Application 4: Applications to Genomics and Beyond Algorithmic Thinking (Part2)

Shamsuddin Rehmani

July 13, 2019

Application 4 Description

In Project 4, you implemented dynamic programming algorithms for determining both global and local alignments of pairs of sequences. In this Application, we will demonstrate the utility of these algorithms in two domains. In the first part of the Application, we examine an interesting problem from genomics. (This is based on "Introduction to Computational Genomics", by Nello Cristianini and Matthew W. Hahn). We will compare two sequences that have diverged from a common ancestor sequence due to mutation. (Mutation here includes base-pair substitution, which changes the sequence content, and insertion/deletion, which change the sequence lengths.) In the second part of the Application, we consider words that have spelling mistakes.

For the genomics part of the Application, you will load several protein sequences and an appropriate scoring matrix. For the spelling correction part of the Application, you will load a provided word list. To simplify these tasks, you are welcome to use this provided code.

Comparing two proteins

In 1994, Walter Gehring and colleagues at the University of Basel carried out an "interesting" experiment: they were able to turn on a gene called eyeless in various places on the body of the fruit fly,Drosophila melanogaster. The result was astonishing - fruit flies developed that had whole eyes sprouting up all over their bodies. It turned out that the eyeless is a master regulatory gene - it controls a cascade that contains more than 2000 other genes. Turning it on anywhere in the body activates the cascade and produces a fully formed, but non-functioning, eye. Humans, as well as many other animals, have a slightly different version of the eyeless gene (that is, a similar, yet not identical sequence of the same gene).

This observation suggests that about 600 million years ago (the estimated time of divergence between humans and fruit flies) there was an ancestral organism that itself used some version of eyeless, and that throughout the evolution of humans and fruit flies this gene continued to be maintained, albeit while accumulating mutations that did not greatly affect its function. In particular, a substring of the eyeless protein of about 130 amino acids, known as the PAX domain, whose function is to bind specific sequences of DNA, is virtually identical between the human and fruit fly versions of eyeless.

In following questions, we compute the similarity between the human and fruit fly versions of the eyeless protein and see if we can identify the PAX domain.

Answer to Question 1

Score of local Alignment = 875

Local alignment of HumanEyelessProtein:

HSGVNQLGGVFVNGRPLPDSTRQKIVELAHSGARPCDISRILQVSNGCVSKILGRYYETGSIRPRAIGGSK
PRVATPEVVSKIAQYKRECPSIFAWEIRDRLLSEGVCTNDNIPSVSSINRVLRNLASEK-QQ

Local alignment of FruitflyEyelessProtein:

HSGVNQLGGVFVGGRPLPDSTRQKIVELAHSGARPCDISRILQVSNGCVSKILGRYYETGSIRPRAIGGSK
PRVATAEVVSKISQYKRECPSIFAWEIRDRLLQENVCTNDNIPSVSSINRVLRNLAAQKEQQ

Answer to Question 2

- percentage of elements global alignment of local human vs consensus PAX domain that agree: 72.9%

- percentage of elements global alignment of local fruitfly vs consensus PAX that agree: 70.2%

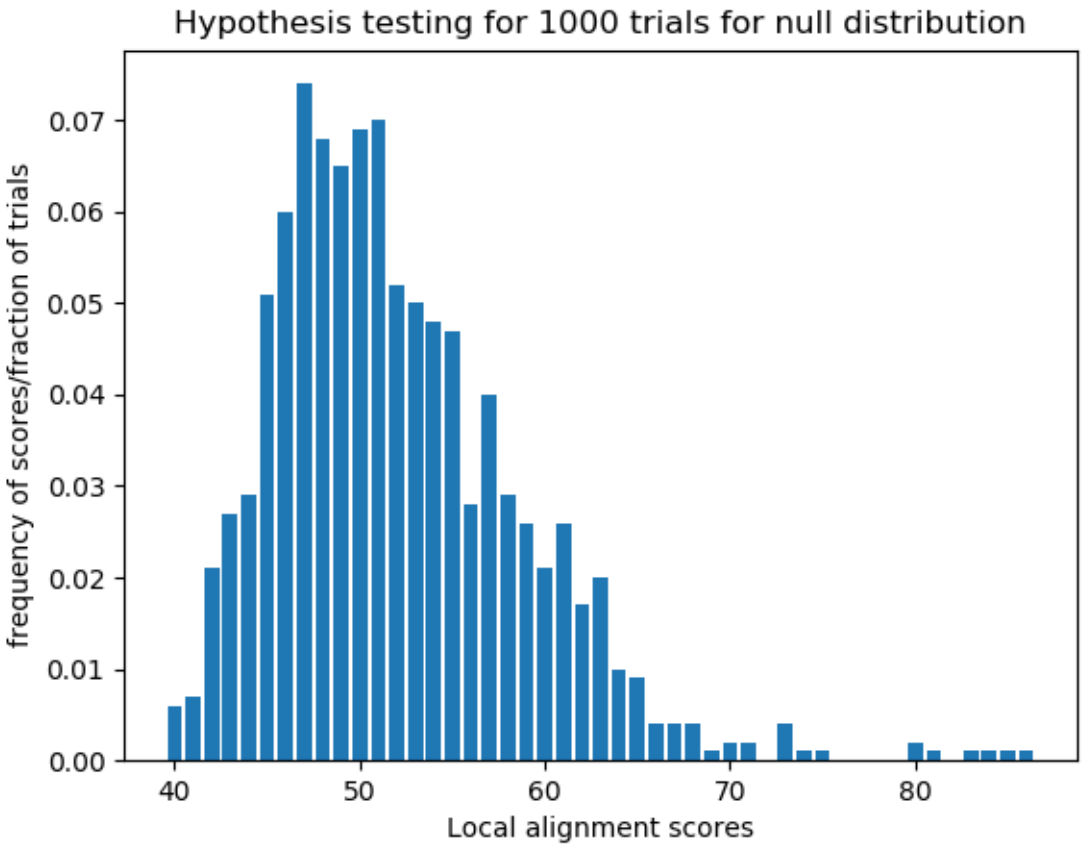
Answer to Question 3

The number of elements matching between each local alignment to the PAX consensus domain is too high to be deemed that they matched due to chance.

For local human vs PAX, 97 out of 133 elements match and the probability for all 97 of them to match due to chance would be quite small given than we have to randomly choose between 23 alphabets and that the probability of choosing an element is equal for all alphabets

The same can be said for the local fly vs PAX where 94 out of the 134 elements match

Answer to Question 4



Answer to Question 5

- mean of the distrubution: 52.0
- standard deviation of distribution: 6.72
- z-score of the distribution: 122.5

Answer to Question 6

The z-score is around 100 which means it right at the corner of the bell shaped distribution. 1σ from the mean of around 50 is about 7. And we know that 3σ covers about 99% of bell shape, i.e the likelihood of value falling within 3σ is high and the probability of value to fall outside 3σ is about 1%. The z-score we found tells us that the value of the score we found in Q1 is about 100 standard deviations away from mean (score of 875 is more than $100\sigma + \mu$). We can find the exact probability using the z test for normal distribution which gives us a probability on the order of 10^{-2000} (source) which is much smaller than the chance of winning a lottery

Answer to Question 7

- Let $x =$ "ABC" and $y =$ "". Then global alignment will return $x' =$ "ABC" and $y' =$ "$---$" so $|x| = 3$ and $|y| = 0$ and edit distance need 3 substitution operation to substitute "$---$" to "ABC" and $score(x', y') = |x'|+|y'|-3 = 0$. Therefore, `dash_score`$= 0$
- Let $x =$ "ABC" and $y =$ "ABC". Then global alignment returns the same strings and edit distance need zero operations. Hence $score(x, y) = |x| + |y| - 0 = 3 + 3 = 6$. Thus `diag_score` must be 2 for the score of x,y to be 6

  `diag_score` $= 2$
- Let $x =$ "ABC" and $y =$ "ABT". Then global alignment returns the same strings with $score(x, y) = score($"C", "T"$) + 2\times$`diag_score`. Hence edit distance needs substitution of "T" to "C" i.e one operation. Thus we have:

  $score($"C", "T"$) + 2\times$`diag_score` $= |x| + |y| - 1$

  `off_diag_score` $= 3 + 3 - 1 - 2 * 2 = 1$.

Answer to Question 8

- words within 1 edit distance of string "humble":

  {"bumble", "humbled", "tumble", "humble", "rumble", "humbler", "humbles", "fumble", "humbly", "jumble", "mumble"}
- words within 2 edit distance of string "firefly":

  {"firefly", "tiredly", "freely", "fireclay", "direly", "finely", "firstly", "liefly", "fixedly", "refly", "firmly"}

A Python code used to answer the Application Questions

```
"""
All answers to questions in Application #4 -
Applications to Genomics and Beyond
"""

import random
import matplotlib.pyplot as plt
import alg_application4_provided as alg_app4_prov
import alg_project4_solution as alg_proj4_sol

###############################################################################
# Solution Q1

# Q1: First, load the files HumanEyelessProtein and FruitflyEyelessProtein
# using the provided code. These files contain the amino acid sequences that
# form the eyeless proteins in the human and fruit fly genomes, respectively.
# Then load the scoring matrix PAM50 for sequences of amino acids. This scoring
# matrix is defined over the alphabet {A,R,N,D,C,Q,E,G,H,I,L,K,M,F,P,S,T,W,Y,V,B,Z,X,-}
# which represents all possible amino acids and gaps (the "dashes" in the alignment).
# Next, compute the local alignments of the sequences of HumanEyelessProtein and
# FruitflyEyelessProtein using the PAM50 scoring matrix and enter the score and
# local alignments for these two sequences below. Be sure to clearly distinguish
# which alignment is which and include any dashes ('-') that might appear in the
# local alignment.

# load the two amino acid sequences that form the eyeless proteins in the human
# and fruit fly genomes
human_seq = alg_app4_prov.read_protein(alg_app4_prov.HUMAN_EYELESS_URL)
fly_seq = alg_app4_prov.read_protein(alg_app4_prov.FRUITFLY_EYELESS_URL)

# load their scoring matrix
scoring_matrix = alg_app4_prov.read_scoring_matrix(alg_app4_prov.PAM50_URL)

# compute the local alignments matrix for these 2 sequences
local_align_matrix = alg_proj4_sol.compute_alignment_matrix(human_seq, fly_seq, scoring_matrix, False)

# compute the local alignments and the score for the two sequences
(score_q1, align_human, align_fly) = alg_proj4_sol.compute_local_alignment(human_seq,
                                                                           fly_seq, scoring_matrix,
                                                                           local_align_matrix)

print "Q1 Answers: "
print "Score: ",score_q1
print "local alignment of human: ", align_human
print "local alignment of fly:   ", align_fly
print

###############################################################################
# Solution Q2
# Q2: To continue our investigation, we next consider the similarity of the two
# sequences in the local alignment computed in Question 1 to a third sequence.
# The file ConsensusPAXDomain contains a "consensus" sequence of the PAX domain;
# that is, the sequence of amino acids in the PAX domain in any organism. In this
# problem, we will compare each of the two sequences of the local alignment computed
# in Question 1 to this consensus sequence to determine whether they correspond to
# the PAX domain.
#
# Load the file ConsensusPAXDomain. For each of the two sequences of the local
# alignment computed in Question 1, do the following:
#
# - Delete any dashes '-' present in the sequence.
# - Compute the global alignment of this dash-less sequence with the ConsensusPAXDomain
#   sequence.
# - Compare corresponding elements of these two globally-aligned sequences (local vs.
#   consensus) and compute the percentage of elements in these two sequences that agree.

# load the file ConsensusPAXDomain
pax_domain_seq = alg_app4_prov.read_protein(alg_app4_prov.CONSENSUS_PAX_URL)

# delete the dashes from the 2 local alignments in Q1
hum_nodash_seq = align_human.replace('-', "")

# compute the global alignments matrix for PAX domain and local human alignment seq with no dash
glob_align_mat_hum_pax = alg_proj4_sol.compute_alignment_matrix(pax_domain_seq, hum_nodash_seq,
                                                                scoring_matrix, True)

# compute the global alignment matrix for the PAX domain and local fly alignment seq with no dash
glob_align_mat_fly_pax = alg_proj4_sol.compute_alignment_matrix(pax_domain_seq, align_fly,
                                                                scoring_matrix, True)

(_, align_pax_hum, align_hum_nodash) = alg_proj4_sol.compute_global_alignment(pax_domain_seq,
                                                                              hum_nodash_seq,
                                                                              scoring_matrix,
                                                                              glob_align_mat_hum_pax)

(_, align_pax_fly, align_fly_nodash) = alg_proj4_sol.compute_global_alignment(pax_domain_seq,
                                                                              align_fly,
                                                                              scoring_matrix,
                                                                              glob_align_mat_fly_pax)

count_pax_hum = 0.0
count_pax_fly = 0.0


```

```
for charc_i, charc_j in zip(align_pax_hum, align_hum_nodash):
    if charc_i == charc_j:
        count_pax_hum += 1

print "Q2 Answers: "
print ("percentage of elments global alignment of local human vs PAX that agree: " +
        str(round(count_pax_hum/len(align_pax_hum) * 100, 2)) + "%")

for charc_i, charc_j in zip(align_pax_fly, align_fly_nodash):
    if charc_i == charc_j:
        count_pax_fly += 1

print ("percentage of elments global alignment of local fly vs PAX that agree: " +
        str(round(count_pax_fly/len(align_pax_fly) * 100, 2)) + "%")
print

###############################################################################
# Solution Q3
# Q3: Examine your answers to Questions 1 and 2. Is it likely that the level of
# similarity exhibited by the answers could have been due to chance? In particular,
# if you were comparing two random sequences of amino acids of length similar to
# that of HumanEyelessProtein and FruitflyEyelessProtein, would the level of agreement
# in these answers be likely? To help you in your analysis, there are 23 amino acids
# with symbols in the string "ACBEDGFIHKMLNQPSRTWVYXZ". Include a short justification
# for your answer.
#
# Ans: The number of elements matching between each local alignment to the PAX
# consensus domain is too high to deemed that they matched due to chance.
#
# For local human vs PAX, 97 out of 133 elements match and the probability
# for all 97 of them to match due to chance would be quite small given than we have
# to randomly choose between 23 alphabets and that the probability of choosing
# an element is equal for all alphabets
#
# The same can be said for the local fly vs PAX where 94 out of the 134 elements
# match

print "Q3 Answers:"
print "length of global alignment of human vs pax: ", len(align_pax_hum)
print "number of elements matching in this global alignment: ", int(count_pax_hum)
print "length of global alignment of fly vs pax: ", len(align_pax_fly)
print "number of elements matching in this global alignment: ", int(count_pax_fly)
print

###############################################################################
# Solution Q4
#
# Q4: Write a function generate_null_distribution(seq_x, seq_y, scoring_matrix, num_trials)
# that takes as input two sequences seq_x and seq_y, a scoring matrix scoring_matrix, and a
# number of trials num_trials. This function should return a dictionary scoring_distribution
# that represents an un-normalized distribution generated by performing the following process
# num_trials times:
#
# - Generate a random permutation rand_y of the sequence seq_y using random.shuffle().
# - Compute the maximum value score for the local alignment of seq_x and rand_y using the score
#   matrix scoring_matrix.
# - Increment the entry score in the dictionary scoring_distribution by one.
#
# Use the function generate_null_distribution to create a distribution with 1000 trials using
# the protein sequences HumanEyelessProtein and FruitflyEyelessProtein (using the PAM50 scoring matrix).
# Important: Use HumanEyelessProtein as the first parameter seq_x (which stays fixed) and
# FruitflyEyelessProtein as the second parameter seq_y (which is randomly shuffled) when calling
# generate_null_distribution. Switching the order of these two parameters will lead to a slightly
# different answers for question 5 that may lie outside the accepted ranges for correct answers.
#
# Next, create a bar plot of the normalized version of this distribution using plt.bar in
# matplotlib (or your favorite plotting tool). (You will probably find CodeSkulptor too slow to
# do the required number of trials.) The horizontal axis should be the scores and the vertical axis
# should be the fraction of total trials corresponding to each score. As usual, choose reasonable
# labels for the axes and title. Note: You may wish to save the distribution that you compute in
# this Question for later use in Question 5.

def generate_null_distribution(seq_x, seq_y, scoring_matrix, num_trials):
    """
    Takes as input two sequences seq_x and seq_y, a scoring matrix scoring_matrix, and a
    number of trials num_trials. This function should return a dictionary scoring_distribution
    that represents an un-normalized distribution generated by performing the following process
    num_trials times:

    - Generate a random permutation rand_y of the sequence seq_y using random.shuffle().
    - Compute the maximum value score for the local alignment of seq_x and rand_y using the score
      matrix scoring_matrix.
    - Increment the entry score in the dictionary scoring_distribution by one.

    Arguments:
        seq_x {string} -- a sequence of alphabets representing amino acids
        seq_y {string} -- a sequence of alphabets representing amino acids
        scoring_matrix {dict of dict} -- the scoring matrix for the all alphabet plus "-" combination
        num_trials {integer} -- the number of trial

    Returns:
        dict -- a dictionary representing the scoring distribution
    """

    # initialize the scoring distribution
    scoring_distribution = {}
```

```
191
192     for dummy_idx in range(num_trials):
193         # generate a random permutation for the seq_y
194         rand_y = list(seq_y)
195         random.shuffle(rand_y)
196         rand_y = "".join(rand_y)
197
198         # compute the local alignments matrix for seq_x and rand_y
199         local_align_matrix = alg_proj4_sol.compute_alignment_matrix(seq_x, rand_y, scoring_matrix, False)
200
201         # compute the local alignment score for the two sequences
202         (score, _, _) = alg_proj4_sol.compute_local_alignment(seq_x, rand_y, scoring_matrix, local_align_matrix)
203
204         # increment the entry score in the scoring_distribution
205         if scoring_distribution.has_key(score):
206             scoring_distribution[score] += 1
207         else:
208             scoring_distribution[score] = 1
209
210
211     return scoring_distribution
212
213 # get the scoring distribution for 1000 trial for HumanEyelessProtein and FruitflyEyelessProtein
214 scoring_dist = generate_null_distribution(human_seq, fly_seq, scoring_matrix, 1000)
215
216 sum_of_vals = float(sum(scoring_dist.values()))
217 norm_values = [value/sum_of_vals for value in scoring_dist.values()]
218 plt.bar(scoring_dist.keys(), norm_values)
219 plt.title('Hypothesis testing for 1000 trials for null distribution')
220 plt.xlabel('Local alignment scores')
221 plt.ylabel('frequency of scores/fraction of trials')
222 plt.xlim(None, None)
223 plt.ylim(None, None)
224 plt.show()
225
226 ###############################################################################
227 # Solution Q5
228 #
229 # Q5: What are the mean and standard deviation for the distribution that you
230 # computed in Question 4?
231 # What is the z-score for the local alignment for the human eyeless protein vs.
232 # the fruitfly eyeless protein based on these values?
233
234 # calculate the mean of the distribution
235 num_trials = 1000.0
236 mean_mu = sum([key * value for key, value in scoring_dist.items()])/num_trials
237
238 # calculate the standard deviation
239 sigma = sum([((key - mean_mu) ** 2) * value for key, value in scoring_dist.items()])/num_trials
240 sigma = sigma ** 0.5
241
242 # use mean and standard deviation to calculate the z score
243 z_score = (score_q1 - mean_mu)/sigma
244
245 print "Q5 Answers"
246 print "mean of the distrubution: ", mean_mu
247 print "standard deviation of distribution: ", sigma
248 print "z-score of the distribution: ", z_score
249 print
250
251 ###############################################################################
252 # Solution Q6
253 #
254 # Q6: Based on your answers to Questions 4 and 5, is the score resulting from the local
255 # alignment of the HumanEyelessProtein and the FruitflyEyelessProtein due to chance? As
256 # a concrete question, which is more likely: the similarity between the human eyeless
257 # protein and the fruitfly eyeless protein being due to chance or winning the jackpot
258 # in an extremely large lottery? Provide a short explanation for your answers
259 #
260 # Ans: the z-score is around 100 which means it right at the corner of the bell
261 # shaped. 1 standard deviation from the mean of around 50 is about 7. And
262 # we know that 3 standard deviations(sigma) covers about 99% of bell shape, i.e the
263 # likelihood of value falling within 3 sigma is high and the probability of
264 # value to fall outside 3 sigma is  about 1%. The z-score we found tells us that
265 # the value of the score we found in Q1 is about 100 standard away from mean
266 # (score of 875 is more than 100 * sigma + mean)
267 # We can find the exact probality using the z test for normal distribution
268 # which gives us a probablity on the order of 10^(-2000)
269 # (https://www.wolframalpha.com/input/?i=Probability+of+100+standard+deviations)
270 # which is much smaller than the chance of winning a lottery
271
272 ###############################################################################
273 # Solution Q7
274 #
275 # Not surprisingly, similarity between pairs of sequences and edit distances between pairs
276 # of strings are related. In particular, the edit distance for two strings x and y can be
277 # expressed in terms of the lengths of the two strings and their corresponding similarity
278 # score as follows: |x| + |y| - score(x, y) where score(x, y) is the score returned by the
279 # global alignment of these two strings using a very simple scoring matrix that can be
280 # computed using build_scoring_matrix.
281 # Determine the values for diag_score, off_diag_score, and dash_score such that the score
282 # from the resulting global alignment yields the edit distance when substituted into the
283 # formula above
284 #
285 # Ans:
286 # 1-Let x = "ABC" and y = "". Then global alignment will return x' = "ABC" and y' = "---"
```

```
#   so |x| = 3 and |y| = 0 and edit distance need 3 substitution operation to subsitute "---"
#   to "ABC" and score(x', y') = |x'| +|y'| - 3 = 0. Therefore, dash_score = 0
#   dash_score = 0

# 2-Let x = "ABC" and y = "ABC". Then global alignment returns the same strings and edit
#   distance need zero operations. Hence score(x, y) = |x| + |y| - 0 = 3 + 3 = 6. Thus
#   diag_score must be 2 for the score of x, y to be 6
#   diag_score = 2

# 3-Let x = "ABC" and y = "ABT". Then global alignment resturns the same strings with
#   total score of score of string "C" and "T" + 2 * diag_score. Hence edit distance needs
#   subsitution od "T" to "C" i.e one operation. Thus we have:
#   score("C", T) + 2 * diag_score = |x| + |y| - 1
#   off_diag_score = 3 + 3 - 1 - 2 * 2 = 1
#   off_diag_score = 1

##############################################################################
# Solution Q8
#
# Q8: For this final question, we will implement a simple spelling correction function that
# uses edit distance to determine whether a given string is the misspelling of a word.
# To begin, load this list of 79339 words. Then, write a function
# check_spelling(checked_word, dist, word_list) that iterates through word_list and returns
# the set of all words that are within edit distance dist of the string checked_word.
# Use your function check_spelling to compute the set of words within an edit distance of
# one from the string "humble" and the set of words within an edit distance of two from the
# string "firefly".
#

def check_spelling(checked_word, dist, word_list):
    """[summary]

    Arguments:
        checked_word {[type]} -- [description]
        dist {[type]} -- [description]
        word_list {[type]} -- [description]

    Returns:
        [type] -- [description]
    """

    # initialize the set that will contain a list of words within edit distance dist
    # from word_list
    within_dist = set()

    # initalize a set of alphabets
    alphabets = set(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm',
                     'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z'])

    # compute the scoring matrix for of scores calculated in Q7
    score_mat = alg_proj4_sol.build_scoring_matrix(alphabets, 2, 1, 0)

    for word in word_list:
        # compute the global alignment matrix
        align_mat = alg_proj4_sol.compute_alignment_matrix(word, checked_word, score_mat, True)

        # compute the global alignment score of the word
        (score, _, _) = alg_proj4_sol.compute_global_alignment(word, checked_word, score_mat, align_mat)

        # add the word to the within_dist set if it edit distance is within edit distance dist
        if (len(word) + len(checked_word) - score) <= dist:
            within_dist.add(word)

    return within_dist

# load the word list
word_list = alg_app4_prov.read_words(alg_app4_prov.WORD_LIST_URL)

# compute the set of word that are within edit distance 1 of string "humble"
with_wrds_humble = check_spelling("humble", 1, word_list)

# compute the set of word that are within edit distance 2 of string "firefly"
with_wrds_firefly = check_spelling("firefly", 2, word_list)

print "Q8 Answers:"
print "words within 1 edit distance of string 'humble': \n", with_wrds_humble
print "words within 2 edit distance of string 'firefly': \n", with_wrds_firefly
```

B All functions for project 4 used in the application

```
"""
All the algorithms for Project#4 - Computing
Alignments of Sequences
"""

def build_scoring_matrix(alphabet, diag_score, off_diag_score, dash_score):
    """
    The function computes a scoring matrix(a dictionary of dictionaries)
    whose entries are indexed by pairs of characters in alphabet plus '-'
    The score for any entry indexed by one or more dashes is dash_score.
    The score for the remaining diagonal entries is diag_score. Finally,
    the score for the remaining off-diagonal entries is off_diag_score.

    Arguments:
```

```
        alphabet {set} -- a set of characters
        diag_score {integer} -- the diagnal score for the scoring matrix
        off_diag_score {integer} -- the off diagnal score
        dash_score {[type]} -- the score that includes atleast one dash

    Returns:
        a dictionary of dictionary -- the scoring matrix
    """
    # initialize the scoring matrix
    scoring_matrix = {}
    # add the dash character to the set of alphabets
    all_chars = alphabet.union("-")

    # create a dictionary of dictionary for scores
    for char_i in all_chars:
        scoring_matrix[char_i] = {}
        for char_j in all_chars:
            if (char_i == "-" or char_j == "-"):
                scoring_matrix[char_i][char_j] = dash_score
            elif (char_i == char_j):
                scoring_matrix[char_i][char_j] = diag_score
            else:
                scoring_matrix[char_i][char_j] = off_diag_score


    return scoring_matrix

def compute_alignment_matrix(seq_x, seq_y, scoring_matrix, global_flag):
    """
    Takes as input two sequences seq_x and seq_y whose elements share a common
    alphabet with the scoring matrix scoring_matrix. The function computes and
    returns the alignment matrix for seq_x and seq_y. If global_flag is True,
    the global alignment matrix is computed. If global_flag is False, local
    alignment matrix is computed

    For global alignment matrix S, each entry S[i][j] contains the maximum score
    over every possible global alignment of the pair of sequences seq_x[0...i-1] and
    seq_y[0...j-1]

    For local alignment matrix S, each entry entry S[i][j] contains the maximum score
    over every possible alignment of the pair of sequences seq_x[0...i-1] and
    seq_y[0...j-1] except for the case when  S[i][j] < 0, S[i][j]  is set to zero

    Arguments:
        seq_x {string} -- a string of alphabets whose elements share a common alphabet
                          with scoring matrics
        seq_y {string} -- a string of alphabets whose elements share a common alphabet
                          with scoring matrics
        scoring_matrix {a dictionary of dictionaries} -- the scoring matrix with each
                                                         alphabet plus '-' combination
                                                         scores
        global_flag {bolean -- flag to choose between global and local alignment

    Returns:
        list of lists -- an alignment matrix for seq_x and seq_y
    """
    # let align_matrix = S. Then initialize S[0][0] = 0
    align_matrix = [[0]]

    # set the alignment matrix's column 0 to appropriate score either based on local or global
    # alignment selected
    for idx in range(1, len(seq_x) + 1):
        align_matrix.append([align_matrix[idx - 1][0] + scoring_matrix[seq_x[idx - 1]]["-"] ])
        if not global_flag and align_matrix[idx][0] < 0:
            align_matrix[idx][0] = 0

    # set the alignment matrix's row 0 to appropriate score either based on local or global
    # alignment selected
    for idx in range(1, len(seq_y) + 1):
        align_matrix[0].append(align_matrix[0][idx - 1] + scoring_matrix["-"][seq_y[idx - 1]])
        if not global_flag and align_matrix[0][idx] < 0:
            align_matrix[0][idx] = 0

    # set the remaining alignment matrix(S) values based on previous 3 values of S[i-1][j-1], S[i-1][j],
    # and S[i][j-1]
    for idx_x in range(1, len(seq_x) + 1):
        for idx_y in range(1, len(seq_y) + 1):
            val1 = align_matrix[idx_x - 1][idx_y - 1] + scoring_matrix[seq_x[idx_x - 1]][seq_y[idx_y - 1]]
            val2 = align_matrix[idx_x - 1][idx_y] + scoring_matrix[seq_x[idx_x - 1]]["-"]
            val3 = align_matrix[idx_x][idx_y - 1] + scoring_matrix["-"][seq_y[idx_y - 1]]
            align_matrix[idx_x].append(max(set([val1, val2, val3])))
            if not global_flag and align_matrix[idx_x][idx_y] < 0:
                align_matrix[idx_x][idx_y] = 0

    return align_matrix

def compute_global_alignment(seq_x, seq_y, scoring_matrix, alignment_matrix):
    """
    Takes as input two sequences seq_x and seq_y whose elements share a common alphabet
    with the scoring matrix scoring_matrix. This function computes a global alignment of
    seq_x and seq_y using the global alignment matrix alignment_matrix. The function returns
    a tuple of the form (score, align_x, align_y) where score is the score of the global
    alignment align_x and align_y. Note that align_x and align_y should have the same
    length and may include the padding character '-'.

    Arguments:
```

```
        seq_x {string} -- a string of alphabets
        seq_y {string} -- a string of alphabets
        scoring_matrix {dictionary of dictionaries} -- the scoring matrix
        alignment_matrix {list of lists} -- the global alignment matrix

    Returns:
        tuple -- returns (score, align_x, align_y) where score is the score of the global
                 alignment align_x and align_y.
    """

    # intialize the indicies to the lengths of the sequences
    idx_x = len(seq_x)
    idx_y = len(seq_y)
    align_x = ""
    align_y = ""
    score = 0

    while (idx_x != 0 and idx_y != 0):
        if (alignment_matrix[idx_x][idx_y] == (alignment_matrix[idx_x - 1][idx_y - 1]
            + scoring_matrix[seq_x[idx_x - 1]][seq_y[idx_y - 1]])):
            align_x = seq_x[idx_x - 1] + align_x
            align_y = seq_y[idx_y - 1] + align_y
            score += scoring_matrix[seq_x[idx_x - 1]][seq_y[idx_y - 1]]
            idx_x -= 1
            idx_y -= 1

        else:
            if (alignment_matrix[idx_x][idx_y] == (alignment_matrix[idx_x - 1][idx_y]
                + scoring_matrix[seq_x[idx_x - 1]]["-"])):
                align_x = seq_x[idx_x - 1] + align_x
                align_y = "-" + align_y
                score += scoring_matrix[seq_x[idx_x - 1]]["-"]
                idx_x -= 1

            else:
                align_x = "-" + align_x
                align_y = seq_y[idx_y - 1] + align_y
                score += scoring_matrix["-"][seq_y[idx_y - 1]]
                idx_y -= 1

    while idx_x != 0:
        align_x = seq_x[idx_x - 1] + align_x
        align_y = "-" + align_y
        score += scoring_matrix[seq_x[idx_x - 1]]["-"]
        idx_x -= 1

    while idx_y != 0:
        align_x = "-" + align_x
        align_y = seq_y[idx_y - 1] + align_y
        score += scoring_matrix["-"][seq_y[idx_y - 1]]
        idx_y -= 1


    return (score, align_x, align_y)

def compute_local_alignment(seq_x, seq_y, scoring_matrix, alignment_matrix):
    """
    Takes as input two sequences seq_x and seq_y whose elements share a common alphabet
    with the scoring matrix scoring_matrix. This function computes a local alignment of
    seq_x and seq_y using the local alignment matrix alignment_matrix. The function returns
    a tuple of the form (score, align_x, align_y) where score is the score of the local
    alignment align_x and align_y. Note that align_x and align_y should have the same
    length and may include the padding character '-'.

    Arguments:
        seq_x {string} -- a string of alphabets
        seq_y {string} -- a string of alphabets
        scoring_matrix {dictionary of dictionaries} -- the scoring matrix
        alignment_matrix {list of lists} -- the global alignment matrix

    Returns:
        tuple -- returns (score, align_x, align_y) where score is the score of the global
                 alignment align_x and align_y.
    """

    # initialize the variables
    max_value = float("-inf")
    idx_x = -1
    idx_y = -1
    align_x = ""
    align_y = ""
    score = 0

    # find the location (row, col) of the maximum value in alignment_matrix
    for row in range(len(alignment_matrix)):
        for col in range(len(alignment_matrix[row])):
            if alignment_matrix[row][col] > max_value:
                max_value = alignment_matrix[row][col]
                idx_x = row
                idx_y = col

    while alignment_matrix[idx_x][idx_y] != 0 and idx_x != 0 and idx_y != 0:
        if (alignment_matrix[idx_x][idx_y] == (alignment_matrix[idx_x - 1][idx_y - 1]
            + scoring_matrix[seq_x[idx_x - 1]][seq_y[idx_y - 1]])):
            align_x = seq_x[idx_x - 1] + align_x
            align_y = seq_y[idx_y - 1] + align_y
```

```
207                 score += scoring_matrix[seq_x[idx_x - 1]][seq_y[idx_y - 1]]
208                 idx_x -= 1
209                 idx_y -= 1
210 
211             else:
212                 if (alignment_matrix[idx_x][idx_y] == (alignment_matrix[idx_x - 1][idx_y]
213                     + scoring_matrix[seq_x[idx_x - 1]]["-"])):
214                     align_x = seq_x[idx_x - 1] + align_x
215                     align_y = "-" + align_y
216                     score += scoring_matrix[seq_x[idx_x - 1]]["-"]
217                     idx_x -= 1
218 
219                 else:
220                     align_x = "-" + align_x
221                     align_y = seq_y[idx_y - 1] + align_y
222                     score += scoring_matrix["-"][seq_y[idx_y - 1]]
223                     idx_y -= 1
224 
225         return (score, align_x, align_y)
```