Biologia Computacional 2019/2 - Lista de Exercícios 3 - Parte II

Aluno: Leonardo Oliveira Wellausen

1) Código:

```
class Node:
    def __init__(self, left, leftval, right, rightval, val):
        self.left = left
        self.leftval = leftval
        self.right = right
        self.rightval = rightval
        self.val = val

    def print(self, gap):
        if self.right is not None:
            print(gap + self.val)
            self.left.print(gap + self.val + "-left----" + str(self.leftval) + '---')
            self.right.print(gap + self.val + "-right----" + str(self.rightval) + '---')
        else:
            print(gap + "OTU:" + str(self.val))

        return
```

Adicionalmente à parte I, essa estrutura contém um peso para cada filho, representando a distância de um nodo a outro. Estrutura utilizada para criação e armazenamento da árvore. Assim como um método para a impressão da mesma.

```
distances = {
    ('gori', 'oran'): 0.1890,
    ('gori', 'huma'): 0.1100,
    ('gori', 'chim'): 0.1130,
    ('gori', 'giba'): 0.2150,
    ('oran', 'huma'): 0.1790,
    ('oran', 'chim'): 0.1920,
    ('oran', 'giba'): 0.2110,
    ('huma', 'chim'): 0.0940,
    ('huma', 'giba'): 0.2050,
    ('chim', 'giba'): 0.2140
}
keys = [k for k in distances.keys()]
for k in keys:
    distances[(k[1], k[0])] = distances[k]

otus = {i[0] for i in distances.keys()}
tree = {i: Node(None, None, None, None, i) for i in otus}
root = Node(None, None, None, None, None)
```

Inicialização da matriz de distâncias (hard-coded) e outras variáveis do algoritmo. OTUs é a lista com todos os OTUs e grupos encontrados. Tree é a árvore montada com raiz root. Todos as entradas no dicionário de distância são espelhados.

```
while len(otus) > 2:
    u_list = compute_ulist(distances, otus)

    new_group = find_lowest(distances, u_list, otus)
    new_otu = '(' + new_group[0] + ',' + new_group[1] + ')'

    v0 = 0.5 * distances[new_group] + 0.5 * (u_list[new_group[0]] - u_list[new_group[1]])
    v1 = 0.5 * distances[new_group] + 0.5 * (u_list[new_group[1]] - u_list[new_group[0]])

    root = tree[new_otu] = Node(tree[new_group[0]], v0,
                                tree[new_group[1]], v1,
                                new_otu)

    otus.remove(new_group[0])
    otus.remove(new_group[1])

    for i in otus:
        distances[(new_otu, i)] = distances[(i, new_otu)] = (distances.pop((i, new_group[0])) + distances.pop((i, new_group[1])) - distances[new_group]) / 2
        distances.pop((new_group[0], i))
        distances.pop((new_group[1], i))

    distances.pop((new_group[0], new_group[1]))
    distances.pop((new_group[1], new_group[0]))

    otus.add(new_otu)
```

Criação da árvore. Primeiro geramos a lista com valores U para cada OTU. Após, encontramos o novo grupo a ser criado (quais OTUs devem ser unidas). Criamos o novo OTU a ser inserido na árvore e inserimos este novo nodo como raiz, considerando os pesos v0 e v1 nas arestas. Então, removemos os dois OTUs unidos da lista de OTUs e também da matriz de distâncias, após ajustarmos as distâncias.

Após o novo nodo criado, atualizamos as distâncias na matriz, já removendo todas entradas referentes aos OTUs removidos. Por último, inserimos o novo grupo na lista de OTUs.

Paramos quando restam dois OTUs na lista, pois isso impossibilita alguns cálculos que causariam divisão por 0.

```
new_group = tuple(otus)
new_otu = '(' + new_group[0] + ',' + new_group[1] + ')'

v0 = distances[new_group]
v1 = distances[new_group]

root = tree[new_otu] = Node(tree[new_group[0]], v0,
                            tree[new_group[1]], v1,
                            new_otu)


root.print('')
```

Após restarem apenas dois grupos de OTUs, os juntamos formando a árvore completa. O termo usado para este ponto de ligação é raiz mesmo sabendo-se que esta árvore, na realidade, não tem raiz.

Imprimimos a árvore

```python
def compute_ulist(dic, lst):
    n = len(lst)
    u_list = {}
    for i_otu in lst:
        u_i = 0
        for j_otu in lst:
            if i_otu == j_otu:
                continue
            u_i += dic[(i_otu, j_otu)]
        u_i = u_i / (n - 2)
        u_list[i_otu] = u_i

    return u_list
```

Função que percorre toda a lista de OTUs calculando o valor de U para cada um, de acordo com a fórmula da etapa 1 do algoritmo nos slides.

```python
def find_lowest(dst, ulst, lst):
    min_v = np.inf
    min_k = ()
    for i_otu in lst:
        ui = ulst[i_otu]
        for j_otu in lst:
            if i_otu == j_otu:
                continue
            val = dst[(i_otu, j_otu)] - ui - ulst[j_otu]
            if val < min_v:
                min_v = val
                min_k = (i_otu, j_otu)

    return min_k
```

Função que encontra o par de OTUs (ou grupo) com menor distância entre si, de acordo com a fórmula da etapa 2 do algoritmo nos slides.

Resultados:

```
C:\Users\lwell\AppData\Local\Programs\Python\Python37\python.exe D:/Documents/CIC/biocomp/lista3/e2.py
(((huma,chim),gori),(giba,oran))
(((huma,chim),gori),(giba,oran))-left----0.038500000000000006---((huma,chim),gori)
(((huma,chim),gori),(giba,oran))-left----0.038500000000000006---((huma,chim),gori)-left----0.006499999999999992---(huma,chim)
(((huma,chim),gori),(giba,oran))-left----0.038500000000000006---((huma,chim),gori)-left----0.006499999999999992---(huma,chim)-left----0.0435---OTU:huma
(((huma,chim),gori),(giba,oran))-left----0.038500000000000006---((huma,chim),gori)-left----0.006499999999999992---(huma,chim)-right----0.0505---OTU:chim
(((huma,chim),gori),(giba,oran))-left----0.038500000000000006---((huma,chim),gori)-right----0.05800000000000001---OTU:gori
(((huma,chim),gori),(giba,oran))-right----0.038500000000000006---(giba,oran)
(((huma,chim),gori),(giba,oran))-right----0.038500000000000006---(giba,oran)-left----0.1178333333333333---OTU:giba
(((huma,chim),gori),(giba,oran))-right----0.038500000000000006---(giba,oran)-right----0.09316666666666669---OTU:oran

Process finished with exit code 0
```

```
C:\Users\lwell\AppData\Local\Programs\Python\Python37\python.exe D:/Documents/CIC/biocomp/lista3/e2.py
(chim,(huma,((oran,giba),gori)))
(chim,(huma,((oran,giba),gori)))-left----0.0505---OTU:chim
(chim,(huma,((oran,giba),gori)))-right----0.0505---(huma,((oran,giba),gori))
(chim,(huma,((oran,giba),gori)))-right----0.0505---(huma,((oran,giba),gori))-left----0.0435---OTU:huma
(chim,(huma,((oran,giba),gori)))-right----0.0505---(huma,((oran,giba),gori))-right----0.006500000000000002---((oran,giba),gori)
(chim,(huma,((oran,giba),gori)))-right----0.0505---(huma,((oran,giba),gori))-right----0.006500000000000002---((oran,giba),gori)-left----0.03850000000000001---(oran,giba)
(chim,(huma,((oran,giba),gori)))-right----0.0505---(huma,((oran,giba),gori))-right----0.006500000000000002---((oran,giba),gori)-left----0.03850000000000001---(oran,giba)-left----0.09316666666666663---OTU:oran
(chim,(huma,((oran,giba),gori)))-right----0.0505---(huma,((oran,giba),gori))-right----0.006500000000000002---((oran,giba),gori)-left----0.03850000000000001---(oran,giba)-right----0.11783333333333336---OTU:giba
(chim,(huma,((oran,giba),gori)))-right----0.0505---(huma,((oran,giba),gori))-right----0.006500000000000002---((oran,giba),gori)-right----0.058---OTU:gori

Process finished with exit code 0
```

```
C:\Users\lwell\AppData\Local\Programs\Python\Python37\python.exe D:/Documents/CIC/biocomp/lista3/e2.py
((chim,(gori,(giba,oran))),huma)
((chim,(gori,(giba,oran))),huma)-left----0.0435---(chim,(gori,(giba,oran)))
((chim,(gori,(giba,oran))),huma)-left----0.0435---(chim,(gori,(giba,oran)))-left----0.0505---OTU:chim
((chim,(gori,(giba,oran))),huma)-left----0.0435---(chim,(gori,(giba,oran)))-right----0.006500000000000002---(gori,(giba,oran))
((chim,(gori,(giba,oran))),huma)-left----0.0435---(chim,(gori,(giba,oran)))-right----0.006500000000000002---(gori,(giba,oran))-left----0.058---OTU:gori
((chim,(gori,(giba,oran))),huma)-left----0.0435---(chim,(gori,(giba,oran)))-right----0.006500000000000002---(gori,(giba,oran))-right----0.03850000000000001---(giba,oran)
((chim,(gori,(giba,oran))),huma)-left----0.0435---(chim,(gori,(giba,oran)))-right----0.006500000000000002---(gori,(giba,oran))-right----0.03850000000000001---(giba,oran)-left----0.1178333333333333---OTU:giba
((chim,(gori,(giba,oran))),huma)-left----0.0435---(chim,(gori,(giba,oran)))-right----0.006500000000000002---(gori,(giba,oran))-right----0.03850000000000001---(giba,oran)-right----0.09316666666666666---OTU:oran
((chim,(gori,(giba,oran))),huma)-right----0.0435---OTU:huma

Process finished with exit code 0
```

Devido ao fato da etapa 2 do algoritmo gerar diversos pares de OTUs que possuem uma mesma distância e à natureza de algumas estruturas de python utilizadas (como dicionários e conjuntos, que não mantém os itens ordenados na mesma ordem de inserção), o algoritmo retorna árvores diferentes (umas 3 possíveis) para execuções distintas, porém todas construídas de forma válida pelo algoritmo.

Entendendo o resultado: Na primeira linha vemos o nodo raiz, que por si só já contém toda a informação necessária para entender os grupos gerados. Esta informação é na forma de nodos representados por pares identados de parênteses, onde “(n1, n2)” é um noso que tem n1 e n2 como filhos esquerdo e direito, respectivamente.

Em seguida montamos a topologia da árvore; em cada linha um nodo é expandido para a esquerda até chegar em uma folha (simbolizada por OTU), e em seguida expandido para a direita. Também é exibida a distância de um nodo até seus filhos.

Exemplo demonstrativo:

(origem do nodo) nodo

(origem do nodo) nodo + filho à esquerda -- distância -- (expande o filho à esquerda)

(origem do nodo) nodo + filho à direita -- distância -- (expande o filho à direita)