Bioinformatika Lab. 1

Atliko: Rytis Kaplūnas, PS 6, 4 Kursas

In [1]:

```
from Bio import SeqIO
from Bio.Seq import Seq
from Bio.Alphabet import generic_dna, generic_protein
from bio_utils import *
from Bio import Align
from scipy.spatial import distance
import pandas as pd
from scipy.spatial import distance_matrix
import numpy as np
import glob
import os
from os import listdir
from os.path import isfile, join
from Bio.SeqUtils.ProtParam import ProteinAnalysis
```

Konstantos

In [2]:

```
start = ["ATG"]
stop = ["TAA", "TAG", "TGA"]
convert = {"C" : "G", "G" : "C", "T" : "A", "A" : "T"}
data_path = "./data/"
output_path = "./out/"
file_format = "fasta"
proteins_out = output_path + "proteins.txt"
```

FASTA sekų nuskaitymas iš failo

In [3]:

```
sequences = []
onlyfiles = [f for f in listdir(data_path) if isfile(join(data_path, f))]
for file in onlyfiles:
    if os.path.splitext(file)[1] == '.fasta':
        rec = SeqIO.read(data_path + file, file_format)
        sequences.append({'seq': rec.seq, 'name' : os.path.splitext(file)[0]})
```

Sekos apvertimo funkcijos

In [4]:

```
def complement(data):
    compl = []
    for var in data:
        compl.append(convert[var])
    return compl

def reverse(data):
    return "".join(list(reversed(data)))
```

Funkcija rasti ORF'ams sekoje

In [5]:

```
def find_orfs(seq, st, numb, revert=False):
    frame = "-" if revert else ""
    frame += str(st)
    orfstart = -1
    orfend = -1
    orfs = []
    dna = seq['seq']
    data = reverse(complement(dna)) if revert else dna
    for index in range(st, len(data), 3):
        codon = data[index:index+3]
        last_start = 0
        if codon in start:
            if orfstart == -1:
                orfstart = index
        if codon in stop:
            orfend = index + 3
            if orfend > orfstart and orfstart != -1:
                value = data[orfstart:orfend]
                if orfend - orfstart > 100:
                    length =(orfend - orfstart)
                    name = seq['name']+str(len(orfs)+numb+1)
                    orfs.append(Orf(orfstart, orfend, frame, value, length, name))
                orfstart = -1
    return orfs
```

Funkcija rasti visiems ORF'ams

In [6]:

```
def getAllOrfs(dna, start=0):
    orfs = []
    numb = start
    # +1, +2 ir +3 poslinkio frame'ai
    for frame in range(0, 3):
        orfs.extend(find_orfs(dna, frame, numb))
        numb = len(orfs) + start
    # -1, -2 ir -3 poslinkio frame'ai
    for frame in range(0, 3):
        orfs.extend(find_orfs(dna, frame, numb, True))
        numb = len(orfs) + start
    return orfs
```

Funkcija išskirianti baltymus iš rastų ORF'ų

In [7]:

```
def to_protein(orf):
    data = Seq(str(orf.value), generic_dna)
    return data.translate(to_stop=True)

def to_proteins(orfs):
    proteins = []
    for orf in orfs:
        protein = to_protein(orf)
        proteins.append(protein)
    return proteins
```

In [8]:

```
def format_data(names, orfs):
    value = ""
    for name in names:
        orf = next(orf for orf in orfs if orf.name == name)
        value += "{}:\n\tORF: {}\n\tBaltymas: {}\n\n".format(name, orf.value, to_protein(orf))
    return value
```

In [21]:

```
orfs = []
for seq in sequences:
    orfs.extend(getAllOrfs(seq, len(orfs)))

proteins = to_proteins(orfs)
writeToFile(proteins_out, proteins)

print("Rastų ORF'ų skaičius: {}".format(len(orfs)))
```

```
Rastų ORF'ų skaičius: 35
```

Kodonų ir Dikodonų dažnių skaičiavimai

In [22]:

```
CodonsDict = {
 'TTT': 0, 'TTC': 0, 'TTA': 0, 'TTG': 0, 'CTT': 0,
 'CTC': 0, 'CTA': 0, 'CTG': 0, 'ATT': 0, 'ATC': 0,
 'ATA': 0, 'ATG': 0, 'GTT': 0, 'GTC': 0, 'GTA': 0,
 'GTG': 0, 'TAT': 0, 'TAC': 0, 'TAA': 0, 'TAG': 0,
 'CAT': 0, 'CAC': 0, 'CAA': 0, 'CAG': 0, 'AAT': 0,
 'AAC': 0, 'AAA': 0, 'AAG': 0, 'GAT': 0, 'GAC': 0,
 'GAA': 0, 'GAG': 0, 'TCT': 0, 'TCC': 0, 'TCA': 0,
 'TCG': 0, 'CCT': 0, 'CCC': 0, 'CCA': 0, 'CCG': 0,
 'ACT': 0, 'ACC': 0, 'ACA': 0, 'ACG': 0, 'GCT': 0,
 'GCC': 0, 'GCA': 0, 'GCG': 0, 'TGT': 0, 'TGC': 0,
 'TGA': 0, 'TGG': 0, 'CGT': 0, 'CGC': 0, 'CGA': 0,
 'CGG': 0, 'AGT': 0, 'AGC': 0, 'AGA': 0, 'AGG': 0,
 'GGT': 0, 'GGC': 0, 'GGA': 0, 'GGG': 0
}
```

In [23]:

```
amino_acids = [
    'A', 'C', 'E', 'D', 'G',
    'F', 'I', 'H', 'K', 'M',
    'L', 'N', 'Q', 'P', 'S',
    'R', 'T', 'W', 'V', 'Y']
DicodonsDict = {}
for i in range(0, len(amino_acids)):
    for j in range(0, len(amino_acids)):
        dicodon = amino_acids[i] + amino_acids[j]
        DicodonsDict[dicodon] = 0
```

Suskaičiuojami kodonų dažniai kiekvienama ORF'e

In [24]:

```
def count_codon_frequencies(orfs):
    all_codon_freqs = []
    for o in orfs:
        codons = CodonsDict.copy()
        for index in range(0, len(o.value), 3):
            codon = o.value[index:index+3]
            codons[str(codon)] += 1
        all_codon_freqs.append(codons)
    all_codon_frequencies = []
    for cf in all_codon_freqs:
        numb = 0
        freqs = []
        for codon, freq in cf.items():
            numb += freq
        for codon, freq in cf.items():
            freqs.append(freq/numb)
        all_codon_frequencies.append(freqs)
    return all_codon_frequencies
```

Suskaičiuojami dikodonų dažniai kiekviename ORF'e

In [25]:

```
def get_dicodon_freq_dict(proteins):
    all_dicodon_freqs = []
    for protein in proteins:
        dicodons = DicodonsDict.copy()
        for frame in range(0,3):
            for index in range(frame, len(protein), 2):
                if index+2 <= len(protein):
                    dicodon = protein[index:index+2]
                    dicodons[str(dicodon)] += 1
        all_dicodon_freqs.append(dicodons)
    return all_dicodon_freqs
```

In [26]:

```
def get_dicodon_frequencies(all_dicodon_freqs):
    all_dicodon_frequencies = []
    for df in all_dicodon_freqs:
        numb = 0
        freqs = []
        for dicodon, freq in df.items():
            numb += freq
        for dicodon, freq in df.items():
            freqs.append(freq/numb)
        all_dicodon_frequencies.append(freqs)
    return all_dicodon_frequencies
```

In [27]:

```
def count_dicodon_frequencies(proteins):
    all_dicodon_freq_dict = get_dicodon_freq_dict(proteins)
    return get_dicodon_frequencies(all_dicodon_freq_dict)
```

Atstumų matricų skaičiavimai

Įvertinamas atstumas tarp dviejų kodonų/dikodonų dažnių vektorių pagal Euklidinio atstumo formulę

In [28]:

```
def get_scores(values):
    arr = np.zeros((len(values), len(values)))
    for i in range(0, len(values)):
        for j in range(0, len(values)):
            arr[i,j] = distance.euclidean(values[i], values[j])
    return arr
```

In [29]:

```
codon_scores = get_scores(count_codon_frequencies(orfs))
dicodon_scores = get_scores(count_dicodon_frequencies(proteins))
```

Sudedama atstumų matrica į DataFrame

In [30]:

```
def calculate_distances(scores):
    data = np.asarray(scores)
    df = pd.DataFrame(data)
    distances = pd.DataFrame(distance_matrix(df.values, df.values))
    return distances
```

Išsaugoma atstumų matrica Phylip formatu

In [31]:

```
def save_matrix_to_phy(distances, phy_name):
    phylip_out = output_path + phy_name + "_phylip.phy"
    rows = list(map(lambda x: x.name, orfs))
    rows = np.array(rows)[:, np.newaxis]
    values = np.hstack((rows, distances.values))
    header = str(len(rows))
    np.savetxt(phylip_out, values, fmt='%s', header=header, comments='')
```

In [32]:

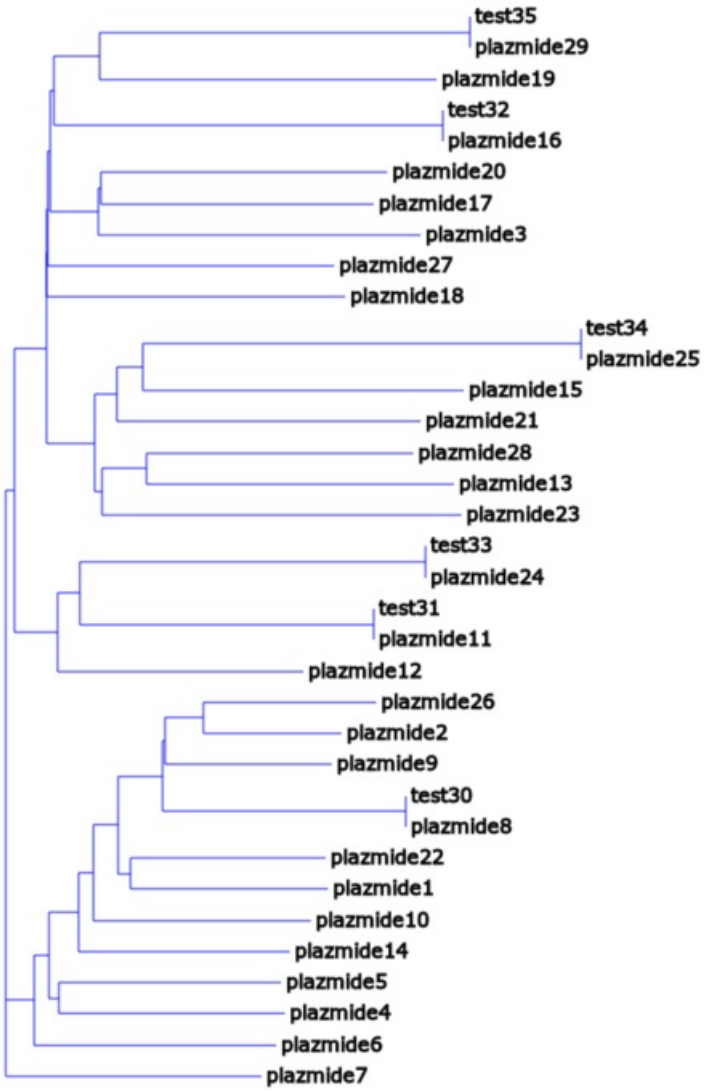
```
dicodon_dist = calculate_distances(dicodon_scores)
save_matrix_to_phy(dicodon_dist, "dicodon")
```

In [33]:

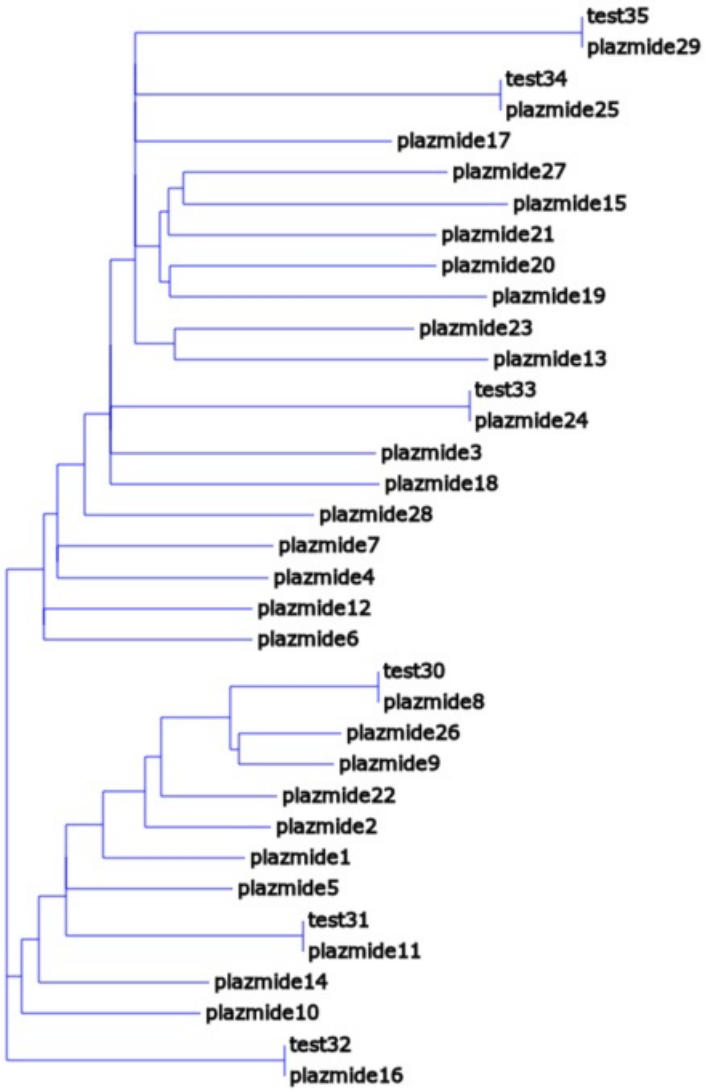
```
codon_dist = calculate_distances(codon_scores)
save_matrix_to_phy(codon_dist, "codon")
```

Sugeneruoti medžiai pagal gautas atstumų matricas

Kodonų atstumų matricos medis:



Dikodonų atstumų matricos medis:



Išvados

Pagal kodonų dažnumą, iš medžio matyti, kad test34 ir plazmide25 išsiskyrė labiausiai iš kitų ORF'ų: jie yra labiausiai nutolę nuo šaknies.Dikodonų atstumų matricos medis parodo, kad labiausiai išsiskyrė test35 ir plazmide29. Tačiau test34 ir plazmide25 (labiausiai išsiskyrusios pagal kodonų medį) taip pat nemažai išsiskyrė iš kitų.

In [42]:

```
print(format_data(["test34", "plazmide25", "test35", "plazmide29"], orfs))
```

```
test34:
        ORF: ATGGCAAATTCCTTGACCTTGGTGGAGTGGGATGAGAACTTGACCACAGCGGGCTCCCGGCTGAAGCTGTCGTTCCCGCAGG
ACTCGCAGCCGTCGTCGTACTGGTAG
        Baltymas: MANSLTLVEWDENLTTAGSRLKLSFPQDSQPSSYW

plazmide25:
        ORF: ATGGCAAATTCCTTGACCTTGGTGGAGTGGGATGAGAACTTGACCACAGCGGGCTCCCGGCTGAAGCTGTCGTTCCCGCAGG
ACTCGCAGCCGTCGTCGTACTGGTAG
        Baltymas: MANSLTLVEWDENLTTAGSRLKLSFPQDSQPSSYW

test35:
        ORF: ATGCAGCTTGGAAGTCAAAGGGAGCGGCCGAGCCAGGAACCACAGAACTCTGGAGCAGAGACCCTGCGACCACGATCCTGTC
ATAGGCGCAGTTCGTGGACGTAG
        Baltymas: MQLGSQRERPSQEPQNSGAETLRPRSCHRRSSWT

plazmide29:
        ORF: ATGCAGCTTGGAAGTCAAAGGGAGCGGCCGAGCCAGGAACCACAGAACTCTGGAGCAGAGACCCTGCGACCACGATCCTGTC
ATAGGCGCAGTTCGTGGACGTAG
        Baltymas: MQLGSQRERPSQEPQNSGAETLRPRSCHRRSSWT
```