Biologia Computacional 2019/2 - Lista de Exercícios 2 parte 2

Aluno: Leonardo Oliveira Wellausen

1a)

```python
import numpy as np

word1 = list('MFVLKGSVVQAFVLLSIVCLEITIADDGVRYVNAEWKRPEQSQEGRHSCTARRLEDNSEEVACSTEVKFRQRAPAEYANKIKKAKDKLRRLESQFDDCQQENDRKDRLIQLQANL
word2 = list('MEKVPGEMEIERRERSEELSEAERKAVQAMWARLYANCEDVGVAILVRFFVNFPSAKQYFSQFKHMEDPLEMERSPQLRKHACRVMGALNTVVENLHDPDKVSSVLALVGKAHAL

#word1 = 'aaaaabb'
#word2 = 'bbccccc'

size1 = len(word1)
size2 = len(word2)

gap = -2
match = 1
mismatch = -1

matrix = np.zeros((size1 + 1, size2 + 1))

def matching(a, b):
    if a == b:
        return match
    else:
        return mismatch

maxval = -1000

for i in range(1, size1 + 1):
    for j in range(1, size2 + 1):
        val = max(matching(word2[j-1], word1[i-1]) + matrix[i-1][j-1], gap + matrix[i-1][j], gap + matrix[i][j-1], 0)
        if val >= maxval:
            maxval = val
        matrix[i][j] = val

```

Como nos exercícios da lista anterior, as sequências a serem alinhadas já estão inseridas no código, assim como os pesos do algoritmo. O algoritmo é implementado em um laço percorrendo e preenchendo a matriz.

```
def back_trace(starti, startj):
    res1 = ''
    res2 = ''
    i = starti
    j = startj
    matches = 0
    while matrix[i][j] != 0:
        up = matrix[i-1][j]
        left = matrix[i][j-1]
        diag = matrix[i-1][j-1]

        if diag >= up and diag >= left:
            res1 += word1[i-1]
            res2 += word2[j-1]
            i -= 1
            j -= 1
            if word1[i] == word2[j]:
                matches += 1
        elif up >= diag and up >= left:
            res1 += word1[i-1]
            res2 += '-'
            i -= 1
        else:
            res1 += '-'
            res2 += word2[j-1]
            j -= 1
    res1 = res1[::-1]
    res2 = res2[::-1]
    print('Cadeias alinhadas: Identidade do alinhamento: %2.2f' % (matches/len(res1)*100) + '%')
    print(res1)
    print(res2)
    print("")
```

O back trace para recuperar os alinhamentos de sequência de interesse pode começar em qualquer ponto da matriz, sendo os pontos de interesse os de valor máximo, e termina ao encontrar um valor 0.

Para cálculo de identidade considerei o comprimento das subsequências alinhadas localmente, com relação ao número de matches dentro delas (valor alto, em geral).

```
for i in range(size1):
    for j in range(size2):
        if matrix[i][j] == maxval:
            back_trace(i, j)

print('Tabela de alinhamento:')
print(matrix)
print('\nScore do alinhamento: %d' % maxval)
```

Como o valor máximo de score na matriz se apresenta diversas vezes, há mais de um alinhamento possível que contém esse score máximo. Todos estes alinhamentos são retornados. Ao final é retornada a tabela e o score máximo.

Resultado:

```
Cadeias alinhadas: Identidade do alinhamento: 100.00%
GALN
GALN

Cadeias alinhadas: Identidade do alinhamento: 83.33%
GALNKV
GALNTV

Cadeias alinhadas: Identidade do alinhamento: 80.00%
Q-VPN
QQVPN

Cadeias alinhadas: Identidade do alinhamento: 100.00%
SEAE
SEAE

Cadeias alinhadas: Identidade do alinhamento: 75.00%
LSEEDRKA
LSEAERKA

Tabela de alinhamento:
[[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.]]

Score do alinhamento: 4
```

1e) Observando que o valor obtido para score máximo foi 4, e as cadeias são extremamente pequenas em relação à sequência original posso concluir duas coisas: ou as sequências possuem baixa semelhança, possuindo muito poucas regiões bem alinhadas; ou os pesos utilizados para o preenchimento da matriz são inadequados para estas sequências.

2a)

```python
import numpy as np

word1 = 'ATAGCAGAGTCTGGGAGGCTGTCCGCTTTTCAGTTTGTCTAAAGGTTAGGGCAGTTGAT
word2 = 'ATAGCAGAGTCTGGGAGGCTGTCCGCTTTTCAGTTTGTCTAAAGGTTAGGGCAGTTGAT

#word1 = 'abcdefghijklmnopqrstuvwxyz'
#word2 = 'abcdefghijklmnopqrstuvwxyz'

size1 = len(word1)
size2 = len(word2)

print(size2, size1)
matrix_points = np.zeros((size1, size2))



for i in range(size1):
    for j in range(size2):
        if word1[i] == word2[j]:
            matrix_points[i][j] = 1
        else:
            matrix_points[i][j] = 0

```

Implementação da matriz de pontos para as sequências fornecidas. Pela tabela final, seria possível observar onde há mutações com base em valores 0 na diagonal principal. Porém não é possível observar gaps, pois não há como diferenciar uma mutação de um gap quanto se observa valores 0 na diagonal.

Para as sequências fornecidas, como há gaps (a sequência mutada tem tamanho menor que a sequência original), caímos nesse caso de não saber onde ocorre mutações e onde ocorre gaps.