Algoritmos para Análise de Sequências Biológicas

Exame de Época Especial Duração 3 horas Cada pergunta do teste vale 4 pontos

12 de Julho de 2024

- Entregue um ficheiro zip com duas pastas:
  1. G1 contendo a resolução do grupo 1, e
  2. G2 contendo a resolução do grupo 2
- Cada pasta deve conter dois ficheiros:
  1. Um ficheiro chamado `resolucao.py` contendo a resolução, e
  2. Um ficheiro chamado `testes.py` com os testes.

Deve ser possível correr:

```
pytest testes.py
```

Para correr todos os testes.

Grupo 1

Pretende-se que implemente um módulo `Python` com algumas funções. Deve entregar um módulo com as funções e um módulo com os testes de unidade que ache necessários para garantir que as funções estão implementadas corretamente.

1. Escreva uma função chamada `cols_conservadas(alinhamento, perc = 0.7, ignore_gap = True)`, que dado um alinhamento de sequências, retorne uma lista com os índices de todas as colunas que sejam conservadas. Define-se como conservada uma coluna que tenha o mesmo símbolo numa proporção de linhas (sequências) superior ou igual a `perc`. O método recebe ainda o argumento `ignore_gap`, que define se os espaçamentos devem ser ignorados neste cálculo (se for True) ou não (se for False);
2. Escreva uma função chamada `perc_conservacao(alinhamento, num, ignore_gap = True)` que devolve a percentagem de conservação máxima para a qual pelo menos `num` colunas sejam conservadas. Tal como na função anterior, esta também recebe o argumento opcional `ignore_gap`;
3. Escreva uma função chamada `maior_inversao(seq)` que recebe uma sequência e devolve um tuplo com três posições:
   1. O tamanho da maior sequência que também aparece invertida;
   2. Uma lista com todos os índices de ocorrência da sequência;
   3. Uma lista com todos os índices de ocorrência da sequência invertida.
4. Escreva uma função chamada `prot2re(proteina)` que recebe uma proteína e devolve a expressão regular que faz match com as possíveis cadeias de DNA que possam ser traduzidas nessa proteína.

Eis um exemplo *básico* da utilização das funções:

```
>>> alin = """
ATGA-AC
AA-AT-C
ATCA-AG
ATGATAG
ATCTCCG
""".splitlines()
>>> alin = [l for l in alin if l.strip()]
>>> cols_conservadas(alin)
[0,1,3,4]
>>> cols_conservadas(alin, ignore_spaces = False)
[0,1,3]
>>> perc_conservacao(alin, 2)
80
>>> perc_conservacao(alin, 4)
75
>>> maior_inversao("ATAGTGATCGAT")
(3, [1], [5, 9])
>>> prot2re('MSRL')
"ATGTC[ACGT]AG[AG](TT[AG]|CT[ACGT])"
```

Grupo 2

Considere o seguinte objeto que representa árvores `UPGMA`. Pretende-se que:

1. Implemente o método `dist` que devolve a distância entre duas sequências na árvore;
2. Implemente testes de unidade para testar os métodos `depth` e `dist`.

```
import math
import pprint
import io

class Tree:
    def __init__(self, *args):
        def is_seq(seq):
            return  type(seq) is str and all(b in "ACGT" for b in seq)
        if len(args) == 1:
            assert is_seq(args[0])
            self.leaf = args[0]
        else:
            assert len(args) == 2
            assert all(type(s) is Tree for s in args)
            self.left, self.right = args
    def dist(self, seq1, seq2):
        """
        Devolve a distância na árvore entre seq1 e seq2 ou infinito
            caso ambas as sequências não se encontrem na árvore"""
        return math.inf
    def depth(self, seq):
        """
        Devolve a profundidade na árvore onde está a sequência seq
            ou infinito caso não esteja na árvore
        """
        if hasattr(self, 'leaf'):
            return 0 if self.leaf == seq else math.inf
```

```
        else:
            return 1 + min(self.left.depth(seq), self.right.depth(seq))
    def __str__(self, indent = ''):
        if hasattr(self, 'leaf'):
            return indent + self.leaf
        return f"""{indent}[
{indent} {self.left.__str__(indent = indent + ' ')}
{indent} {self.right.__str__(indent = indent + ' ')}
{indent}]"""
    def __repr__(self):
        return str(self)


if __name__ == '__main__':
    """
    +-- ACGT
 +--|
 |  +--  ACG
 +-----  ACT
"""
    s1 = Tree('ACGT')
    s2 = Tree('ACT')
    s3 = Tree('ACG')
    b1 = Tree(s1, s3)
    r = Tree(b1, s2)

    print(r.dist('ACG', 'ACGT')) # Devia dar 1
    print(r.dist('ACT', 'ACG')) # Devia dar 2

    T = Tree(Tree(Tree('CGC'),Tree('CGT')), Tree(Tree('ACG'),Tree('ACA')))
```