Homework 1 Answers:

A few things to remember:

1. With the naïve algorithm, it is important to ensure that you don’t skip over a match after you find a partial match. In practice, this means that you need to keep track of your frame location in the database string separately from the letters you’re comparing to the query string.
2. The BM algorithm searches the database string from left to right, but it checks for matches in the query string from right to left.
3. Unless otherwise indicated, don’t use prepackaged functions like ‘find’ or ‘regex’
4. You can save us a bit of time by commenting your code well, and by submitting your code as individual files, rather than as a compressed folder (.zip, etc). This allows us to view it online, which can speed things up considerably on our end. Thanks!

Note: there are many acceptable ways to accomplish the homework, these are just examples

Part 1 Naïve Algorithm:

with open('string\_file.txt','r') as string\_file:

string=string\_file.read().replace('\n', '')

with open('query\_file.txt','r') as query\_file:

query=query\_file.read().replace('\n', '')

string\_length = len(string)

query\_length = len(query)

if query\_length > string\_length:

print -1

quit()

offset\_positions = ""

i=0

j=0

while i+j < string\_length:

while j < query\_length and i+j < string\_length:

if string[i+j] == query[j]:

if j == query\_length-1:

if not offset\_positions:

offset\_positions += str(i)

else:

offset\_positions += ",%s" % (str(i))

j=0

break

j +=1

else:

j = 0

break

i += 1

if not offset\_positions:

print "-1"

else:

print offset\_positions

**Part 2-3 Boyer-Moore Algorithm:**

import numpy as np

my\_library = ['A', 'T', 'G', 'C']

def match\_characters(reference, substring, position):

for i in list(range((len(substring)-1),-1,-1)):

if substring[i] != reference[i + position] and reference[i + position] != "N" and substring[i] != "N":

return([False, i, substring[i], reference[i + position]])

return([True])

def make\_gs\_mismatch\_set(substring):

mismatch\_set = []

mismatch\_set.append([["N"],[substring[1:]]])

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

substring = substring[1:]

mismatch\_set.append([[substring[0]],[substring[1:]]])

mismatch\_set.append([[substring[1]],["N"]])

return(mismatch\_set)

def make\_good\_suffix\_list(string):

mismatch\_set = make\_gs\_mismatch\_set(string)

#print(mismatch\_set)

reference = "N"\*len(string) + string

shift\_vector = []

for i in list(range(0,len(mismatch\_set))):

start = len(reference)-len(mismatch\_set[i][1][0])-1

end = (start - len(string))

shift = 1

skip = []

for j in list(range(start, end, -1)):

end\_of\_start = j + len(mismatch\_set[i][1][0])

reference\_subset = reference[j:end\_of\_start]

#print(reference\_subset)

match = match\_characters(reference\_subset, mismatch\_set[i][1][0], 0)

#print(match)

if match[0] == True and skip == []:

match2 = match\_characters(reference[j-1], mismatch\_set[i][0][0], 0)

if match2[0] == False or reference[j-1] == "N":

shift\_vector.append(shift)

skip.append(True)

#print(shift\_vector)

else:

shift += 1

elif skip == []:

shift += 1

if shift == len(string):

shift\_vector.append(len(string))

return(shift\_vector)

def make\_bad\_character\_matrix(substring, library):

bad\_character\_matrix = {}

for char in library:

bad\_character\_matrix[char] = []

n = 1

for i in list(range(0,len(substring))):

if char != substring[i]:

bad\_character\_matrix[char].append(n)

n += 1

else:

n = 1

bad\_character\_matrix[char].append(0)

return(bad\_character\_matrix)

def match(reference, substring, library):

bad\_character\_matrix = make\_bad\_character\_matrix(substring, library)

shift\_vector = make\_good\_suffix\_list(substring)

final\_position = len(reference) - len(substring)

reference\_match\_start\_positions = []

position = 0

while position <= final\_position:

match\_characteristics = match\_characters(reference, substring, position)

if match\_characteristics[0] == True:

reference\_match\_start\_positions.append(position)

position += 1

#print(position)

else:

mismatch\_position = match\_characteristics[1]

correct\_character = match\_characteristics[3]

bad\_character\_shift = bad\_character\_matrix[correct\_character][mismatch\_position]

good\_suffix\_shift = shift\_vector[mismatch\_position]

#print([good\_suffix\_shift, bad\_character\_shift])

best\_shift = max(good\_suffix\_shift, bad\_character\_shift)

position += best\_shift

#print(position)

if reference\_match\_start\_positions == []:

return(-1)

else:

return(reference\_match\_start\_positions)

def test\_match(substring, library):

mean = []

for n in list(range(20)):

reference\_size = (4\*\*len(substring))\*10

reference = ""

for i in list(range(0, reference\_size)):

position = np.random.random\_integers(0, 3)

reference += library[position]

mean.append(len(match(reference, substring, library)))

print(np.mean(mean))

substring = "GCGACG"

reference = "GCGACGGCGACGAGCTTTGCGCAGCGACGGCTAGCCTGACTGCGACGGCGACGCTACGTTAGCCGCATCTACGCGACGCTAGGTTGTTTGCTAGCTACCCGATTAAAGCTTCAGTCAGATCGATCTGATCGCGACGGAGCTGCGACGTTAGGCGACGGCGACG"

print(match(reference, substring, my\_library))