Instituto de Informática, UFRGS
INF05018 - Biologia Computacional - Turma: U (2019/2)
Prof. Márcio Dorn
Lista de exercícios 2 parte 1
Nome: Pedro Foletto Pimenta Nº Cartão UFRGS: 229778

1b, 1c, 1d. Código fonte do algoritmo Smith-Waterman:

```
# Smith–Waterman algorithm: global alignment of two sequences
def smith_waterman(seq1, seq2):

    # size of the grid
    size1 = len(seq1) + 1 # +1 for the extra column/row in the points table
    size2 = len(seq2) + 1

    # construct the grid
    point_matrix = np.zeros((size1, size2))
    for i in range(size1):
            for j in range(size2):
                    if(i==0 or j == 0):     # first row or first column
                            point_matrix[i][j] = 0
                    else:                   # rest of the table
                            if(seq1[i-1] == seq2[j-1]): # match
                                    match_score = MATCH_SCORE
                            else:       # mismatch
                                    match_score = MISMATCH_SCORE
                            expr1 = point_matrix[i-1][j-1] + match_score
                            expr2 = point_matrix[i][j-1] + GAP_SCORE # -4 == +gap(seq1)
                            expr3 = point_matrix[i-1][j] + GAP_SCORE # -4 == +gap(seq2)
                            point_matrix[i][j] = max(0, expr1, expr2, expr3)

    # find the best alignment
    alignment1 = ""
    alignment2 = ""

    match_count = 0

    # start at the bottom right
    i, j = index_highest_value(point_matrix)
    end_i, end_j = i, j
    alignment_score = point_matrix[i][j] # inicializar total score

    # traceback starting at the element with the highest score until 0 is encountered
    while (point_matrix[i][j] != 0):
            up = point_matrix[i-1][j]
            left = point_matrix[i][j-1]
            diag = point_matrix[i-1][j-1]
            # find out best direction
            if(diag >= up and diag >= left):
                    i = i - 1
                    j = j - 1
                    alignment1 = seq1[i] + alignment1
                    alignment2 = seq2[j] + alignment2
            elif(up >= diag and up >= left):
                    i = i - 1
                    alignment1 = seq1[i] + alignment1
                    alignment2 = "-" + alignment2
            else: # (left >= diag and left >= up)
                    j = j - 1
                    alignment1 = "-" + alignment1
                    alignment2 = seq2[j] + alignment2
            # count score
            alignment_score = alignment_score + point_matrix[i][j]


    # guardar os indices de começo do alinhamento local
```

```
    begin_i, begin_j = i, j

    # identidade
    match_count = 0
    for i in range(len(alignment1)):
            if(alignment1[i] == alignment2[i]):
                    match_count += 1

    # print point matrix
    print("...printing point matrix of dimensions " + str(point_matrix.shape))
    print(point_matrix)

    # print aligned sequences
    print("Alinhamento final de tamanho "+str(end_i-begin_i)+" :")
    print(alignment1 + " (indices de " + str(begin_i) + " ate " + str(end_i) + ")")
    print(alignment2 + " (indices de " + str(begin_j) + " ate " + str(end_j) + ")")

    # print "identidade do alinhamento" (?)
    print("Identidade: " + str(match_count))

    # print score
    print("Alignment score: " + str(alignment_score))

    return alignment_score
```

1e. Resultado do alinhamento:

```
...printing point matrix of dimensions (2149, 191)
[[ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  1.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 ...,
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]]
Alinhamento final de tamanho 4 :
GALN (indices de 420 ate 424)
GALN (indices de 86 ate 90)
Identidade: 4
Alignment score: 10.0
```

Como o melhor alinhamento local é de 4 caracteres e as sequências têm tamanhos de 2149 e 191, podemos concluir que elas não são muito semelhantes.

2. O método de matriz de pontos escolhido para alinhar as duas sequências foi o Needleman-Wunsch, pois nesse caso estamos interessados no alinhamento global entre as sequências.
Sim, é possível identificar quantas e quais foram as mutações realizadas na sequência *"Organismo MusMusculus – PKp2 com mutacao"*. Para isso, basta executar o alinhamento das sequências:

```
VLSAANKSNVKAAWGKVGGNAPAY-GAQALQRMFLSFPTTKTYFPHFDLSHGSAQQKAHGQKVANA-LTKAQGHLND-LPGTLSNLSNLHAHKLRVNPVNFKLLSHSLLVTLASHLPTNFTPAVHANLNKFLANDSTVLTSKYR
VLSAADKGNVKAAWGKVGGHAA-EYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHGAKVAA-ALTKAVEHLD-DLPGALSELSDLHAHKLRVDPVNFKLLSHSLLVTLASHLPSDFTPAVHASLDKFLANVSTVLTSKYR
```

Então, podemos facilmente percorrer o alinhamento gerado e contar ou identificar as posições das mutações. Nesse caso, também é possível de identificar e contar as mutações ao olhar para o resultado, já que a sequência não é tão longa.
Do mesmo jeito, também é possível identificar a inserção de GAPS ou deleções.
Output completo do programa feito em Python para o problema:

```
...printing point matrix of dimensions (142, 142)
[[   0.   -4.   -8. ..., -556. -560. -564.]
 [  -4.    5.    1. ..., -547. -551. -555.]
 [  -8.    1.   10. ..., -538. -542. -546.]
 ...,
 [-556. -547. -538. ...,  514.  510.  506.]
 [-560. -551. -542. ...,  510.  519.  515.]
 [-564. -555. -546. ...,  506.  515.  524.]]
Alinhamento final:
OOOOOXOXOOOOOOOOOOOXOX-X-OOXOOXOOOOOOOOOOOOOOOOOOOOOOOXOXOOXOOOX--OOOOXXOOX--OOOXOOXOOXOOOOOOOXOOOOOOOOOOOOOOOOOOXXOOOOOOOXOXOOOOOXOOOOOOOO
VLSAANKSNVKAAWGKVGGNAPAY-GAQALQRMFLSFPTTKTYFPHFDLSHGSAQQKAHGQKVANA-LTKAQGHLND-LPGTLSNLSNLHAHKLRVNPVNFKLLSHSLLVTLASHLPTNFTPAVHANLNKFLANDSTVLTSKYR
VLSAADKGNVKAAWGKVGGHAA-EYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHGAKVAA-ALTKAVEHLD-DLPGALSELSDLHAHKLRVDPVNFKLLSHSLLVTLASHLPSDFTPAVHASLDKFLANVSTVLTSKYR
Identidade (?): 115
Alignment score: 37191.0
```

A linha acima das sequências informa, para cada posição, se ocorreu um match (marcado por um **O**), um mismatch (marcado por um **X**), ou um gap (marcado por um **-**).

Código fonte:

```
# "regras" do alinhamento
GAP_SCORE = -4
MATCH_SCORE = 5
MISMATCH_SCORE = -3

# prints an alignment in an organized way
def print_alignment(alignment0, alignment1, alignment2):
    print("Alinhamento final:")
    print(alignment0)
    print(alignment1)
    print(alignment2)

# needleman_wunsch algorithm: global alignment of two sequences
def needleman_wunsch(seq1, seq2):

    # size of the grid
    size1 = len(seq1) + 1 # +1 for the extra column/row in the points table
    size2 = len(seq2) + 1

    # construct the grid
    point_matrix = np.zeros((size1, size2))
    for i in range(size1):
            for j in range(size2):
                    if(i==0):     # first row
                            point_matrix[i][j] = j * GAP_SCORE
                    elif(j==0):     # first column
                            point_matrix[i][j] = i * GAP_SCORE
                    else:       # rest of the table
                            if(seq1[i-1] == seq2[j-1]): # match
                                    match_score = MATCH_SCORE
                            else: # mismatch
                                    match_score = MISMATCH_SCORE
                            expr1 = point_matrix[i-1][j-1] + match_score
                            expr2 = point_matrix[i][j-1] + GAP_SCORE # -4 == +gap(seq1)
                            expr3 = point_matrix[i-1][j] + GAP_SCORE # -4 == +gap(seq2)
                            point_matrix[i][j] = max(expr1, expr2, expr3)

    # find the best alignment
    alignment0 = "" # string q indica se ouve match, mismatch ou gap para cada posicao
    alignment1 = ""
    alignment2 = ""

    match_count = 0

    # start at the bottom right
    i, j = size1-1, size2-1
    alignment_score = point_matrix[i][j] # init total score

    # trace back to the upper left corner
    while (i > 0 and j > 0):
            up = point_matrix[i-1][j]
            left = point_matrix[i][j-1]
            diag = point_matrix[i-1][j-1]
            # find out best direction
```

```
            if(diag >= up and diag >= left):
                        i = i - 1
                        j = j - 1
                        alignment1 = seq1[i] + alignment1
                        alignment2 = seq2[j] + alignment2
                        if( seq1[i] ==  seq2[j]):
                                    alignment0 = "O" + alignment0 # match
                        else:
                                    alignment0 = "X" + alignment0 # mismatch
            elif(up >= diag and up >= left):
                        i = i - 1
                        alignment0 = "-" + alignment0 # gap
                        alignment1 = seq1[i] + alignment1
                        alignment2 = "-" + alignment2
            else: # (left >= diag and left >= up)
                        j = j - 1
                        alignment0 = "-" + alignment0 # gap
                        alignment1 = "-" + alignment1
                        alignment2 = seq2[j] + alignment2
            # count score
            alignment_score = alignment_score + point_matrix[i][j]

   # get last remaining letters
   while(i > 0):
            i = i - 1
            alignment1 = seq1[i] + alignment1
            alignment2 = "-" + alignment2
   while(j > 0):
            j = j - 1
            alignment1 = "-" + alignment1
            alignment2 = seq2[j] + alignment2

   # identidade
   match_count = 0
   for i in range(len(alignment1)):
            if(alignment1[i] == alignment2[i]):
                        match_count += 1

   # print point matrix
   print("...printing point matrix of dimensions " + str(point_matrix.shape))
   print(point_matrix)

   # print aligned sequences
   print_alignment(alignment0, alignment1, alignment2)

   # print "identidade do alinhamento" ?
   print("Identidade (?): " + str(match_count))

   # print score
   print("Alignment score: " + str(alignment_score))

   return alignment_score




####################################
# main


score = needleman_wunsch(seq_musmusculus, seq_musmusculus_mut)
```