DAT158 Assignment 1
587728 Joachim Leiros

Problem 1:
Text is:
aabaadaabaaa
String to match is aabaaa.

Brute-force

a	a	a	b	a	a	d	a	a	b	a	a	a
a	a	b	a	a	a							
	a	a	b	a	a	a						
		a	a	b	a	a	a					
			a	a	b	a	a	a				
				a	a	b	a	a	a			
					a	a	b	a	a	a		
						a	a	b	a	a	a	
							a	a	b	a	a	a

Problem 2:
a)
Boyer-moore

a	a	a	b	a	a	d	a	a	b	a	a	a
a	a	b	a	a	a							
	a	a	b	a	a	a						
							a	a	b	a	a	a

b)

```
def BoyerMoreComparison(text, pattern):
      x = len(pattern)
      y = len(text)
      a = x-1
      b = x-1
      comparisons = 0

      while a <= y-1:
      comparisons += 1
      if pattern[b] == text[a]:
            if b == 0:
                  return str(comparisons/x)

            else:
                  a -= 1
                  b -= 1
      else:
            a = a+x - min(b, 1 + lo(text[a], pattern))
            b = x-1
      return "No pattern"

def lo(character, pattern):
      index = -1
      for pos, char in enumerate(pattern):
      if character == char:
            index = pos
      return index

def main():
      text = "joachim gjør ting han ikke kan"
      pattern = "oachi"
      result = BoyerMoreComparison(text, pattern)
      print(result)

if __name__ == '__main__':
      main()
```

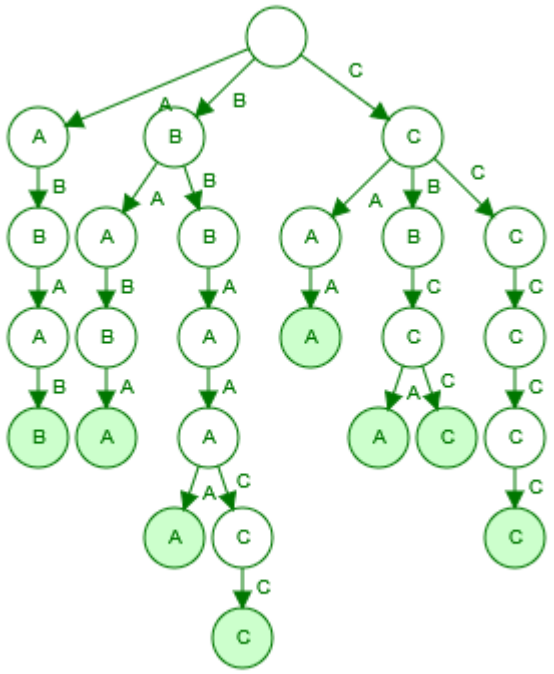
Result:
Pattern starts at position 1.
There was 1.2 comparisons per character.

Problem 3:
Knuth-Morris-Pratt

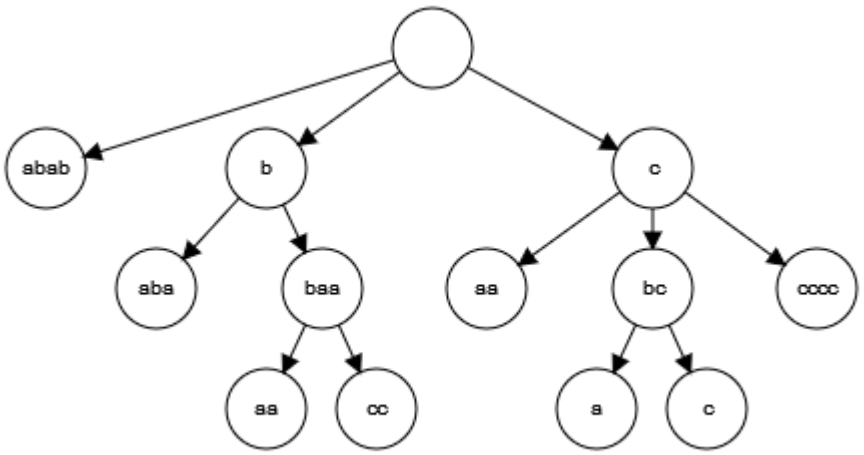
a	a	a	b	a	a	d	a	a	b	a	a	a
a	a	b	a	a	a							
	a	a	b	a	a	a						
				a	a	b	a	a	a			
					a	a	b	a	a	a		
							a	a	b	a	a	a

Problem 4:
In the string P=”aaabbaaa” there are 3 nonempty prefixes which are also suffixes of the string P.
a, aa and aaa.

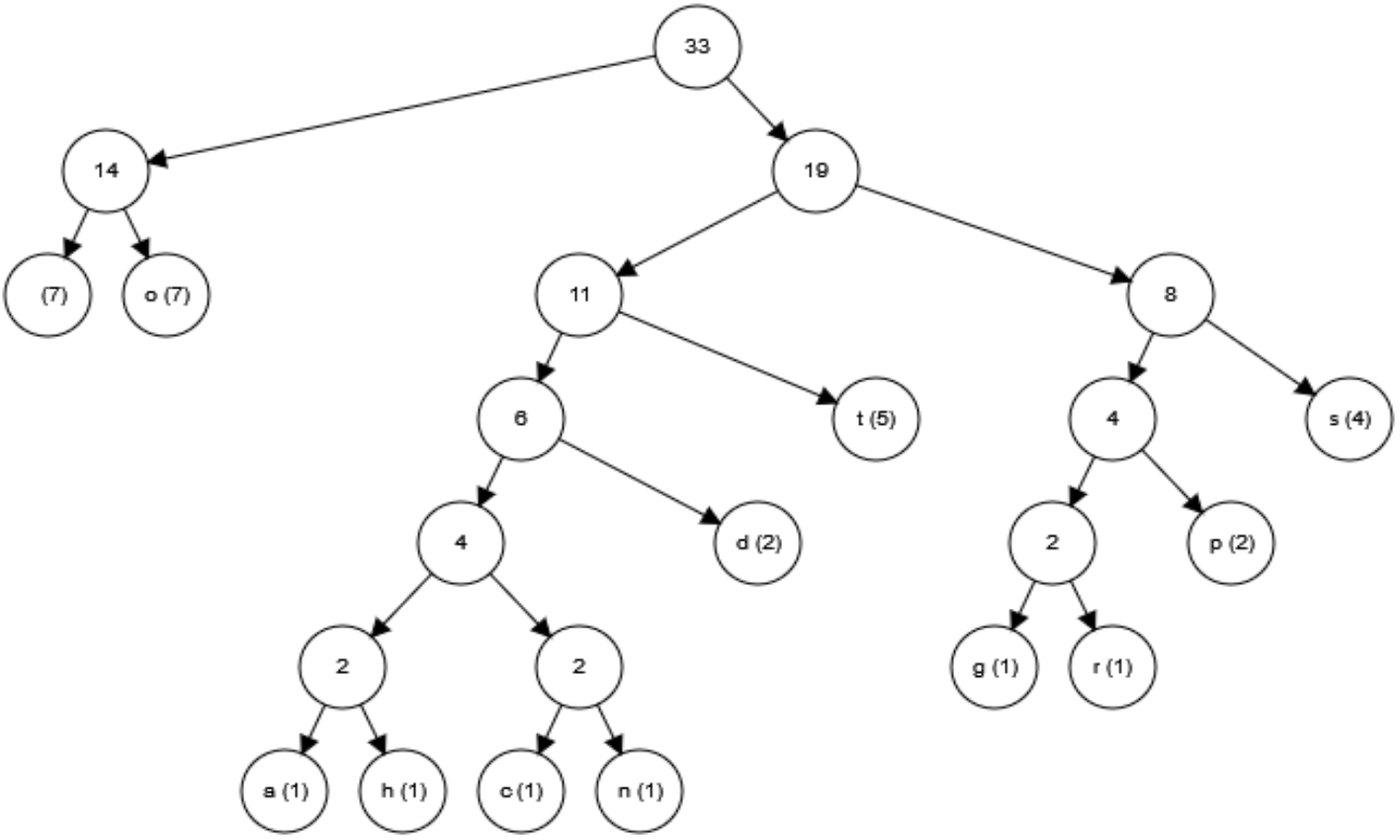
Problem 5:
Standard trie



Problem 6:
Compressed trie



Problem 7 Huffman and frequency tree



Char	Frequency
Space	7
o	7
t	5
s	4
d	2
p	2
a	1
h	1
c	1
m	1
g	1:L
r	1

Problem 8:
To find the longest common subsequence we can find it by:

We have string `X[0..i]` and `Y[0..j]`

`LCS L[i,j]`

if the character `b = X[i] = Y[j]` then the LCS would end with a b
then `L[i,j] = L[i-1,j-1]+1`
`if X[i] =/= Y[j]`, then the sequence cannot end with the character b
then `L[i,j] = max{L[i-1,j], L[i,j-1]`

There are also some boundary cases

```
L[i,-1] = 0 for i= -1,0,...,n-1
L[-1,j] = 0 for j = -1,0,...,m-1
```

L	-1	b	b	a	b	b	a	a	a	b
-1	-1	0	0	0	0	0	0	0	0	0
b	0	1	1	1	1	1	1	1	1	1
a	0	1	1	2	2	2	2	2	2	2
b	0	1	2	2	3	3	3	3	3	3
b	0	1	2	2	3	4	4	4	4	4
a	0	1	2	3	3	4	5	5	5	5
b	0	1	2	3	4	4	5	5	5	6
a	0	1	2	3	4	4	5	6	6	6
b	0	1	2	3	4	5	5	6	6	7

= babbaab

Problem 9:

a)

```
def longestseq(s1, s2, a, b):
      if a == 0 or b == 0:
            return 0;
      elif s1[a-1] == s2[b-1]:
            return 1 + longestseq(s1, s2, a-1, b-1);
      else:
            return max(longestseq(s1, s2, a, b-1), longestseq(s1,
s2, a-1, b));

def main():
      s1 = "babbabab"
      s2 = "bbabbaaab"
      l1 = len(s1)
      l2 = len(s2)
      result = longestseq(s1, s2, l1, l2)
      print(result)

if __name__ == '__main__':
      main()
Longest common subsequence is 7.
```

b)

```
def dynamiclongestseq(s1, s2):
      l1 = len(s1)
      l2 = len(s2)

      L = [[None]*(l2+1) for a in range(l1+1)]

      for a in range(l1+1):
      for b in range(l2+1):
            if a == 0 or b == 0 :
                  L[a][b] = 0
            elif s1[a-1] == s2[b-1]:
                  L[a][b] = L[a-1][b-1]+1
            else:
                  L[a][b] = max(L[a-1][b] , L[a][b-1])

      return L[l1][l2]


def main():
      s1 = "babbabab"
      s2 = "bbabbaaab"
      result =  dynamiclongestseq(s1, s2)
      print(result)

if __name__ == '__main__':
      main()
```

c)

I tried running the function with an increasing number of characters to compare. Around ~27-30 characters I started to notice the recursive version lagging significantly behind.