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.

Algorithm

The algorithm first reads the sequences from the “***sequence.fasta***” file and stores each sequence in a list while skipping every other line, since the skipped line contains the sequence number and not the actual sequence. Then we take the first item in the sequence list and get 15 characters from the sequence at a time and pass that pattern to the (***setupLookup***) function.

The (***setupLookup***) function builds the prefix list by defining two indexes, one (***i***) that starts from the first element of the list and the second (***j***) that starts from the second element of the list. The prefix list is first initialized as an empty list that has the first element as zero. The process is first started by comparing the element at the ***i*th** index against the element at the ***j*th** index, if there is a match, the prefix list at the ***i*th** index is assigned the value of ***j*** plus 1 and increment both ***i*** and ***j*** by 1. If the elements do not match and if the value of ***j*** is not zero, the value of ***j*** is decremented by 1 and is assigned the value of the prefix list at the ***j*th** index, and if the value of ***j*** is zero, the value of the prefix list at the ***i*th** index is assigned to zero and ***i*** is incremented by 1. The process is repeated until the ***i***th value is equal to the length of the pattern passed to the function. This process is the first part of KMP algorithm.

The second part of the algorithm uses the pattern to match against the remaining sequences by passing the pattern, the prefix list, and the sequence to the (***check[[2]](#footnote-2)***) function. The function first defines two indexes, one (***i***) that starts from the first element of the sequence and the second (***j***) that starts from the first element of the pattern. The process is first started by comparing the element at the ***i*th** index of the sequence against the element at the ***j*th** index of the pattern, if there is a match, the prefix list at the ***i*th** index is assigned the value of ***j*** plus 1 and increment both ***i*** and ***j*** by 1 and the value at the ***i*th** index of the sequence is appended to a string (***result***). If the elements do not match and if the value of ***j*** is not zero, the value of ***j*** is decremented by 1 and ***j*** is assigned the prefix list’s at ***j*th** value, and if the value of ***j*** is zero, the value of ***i*** is incremented by 1. The process is repeated until the ***j***th value is equal than the length of the prefix list. The function returns the last part of the string (***result***) of size of the length of the pattern. This process is the second part of KMP algorithm.

Then we append the returned (***result***) string to a new list (***matched***). Then we check if the pattern exists in all of the elements of the list (***matched***), if found, the pattern exists in all the sequences.

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)
2. Implemented function in the code below. [↑](#footnote-ref-2)