at the bottom right corner and backtrace
    # Find if it was from diagonal, from the above, and from right, and
backtrace appropriately
    while matrix_path[j][k] != '':
        path = path + matrix_path[j][k]
        if matrix_path[j][k] == 'd':
            j = j - 1
            k = k - 1
        elif matrix_path[j][k] == 'r':
            k = k - 1
        elif matrix_path[j][k] == 't':
            j = j - 1

    path = path[::-1]
    index_r, index_c = 0,0
    for i in range(len(path)):
        if path[i] == 't':
            i_final += '-'
        else:

```

```

        i_final += sequence_i[index_r]
        index_r = index_r + 1
    if path[i] == 'r':
        j_final += '-'
    else:
        j_final += sequence_j[index_c]
        index_c = index_c + 1
    return i_final, j_final


# return -3 if match, 1 if sub
# Would take O(1) time and constant space
def get_difference(self, sequence_i, sequence_j, count, index):
    if sequence_i[index] == sequence_j[count]:
        return MATCH
    return SUB

```

1. The time complexity would be $O(m*n)$. It'll take $O(mn)$ time to initialize the matrix, and $O(mn)$ time to backtrack. The Space complexity would be also $O(mn)$ because it's using two 2-d matrices.

2. For the alignment extraction, the characters are used to know if the minimum cost was from the above, diagonal, or from the right side. The extraction starts from the bottom right corner of the matrix and start backtrack as it reduces the matrix. Once it hits the top left corner, the alignments are made by looping through the path. I put dash when it was INDEL and append it to each alignment 1 and 2.

3. Results

 Gene Sequence Alignment

	sequence1	sequence2	sequence3	sequence4	sequence5	sequence6	sequence7	sequence8	sequence9	sequence10
sequence1	-30	-1	inf	inf	inf	inf	inf	inf	inf	inf
sequence2		-33	inf	inf	inf	inf	inf	inf	inf	inf
sequence3			-3000	-2996	-2956	-2944	-1431	-1448	-1399	-1448
sequence4				-3000	-2960	-2948	-1431	-1448	-1399	-1448
sequence5					-3000	-2988	-1423	-1452	-1391	-1448
sequence6						-3000	-1426	-1452	-1394	-1448
sequence7							-3000	-2771	-2814	-2767
sequence8								-3000	-2731	-2996
sequence9									-3000	-2727
sequence10										-3000

Label I:
 Sequence I:
 Sequence J:
 Label J:

☐ Banded Align Length:

Done. Time taken: 58.858 seconds.

	sequence1	sequence2	sequence3	sequence4	sequence5	sequence6	sequence7	sequence8	sequence9	sequence10
sequence1	-30	-9	inf	inf	inf	inf	inf	inf	inf	inf
sequence2		-33	inf	inf	inf	inf	inf	inf	inf	inf
sequence3			-9000	-8984	-8888	-8848	-2736	-2749	-1433	-2741
sequence4				-9000	-8888	-8848	-2740	-2754	-1430	-2746
sequence5					-9000	-8960	-2712	-2745	-1431	-2733
sequence6						-9000	-2709	-2734	-1420	-2722
sequence7							-9000	-8108	-1261	-8104
sequence8								-9000	-1310	-8980
sequence9									-9000	-1315
sequence10										-9000

Label I:

Sequence I:

Sequence J:

Label J:

Process

Clear

☒ Banded Align Length: 3000

Done. Time taken: 29.312 seconds.

4.

	sequence1	sequence2	sequence3	sequence4	sequence5	sequence6	sequence7	sequence8	sequence9	sequence10
sequence1	-30	-1	inf	inf	inf	inf	inf	inf	inf	inf
sequence2		-33	inf	inf	inf	inf	inf	inf	inf	inf
sequence3			-3000	-2996	-2956	-2944	-1431	-1448	-1399	-1448
sequence4				-3000	-2960	-2948	-1431	-1448	-1399	-1448
sequence5					-3000	-2988	-1423	-1452	-1391	-1448
sequence6						-3000	-1426	-1452	-1394	-1448
sequence7							-3000	-2771	-2814	-2767
sequence8								-3000	-2731	-2996
sequence9									-3000	-2727
sequence10										-3000

Label 3:

Bovine coronavirus isolate BCoV-ENT, complete genome.

Sequence 3:

ctaagaaggctatgcttt-aatt-agaggatac-cgt-ggtgt---taa-a-cgc-c---ttcttt-a-tgta-g--a-ccag--tatgggt--gtgattatac---tg---g---t-ggtc

Sequence 10:

agggttcatttaac-ccgaagggttaagtactccgcaaggcgtatgctcttctaagggtatcgcggtgttaaatccatcctattcttgaccagtatggtgtgactatactggcg

Label 10:

Murine hepatitis virus strain Penn 97-1, complete genome.

Process

Clear

☐ Banded Align Length: 1000

Done. Time taken: 51.841 seconds.

	sequence1	sequence2	sequence3	sequence4	sequence5	sequence6	sequence7	sequence8	sequence9	sequence10
sequence1	-30	-9	inf	inf	inf	inf	inf	inf	inf	inf
sequence2		-33	inf	inf	inf	inf	inf	inf	inf	inf
sequence3			-9000	-8984	-8888	-8848	-2736	-2749	-1433	-2741
sequence4				-9000	-8888	-8848	-2740	-2754	-1430	-2746
sequence5					-9000	-8960	-2712	-2745	-1431	-2733
sequence6						-9000	-2709	-2734	-1420	-2722
sequence7							-9000	-8108	-1261	-8104
sequence8								-9000	-1310	-8980
sequence9									-9000	-1315
sequence10										-9000

Label 3:

Bovine coronavirus isolate BCoV-ENT, complete genome.

Sequence 3:

ggcgtagatttcatagtggtgtctatattcattctgctgtaacagcttcagcc-agggacgtgtgtatcctaggcagtgcccacccatagggtcacaatgtc-gaatgca-a

Sequence 10:

gtg-ggcttg-gtc-ttgtcatagtgtgacattgtggttccttggtttgttctctgccagtgcgtgtccattcggcgccagcagccacccatagggtgcataatggc-aaagat

Label 10:

Murine hepatitis virus strain Penn 97-1, complete genome.

Process

Clear

☒ Banded Align Length: 3000

Done. Time taken: 29.312 seconds.