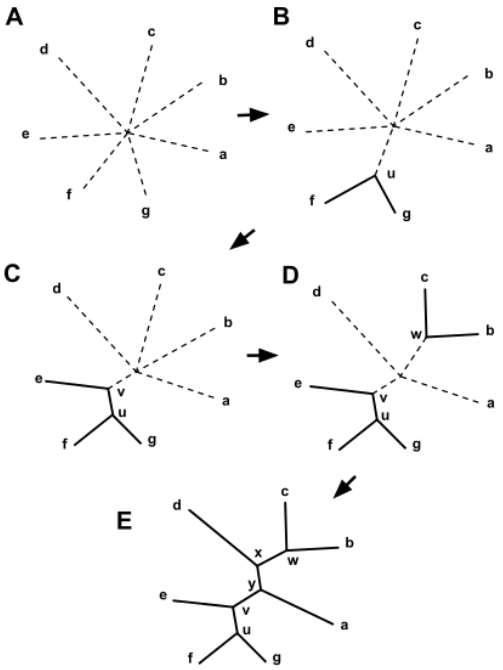
Lista III - Parte II

Construção de árvores filogenéticas; implementação do método Agglomerative methods for ultrametric trees (Neighbour Joining).

Pedro Foletto Pimenta - 00229778
INF05018 - Biologia Computacional - Turma: U (2019/2)
Prof. Márcio Dorn

Neighbour Joining (1987)

- Método hierárquico de clustering para inferência de árvores filogenéticas
- **Entrada:** matriz de distâncias
- **Saída:** árvore* filogenética estimada
  - * "*árvore sem raíz*"
- **Critério de agrupamento:** a cada passo é escolhido o par de clusters com menor média das distâncias de seus elementos.
- Pode ser visto como um algoritmo ganancioso por querer otimizar uma árvore seguindo o critério da "*Evolução Mínima Equilibrada*"



Neighbor Joining: Pseudo-código

1. inicializar lista de clusters
2. enquanto a arvore nao tiver completa:
   a. calcular matriz Q
   b. achar par de cluster com menor valor Q
   c. calcular tamanho das bifurcações do novo nodo da árvore
   d. atualizar matriz de distâncias
   e. fusionar os dois clusters e atualizar lista de clusters
3. retornar a árvore resultante

Entrada: Matriz de distâncias

	Gorila	*Orangotango*	*Humano*	*Chimpanzé*	*Gibão*
Gorila	0	0.1890	0.1100	0.1130	0.2150
Orangotango	0.1890	0	0.1790	0.1920	0.2110
Humano	0.1100	0.1790	0	0.09405	0.2050
Chimpanzé	0.1130	0.1920	0.0940	0	0.2140
Gibão	0.2150	0.2110	0.2050	0.2140	0

Dados obtidos do artigo J Mol Evol. 1982;18(4):225-39. Mitochondrial DNA sequences of primates: tempo and mode of evolution. Brown WM, Prager EM, Wang A, Wilson AC.

Implementação

```
# tree class
class Tree:

    # initialization
    def __init__(self):…

    # distance from this point to the leaves
    def get_leaves_dist(self):…

    # printing function ( CALL THIS )
    def print_tree(self): …


    # auxiliary printing function
    def print_subtree(self, subtree, level, dist):…
```

Implementação

```
# returns the q_matrix generated from dist_matrix
# according to the neighbour joining algorithm
def get_q_matrix(dist_matrix, otu_list):
  # dist_matrix : dicionario com as distancias entre as OTUs
  # otu_list : lista das "OTUs" no passo atual ("clusters")

  # q_matrix is a dict
  q_matrix = {}

  # create a q_matrix element for each dist_matrix element
  for otu_pair in dist_matrix.keys():

    # get otu pair
    otu_a, otu_b = otu_pair

    # sum of the distances for otu_a and otu_b
    sum_a, sum_b = 0, 0
    for otu in otu_list:
      if(otu != otu_a):
        sum_a = sum_a + dist_matrix[(otu_a, otu)]
      if(otu != otu_b):
        sum_b = sum_b + dist_matrix[(otu_b, otu)]

    # q_matrix formula
    q_matrix[otu_pair] = (len(otu_list) - 2) * dist_matrix[otu_pair]
     - sum_a - sum_b

  return q_matrix
```

Implementação

```python
# returns dist_matrix with otu_a and otu_b fused into a new otu
def update_dist_matrix(dist_matrix, otu_list, otu_a, otu_b):
  # dist_matrix : dicionario com as distancias entre as OTUs
  # otu_list : lista das "OTUs" no passo atual ("clusters")
  # otu_a, otu_b : OTUs a serem fusionadas em uma nova OTU

  # add new otu
  new_otu = otu_a + '-' + otu_b
  for otu in otu_list:
    if otu != otu_a and otu != otu_b:
      new_dist = (dist_matrix[(otu, otu_a)] + dist_matrix[(otu,
      otu_b)] - dist_matrix[(otu_a, otu_b)])/2
      new_key = (new_otu, otu)
      dist_matrix[new_key] = new_dist
      new_key = (otu, new_otu)
      dist_matrix[new_key] = new_dist

  # remove otu_a and otu_b
  for otu in otu_list:
    # remove otu_a distances
    key = (otu, otu_a)
    dist_matrix.pop(key, None)
    key = (otu_a, otu)
    dist_matrix.pop(key, None)
    # remove otu_b distances
    key = (otu, otu_b)
    dist_matrix.pop(key, None)
    key = (otu_b, otu)
    dist_matrix.pop(key, None)

  return dist_matrix
```

Implementação

```
# Agglomerative methods for ultrametric trees (Neighbour Joining)
def neighbour_joining(dist_matrix):
  # dist_matrix : dicionario com as distancias entre as OTUs

  # inicializacao da lista de OTUs
  otu_list = []
  for key in dist_matrix:
    if(key[0] not in otu_list):
      otu_list.append(key[0])
    if(key[1] not in otu_list):
      otu_list.append(key[1])

  # initialize tree clusters
  tree_clusters = {}
  for otu in otu_list:
    tree_clusters[otu] = otu

  # enquanto a arvore nao tiver completa
  while(len(otu_list)>1):

    # calcular matriz Q
    q_matrix = get_q_matrix(dist_matrix, otu_list)

    # find smallest distance for clustering
    otu_a, otu_b = min(q_matrix, key=q_matrix.get)

    # branch lenght estimation
    sum_a, sum_b = 0, 0
    for otu in otu_list:
      if(otu != otu_a):
        sum_a = sum_a + dist_matrix[(otu_a, otu)]
      if(otu != otu_b):
        sum_b = sum_b + dist_matrix[(otu_b, otu)]
    branch_lenght_a = (dist_matrix[(otu_a, otu_b)])/2 + (1.0/(2*len(otu_list) - 2))*
    (sum_b - sum_a)
    branch_lenght_b = (dist_matrix[(otu_a, otu_b)]) - branch_lenght_a

    # update distance matrix
    dist_matrix = update_dist_matrix(dist_matrix, otu_list, otu_a, otu_b)

    # update OTU list
    new_otu = otu_a + '-' + otu_b
    otu_list.append(new_otu)
    otu_list.remove(otu_a)
    otu_list.remove(otu_b)

    # update tree : new tree node
    new_tree_node = Tree()
    new_tree_node.right = tree_clusters[otu_a]
    new_tree_node.right_dist = branch_lenght_a
    new_tree_node.left = tree_clusters[otu_b]
    new_tree_node.left_dist = branch_lenght_b

    # update tree clusters
    tree_clusters.pop(otu_a)
    tree_clusters.pop(otu_b)
    tree_clusters[new_otu] = new_tree_node

  return tree_clusters[otu_list[0]]
```

Implementação - "main"

```
# construcao de uma arvore ultrametrica filogenetica a partir da
matriz de distancias usando UPGMA
tree = neighbour_joining(dist_matrix)

# printar resultados
print("printing resulting tree:")
tree.print_tree()
```

Resultado

```
pfpimenta@s-72-204-14:~/biocomp2019$ python lista3parte2/e3-2.py
printing resulting tree:
    -
                -   - 0.11475 -   -   ora
    -0.01925-
                -   - 0.09625 -   -   gib
    -
    -
        -
                        -   -   - 0.0493583333333 -   -   -       hum
        --0.04511875--
                        -   -   - 0.0446916666667 -   -   -       chi
        -
    -0.01925-
                -   - 0.01935625 -   -       gor
    -
```

Resultado

A partir do resultado podemos ver quais as OTUs mais próximas:

Chimpanzé é relativamente próximo de Humano
Orangotango é relativamente próximo de Gibão
entre as 5 OTUs, o Gorila é o mais diferente

**OBS: também foi implementada uma versão onde as distâncias são arredondadas, mas isso não retorna bons resultados se a tabela de distâncias for normalizada entre 0 e 1*

Lista III - Parte II

Construção de árvores filogenéticas; implementação do método Agglomerative methods for ultrametric trees (Neighbour Joining).

Pedro Foletto Pimenta - 00229778
INF05018 - Biologia Computacional - Turma: U (2019/2)
Prof. Márcio Dorn