Aluno: Felipe Colombelli Cartão: 00287698

Nota inicial: para a resolução da lista de exercícios, foi utilizado Python 3.

Questão 1:

Pseudocódigo

Foi utilizado o editor VS Code pela facilidade de formatação que inclui também o número de linhas.

```
// Primeiro definimos a classe responsável pelas operações de alinhamento

Classe: Alinhador
Recebe os parâmetros:
    seq1 - sequência 1
    seq2 - sequencia 2
    penalidadeGap
    penalidadeMiss
    scoreDeMatch

Construtor:
    Todos os parâmetros viram atributo do objeto, além dos atributos
    * scoreFinal   // que começa com 0
    * identidade    // que começa com 0
    * matrizAlinhamento     // matriz de zeros com (n x m) = (len(seq1)+1, len(seq2)+1)
    * matrizDeTraceBack     // matriz de zeros com (n x m) = (len(seq1)+1, len(seq2)+1)
    // a matriz de trace back será preenchida com caracteres indicando como foi construída
    // a matriz de alinhamento. Onde d: diagonal; l: esquerda; u: cima; f: posição (0,0)
    * string1   // Resultado de alinhamento da seq1
    * string2   // Resultado de alinhamento da seq2

Métodos:

    pegaValor(i, j)     // recebe uma posição i,j na matrix e compara a posição i-1,j-1
                        // das sequências, retornando o máximo valor de acordo com a fórmula dada
                        // além da operação escolhida (match/missmatch, gap na seq1, gap na seq2, 0)


    alinha():
        Preenche a coluna 0 da matrizDeTraceBack com 'u'
        Preenche a linha 0 da matrizDeTraceBack com 'l'
        Assina a posição (0, 0) da matrizDeTraceBack com 'f'

        Preenche a coluna 0 da matrizAlinhamento com 0
        Preenche a linha 0 da matrizAlinhamento com 0

        Para i, j da matrizAlinhamento:
            Se i ou j forem iguais a 0:
                Pula iteração   // pois já foram preenchidos

            matrizAlinhamento[i][j], operação = pegaValor(i, j)
            matrizDeTraceBack[i][j] = operação

        scoreFinal = valor máximo da matrizAlinhamento
        maxIndices = pega valor dos índices onde a célula é == scoreFinal
        fatiaMatriz()
```

```
47
48
49      fatiaMatriz():
50
51          para cada (x,y) em maxIndices:
52              matrizFatiada = matrizAlinhamento[0 a x][0 a y]
53              fazAlinhamento()
54
55
56
57      fazAlinhamento():
58          string1 = ''
59          string2 = ''
60
61          OperaçãoAtual = matrizDeTraceBack[último i da matrizFatiada][último j da matrizFatiada]
62          i,j = últimos i,j da matrizFatiada
```

```
63
64              Enquanto OperaçãoAtual != 'f':
65
66                  Se OperaçãoAtual for 'd' (diagonal):
67                      string1 recebe o caractere i da seq1    // ou seja, alinha
68                      string2 recebe o caractere j da seq2    // ou seja, alinha
69                      i = i-1
70                      j = j-1
71
72                  Se OperaçãoAtual for 'l' (esquerda):
73                      string1 recebe -    // ou seja, recebe um gap
74                      string2 recebe o caractere j da seq2    // ou seja, alinha
75                      j = j-1
76
77                  Se OperaçãoAtual for 'u' (cima):
78                      string1 recebe o caractere i da seq1    // ou seja, alinha
79                      string2 recebe -    // ou seja, recebe um gap
80                      i = i-1
81
82              calculaIdentidade()
83              printaResultados()
84
85          calculaIdentidade():
86              Conta o número n da maior string final entre as duas alinhadas
87              identidade =    número de caracteres  iguais nas mesmas posições das
88                              das strings 1 e 2 alinhadas, dividido por n
89
90      // Com a classe definida, podemos prosseguir com os alinhamentos
91      gap = -2
92      match = 1
93      missmatch = -1
94
95      Abre sequencia humana
96      Abre sequencia biomphalaria
97
98      Cria objeto passando os parâmetros
99      objeto.alinha()
```

Considerações: como a matriz de score final ficou com um valor muito alto de linhas e colunas (191 por 2149), fica inviável imprimí-la na tela de maneira a extrair alguma informação de útil. Com este problema em mente foi implementado no algoritmo um pedaço de código em que a matriz final é “cortada” só naqueles índices que interessam para o traceback. Como mais de uma parte da matriz de scores obteve score final de 4, todos estes pedaços foram levados em consideração e printados até o critério de parada (zero). Além disso, só a pequena parte das strings que se alinharam localmente foram printadas junto das suas “submatrizes” pertencentes a matriz de scores final.

Resultados:

```
C:\Users\Pichau\Desktop\Felipe\biocomp\list 2\part 2>python e2-1.py
Indexes from score matrix: (23, 803)
[[0 0 0 0 0]
 [0 1 0 0 0]
 [0 0 2 0 1]
 [0 0 0 3 1]
 [0 0 1 1 4]]

SEAE
SEAE

Final Score: 4
Identity: 0.00186219739292365



Indexes from score matrix: (26, 2005)
[[0 0 0 1 3 1 0 0 0]
 [0 1 0 0 1 2 0 0 0]
 [0 0 2 0 0 0 1 0 0]
 [0 0 0 3 1 0 0 0 0]
 [0 0 0 1 2 0 0 0 1]
 [0 0 0 1 2 1 0 0 0]
 [0 0 0 0 0 1 2 0 0]
 [0 0 0 0 0 0 0 3 1]
 [0 0 0 0 0 0 0 1 4]]

LSEAERKA
LSEEDRKA

Final Score: 4
Identity: 0.002793296089385475



Indexes from score matrix: (90, 424)
[[0 0 0 0 0]
 [0 1 0 0 0]
 [0 0 2 0 0]
 [0 0 0 3 1]
 [0 0 0 1 4]]

GALN
GALN

Final Score: 4
Identity: 0.00186219739292365
```

```
Indexes from score matrix: (92, 426)
[[0 0 0 0 0 0 0]
 [0 1 0 0 0 0 0]
 [0 0 2 0 0 0 0]
 [0 0 0 3 1 0 0]
 [0 0 0 1 4 2 0]
 [0 0 0 0 2 3 1]
 [1 0 0 0 0 1 4]]

GALNTV
GALNKV

Final Score: 4
Identity: 0.0023277467411545625



Indexes from score matrix: (177, 787)
[[0 1 0 0 0]
 [0 1 0 0 0]
 [0 0 2 0 0]
 [0 0 0 3 1]
 [0 0 0 1 4]]

QVPN
QVPN

Final Score: 4
Identity: 0.00186219739292365
```

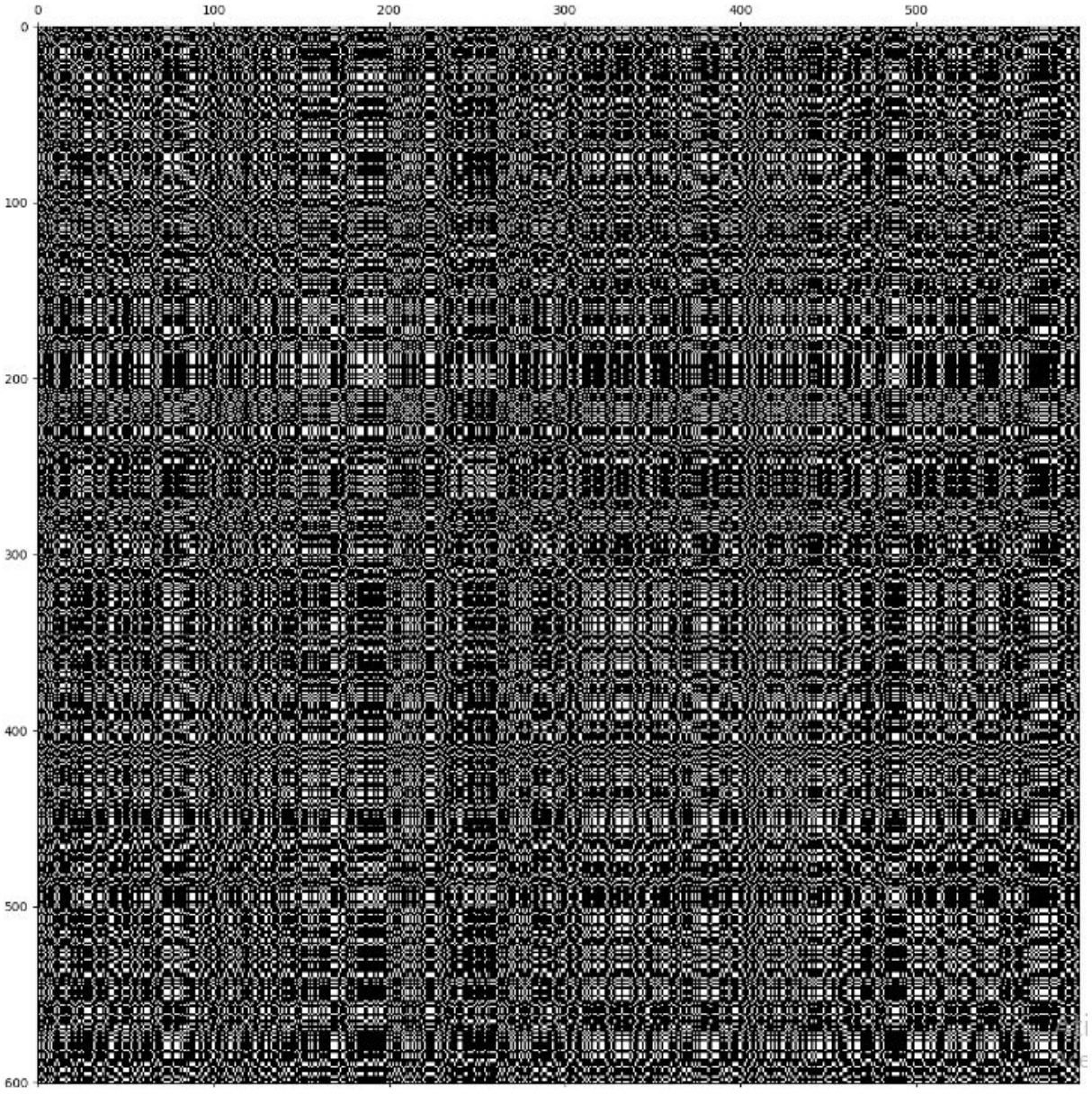
Questão 2:

Pseudocódigo

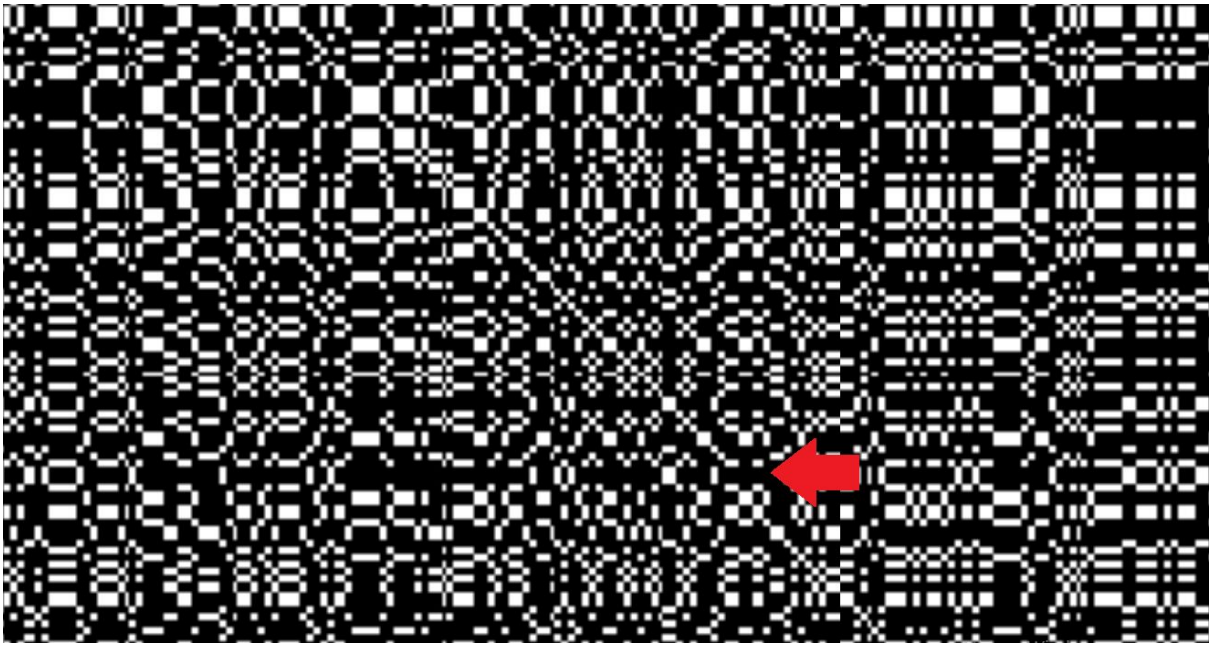
Foi utilizado o editor VS Code pela facilidade de formatação que inclui também o número de linhas.

```
Classe: Alinhador
Recebe os parâmetros:
    seq1 - sequência 1
    seq2 - sequencia 2

Construtor:
    Todos os parâmetros viram atributo do objeto, além dos atributos
    * matrizDePontos    // cujas linhas representa a seq1 e as colunas, seq2
    * mutações  = 0        // parte do algoritmo que não funciona


Métodos:

    alinha():

    Para cada i,j da matrizDePontos:
        Se char i da seq1 == char j da seq2:
            matrizDePontos[i][j] = 1
        Se não:
            matrizDePontos[i][j] = 0
            mutações = mutações + 1
            print caractere i e j mostrando que mutação ocorreu

Abre sequência normal sn
Abre sequência mutada sm

alinhador = Alinhador(sn, sm)
alinhador.alinha()
```

Resultados:



Como podemos ver, a matriz de pontos é muito grande e acaba ficando poluída demais para fazer qualquer análise visual. Porém podemos notar uma característica interessante: em um dado momento, a diagonal é descontinuada e deslocada algumas (duas?) unidades para baixo, o que fortemente sugere o acontecimento de gaps na sequência mutada (que é menor que a sequência original). Segue uma imagem com zoom evidenciando este fenômeno.



Além disso, dado esse deslocamento, não consegui realizar a contagem de mutações pois não tive nenhuma ideia (dentro do meu tempo) de como, algoritmicamente, detectar o shift da diagonal e seguir computando as diferenças (mutações) dos aminoácidos das sequências. Ainda assim, ignorando esse gap, foi implementado no algoritmo um contador de mutações que incrementa toda vez que i == j, mas seq1[i] != seq2[j], e printa esses caracteres diferentes.