ICE 20/21 2º semestre

Trabalho Prático 1

Alinhamento de Sequências

(publicado na 2ª feira, 12 de abril; entrega até domingo, 2 de maio, às 23:59)

1. Introdução

O alinhamento de sequências é um problema que se coloca quando se pretende comparar sequências de "elementos", sujeitas a trocas, inserções e eliminações desses elementos em posições arbitrárias. Neste trabalho, pretende-se resolver o problema do alinhamento de duas sequências de caracteres (*strings*) de forma a maximizar a sua semelhança.

Estes problemas são comuns em biologia quando se comparam sequências de aminoácidos em proteínas ou sequências de bases de ADN (bases A,C,G,T) ou ARN (bases A,C,G,U), para testar uma possível origem comum, ou analisar variantes (por exemplo, dos vírus da COVID). O problema também se coloca na comparação de textos, para se verificar as alterações a que podem estar sujeitos nas suas várias versões.

Para esse efeito, as sequências de caracteres devem ser alinhadas, eventualmente incluindo "hiatos" (*gaps*) antes de se compararem. Por exemplo, analisados os textos **"o bom rato"** e **"o boto raro"**, após o seu alinhamento com a introdução de hiatos (**"-"**)

"o bom- rato"
"o boto raro"

pode concluir-se que houve a troca de um caracter (**"t"** por **"r"**) na palavra final e a inserção de um hiato e uma troca de caracteres na palavra do meio.

A semelhança entre sequências pode ser medida de várias formas, mas vamos assumir o caso mais simples, em que ela é obtida pela soma das semelhanças entre os caracteres na mesma posição. Se os caracteres são idênticos e pelo menos um deles não é um hiato, a semelhança vale +1, caso contrário vale -1. No caso acima a semelhança entre as duas sequências após alinhamento é de +5, pois

"o bom- rato"
"o boto raro"
1111xx111x1 = 8*1-3*1 = +5

Como se pode notar, outros alinhamentos das sequências dariam valores piores para a semelhança. Por exemplo, o alinhamento com um hiato no fim da sequência mais curta teria uma semelhança de -3, pois

"o bom rato-"
"o boto raro"
1111xxxxxxx = 4*1-7*1 = -3

Note que nestes alinhamentos não são consideradas *transposições* (mudança de posição de subsequências) – no exemplo, a maneira mais natural de alinhar as sequências **"o bom rato**" e **"o gato mau"** seria *transpor* a palavra **"bom"** para depois/antes da palavra **"gato"** / **"rato"**, obtendo-se uma semelhança de +2:

"o bom rato"
"o mau gato"
11xxx1x111 = 6*1-4*1 = +2

O melhor alinhamento **sem transposição** seria obtido como em baixo, obtendo-se uma semelhança de -2:

"o b--om rato"
"o gato- mau-"
11xxx1x1x1xx = 5*1-7*1 = -2

2. Algoritmo de Needleman-Wunsch

Neste trabalho pretende-se obter o alinhamento de duas *strings* (sequências de caracteres) que maximiza a sua semelhança, e obter o valor dessa semelhança. Para esse efeito será usado um algoritmo de programação dinâmica, o algoritmo de *Needleman-Wunsch*, constituído por 4 fases:

a) Preparar uma matriz de comparações
b) Preencher a matriz de comparações
c) Obter um caminho na matriz de comparações
d) Alinhar as *strings* com base nesse caminho

O algoritmo é exemplificado no alinhamento das *strings* **M = 'acdea'** e **N = 'abce'**.

a) Fase 1: Preparar uma matriz de comparações

Dadas duas strings **M** e **N** com respetivamente **m** e **n** caracteres, cria-se uma matriz **M**, com **m+1** linhas e **n+1** colunas. Cada elemento **C[i][j]** corresponde ao melhor valor de alinhamento de todos os caracteres de **M** e **N** até às posições **i** e **j**, exclusivé.

Assim, o elemento **C[0][0]** corresponde ao valor inicial de alinhamento, que definimos como nulo **C[0][0] = 0** (nenhuns caracteres foram ainda comparado). Já **C[m][n]** corresponderá ao melhor valor do alinhamento completo das duas sequências (**m** carateres de **M** e **n** caracteres de **N**).

Como justificado à frente, as primeiras linha e coluna da matriz C contêm valores decrescentes de **0** até **-n** e -**m**, respetivamente, como ilustrado ao lado.

M\N		a	b	c	e
	0	-1	-2	-3	-4
a	-1				
c	-2				
d	-3				
e	-4				
a	-5				

b) Preencher a matriz de comparações

Em geral, para determinar o valor de todos os elementos da matriz **C** consideram-se os seus vizinhos à esquerda e acima da seguinte forma.

- Se o alinhamento já tiver incluído os primeiros **i+1** elementos de M e **j** elementos de **N**, **C[i+1][j+1]** corresponderá a avançar a sequência **N**, criando um hiato em **M**, e penalizando o alinhamento em **-1**, isto é,

$$C[i+1][j+1] = C[i+1][j]-1.$$

- Se o alinhamento já tiver incluído os primeiros **i** elementos de **M** e **j+1** elementos de **N**, **C[i+1][j+1]** corresponderá a avançar a sequência **M**, criando um hiato em **N**, e penalizando o alinhamento em **-1**, isto é

$$C[i+1][j+1] = C[i][j+1]-1.$$

- Se o alinhamento já tiver incluído os primeiros **i** elementos de **M** e **j** elementos de **N**, **C[i+1][j+1]** corresponderá a avançar ambas as sequências **M** e **N**, incrementando o valor de **C[i][j]** se **M[i] = N[j]** (são alinhados dois caracteres iguais) ou decrementando-o no caso contrário (se **M[i] ≠ N[j]**). Assim,

$$C[i+1][j+1] = C[i][j] \pm 1.$$

O melhor alinhamento até aos primeiros **i+1** elementos de **M** e **j+1** elementos de **N**, será o melhor dos valores obtidos nos 3 casos considerados acima, isto é:

$$C[i+1][j+1] = \max(C[i+1][j]-1,\ C[i][j+1]-1,\ C[i][j] \pm 1)$$

Aplicando este raciocínio ao exemplo acima, obtemos a matriz apresentada ao lado.

M\N		a	b	c	e
	0	-1	-2	-3	-4
a	-1	1	0	-1	-2
c	-2	0	0	1	0
d	-3	-1	-1	0	0
e	-4	-2	-2	-1	1
a	-5	-3	-3	-2	0

Adicionalmente, ficamos a saber que o melhor alinhamento tem um valor de 0, que é o valor do elemento **`C[m][n] = C[5][4].`**

Apesar de sabermos o valor do melhor alinhamento ainda não o determinamos. Mas esse alinhamento pode ser obtido a partir da matriz C como explicado de seguida.

c) Obter um caminho na matriz de comparações

Para se ter obtido o valor de semelhança em **`C[m][n]`** teve de se proceder a um alinhamento progressivo, avançando quer na sequência M, quer na sequência N, quer nas duas simultaneamente, como explicado atrás. Para se obter o alinhamento que conduziu ao valor na posição **`C[m][n],`** há que fazer o raciocínio inverso.

Mais concretamente, dado um valor **`C[i+1][j+1]`**, esse valor pode ter sido obtido,

- (da esquerda): a partir do valor de **`C[i+1][j]`** subtraindo **`1`**, ou
- (de cima): a partir do valor de **`C[i][j+1]`** subtraindo **`1`**, ou
- (da diagonal): a partir do valor de **`C[i][j]`** somando ou subtraindo **`1`**.

M\N		a	b	c	e
	0	-1	-2	-3	-4
a	-1	1	0	-1	-2
c	-2	0	0	1	0
d	-3	-1	-1	0	0
e	-4	-2	-2	-1	1
a	-5	-3	-3	-2	0

O caminho **`P`** pode então obter-se determinando, por ordem inversa (isto é, a partir do fim), quais as posições a partir das quais se podem ter obtido os valores nas posições **`C[i][j]`**, começando na posição **`C[m][n].`** Em geral, um caminho consistirá em vários pares **(linha, coluna)**, em que o par **`P[i]`** corresponde à linha e coluna de uma posição no caminho **`P.`**

No exemplo, o caminho que se obtém está ilustrado na figura ao lado, correspondendo à sequência de posições

`P = [(0,0),(1,1),(1,2),(2,3),(3,3),(4,4),(5,4)]`

NOTA: Em geral poderá haver mais de um caminho. Neste trabalho admitiremos qualquer caminho que tenha sido obtido seguindo as regras acima.

d) Alinhar as strings com base nesse caminho

A partir do caminho obtido o alinhamento pode ser determinado, obtendo-se duas sequências **`M'`** e N', correspondentes às sequências **`M`** e **`N`** com os hiatos identificados pelo caminho. Mais concretamente, se o caminho **`P`** tiver **`k`** pares, e partindo-se de strings **`M'`** e **`N'`** vazias, elas completam-se em **`k-1`** iterações, variando uma variável **`i`** entre **`1`** e **`k-1`**, da seguinte forma:

- se a linha indicada em **`P[i]`** (i.e. **`P[i][0]`**) for igual à linha em **`P[i-1]`** (i.e. **`P[i-1][0]`**), não existe avanço da string **`M`** e é acrescentado um hiato à string **`M'`**. Caso contrário, a **`M'`** é acrescentada o caracter da *string* **`M`** indicada na linha de **`P[i-1]`** (i.e. **`P[i-1][0]`**);
- se a coluna indicada em **`P[i]`** (i.e. **`P[i][1]`**) for igual à coluna em **`P[i-1]`** (i.e. **`P[i-1][1]`**), não existe avanço da string **`N`** e é acrescentado um hiato à string **`N'`**. Caso contrário, a **`N'`** é acrescentada o caracter da *string* **`N`** indicada na coluna de **`P[i-1]`** (i.e. **`P[i-1][1]`**).

No exemplo acima obter-se-iam as sequências:

`M' = a-cdea`
`N' = abc-e-`

Com efeito, sendo **`P[2][0] = P[1][0] = 1`**, no caminho não houve avanço de linha (o caminho passou da linha 1 coluna 1, para a linha 1 coluna 2), e o caracter **"–"** é acrescentado à string **`M'`** da iteração anterior.

Já **`N'[4]`** = **"e"** porque sendo **`P[5][1] != P[4][1] = 3`**, houve avanço da coluna no caminho (que passou da linha 3 coluna 3, para a linha 4 coluna 4, pelo que o caracter **`N[3]`** = **"e"** foi acrescentado à string **`M'`** da iteração anterior.

Em conclusão, no exemplo apresentado, o melhor alinhamento das sequências **M** e **N** é aquele que é obtido com as sequências **M'** e **N'**, tendo este alinhamento um valor de 0, já que **M'** e **N'** têm os mesmos caracteres em 3 posições, e nas outras 3 posições têm ou caracteres diferentes ou um hiato.

Neste caso, não existem alinhamentos alternativos com o mesmo valor, mas este não é necessariamente o caso noutras sequências. Por exemplo, é fácil de verificar que as sequências **M = 'aba'** e **N = 'bab'** poderiam ser otimamente alinhadas através de quaisquer das sequências

M' = "aba-"	ou	**M'' = "-aba"**
N' = "-bab"		**N'' = "bab-".**

3. Objetivo

Para implementar o algoritmo de *Needleman-Wunsch*, implemente as funções indicadas abaixo. Pode verificar a correção dessas funções com o exemplo usado na secção anterior.

a) Preparar uma matriz de comparações

```
def init_matrix(M, N):
```

Dadas duas strings, **M** e **N**, retorna a matriz inicial de comparações C, conforme explicação na secção 2. a) anterior.

b) Preencher a matriz de comparações

```
def fill_matrix(M, N, C):
```

Dadas duas *strings*, **M** e **N**, e a correspondente matriz inicial de comparações **C**, retorna a matriz **C** completamente preenchida, conforme explicação na secção 2. b) anterior.

c) Obter um caminho na matriz de comparações

```
def path_in_matrix(M, N, C):
```

Dadas duas *strings*, **M** e **N**, e a correspondente matriz de comparações **C**, retorna a lista **P**, definidora de um alinhamento ótimo, conforme explicação na secção 2. c) anterior.

d) Alinhar as strings a partir do caminho na matriz de comparações

```
def align_by_matrix(M, N, P):
```

Dadas duas *strings*, **M** e **N**, a correspondente matriz de comparações **C**, e o caminho **P** obtido nas alíneas anteriores, retornar as strings **M'** e **N'**, que correspondem a um alinhamento ótimo das strings **M** e **N**, conforme explicado na secção 2. d) anterior.

e) Alinhar duas strings

```
def align_sequences(M, N):
```

Dadas duas *strings*, **M** e **N**, usar as funções definidas anteriormente para retornar

1. as strings **M'** e **N'**, que correspondem a um alinhamento ótimo das strings **M** e **N**, obtidas pelo algoritmo de **Needleman-Wunsch**, explicado na secção 2; e
2. o valor do alinhamento assim obtido.

Para as strings **M** e **N'** do exemplo, a função deverá retornar o tuplo:

(('a-cdea', 'abc-e-'), 0)

4. Notas Finais

1. O trabalho deve ser feito em grupos de 2 alunos (que podem não estar inscritos no mesmo turno prático).
2. Cada grupo deverá entregar um ficheiro com as funções implementadas (idealmente serão as 5 funções, mas podem não implementar alguma – com a correspondente penalização na nota). Atenção que as funções deverão ter a assinatura exatamente igual à que é pedida, nomeadamente os nomes das funções devem ser em minúsculas e não devem ter quaisquer acentos.
3. A entrega do trabalho deverá ser feita seguindo as instruções indicadas em separado.
4. As funções implementadas deverão ser testadas o mais exaustivamente possível, e no mínimo com os exemplos do enunciado.
5. A classificação do trabalho tem em conta não só a correção do código das funções, mas também a qualidade do código apresentado, nomeadamente a sua estrutura e facilidade de leitura, medida em aspetos como a documentação das funções pedidas, a escolha do nome das variáveis utilizadas e a decomposição adequada dos problemas.
6. Este trabalho cobre a matéria dada sobre funções, listas e vetores, ciclos e condições. Além do material das aulas, a documentação do Python e o livro de texto serão boas fontes para estudar o que precisam para resolver estes problemas.