Abstract

Sequence Alignment is a sub problem in bioinformatics that takes root in sequence processing from computer science, the basic idea from computer science is finding matches or mismatches or positions or patterns in any number of given sequences.
What is added in Sequence Alignment is the value derived from a certain result of sequence processing and its meaning in biology.
A certain match or difference in alignment between two characters in a sequence could have a different implication from another when it comes to sequences that represent DNA or Proteins.
What is important to know is that alignments have certain scores for each match or mismatch between two characters from each two sequences being aligned, when added up they compose the ultimate score for a certain alignment between the two sequences being aligned.

```
# Crude FastA Implementation
# Author: Aly Shmahell
# Copyrights: Aly Shmahell 2017
# Usage: python fasta.py <database filename> <sequences filename>
# Example: python fasta.py ./databaseFiles/database ./sequenceFiles/sequences

import sys
import time
import datetime

scoreMatrix = []
database = []
sequences = []
alphabet = []

# retrieves 'correspondent pairwise substitution score' from the score matrix
def findScore(nucSeq, nucDB):
  return scoreMatrix[alphabet.index(nucSeq)][alphabet.index(nucDB)]

# display and log
def output(array,score,computeTime):
  string = ""
  for value in array:
    string+=str(value)
  string += (" ... sequence scored: " + str(score) + " ... time to compute: " + str(computeTime) + " seconds \n")
  print string
  with open('log.txt','a') as log:
    log.write(string)

# logging session upstart
def log():
  with open('log.txt','a') as log:
    log.write("Session started at: " + str(datetime.datetime.now()) + "\n")

def fasta():
  # starting session log
  log()
  for inputSequence in sequences:
    startTime = time.time()
    maxScore = -1000000000
    highScoreSequence = []
    # populating seqWordTable
    seqWordTable = [[] for i in range(len(alphabet))]
    for nucSeqPos,nucSeq in enumerate(inputSequence):
      seqWordTable[alphabet.index(nucSeq)].append(nucSeqPos)
    # testing each database sequence against the input sequence at hand
    for databaseSequence in database:
      subScore = -1000000000
      maxScore = max(maxScore,subScore)
      # populating dbWordTable
      dbWordTable = [[] for i in range(len(alphabet))]
      for nucDbPos,nucDb in enumerate(databaseSequence):
        dbWordTable[alphabet.index(nucDb)].append(nucDbPos)
      # populating offset table
      offsetTable = [[] for i in range(len(alphabet))]
      for nucCount in range(4):
        for seqWordTableVal in seqWordTable[nucCount]:
          for dbWordTableVal in dbWordTable[nucCount]:
```

```
            offsetTable[nucCount].append(seqWordTableVal-dbWordTableVal)
        # scoring
        for offset in offsetTable:



if __name__=='__main__':
  # using list mapping to reduce a file to a 1 dimentional array of alphabet
  with open('metadata') as metadataFile:
    alphabet =[item for sublist in map(list,(value.rstrip('\n') for value in metadataFile)) for item in sublist]
  # using line splitting and integer conversion to input the score matrix
  with open('scoreMatrix') as scoreMatrixFile:
    scoreMatrix = [[int(value) for value in line.split()] for line in scoreMatrixFile]
  # inputting the database as a terminal parameter with paying attention to new lines
  with open(sys.argv[1],'r') as databaseFile:
    database = [[value for value in line.rstrip('\n')] for line in databaseFile]
  # inputting the sequencesFile as a terminal parameter with paying attention to new lines
  with open(sys.argv[2],'r') as sequencesFile:
    sequences = [[value for value in line.rstrip('\n')] for line in sequencesFile]
  fasta()
```