Biologia Computacional 2019/2 - Lista de Exercícios 2 parte 1

Aluno: Leonardo Oliveira Wellausen

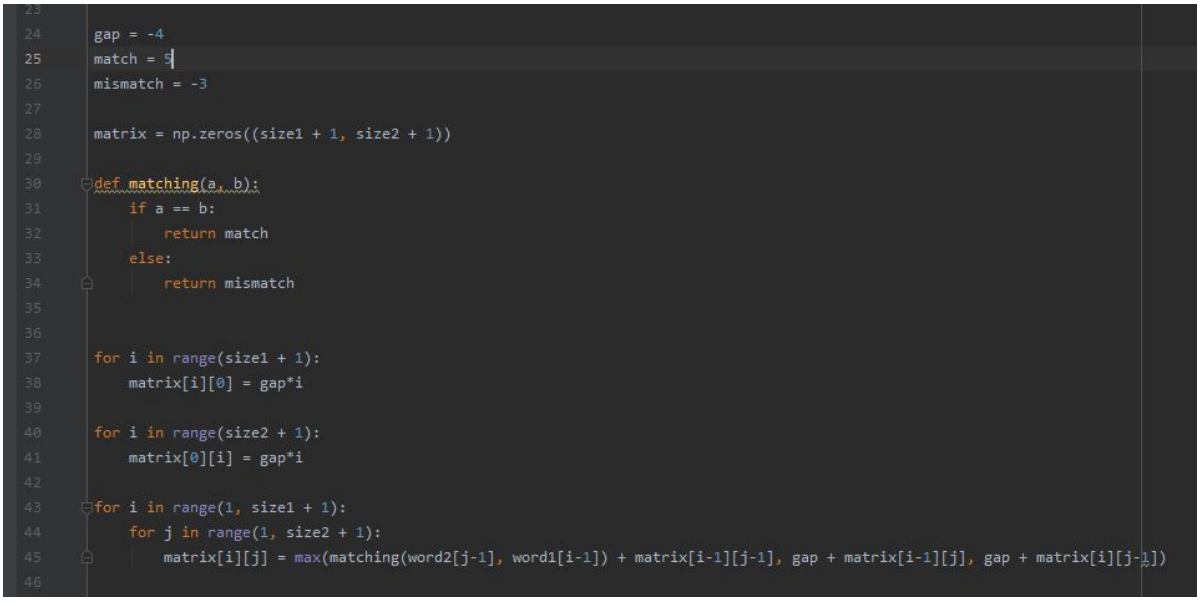
CORREÇÃO ~~

Na primeira entrega havia um erro no meu cálculo de identidade. No laço de back trace todo salto para a diagonal era considerado um match, transformando assim o valor identidade em uma proporção de alinhamentos para gap. As tabelas de resultado (e a conclusão obtida a partir destes) foram atualizadas com os valores corretos, muito menores e mais consistentes com o score (este estava correto).

1a)

```
import numpy as np

hemos = {}

hemos['human'] = 'VLSPADKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLASVSTVLTSKY',
hemos['horse'] = 'VLSAADKTNVKAAWSKVGGHAGEYGAEALERMFLGFPTTKTYFPHFDLSHGSAQVKAHGKKVGDALTLAVGHLDDLPGALSNLSDLHAHKLRVDPVNFKLLSHCLLSTLAVHLPNDFTPAVHASLDKFLSSVSTVLTSKYR',
hemos['deer'] = 'VLSAANKSNVKAAWGKVGGNAPAYGAQALQRMFLSFPTTKTYFPHFDLSHGSAQQKAHGQKVANALTKAQGHLNDLPGTLSNLSNLHAHKLRVNPVNFKLLSHSLLVTLASHLPTNFTPAVHANLNKFLANDSTVLTSKYR',
hemos['cow'] = 'VLSAADKGNVKAAWGKVGGHAAEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHGAKVAAALTKAVEHLDDLPGALSELSDLHAHKLRVDPVNFKLLSHSLLVTLASHLPSDFTPAVHASLDKFLANVSTVLTSKYR',
hemos['pig'] = 'VLSAADKANVKAAWGKVGGQAGAHGAEALERMFLGFPTTKTYFPHFNLSHGSDQVKAHGKKVADALTKAVGHLDDLPGALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHHPDDFNPSVHASLDKFLANVSTVLTSKYR',
hemos['wolf'] = 'VLSPADKTNIKSTWDKIGGHAGDYGGEALDRTFQSFPTTKTYFPHFDLSPGSAQVKAHGKKVADALTTAVAHLDDLPGALSALSDLHAYKLRVDPVNFKLLSHCLLVTLACHHPTEFTPAVHASLDKFFTAVSTVLTSKYR',
hemos['chicken'] = 'MLTAEDKKLIQQAWEKAASHQEEFGAEALTRMFTTYPQTKTYFPHFDLSPGSDQVRGHGKKVLGALGNAVKNVDNLSQAMAELSNLHAYNLRVDPVNFKLLSQCIQVVLAVHMGKDYTPEVHAAFDKFLSAVSAVLAEKYR',
hemos['trout'] = 'XSLTAKDKSVVKAFWGKISGKADVVGAEALGRMLTAYPQTKTYFSHWADLSPGSGPVKKHGGTIMGAIGKAVGLMDDLVGGMSALSDLHAFKLRVDPGNFKILSHNILVTLATHFPSDFTPEVHIAVDKFLAAVSAALADKYR',

print('Opções de entrada: human, horse, deer, cow, pig, wolf, chicken, trout')
word1 = input("Entre uma espécie: ")
word2 = input("Entre outra espécie: ")

word1 = list(hemos[word1])
word2 = list(hemos[word2])

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

Todas as sequências fornecidas estão hard-coded no código fonte, ao se rodar o código deve inserir o nome das duas espécies que se deseja comparar.



```
gap = -4
match = 5
mismatch = -3

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

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


for i in range(size1 + 1):
    matrix[i][0] = gap*i

for i in range(size2 + 1):
    matrix[0][i] = gap*i

for i in range(1, size1 + 1):
    for j in range(1, size2 + 1):
        matrix[i][j] = max(matching(word2[j-1], word1[i-1]) + matrix[i-1][j-1], gap + matrix[i-1][j], gap + matrix[i][j-1])
```

Também está inserido no código a informação de pesos para o algoritmo de Needleman-Wunsch. Com as sequências fornecidas e o pesos podemos executar um laço para preencher a matriz de programação dinâmica com os possíveis scores de alinhamento. Primeiro preenchemos a primeira e última coluna com sucessivos gaps, e após preenchemos o resto considerando matches e mismatches.

```
res1 = ''
res2 = ''
i = size1
j = size2
matches = 0
while i!=0 and 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
        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]
```

Após preenchermos a matriz realizamos o back-trace para encontrar o alinhamento ótimo e montar as sequências resultantes.

```
72      res1 = res1[::-1]
73      res2 = res2[::-1]
74
75      print('\n\nTabela de alinhamento:')
76      print(matrix)
77
78      print('\n\nCadeias alinhadas:')
79      print(res1)
80      print(res2)
81
82      print('\nIdentidade do alinhamento: %2.2f' % (matches/size1*100) + '%')
83      print('\nScore do alinhamento: %d' % (matrix[size1][size2]))
84
```

Por último, exibição dos resultados.

Exemplo para humano e cavalo:

```
Opções de entrada: human, horse, deer, cow, pig, wolf, chicken, trout
Entre uma espécie: human
Entre outra espécie: horse


Tabela de alinhamento:
[[   0.   -4.   -8. ... -556. -560. -564.]
 [  -4.    5.    1. ... -547. -551. -555.]
 [  -8.    1.   10. ... -538. -542. -546.]
 ...
 [-552. -543. -534. ...  550.  546.  542.]
 [-556. -547. -538. ...  559.  555.  551.]
 [-560. -551. -542. ...  555.  564.  560.]]


Cadeias alinhadas:
VLSPA-DKTNVKAAWGKVGAHAGEYGAEALERMFLSFPTTKTYFPHFDLSHGSAQVKGHGKKVADALTNAVAHVDDMPNALSALSDLHAHKLRVDPVNFKLLSHCLLVTLAAHLPAEFTPAVHASLDKFLAS-VSTVLTSKY-
VLSA-ADKTNVKAAWSKVGGHAGEYGAEALERMFLGFPTTKTYFPHFDLSHGSAQVKAHGKKVGDALTLAVGHLDDLPGALSNLSDLHAHKLRVDPVNFKLLSHCLLSTLAVHLPNDFTPAVHASLDKFLS-SVSTVLTSKYR

Identidade do alinhamento: 97.87%

Score do alinhamento: 560
```

Resultados para todos alinhamentos:

Alinhamento (HumanoX)	**Identidade (%)**	**Score**
Cavalo	85.82	560
Veado	74.47	440
Vaca	85.11	560
Porco	82.27	520
Lobo	81.56	507

Galinha	53.90	253
Truta	48.95	220

Utilizando estes pesos para o algoritmo, e considerando ambos os valores de identidade e score, ~~não é óbvio qual das espécies apresenta maior semelhança. Cavalo apresenta ambos valores de identidade e score mais altos, porém Lobo apresenta a maior identidade.~~ notamos que a espécie que apresenta maior similaridade com a humana, tanto em score quanto em identidade, e o cavalo, seguido de perto pela vaca.

2a)

Todo o código é muito semelhante ao primeiro (basicamente igual).

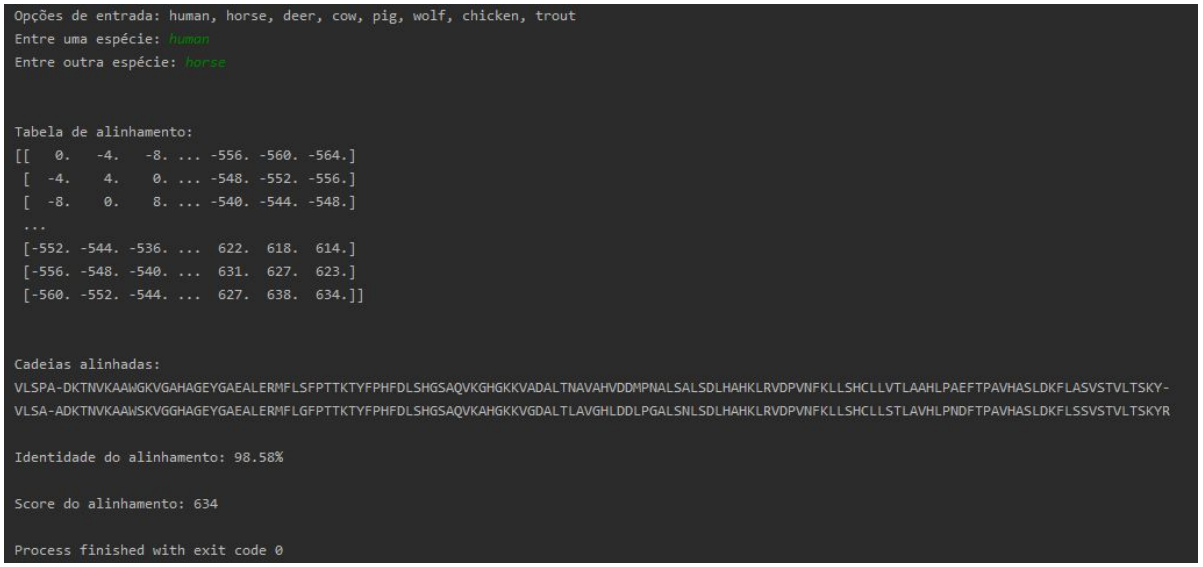
```
15
16  blos = {
17  frozenset(['A','A']):4,
18  frozenset(['A','R']):-1,
19  frozenset(['R','R']):5,
20  frozenset(['N','A']):-2,
21  frozenset(['N','R']):0,
22  frozenset(['N','N']):6,
23  frozenset(['D','A']):-2,
24  frozenset(['D','R']):-2,
25  frozenset(['D','N']):1,
26  frozenset(['D', 'D']):6,
27  frozenset(['C','A']):0,
28  frozenset(['C','R']):-3,
29  frozenset(['C','N']):-3,
30  frozenset(['C', 'D']):-3,
31  frozenset(['C','C']):9,
32  frozenset(['Q','A']):-1,
33  frozenset(['Q','R']):1,
34  frozenset(['Q','N']):0,
35  frozenset(['Q', 'D']):0,
```

A matriz BLOSUN62 é inserida também de forma hard-coded no código.

```
238
239  gap = -4
240
241  matrix = np.zeros((size1 + 1, size2 + 1))
242
243  def blos_matching(a, b):
244      return blos[frozenset([a, b])]
245
246
247  for i in range(size1 + 1):
248      matrix[i][0] = gap*i
249
250  for i in range(size2 + 1):
251      matrix[0][i] = gap*i
252
253  for i in range(1, size1 + 1):
254      for j in range(1, size2 + 1):
255          matrix[i][j] = max(blos_matching(word2[j-1], word1[i-1]) + matrix[i-1][j-1], gap + matrix[i-1][j], gap + matrix[i][j-1])
256
```

Esta matriz BLOSUN62 é agora utilizada para novo cálculo de match e mismatch. O restante do programa é exatamente o mesmo do exercício anterior.

Exemplo para humano e cavalo:



Resultados para todos alinhamentos:

Alinhamento (HumanoX)	**Identidade (%)**	**Score**
Cavalo	86.52	634
Veado	75.18	557
Vaca	85.11	630
Porco	82.98	609
Lobo	82.27	611
Galinha	56.03	437
Truta	52.45*	391*

Truta hacia um aminoácido X não presente na matriz BLOSUN, substituí este aminoácido por algum outro aleatório somente para poder testar.

Nota-se uma grande diferença nos scores por utilizar a matriz nova! Porém a conclusão na qual consigo chegar é a mesma: ~~Lobo possui uma identidade muito grande com o humano, porém Cavalo possui um score maior e identidade também mais alta que os demais, no geral.~~ o cavalo é a espécie que apresenta maior similaridade com a humana, seguida de perto pela vaca, tanto em score quanto em identidade.