DAT158 Compulsory Algorithms exercise 1

The git repository for this assignment is https://github.com/aare-mikael/DAT158_Algorithms

Problem 1:

Draw a figure illustrating the comparisons done by the brute-force pattern matching algorithm for the case when the text is “aaabaadaabaaa” and the pattern is “aabaaa”.

	Text	a	a	a	b	a	a	d	a	a	b	a	a	a
Pattern														
1		a	a	b										
2			a	a	b	a	a	a						
3				a	a									
4					a									
5						a	a	b						
6							a	a						
7								a						
8									a	a	b	a	a	a

Problem 2

a) Repeat the previous problem for the BM matching algorithm, not counting the comparisons made to compute the last function.

	a	a	a	b	a	a	d	a	a	b	a	a	a
1	a	a	b	a	a	a							
2		a	a	b	a	a	a						
3								a	a	b	a	a	a

b) The theory says that experiments shows that on English text, the average number of comparisons done per text character Is approximately 0.24 for a five character pattern string. Implement the algorithm in Python and run it on a Norwegian (or another language) text and compare the results. The reason for using Python is that the Machine learning part of the course use Python.

From my testing I found that the average number of comparisons per character for a pattern string of length 5 was in the range of 0.15 to 0.4 comparisons per character. This was both in Norwegian and English. I tested with variable comparison strings, but always a 5 characters long pattern. See the repository for the code written for this problem.

Problem 3

Repeat the previous problem for the KMP matching algorithm, not counting the comparisons made to compute the failure function.

	a	a	a	b	a	a	d	a	a	b	a	a	a
1	a	a	b	a	a	a							
2		a	a	b	a	a	a						
3					a	a	b	a	a	a			
4						a	a	b	a	a	a		
5							a	a	b	a	a	a	
6								a	a	b	a	a	a

Problem 4

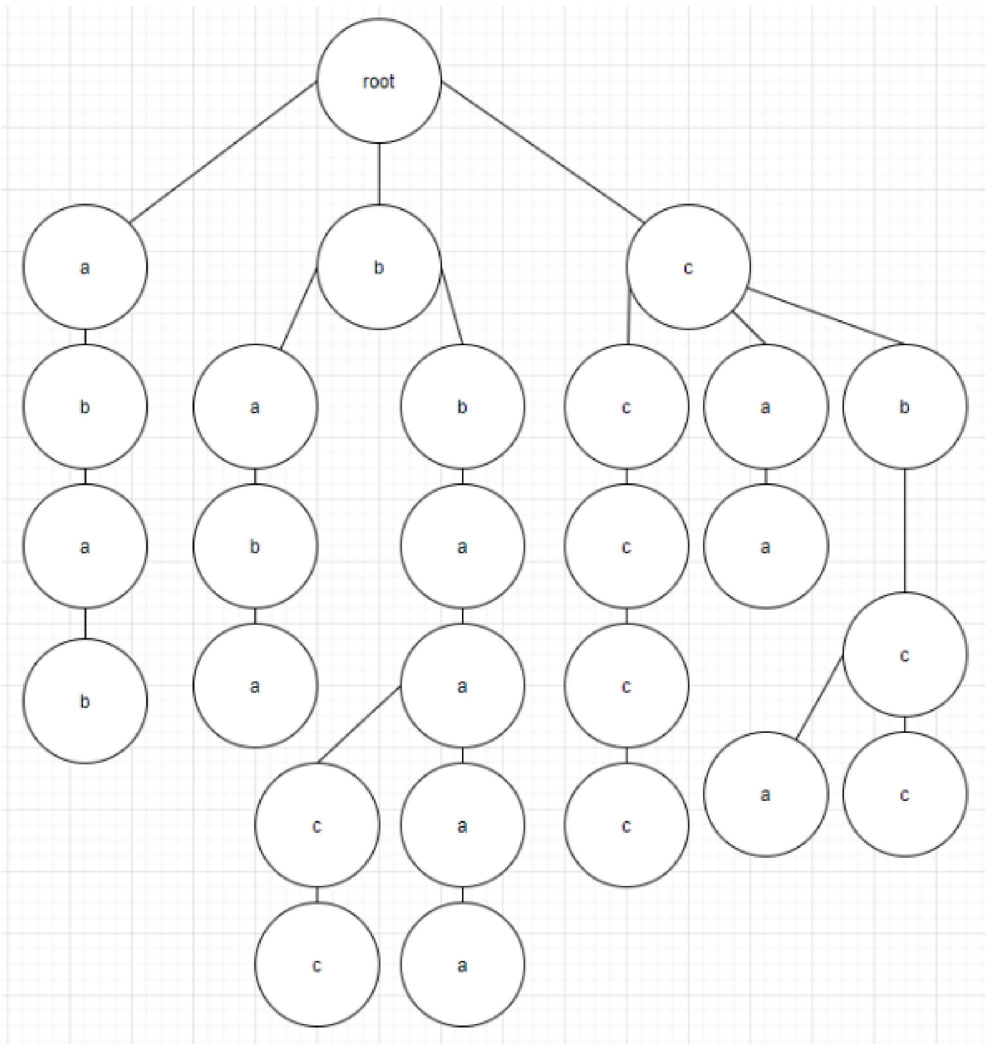
How many nonempty prefixes of the string P= "aaabbaaa" are also suffixes of P?

Three: “a”, “aa” and “aaa” are the only non-empty prefixes of P that are also suffixes of P.

Problem 5

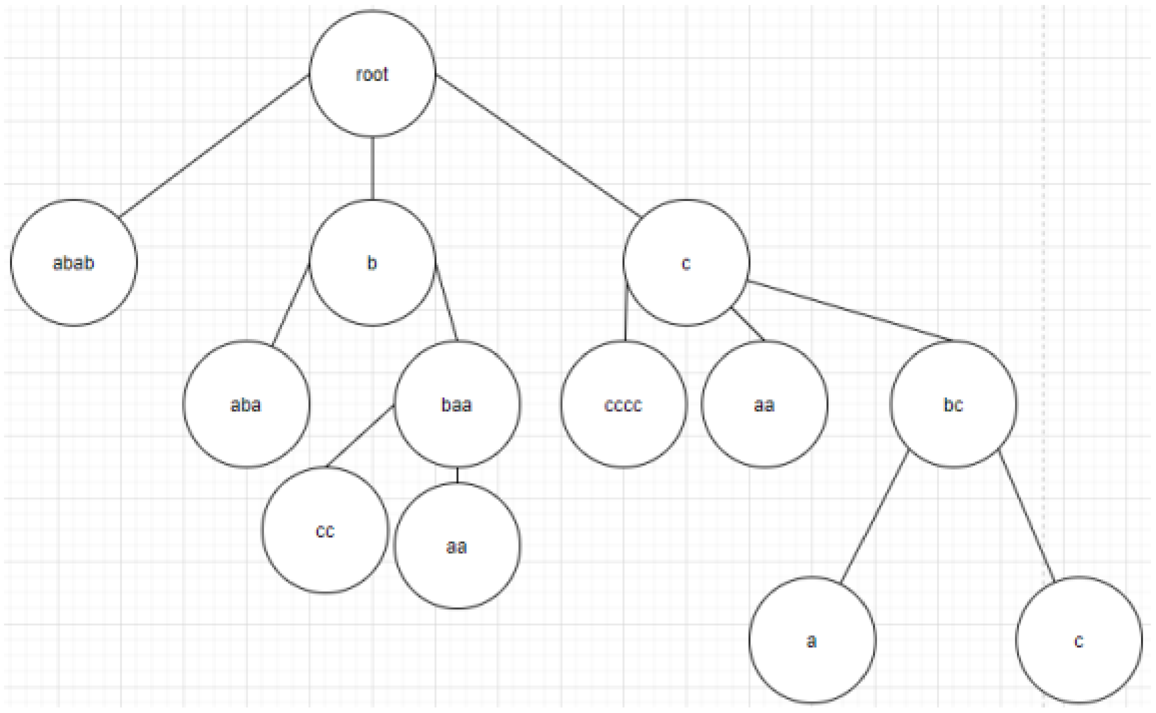
Draw a standard trie for the following set of strings

{abab, baba, ccccc, bbaaaa, caa, bbaacc, cbcc, cbca}



Problem 6:

Draw a compressed trie for the set of strings of the previous problem:

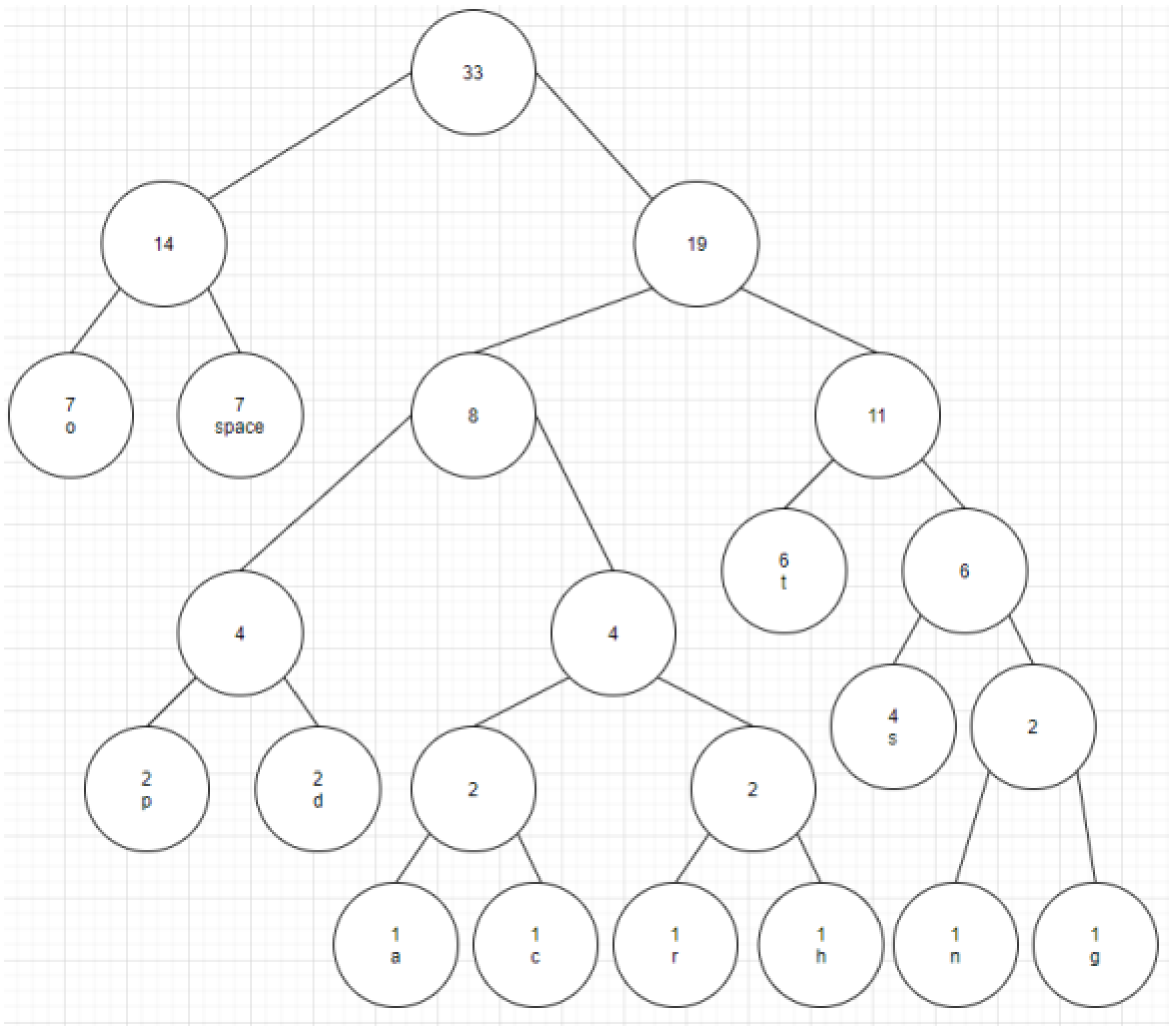


Problem 7

Draw the frequency table and Huffman tree for the following string:

"dogs do not spot hot pots or cats"

Char	Frequency
O	7
Space	7
T	5
S	4
P	2
D	2
A	1
C	1
R	1
H	1
N	1
G	1



Problem 8

Show how to use dynamic programming to compute the number of characters in the longest common subsequence between the two strings "babbabab" and "bbabbaaab".

LCS		b	b	a	b	b	a	a	a	B
	0	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

The LCS between the strings is “babbaab”

Problem 9

a) Implement a recursive version in Python of the longest common subsequence problem.

See lcs_rec.py in the github repository for implementation. Answer is commented in at the bottom.

b) Implement the dynamic version of the longest common subsequence problem in Python.

See lcs_dp.py in the github repository for implementation. Answer is commented in at the bottom.

c) Experiment with strings of different length. At some point the recursive version will start using "a lot of time" while the dynamic version is still fast. Approximately, at what point does this happen?

See impl_comp.py in the github repository for implementation. Answer is commented in at the bottom.