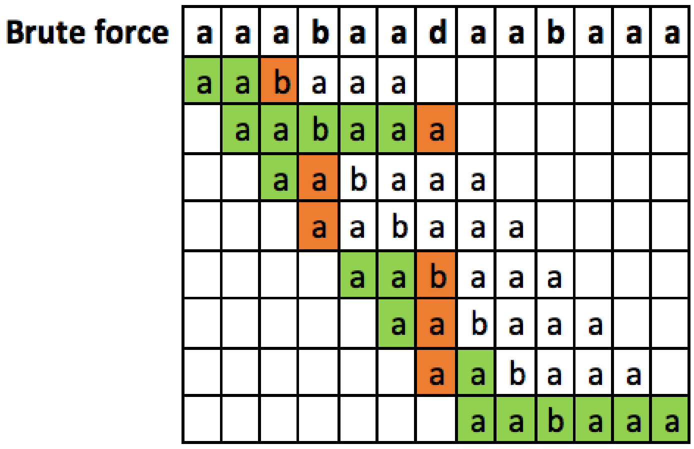
Assignment 1 - Oblig

Problem 1

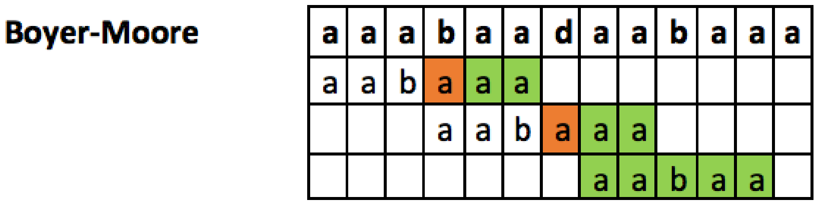
Brute force algorithm on the text `aaabaadaabaaa` with the search string `aabaaa`



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

Boyer-Moore algorithm on the text `aaabaadaabaaa` with the search string `aabaaa`

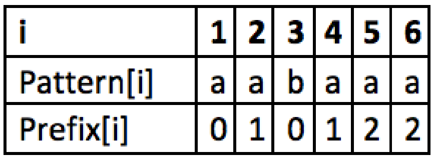


Boyer-Moore	a	a	a	b	a	a	d	a	a	b	a	a	a
	a	a	b	a	a	a							
				a	a	b	a	a	a				
								a	a	b	a	a	

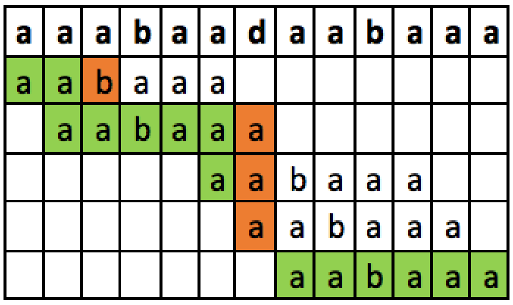
Problem 3

Knut-Morris-Pratt algorithm on the text `aaabaadaabaaa` with the search string `aabaaa`

KMP



i	1	2	3	4	5	6
Pattern[i]	a	a	b	a	a	a
Prefix[i]	0	1	0	1	2	2



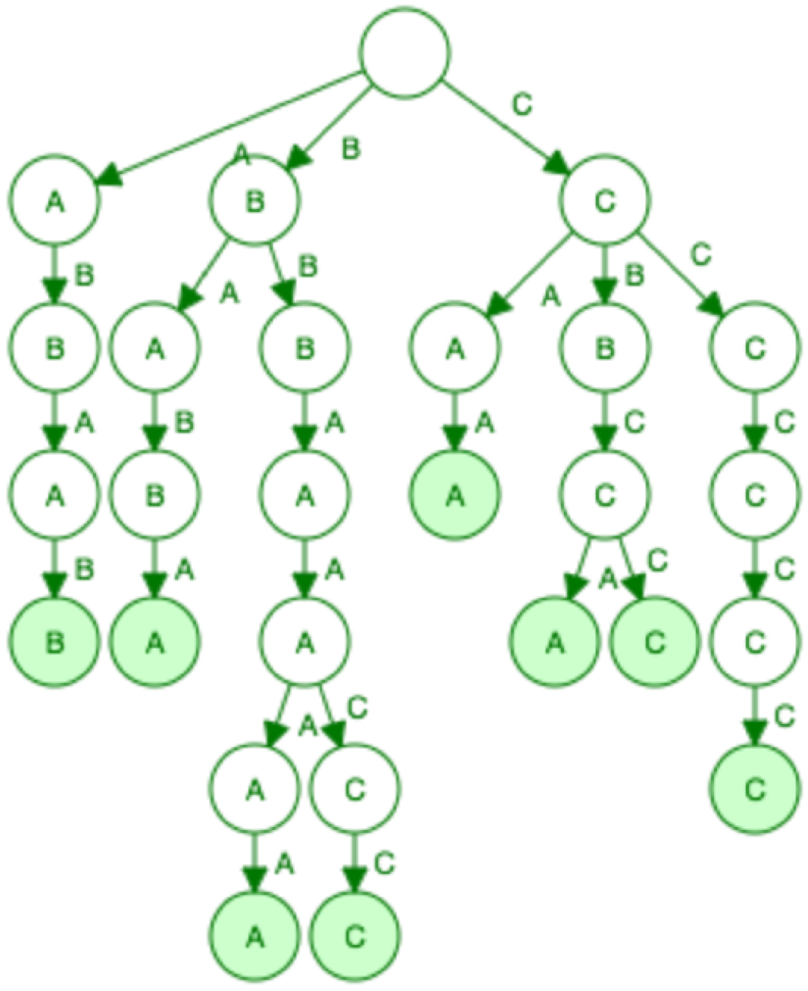
a	a	a	b	a	a	d	a	a	b	a	a	a
a	a	b	a	a	a							
	a	a	b	a	a	a						
					a	a	b	a	a	a		
						a	a	b	a	a	a	
							a	a	b	a	a	a

Problem 4

There are 3 sets of prefixes that are also suffixes of `P = "aaabbaa"`
`{a} {aa} {aaa}`

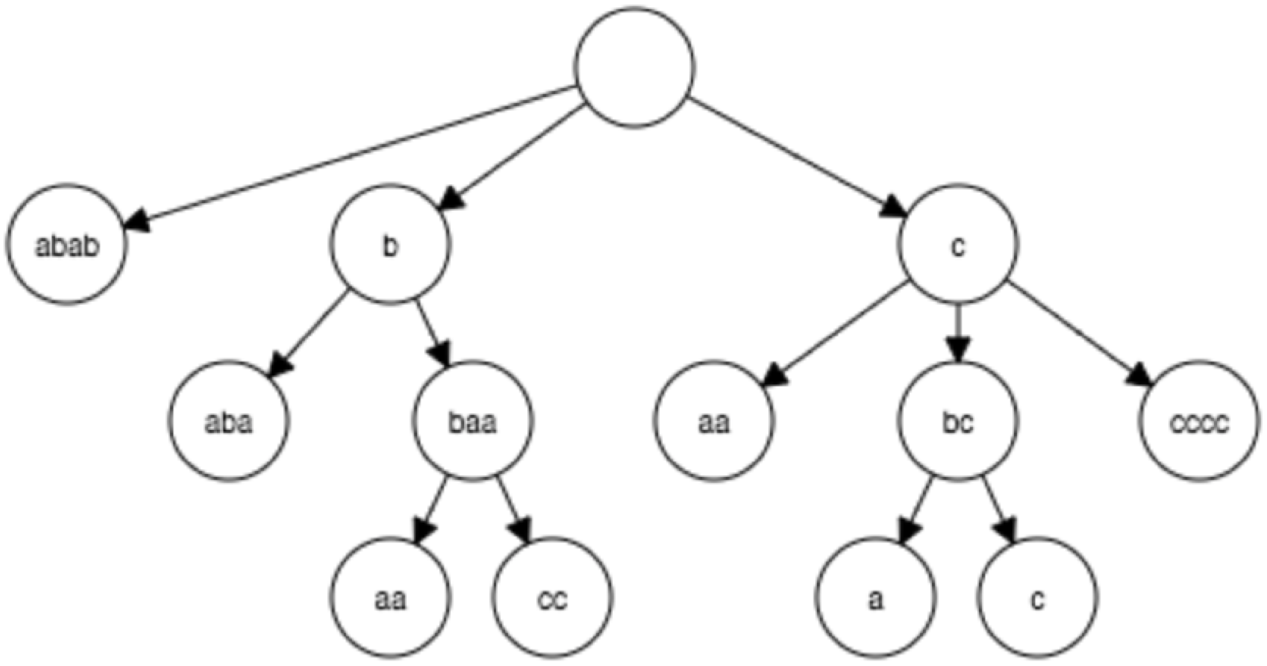
Problem 5

Standard trie for the following set of strings: `{abab, baba, ccccc, bbaaaa, caa, bbaacc, cbcc, cbca}`



Problem 6

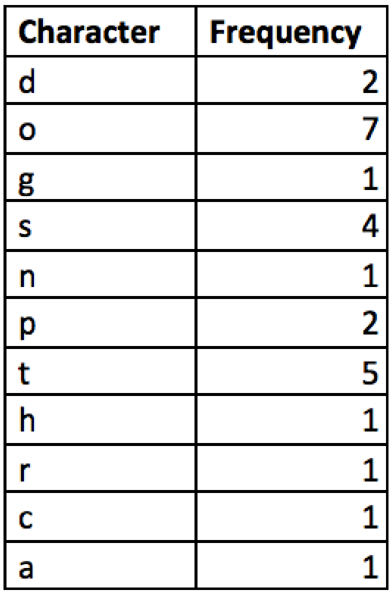
Compressed trie for the following set of strings: `{abab, baba, ccccc, bbaaaa, caa, bbaacc, cbcc, cbca}`



Problem 7

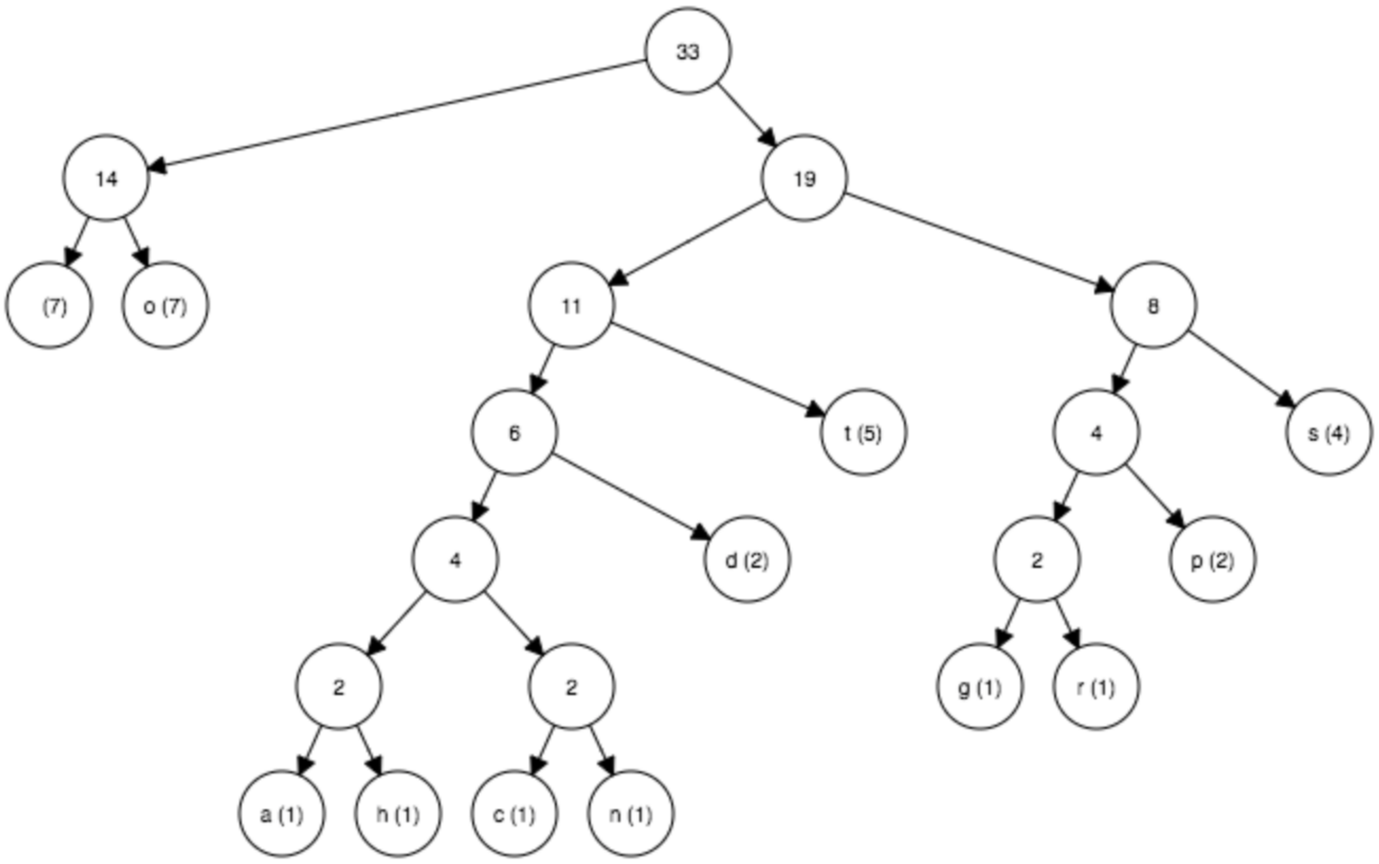
String to analyze: `dogs do not spot hot pots or cats`

Frequency table

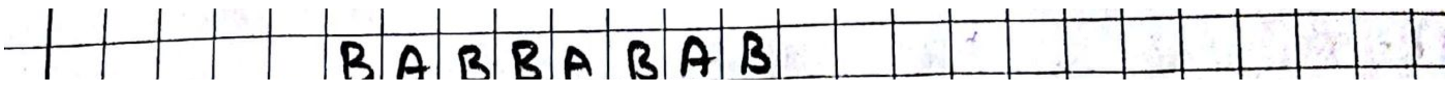


Character	Frequency
d	2
o	7
g	1
s	4
n	1
p	2
t	5
h	1
r	1
c	1
a	1

Huffman tree



Problem 9



B B A B B A A A B

			B	A	B	B	A	B	A	B
		0	1	2	3	4	5	6	7	8
	0	0	0	0	0	0	0	0	0	0
B	1	0	1	1	1	1	1	1	1	1
B	2	0	1	1	2	2	2	2	2	2
A	3	0	1	2	2	2	3	3	3	3
B	4	0	1	2	3	3	3	4	4	4
B	5	0	1	2	3	4	4	4	4	5
A	6	0	1	2	3	4	5	5	5	6
A	7	0	1	2	3	4	5 ←	5	6	6
A	8	0	1	2	3	4	5	5	6	6
B	9	0	1	2	3	4	5	6	6	7
			B	A	B	B	A		A	B

CLS = BABBAAB

As you can see here we get the longest common subsequence `babbaab`

Problem 10

This problem has been solved using "kotlin" as programming language

a) Recursive version

```
fun lcs(a: String, b: String): String {

    val bLen = b.length - 1
    val aLen = a.length - 1
    return when {
        a.isEmpty() || b.isEmpty() -> ""
        a[aLen] == b[bLen] -> lcs(a.substring(0, aLen), b.substring(0, bLen)) + a[aLen]
        else -> {
            val x = lcs(a, b.substring(0, bLen))
            val y = lcs(a.substring(0, aLen), b)
            if (x.length > y.length) x else y
        }
    }
}
```

b) Dynamic version

```
fun lcsDynamic(a: String, b: String): String {
    val lengths = Array(a.length + 1) { IntArray(b.length + 1) }

    // row 0 and column 0 are initialized to 0 already

    for (i in 0 until a.length)
        for (j in 0 until b.length)
            if (a[i] == b[j])
                lengths[i + 1][j + 1] = lengths[i][j] + 1
            else
                lengths[i + 1][j + 1] = Math.max(lengths[i + 1][j], lengths[i][j + 1])

    // read the substring out from the matrix
    val sb = StringBuffer()
    var x = a.length
    var y = b.length
    while (x != 0 && y != 0) {
        when {
            lengths[x][y] == lengths[x - 1][y] -> x--
            lengths[x][y] == lengths[x][y - 1] -> y--
            else -> {
                assert(a[x - 1] == b[y - 1])
                sb.append(a[x - 1])
                x--
                y--
            }
        }
    }
    return sb.reverse().toString()
}
```

c) Main to test the program

```
fun main(args: Array<String>) {
    val a = "aabbaaaaabaaaabaadaaacaacccaadaabbaaabcccdddaasa"
    val b = "aabbaaab"


    println(measureTimeMillis {
        lcs(a, b)
    })
    println(measureTimeMillis {
        lcsDynamic(a, b)
```

```
      })
  }
```

Recursive version uses 24ms

Dynamic version uses 4ms