Bibliotecas utilizadas inicialmente

```
import csv
import numpy as np
import pandas as pd
```

Preenchimento da matriz: foi realizado de duas formas:

Forma 1: importação por .csv

Importação

Conversão de string para float dos valores das células da matriz

```
# Preenchimento da matriz
matriz_neighbor_joining = np.zeros((8, 8)) # A-H X A-H

with open("matriz distancia - Q1.csv") as fp:
    reader = csv.reader(fp, delimiter=",", quotechar='"')
    # next(reader, None)  # skip the headers
    matriz_neighbor_joining = [row for row in reader]

for i in range(8):
    for j in range(8):
        matriz_neighbor_joining[i][j] = float(matriz_neighbor_joining[i][j])

```

Forma 2: atribuição manual

```
matriz_neighbor_joining[0][1] = 2.2266620
matriz_neighbor_joining[0][2] = 3.3353841
matriz_neighbor_joining[0][3] = 3.0926941
matriz_neighbor_joining[0][4] = 1.8896296
matriz_neighbor_joining[0][5] = 4.9589915
matriz_neighbor_joining[0][6] = 4.3943885
matriz_neighbor_joining[0][7] = 5.4122139
matriz_neighbor_joining[1][0] = matriz_neighbor_joining[0][1]
matriz_neighbor_joining[1][2] = 1.1567076
matriz_neighbor_joining[1][3] = 0.9140176
matriz_neighbor_joining[1][4] = 1.3807486
matriz_neighbor_joining[1][5] = 4.4501105
matriz_neighbor_joining[1][6] = 3.8855074
matriz_neighbor_joining[1][7] = 4.9033329
matriz_neighbor_joining[2][0] = matriz_neighbor_joining[0][2]
matriz_neighbor_joining[2][1] = matriz_neighbor_joining[1][2]
matriz_neighbor_joining[2][3] = 1.8198256
matriz_neighbor_joining[2][4] = 2.4894707
matriz_neighbor_joining[2][5] = 5.5588327
matriz_neighbor_joining[2][6] = 4.9942296
matriz_neighbor_joining[2][7] = 6.0120551
matriz_neighbor_joining[3][0] = matriz_neighbor_joining[0][3]
matriz_neighbor_joining[3][1] = matriz_neighbor_joining[1][3]
matriz_neighbor_joining[3][2] = matriz_neighbor_joining[2][3]
matriz_neighbor_joining[3][4] = 2.2467807
matriz_neighbor_joining[3][5] = 5.3161427
matriz_neighbor_joining[3][6] = 4.7515396
matriz_neighbor_joining[3][7] = 5.7693651
matriz_neighbor_joining[4][0] = matriz_neighbor_joining[0][4]
matriz_neighbor_joining[4][1] = matriz_neighbor_joining[1][4]
matriz_neighbor_joining[4][2] = matriz_neighbor_joining[2][4]
matriz_neighbor_joining[4][3] = matriz_neighbor_joining[3][4]
matriz_neighbor_joining[4][5] = 3.3471233
matriz_neighbor_joining[4][6] = 2.7825202
matriz_neighbor_joining[4][7] = 3.8003457
matriz_neighbor_joining[5][0] = matriz_neighbor_joining[0][5]
matriz_neighbor_joining[5][1] = matriz_neighbor_joining[1][5]
matriz_neighbor_joining[5][2] = matriz_neighbor_joining[2][5]
matriz_neighbor_joining[5][3] = matriz_neighbor_joining[3][5]
matriz_neighbor_joining[5][4] = matriz_neighbor_joining[4][5]
matriz_neighbor_joining[5][6] = 1.1966232
matriz_neighbor_joining[5][7] = 2.2144487
matriz_neighbor_joining[6][0] = matriz_neighbor_joining[0][6]
matriz_neighbor_joining[6][1] = matriz_neighbor_joining[1][6]
matriz_neighbor_joining[6][2] = matriz_neighbor_joining[2][6]
matriz_neighbor_joining[6][3] = matriz_neighbor_joining[3][6]
matriz_neighbor_joining[6][4] = matriz_neighbor_joining[4][6]
matriz_neighbor_joining[6][5] = matriz_neighbor_joining[5][6]
matriz_neighbor_joining[6][7] = 1.2176200
matriz_neighbor_joining[7][0] = matriz_neighbor_joining[0][7]
matriz_neighbor_joining[7][1] = matriz_neighbor_joining[1][7]
matriz_neighbor_joining[7][2] = matriz_neighbor_joining[2][7]
matriz_neighbor_joining[7][3] = matriz_neighbor_joining[3][7]
matriz_neighbor_joining[7][4] = matriz_neighbor_joining[4][7]
matriz_neighbor_joining[7][5] = matriz_neighbor_joining[5][7]
matriz_neighbor_joining[7][6] = matriz_neighbor_joining[6][7]
```

Obtenção dos valores das somas das distâncias (ex: soma das distâncias da Seq A com todas as demais seq, incluindo a si mesma). Salva em um vetor

```
print(matriz_neighbor_joining)
# Cálculo da coluna de Distância total
vetor_dist_total = []
for x in range (8):
    n = 0
    for y in range (8):
        n = n + matriz_neighbor_joining[x][y]
    vetor_dist_total.append(n)

```

Criação da matriz de distância

```
matriz_distancia = np.zeros((8, 8)) # A-H X A-H

```

Preenchimento da matriz de distância baseada nas distâncias totais

```
for k in range(8): # linhas
    for w in range(8): # colunas
        if(k != w):
            matriz_distancia[k][w] = matriz_neighbor_joining[k][w] - (vetor_dist_total[k] + vetor_dist_total[w])/(8-2)

```

CICLO DE REPETIÇÃO

Encontrar o menor valor dentro da matriz e salvar em coordenadas linha e coluna

```
# PRIMEIRA RODADA
# Encontra a posição de menor distância
menor = 0
coluna, linha = 0, 0
for k in range(8): # linhas
    for w in range(8): # colunas
        if(matriz_distancia[k][w] < menor):
            menor = matriz_distancia[k][w]
            coluna = w # Guarda coluna
            linha = k # Guarda linha

```

Cálculo das distâncias dos ramos. Embora o código seja capaz de calcular, não sei como se atribui estes valores à árvore. Logo, estes dados são perdidos com a execução.

```
S1 = (matriz_distancia[linha][coluna]/2) + ((vetor_dist_total[linha] - vetor_dist_total[coluna])/(2*6))
S2 = matriz_distancia[linha][coluna] - S1
```

Criação de uma matriz no Pandas para poder observar a matriz com a identificação por letra (A a H). Inserção de uma coluna nova, que dará origem à coluna com as sequências mais próximas em ambas matrizes.

```
MD = pd.DataFrame(matriz_distancia)
MD = MD.rename(columns={0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',6:'G', 7: 'H'}) # Muda os nomes das colunas
MD.index = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', ] # Muda os nomes das linhas

MD.insert(8,"CD", [1.0, 1, 1, 1, 1, 1, 1, 1], True) # Adicionando coluna
matriz_distancia = np.append(matriz_distancia, [[1], [1], [1], [1], [1], [1], [1], [1]], axis=1) # Adicionando coluna
```

Criação das variáveis que guardarão a árvore newick

```
coluna_nova = []
newick = ""
newick2 = ""
newick = newick + "(C, D)"
```

Cálculo dos valores da nova coluna que junta as sequências mais próximas. Atribuição, inicialmente, a um vetor e, posteriormente, à coluna na matriz.

```
for i in range(8): # Existem 8 linhas por enquanto
    coluna_nova.append([matriz_distancia[i][coluna] + matriz_distancia[i][linha] - matriz_neighbor_joining[linha][coluna]][0]/2.0) # Salva os valores da nova coluna

for i in range(8):
    MD["CD"][i] = coluna_nova[i] #Atribui os novos valores
    matriz_distancia[i][8] = coluna_nova[i] #Atribui os novos valores
```

Deleção das colunas e linhas repetidas (redundantes)

```
MD = MD.drop(MD.index[linha])
MD = MD.drop(MD.index[coluna-1])
MD = MD.drop(columns=["C"])
MD = MD.drop(columns=["D"])

matriz_distancia = np.delete(matriz_distancia, coluna, 0) # Deleta a linha H
matriz_distancia = np.delete(matriz_distancia, linha, 0) # Deleta a linha G
matriz_distancia = np.delete(matriz_distancia, coluna, 1) # Deleta a coluna H
matriz_distancia = np.delete(matriz_distancia, linha, 1) # Deleta a coluna G
print(MD)
```

Obs: este ciclo se repete 6 vezes
Então começa o último estágio: resultados.
mostra a árvore no formato NEWICK e uma árvore simples através do Biopython.

```
# SÉTIMA RODADA

newick = "(" + newick2 + newick + ")"
print(newick)

from Bio import Phylo
from Bio.Phylo.PhyloXML import Phylogeny
from io import StringIO

handle = StringIO(newick)
tree = Phylo.read(handle, "newick")
Phylo.draw(tree)
```

Este código ficou bem menos genérico que o UPGMA, funcionando apenas para este caso específico. Ainda, conforme a forma de entrada da matriz de distância, os resultados finais foram diferentes e não sei o motivo.

Não tive tempo para organizar o código de forma modularizada por funções, como o UPGMA, nem entendi como atribuir o valor das distâncias dos ramos.

Estas atividades são muito complexas comparadas ao que os alunos da Biotecnologia foram preparados em Fundamentos de Algoritmos e Algoritmos e Programação, o primeiro (e provavelmente único) contato com programação.