Laboratório 2

The longest common subsequence

Deverá utilizar nesta aula o google colab.

1) Realize a importação de algumas dependências para esta aula:

```
import itertools

import numpy as np

import bokeh.io
import bokeh.plotting

bokeh.io.output_notebook()
```

O objetivo desta aula é calcular o comprimento da maior subsequência comum (LCS) entre duas sequências. Vamos começar por definir alguns testes que irão ser utilizados para testar o nosso código. O que é que este código está a testar?

```
def test_lcslen(lcslen):
    """Test a given function to computing LCS"""
    # Define test sequences
    seq1 = 'abcd'
    seq2 = 'abcdefg'
    seq3 = 'adef'

    assert lcslen('', '') == 0
    assert lcslen('', seq1) == 0
    assert lcslen(seq1, seq1) == 4
    assert lcslen(seq1, seq2) == 4
    assert lcslen(seq1, seq3) == 2
    assert lcslen(seq2, seq3) == 4
```

2) A nossa primeira tentativa ser realizar uma implementação naive. Iremos começar até por fazer com que o teste falhe.

```
def naive_lcslen(seq1, seq2):
    """
    Return length of longest common subsequence
    between two subsequences.
    """

    return 0
```

3) Se executarmos esta função irá naturalmente falhar para o nosso teste

```
test_lcslen(naive_lcslen)
```

4) Um algoritmo naive será algo deste género:

   Seja m o comprimento de seq1 e n o comprimento de seq2, com $m \geq n$.
   Passo 1: Gerar todas as subsequências de seq1 com comprimento n .
   Passo 2: Veja se alguma dessas subsequências corresponde à seq2.
   Passo 3: Gere todas as subsequências de seq1 e de seq2 de comprimento $q=n-1$
   Passo 4: Compare-as com conjuntos de subsequências e veja se alguma das subsequências corresponde. Se o fizerem, retornar.
   Passo 5: Vá para a passo 3, diminuindo q por um em cada ciclo até chegarmos a $q=1$.
   Passo 6: Se nenhuma subsequência comum for encontrada quando $q=1$, retorne uma string vazia.

5) Para implementar esta estratégia, podemos reutilizar a função `itertools.combinations(seq, n)`.

```
def allsubseqs(seq, n):
    """Return a set containing all subsequences of length n"""
    return set(itertools.combinations(seq, n))
```

6) Exemplo de funcionamento:

```
allsubseqs('abcd', 3)
```

7) Implementação

```
def naive_lcslen(seq1, seq2, return_lcs=False):
    """Naive implementation of LCS length calculator."""
    # For convenience, make sure seq1 is longer than seq2
    if len(seq1) < len(seq2):
        seq1, seq2 = seq2, seq1

    # Initialize length of subsequence to consider
    q = len(seq2)

    # Compare subsequences for a match
    while q > 0:
        # Generate subsequences
        seq1_subseqs = allsubseqs(seq1, q)
        seq2_subseqs = allsubseqs(seq2, q)

        # Compute the intersections (common subsequences)
        lcs_set = seq1_subseqs.intersection(seq2_subseqs)

        # If the intersection is non-empty, we've found the LCS
```

```
        if len(lcs_set) > 0:
            break

        # Decrement length of subsequence
        q -= 1

    if return_lcs:
        return q, tuple(["".join(lcs) for lcs in lcs_set])
    else:
        return q
```

O que é que este algoritmo na realidade faz?

8) Vamos testar esta código:

```
test_lcslen(naive_lcslen)
```

9) Vamos agora testar com dados biológicos típicos. Para tal, iremos implementar um gerador de dados de proteínas simples

```
rg = np.random.default_rng(3252)

def random_protein_sequence(n):
    """Generate a random protein sequence."""
    return "".join(rg.choice(list("ACDEFGHIKLMNPQRSTVWY"),
size=n))
```

10) Vamos testar agora a solução naíve anterior (varie o n nos testes). O que observa?

```
# Generate random sequences and compute LCS
seq1 = random_protein_sequence(20)
seq2 = random_protein_sequence(24)
len_lcs, lcs = naive_lcslen(seq1, seq2, return_lcs=True)

print("""
      seq1: {seq1:s}
      seq2: {seq2:s}
LCS length: {len_lcs:d}
""".format(seq1=seq1, seq2=seq2, len_lcs=len_lcs))

print("The LCS(s) is/are:", lcs)
```

11) Tendo em conta as aulas, considere a seguinte implementação dinâmica para este problema

```
def dp_lcslen(seq1, seq2):
    """Length of LCS using a dynamic program."""
    # Initialize matrix of substring LCSs.
```

```
    lcs_mat = np.zeros((len(seq1) + 1, len(seq2) + 1), dtype=int)

    # Build matrix, left to right, row by row
    for i in range(1, len(seq1) + 1):
        for j in range(1, len(seq2) + 1):
            if seq1[i - 1] == seq2[j - 1]:
                lcs_mat[i, j] = lcs_mat[i - 1, j - 1] + 1
            else:
                lcs_mat[i, j] = max(lcs_mat[i - 1, j], lcs_mat[i, j - 1])

    return lcs_mat[-1, -1]
```

12) Teste o algoritmo anterior com os testes iniciais

```
test_lcslen(dp_lcslen)
```

13) Teste esta nova implementação para os mesmos valores (usando o gerador de sequências) que utilizou para a naive. O que observa?

14) Tenta em conta os seguintes testes, analise os mesmos. O que se conclui?

```
n = np.arange(2, 1001)
expon_complexity = 2 ** n
polynomial_complexity = n ** 2

p = bokeh.plotting.figure(
    x_axis_label="n",
    y_axis_label="time to compute (a.u.)",
    x_axis_type="log",
    y_axis_type="log",
    frame_height=225,
    frame_width=325,
    x_range=[n.min(), n.max()],
    y_range=[0.1, 1e11],
)

p.line(n, expon_complexity, line_width=2, legend_label="naive")
p.line(n, polynomial_complexity, line_width=2, color='orange',
legend_label="dynamic program")

p.legend.location = 'bottom_right'

bokeh.io.show(p)
```

15) E se quisermos obter a LCS com esta nova abordagem?

```
def dp_lcslen(seq1, seq2, return_lcs=False):
    """Length of LCS using a dynamic program."""
    # Initialize matrix of substring LCSs.
    lcs_mat = np.zeros((len(seq1) + 1, len(seq2) + 1), dtype=int)
```

```
    # Build matrix, left to right, row by row
    for i in range(1, len(seq1) + 1):
        for j in range(1, len(seq2) + 1):
            if seq1[i - 1] == seq2[j - 1]:
                lcs_mat[i, j] = lcs_mat[i - 1, j - 1] + 1
            else:
                lcs_mat[i, j] = max(lcs_mat[i - 1, j], lcs_mat[i, j - 1])

    if return_lcs:
        lcs = ""
        i = len(seq1)
        j = len(seq2)
        while i != 0 and j != 0:
            if lcs_mat[i, j] == lcs_mat[i - 1, j]:
                i -= 1
            elif lcs_mat[i, j] == lcs_mat[i, j - 1]:
                j -= 1
            else:
                lcs = seq1[i - 1] + lcs
                i -= 1
                j -= 1
        return lcs_mat[-1, -1], lcs
    else:
        return lcs_mat[-1, -1]
```

16) Realize uma terceira implementação, seguindo a abordagem de considerar um problema sob uma rede (usando grafos).