305 SmithWaterman.py

```
@@ -1,13 +1,32 @@
```

```
+ #!/usr/bin/python3
+
+ ### Bioinformatics Algorithms ICA Script by B236494 ###
+
+ ### Adapted from BA4 class code written by Simon Tomlinson ###
+
+ ### print statements were used throughout execution to test the outputs of the program ###
+
+ ### A test has also been implemented at the end to compare the output witht the EMBOSS water Gold-standard ###
+
+ ### B236494_SmithWaterman_Final_Script.py script adapted from SmithWaterman.py Version 1.9 SRT ###
+
  # Simon Tomlinson Bioinformatics Algorithms
  # Perform Smith Waterman Alignment in Python (from first principles)
  # contains rows lists each of length cols initially set to 0
  # index as my_matrix[1][2] my_matrix[R][C]
- # Teaching code not production code!!
- #Version 1.9 SRT

+ #
+ # Version 1.9 SRT

+ import time
+ import os
+ import sys
+ from time import sleep
  from enum import Enum
+ import pandas as pd
+ import re
+ import argparse

- #Global enumeration used for tracement
+ # Global enumeration used for tracement
  TypeB = Enum('TypeB', ['INSERT', 'DELETE', 'MISMATCH', 'MATCH', 'END'])


  def create_matrix(rows, cols):
      my_matrix = [[0 for col in range(cols + 1)] for row in range(rows + 1)]
      return my_matrix


  # x is row index, y is column index
  # follows[r][c]

  def calc_score(matrix, x, y):
-     # print("seq1:",sequence1[y- 1]," seq2: "+sequence2[x - 1],"x:",x," y:",y)
+     print("seq1:",sequence1[y- 1]," seq2: "+sequence2[x - 1],"x:",x," y:",y)

      sc = seqmatch if sequence1[y - 1] == sequence2[x - 1] else seqmismatch

      base_score = matrix[x - 1][y - 1] + sc
      insert_score = matrix[x - 1][y] + seqgap
      delete_score = matrix[x][y - 1] + seqgap
      v = max(0, base_score, insert_score, delete_score)
      return v


  # makes a single traceback step
  def traceback(mymatrix, maxv):
      x = maxv[0]
      y = maxv[-1]
      val = mymatrix[x][y]

      # todo add some outputs for checking errors
      sc = seqmatch if sequence2[x - 1] == sequence1[y - 1] else seqmismatch
+     # print(sc)

      base_score = mymatrix[x - 1][y - 1] + sc
+     # print(base_score)
      if base_score == val:
          if sc==seqmatch:
              return [x - 1,TypeB.MATCH, y - 1]
          else:
              return [x - 1,TypeB.MISMATCH, y - 1]

      insert_score = mymatrix[x - 1][y] + seqgap
+     # print(input_score)
      if insert_score == val:
          return [x - 1, TypeB.INSERT, y]
      else:
          return [x, TypeB.DELETE, y - 1]
```

```
# builds the initial scoring matrix used for traceback
def build_matrix(mymatrix):
    rows = len(mymatrix)
    cols = len(mymatrix[0])
-

    for i in range(1, rows):

        for j in range(1, cols):
            mymatrix[i][j] = calc_score(mymatrix, i, j)

    return mymatrix


# gets the max value from the built matrix
# Max is the end of the traceback for SW
def get_max(mymatrix):
    max = mymatrix[0][0]
    mrow = 0
    mcol = 0

    rows = len(mymatrix)
    cols = len(mymatrix[0])

    for i in range(1, rows):
        for j in range(1, cols):
            if mymatrix[i][j] > max:
                max = mymatrix[i][j]
                mrow = i
                mcol = j
-   print("max score: ", max)
    return [mrow, TypeB.END, mcol]


# print out the best scoring path from the SW matrix
def print_matrix(mymatrix):
    rows = len(mymatrix)
    cols = len(mymatrix[0])
    s1 = "  " + sequence1
    s2 = " " + sequence2

-   print("Dimensions: r= %2d , c= %2d" % (rows, cols))

    for a in s1:
        print(a, end="")
        print(" \t", end="")
    print("\n", end="")

    for i in range(0, rows):
        print(s2[i], end="")
        print(" \t", end="")
        for j in range(0, cols):
            print("%02d\t" % (mymatrix[i][j]), end="")
        print("\n", end="")


# print out the traceback of the best scoring alignment
def print_traceback(mymatrix):
    # this will print as expected with internal gaps
-   print("Building traceback...")

    maxv = get_max(mymatrix)

-   #stash the max score for later
    max_score = mymatrix[maxv[0]][maxv[-1]]

    # traverse the matrix to find the traceback elements
    # if more than one path just pick one
    topstring = ""
    midstring = ""
    bottomstring = ""

    # pad the sequences so indexes into the sequences match the matrix indexes
    asequence1 = "#" + sequence1
    asequence2 = "#" + sequence2
```

```
# builds the initial scoring matrix used for traceback
def build_matrix(mymatrix):
    rows = len(mymatrix)
    cols = len(mymatrix[0])
+   row_number=0
+
    for i in range(1, rows):
+       row_number = row_number + 1
+       print("\nRow Number:", row_number)
+       sleep(wait)
        for j in range(1, cols):
            mymatrix[i][j] = calc_score(mymatrix, i, j)

    return mymatrix


# gets the max value from the built matrix
# Max is the end of the traceback for SW
def get_max(mymatrix):
    max = mymatrix[0][0]
    mrow = 0
    mcol = 0

    rows = len(mymatrix)
    cols = len(mymatrix[0])

    for i in range(1, rows):
        for j in range(1, cols):
            if mymatrix[i][j] > max:
                max = mymatrix[i][j]
                mrow = i
                mcol = j
+   print("The Maximum Score was: ", max,"\n")
    return [mrow, TypeB.END, mcol]


# print out the best scoring path from the SW matrix
def print_matrix(mymatrix):
    rows = len(mymatrix)
    cols = len(mymatrix[0])
    s1 = "  " + sequence1
    s2 = " " + sequence2
+
+   sleep(wait)

+   print("\nDimensions of The SmithWaterman Matrix: Rows= %2d , Columns= %2d\n" % (rows, cols))
+
+   sleep(wait)

    for a in s1:
        print(a, end="")
        print(" \t", end="")
    print("\n", end="")

    for i in range(0, rows):
        print(s2[i], end="")
        print(" \t", end="")
        for j in range(0, cols):
            print("%02d\t" % (mymatrix[i][j]), end="")
        print("\n", end="")


# print out the traceback of the best scoring alignment
def print_traceback(mymatrix):
    # this will print as expected with internal gaps
+
+   sleep(wait)
+
+   print("\n### We Will Now Build The Traceback... ###\n")
    maxv = get_max(mymatrix)
+   print(maxv)

+   #stash the max score for later
    max_score = mymatrix[maxv[0]][maxv[-1]]

+
    # traverse the matrix to find the traceback elements
    # if more than one path just pick one
    topstring = ""
    midstring = ""
    bottomstring = ""

    # pad the sequences so indexes into the sequences match the matrix indexes
    asequence1 = "#" + sequence1
    asequence2 = "#" + sequence2
```

```
-     #this vector is used to store the traceback results
+     #this vector is used to store the traceback results

      traversal_results = []

      # add the rest of the elements
      search = True
      lastelement = False

+     # Stores the position so it can track if it is an insertion or deletion
+     # Check if it is a gap or not
+     if max_score <1:
+         print ("There is no suitable alignment...Check your inputs please!")
+         exit;
+     old_maxv = maxv
+
+

      while (search):

-         #debug print("position: %d, %s, %d" % (maxv[0], maxv[1], maxv[-1]))
-         #store the results
+         # print(" position:  %d, %d " % (maxv[0], maxv[-1]))
+
+         # print("Testing execution, type is", current_type)
+
+         # store the results
          traversal_results.append(maxv)

+         # type_traversal = type(traversal_results)

          maxv = traceback(mymatrix, maxv)

-         #catch the trivial case that we are at the end of one of the sequences
+         #catch the trivial case that we are at the end of one of the sequences
          if (maxv[-1] < 0 or maxv[0] < 0):
              traversal_results.append(maxv)
              search= False
              continue

          if (mymatrix[maxv[0]][maxv[-1]] == 0 and lastelement == False):
              lastelement = True
              continue

          if(lastelement==True) :
              search= False
              traversal_results.append(maxv)
              continue

      for i in range(0, len(traversal_results)-2):

-         #The TypeB of the next element gives how the current element was reached
-         #in the dynamic programming table
-         #The current element gives the index of the two matching bases to be aligned
+         # print("Testing execution")
+
+         # The TypeB of the next element gives how the current element was reached
+         # in the dynamic programming table
+         # The current element gives the index of the two matching bases to be aligned
+

          curr_el=traversal_results[i]
          next_el=traversal_results[i+1]

-         #Match
+         #Match
          if(next_el[1]==TypeB.MATCH):
              bottomstring += asequence2[curr_el[0]]
              topstring += asequence1[curr_el[-1]]
              midstring +="|"
+             # print(" position: ", i,i )
+             # print("MATCHHHHH")
+

          #Mismatch
          elif(next_el[1]==TypeB.MISMATCH):
              bottomstring += asequence2[curr_el[0]]
              topstring += asequence1[curr_el[-1]]
              midstring += "."
+             # print(" position: ", i,i )
+             # print("MISMATCH/START?")

          elif(next_el[1]==TypeB.INSERT):
              bottomstring += asequence2[curr_el[0]]
              topstring += "-"
              midstring += " "
+             # print(" position: ", i,i )
+             # print("Insertion")
```

```
189             elif(next_el[1]==TypeB.DELETE):
190                 bottomstring += "-"
191                 topstring += asequence1[curr_el[-1]]
192                 midstring += " "
193
194 -       print("\nFinal Alignment, Score: %d" % max_score)
195
196 -       print(topstring[::-1])
197 -       print(midstring[::-1])
198 -       print(bottomstring[::-1])
199
200     # build the SW alignment...
201     def perform_smith_waterman():
202         # values for weights
203         global seqmatch
204         global seqmismatch
205         global seqgap
206         global sequence1
```

```
249             elif(next_el[1]==TypeB.DELETE):
250                 bottomstring += "-"
251                 topstring += asequence1[curr_el[-1]]
252                 midstring += " "
253 +               # print(" position: ", i,i )
254 +               # print("DELETED")
255 +
256 +
257 +       print("")
258 +       for element in traversal_results:
259 +           print(element,"\n")
260 +
261 +       sleep(wait)
262 +
263 +       print("\nFinal Alignment, Score: %d\n" % max_score)
264 +
265 +       sleep(wait)
266
267
268 +       ### Printing the alignment with an effect ###
269 +
270 +       # print(topstring[::-1])
271 +       animated_print(topstring[::-1])
272 +
273 +       # print(midstring[::-1])
274 +       animated_print(midstring[::-1])
275 +
276 +       # print(bottomstring[::-1])
277 +       animated_print(bottomstring[::-1])
278 +
279 +
280 +       sleep(wait)
281 +
282 +       # print("Testing execution")
283 +
284 +
285 +
286 +
287 + def time_to_pause():
288 +       print("How long do you want the pauses in between important steps to be
    in seconds?")
289 +       print("Please input a positive integer value <= 3 OR input 0 to execute
    without pauses")
290 +       while True:
291 +           wait_input = input("What is you choice? \n")
292 +           if not wait_input.isdigit():
293 +               print("TypeError: Please input a numerical value")
294 +               continue;
295 +           wait = int(wait_input)
296 +           if wait < 0 or wait > 3:
297 +               print("Invalid choice. Please choose a positive integer value <=
    3")
298 +           else :
299 +               print("You have chosen a value of ", wait," seconds\n")
300 +               return wait
301 +               break;
302 +
303 +
304 + def animated_print(s):
305 +       for c in s:
306 +           sys.stdout.write(c)
307 +           sys.stdout.flush()
308 +           time.sleep(0.25)
309 +       print("")
310 +
311
312     # build the SW alignment...
313     def perform_smith_waterman():
314         # values for weights
315         global seqmatch
316         global seqmismatch
317         global seqgap
318         global sequence1
```

```
207       global sequence2
208
209 -     # note these are not the exact weights used in the original SW paper
210 -     seqmatch = 1
211 -     seqmismatch = -1
212 -     seqgap = -1
213
214 -     # input sequences- other examples
215 -     sequence1="AGTGATAAACTAGTAATTTTT"
216 -     sequence2="TTGGGGGTAAACAGGGG"
217
218 -     #  sequence1 ="AGTCGGTTAGTAAA"
219 -     #  sequence2 ="TTTTGGGTTTAGGCGC"
220
221 -   #   sequence1 = "GTGTATTTTTTT"
222 -   #   sequence2 = "AAAAGTGTTATT"
223
224 -   #   sequence1 = "TCGTTCTAG"
225 -   #   sequence2 = "TCGTTTTTG"
226
227 -   #   sequence1 = "SimonTomkinson"
228 -   #   sequence2 = "SimonTomlinsonBioinformaticsAlgorithms"
```

```
319       global sequence2
320 +     global wait
321 +
322 +     print("\n\n\nWelcome to B236494's version of the SmithWaterman.py Script \n")
323 +     print("\nThis is an adapted version of SmithWaterman.py V 1.9 by Simon Tomlinson\n")
324 +     print("\nThis script takes command line inputs for the match/mismatch/gap scores and user-input during the start of execcution for the FASTA files to be aligned\n")
325 +     print("\nEnsure that the files to be aligned are present in the current directory where the programme is being executed \n")
326 +     print("\nExample CLI input could be ---> python3 SmithWaterman.py --seqmatch 1 --seqmismatch -1 --seqgap -1 \n")
327 +
328 +
329 +
330 +     wait = time_to_pause()
331 +
332 +
333 +     ### Taking command line input for the match, mismatch and gap penalties ###
334 +
335 +     parser = argparse.ArgumentParser(description='Please provide the parameters to Perform Smith-Waterman Alignment.')
336 +     sleep(wait)
337
338 +     # Add arguments for sequence match, mismatch, and gap penalties
339 +     # note these defaults are not the exact weights used in the original SW paper
340
341 +     parser.add_argument('--seqmatch', type=int, default=1, help='Input the score for sequence matches. Default is 1.')
342 +     parser.add_argument('--seqmismatch', type=int, default=-1, help='Input the Penalty for sequence mismatches. Default is -1.')
343 +     parser.add_argument('--seqgap', type=int, default=-1, help='Input the penalty for a gap. Default is -1.')
344
345 +     args = parser.parse_args()
346
347 +     seqmatch = args.seqmatch
348 +     seqmismatch = args.seqmismatch
349 +     seqgap = args.seqgap
350
351 +     sleep(wait)
352
353 +     print(f"Using match score: {seqmatch}")
354 +     print(f"Using mismatch penalty: {seqmismatch}")
355 +     print(f"Using gap penalty: {seqgap}\n")
356 +
357 +     sleep(wait)
358 +
359 +     ### Function to check if there are any non ATGC characters ###
360 +
361 +     def check_if_any_non_atgc_chars(sequence):
362 +         pattern = re.compile(r'[^ATGC]')
363 +         match = pattern.search(sequence.upper())
364 +         return not bool(match)
365 +
366 +     ### Function to store sequences that need to be aligned ###
367 +
368 +     def read_fasta_filename(filename):
369 +         seq = ""
370 +         with open(filename, 'r') as filehandle:
371 +             for line in filehandle:
372 +                 # Using regular expression search to ignore lines starting with ">"
373 +                 if re.search("^>", line.strip()):
374 +                     continue
375 +                 seq += line.strip()
376 +         return seq
377 +
378 +
379 +     # Taking input for file name
380 +
381 +     seq1_file = input("Enter the filename of the first sequence file: ")
382 +     seq2_file = input("Enter the filename to the second sequence file: ")
383 +
384 +     # Saving sequences to variables to be aligned
385 +
386 +     sequence1 = read_fasta_filename(seq1_file)
387 +     sequence2 = read_fasta_filename(seq2_file)
388 +
389 +
390 +
```

```python

    print("Sequence1: " + sequence1)
    print("Sequence2: " + sequence2)

    mymatrix = create_matrix(len(sequence2), len(sequence1))
    mymatrix = build_matrix(mymatrix)
    print_matrix(mymatrix)

    print_traceback(mymatrix)


##this calls the SW algorithm when the script loads
perform_smith_waterman()
```

```diff
+    ### Alternate options to test sequences ###
+
+ #    sequence1="AGTGATAAACTAGTAATTTTT"
+ #    sequence2="TTGGGGGTAAACAGGGG"
+
+ #    sequence1 ="AGTCGGTTAGTAAA"
+ #    sequence2 ="TTTTGGGTTTAGGCGC"
+
+ #    sequence1 = "GTGTATTTTTTT"
+ #    sequence2 = "AAAAGTGTTATT"
+
+ #    sequence1 = "TCGTTCTAG"
+ #    sequence2 = "TCGTTTTTG"
+
+ #    sequence1 = "SimonTomkinson"
+ #    sequence2 = "SimonTomlinsonBioinformaticsAlgorithms"
+
+
+
+    ### Checking sequence1 & sequence2 if they contain any characters other than A, T, G, C ###
+
+    check_seq1 = check_if_any_non_atgc_chars(sequence1)
+    check_seq2 = check_if_any_non_atgc_chars(sequence2)
+
+    sleep(wait)
+
+    if not check_seq1 or not check_seq2:
+        print("Error in One or both input sequences. There are characters other than A, T, G, and C.")
+        print("Please make sure that your input files contain the correct sequences and try again.")
+        print("Exiting Program...Bye")
+        sleep(wait)
+        sys.exit()
+
+
+    ### Printing the sequences ###
+
+    print("The input sequences are\n")
+    sleep(wait)

     print("Sequence1: " + sequence1)
     print("Sequence2: " + sequence2)
+    print("\n") ### Empty line

     mymatrix = create_matrix(len(sequence2), len(sequence1))
     mymatrix = build_matrix(mymatrix)
     print_matrix(mymatrix)

     print_traceback(mymatrix)

+
+
+    ### Taking the absolute value as EMBOSS water does not permit negative values for parameters ###
+
+    # seqmismatch = abs(seqmismatch)
+    seqgap = abs(seqgap)
+
+
+    print("\n\nComparing output with EMBOSS water")
+
+    sleep(wait)
+
+    water_command = "water -asequence {} -bsequence {} -gapopen {} -gapextend {} -outfile water_{}_{}.water -datafile EDNAFULL_srt".format(seq1_file, seq2_file, seqgap, seqgap, seq1_file, seq2_file)
+    os.system(water_command)
+
+    print("\n\nThis is the output from EMBOSS water using the same files which is our Gold-Standard for alignment")
+
+    sleep(wait)
+
+    display_water_file = "cat water_{}_{}.water".format(seq1_file, seq2_file)
+    os.system(display_water_file)
+
+    sleep(wait)
+
+    print("\nThanks for using this script.\n")
+
+
+

 ##this calls the SW algorithm when the script loads
 perform_smith_waterman()
```