Motif finding using

KMP Algorithm

Hitesh Arora (50489713)

Sulav Regmi (50211843)

CS 6713: Advanced Analysis of Algorithms

03/01/2017

Dr. Xiuzhen Huang

Introduction

KMP algorithm or Knuth-Morris-Pratt algorithm is a string-searching algorithm that searches for a certain pattern in a string. The algorithm performs comparisons from left to right which has a time complexity of *O(m)* for the preprocessing phase where *m* is the size of the pattern and has a time complexity of *O(m+n)* for the searching phase where *n* is the size of the string. This algorithm has a faster time complexity than some other methods such as “naïve string-matching algorithm”.

KMP algorithm works by first defining the prefix function (***setupLookup[[1]](#footnote-1)***) to look at the pattern to be matched which then produces something that represents the length of the longest prefix that can be found at the end of the sequence, up until the position in it that is being looked at. For example, if the pattern to be matched against were “***abcaby***”, then the prefix function (***setupLookup***) would return a list of numbers as “**[0, 0, 0, 1, 2, 0]**”, because when looking at the first character, a whole string cannot count as either a suffix or a prefix, so the result is zero. Looking at the second and third character, the same thing applies resulting in zero; the fourth character “***a***” can be found there, and as a prefix. Therefore, the value “**1**” is returned, as the character that can be found both at the start and the end has length one. Similarly when character “***b***” is found, the value “**2**” is returned. When the position holding the character “***y***” is considered with the algorithm, the suffix does not appear previously in the pattern, so the value is “zero” again. The algorithm looks at how much the pattern can be shifted against itself.

String matching is very important when looking at RNA and DNA sequences. It can lead to being able to search for particular patterns in the RNA and/or DNA sequences referred to as “motifs”. So searching for these particular patterns within the sequences is called Motif finding.

Source code

################################################

# Advanced Analysis of Algorithms

# Project: Motif finding using KMP Algorithm

# Members:

# Hitesh Arora (50489713),

# Sulav Regmi (50211843)

# Dr. Huang

#

# Using KMP algorithm or Knuth-Morris-Pratt string searching algorithm

# to find Motif in the sequences

# ##############################################

def readFromFile():

"""

Read sequences from sequence.fasta file and store each sequence in a list by skipping every other line

:return: list of sequences

"""

sequences = []

line\_idx = 0

with open('sequence.fasta') as f:

content = f.read()

for line in content.splitlines():

if line\_idx % 2 == 0:

line\_idx += 1

continue

else:

line\_idx += 1

sequences.append(line)

return sequences

def check(lookup\_arr, result, pattern, string):

"""

Compares provided pattern against the string passed, part of the KMP

algorithm

:param lookup\_arr: list of

:param result: empty string for returning

:param pattern: pattern to be matched against the sequence

:param string: sequence to be matched against

:return: string of size of pattern length

"""

i, j = 0, 0

while j < len(lookup\_arr):

if i < len(string):

if string[i] == pattern[j]:

result += string[i]

i += 1

j += 1

else:

if j != 0:

j -= 1

j = lookup\_arr[j]

continue

else:

i += 1

continue

else:

break

return result[-len(pattern):]

def setupLookup(pattern):

"""

It build us the prefix list, which is part of the KMP algorithm

:param pattern: pattern to be matched against the sequence

:return: prefix list

"""

i, j = 1, 0

arr = [0] \* len(pattern)

while i < len(pattern):

if pattern[j] == pattern[i]:

arr[i] = j + 1

i += 1

j += 1

else:

if j != 0:

j -= 1

j = arr[j]

continue

else:

arr[i] = 0

i += 1

continue

return arr

PATTERN\_LENGTH = 15

# storing list of all the sequences return by the function

sequences = readFromFile()

# looping through the sequence taking string of length same as pattern length

# and then using KMP to find match in all sequences

idx = 0

while idx < len(sequences[0]) - PATTERN\_LENGTH:

ptrn = sequences[0][idx:(idx + PATTERN\_LENGTH)]

res = ''

arr = setupLookup(ptrn)

matched = []

for sequence in range(1, len(sequences)):

matched.append(check(arr, res, ptrn, sequences[sequence]))

if (all(x == ptrn for x in matched)):

print(ptrn, ' matched in all sequences.')

break

else:

idx += 1

if idx == len(sequences[0]) - 16:

print('Match not found in all sequences!')

continue

1. Implemented function in the code below. [↑](#footnote-ref-1)