Homework 1

Elliott Pryor

7 September 2021

Problem 1

For the following mRNA sequence, can you extract its 5' UTR, 3' UTR and the protein sequence? ACTTGTCATGGTAACTCCGTCGTACCAGTAGGTCATG

So the above is actually DNA, since it has T's. So we assume that the T's are actually U's.

So in RNA we have: ACUUGUCAUGGUAACUCCGUCGUACCAGUAGGUCAUG

We then look for the start codon (Met) which is AUG. So we get:

ACUUGUC AUG GUA ACU CCG UCG UAC CAG UAG GUC AUG

The 5' UTR is everything left of the first AUG which is: ACUUGUC. The stop codons are either UAA, UAG, or UGA. We see UAG, so the 3' UTR is everything right of this: GUCAUG

Thus our coding region is:

AUG GUA ACU CCG UCG UAC CAG UAG $\rightarrow$ MET VAL THR PRO SER TYR GLN Stop

Problem 2 Implement the Z-algorithm in the language of your choice. Also, code up the Exact Pattern Matching algorithm that makes use of the Z algorithm. Demonstrate both algorithms on some test strings (use screen shots to show runs).

Below are some screenshots showing functionality. They were run in a python shell and imported the python file that I made for concision and clarity.

```
>>> import HW1.z_algorithm as z
>>> z.z_alg('aabbabaaa')
[0, 1, 0, 0, 1, 0, 2, 2, 1]
>>> z.z_alg('aaaaaaaa')
[0, 7, 6, 5, 4, 3, 2, 1]
>>> z.z_alg('ababbba')
[0, 0, 2, 0, 0, 0, 1]
>>> z.z_alg('aaaaacaaaaaa')
[0, 4, 3, 2, 1, 0, 5, 5, 4, 3, 2, 1]
```

Figure 1: Examples using base z algorithm

```
>>> import HW1.z_algorithm as z
>>> z.z_pattern_match("aaa", "abaaabaabaaa")
[2, 9]
>>> z.z_pattern_match('aaa', 'aaaaaaaaaaaa')
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> z.z_pattern_match('ab', 'hello world')
[]
```

Figure 2: Examples using pattern matching

I also did the bonus, which runs via the command line. Here is an example below showing functionality.

My example FASTA input:

```
>seq0, example file from
    http://prodata.swmed.edu/promals/info/fasta_format_file_example.htm
    with last sequence modified
FQTWEEFSRAAEKLYLADPMKVRVVLKYRHVDGNLCIKVTDDLVCLVYRTDQAQDVKKIEKF
>seq1
KYRTWEEFTRAAEKLYQADPMKVRVVLKYRHCDGNLCIKVTDDVVCLLYRTDQAQDVKKIEKFHSQLMRLME
    LKVTDNKECLKFKTDQAQEAKKMEKLNNIFFTLM
>seq2
```

```
    EEYQTWEEFARAAEKLYLTDPMKVRVVLKYRHCDGNLCMKVTDDAVCLQYKTDQAQDVKKVEKLHGK
>seq3
MYQVWEEFSRAVEKLYLTDPMKVRVVLKYRHCDGNLCIKVTDNSVCLQYKTDQAQDVK
>seq4
EEFSRAVEKLYLTDPMKVRVVLKYRHCDGNLCIKVTDNSVVSYEMRLFGVQKDNFALEHSLL
>seq5
SWEEFAKAAEVLYLEDPMKCRMCTKYRHVDHKLVVKLTDNHTVLKYVTDMAQDVKKIEKLTTLLMR
>seq6
FTNWEEFAKAAERLHSANPEKCRFVTKYNHTKGELVLKLTDDVVCLQYSTNQLQDVKKLEKLSSTLLRSI
>seq7
SWEEFVERSVQLFRGDPNATRYVMKYRHCEGKLVLKVTDDRECLKFKTDQAQDAKKMEKLNNIFF
>seq8
SWDEFVDRSVQLFRADPESTRYVMKYRHCDGKLVLKVTDNKECLKFKTDQAQEAKKMEKLNNIFFTLM
>seq9
KNWEDFEIAAENMYMANPQNCRYTMKYVHSKGHILLKMSDNVKCVQYRAENMPDLKK
>seq10
THISISABSOLUTGIBBRISHJUSTTOSHOWITSOUTPUTBUTNOMATCH
```

```
PS C:\Users\pryor\Documents\GithubProjects\CompBio\HW1> python .\z_algorithm.py E example.fasta
Opening: example.fasta
Matching pattern: E
Analyzing Sequence: seq0, example file from http://prodata.swmed.edu/promals/info/fasta_format_file_example.htm
Found matches 4, 5, 11, 59
Analyzing Sequence: seq1
Found matches 5, 6, 12, 60, 71, 80, 91, 96
Analyzing Sequence: seq2
Found matches 0, 1, 6, 7, 13, 61
Analyzing Sequence: seq3
Found matches 5, 6, 12
Analyzing Sequence: seq4
Found matches 0, 1, 7, 43, 57
Analyzing Sequence: seq5
Found matches 2, 3, 9, 14, 57
Analyzing Sequence: seq6
Found matches 4, 5, 11, 19, 33, 59
Analyzing Sequence: seq7
Found matches 2, 3, 6, 29, 41, 57
Analyzing Sequence: seq8
Found matches 3, 17, 41, 52, 57
Analyzing Sequence: seq9
Found matches 3, 6, 10, 49
Analyzing Sequence: seq10
No match found
```

Figure 3: Program output on example Fasta input

```
"""
Author - Elliott Pryor
Created on - 2 September 2021
"""
import sys


def print_output(z):
    if len(z) > 0:
        z = [str(i) for i in z]
        output = ', '.join(z)
        print(f"Found matches {output}")
    else:
        print("No match found")

def find_next_match(s1, s2, S):
    """
    Finds the next match in S starting from s1 and s2
    Compares S[s1, n] to S[s2, n]
    We don't pass in slices because this is faster
    :param s1: Starting of first string (base)
    :param s2: Start of second string (prefix to match)
    :param S: String to search
    :return: q=s2 index of failed match in S, zk length of match
    """
    n = len(S)
    zk = 0
    while (s1 < n and s2 < n) and (S[s1] == S[s2]):  # while they match and in bounds
        s1 += 1
        s2 += 1
        zk += 1
    return s2, zk


def z_alg(S):
    """
    Run the z prefix matching algorith
    :param S: String input
    :return: Z score for each index (note first one is always 0)
    """
    l, r = 0, 0  # initialize
    z = [0 for _ in S]  # init z scores
    for k in range(1, len(S)):
        if k > r:  # match them
            _, zk = find_next_match(0, k, S)
            if zk > 0:
                r = k + zk - 1
                l = k
                z[k] = zk
        else:
            kp = k - l
            b = r - k + 1  # ||B|| from alg descrip
            if z[kp] < b:
                z[k] = z[kp]
            else:
```

```
                q, zk = find_next_match(b, r + 1, S)
                z[k] = q - k
                l = k
                r = q - 1
    return z


def z_pattern_match(pattern, text, special_char=None):
    """
    Finds all occurrences of the pattern in the text
    :param pattern: pattern to search for
    :param text: text to search in
    :param special_char: special division characters. Can specify a char, or list of chars.
                         Defaults are $, %, ^, ~, `, #
    :return: Index in text of start of pattern match
    """
    if special_char is None:
        special_char = ['$', '%', '^', '@', '~', '`', '#']  # our default options for special chars

    sep_char = ''
    if not isinstance(special_char, str):  # if it isn't just a string
        for c in special_char:
            if c not in text and c not in pattern:  # 2 * O(n) each time
                sep_char = c
                break
    S = pattern + sep_char + text  # put special char in the middle
    z = z_alg(S)
    m = len(pattern)
    matches = []
    for j in range(1, len(z)):
        if z[j] == m:
            matches.append(j - (m + 1))
    return matches


def analyze_fasta(pattern, filepath):
    file = open(filepath)
    print(f"Opening: {filepath}")
    print(f"Matching pattern: {pattern}")
    started = False
    text = ''
    for line in file:
        if line.startswith('>'):
            if started:
                z = z_pattern_match(pattern, text.replace('\n', ''))
                text = ''
                print_output(z)
            started = True
            print(f"Analyzing Sequence: {line.replace('>', '').strip()}")
        else:
            text += line.strip()
    z = z_pattern_match(pattern, text.replace('\n', ''))
    print_output(z)
```

```
109
110
111 if __name__ == '__main__':
112     if len(sys.argv) == 3:
113         # run fasta file
114         # expect arguments z_algorithm.py <pattern> <path to file>
115         pattern = sys.argv[1]
116         filepath = sys.argv[2]
117         analyze_fasta(pattern, filepath)
118     else:
119         print(z_alg('aabbabaaa'))
```