

Tutorial Week 4

Exercise 1. *The following algorithm calculates the table of prefixes of a string x of length m . The value of position s in the table of prefixes of x is given by the longest common prefix of x and its suffix starting at position s .*

Algorithm 1 Compute table of prefixes(string x ; integer m)

```
1: pref[0] = m
2: g = 0
3: f = 1
4: for  $i = 1$  to  $m - 1$  do
5:   if  $i < g$  and  $\text{pref}[i - f] \neq g - i$  then
6:      $\text{pref}[i] = \min(\text{pref}[i - f], g - i)$ 
7:   else
8:      $g = \max(g, i)$ 
9:      $f = i$ 
10:    while  $g < m$  and  $x[g] == x[g - f]$  do
11:       $g = g + 1$ 
12:    end while
13:     $\text{pref}[i] = g - f$ 
14:  end if
15: end for
16: return pref
```

For each string, fill up the values in the following table of prefixes.

strings	0	1	2	3	4	5	6	7	8	9	10
ababa											
abcacabb											
abcacababc											
abacabacab											

What is the complexity of the algorithm?

Exercise 2. Consider the Boyer-Moore search algorithm, that finds a pattern x of length m in a text y of length n (one of the standard benchmark for the practical string search literature). This algorithm uses two ingredients: bad character heuristics and (strong) good suffix heuristics.

Algorithm 2 BM(string x, y ; integer m, n)

```

1: pos = 0
2: while pos  $\leq n - m$  do
3:    $i = m - 1$ 
4:   while  $i \geq 0$  and  $x[i] == y[\text{pos} + i]$  do
5:      $i = i - 1$ 
6:   end while
7:   if  $i == -1$  then
8:     output:  $x$  ‘occurs in’  $y$  ‘at position’ pos
9:     pos = pos +  $\text{period}(x)$ 
10:  else
11:    pos = pos +  $\max(d[i], DA[y[\text{pos} + i]] - m + i + 1)$ 
12:  end if
13: end while

```

The following algorithm implements the bad-character rule. Basically, this returns for each symbol of the alphabet, the length of the shift, considering the last occurrence.

Algorithm 3 Compute DA(string x ; integer m)

```

1: for all  $\sigma$  in  $\Sigma$  do
2:    $DA[\sigma] = m$ 
3: end for
4: for  $i = 0$  to  $m - 2$  do
5:    $DA[x[i]] = m - i - 1$ 
6: end for
7: return  $DA$ 

```

Fill up the bad-character table for each of the following patterns *ababa*, *abcacabb*, *abcacababc*, and *abacabacab*.

$DA[i]$	a	b	c
<i>ababa</i>			
<i>abcacabb</i>			
<i>abcacababc</i>			
<i>abacabacab</i>			

In the Boyer-Moore algorithm one also needs a displacement table. To compute this table for a given string (aka. the pattern) one needs to make use of the table of suffixes of the string, which is computed analogous to the table of prefixes for that string. Fill in the table of suffixes associated to each of the patterns *ababa*, *abcacabb*, *abcacababc*, and *abacabacab*.

suff[<i>i</i>]	0	1	2	3	4	5	6	7	8	9
<i>ababa</i>										
<i>abcacabb</i>										
<i>abcacababc</i>										
<i>abacabacab</i>										

- a) What is the worst case running time of the algorithm when the pattern is not present in the text.
b) What is the worst case running time of the algorithm when the pattern is present in the text.
c) Considering the pattern $x = aba$ and the text $y = abcacacabac$, and the displacement table associated to the pattern.

	0	1	2
suff[<i>i</i>]			
<i>d[i]</i>	2	2	1

Run the Boyer-Moore algorithm and fill up the values of the following table, including repeats (even if you repeat values, you need to see how pos and *i* increase and decrease, respectively).

pos											
<i>i</i>											

Exercise 3. Construct for each of the following strings, their corresponding string matching automaton: *ababa*, *abcacabb*, *abcacababc*, *abacabacab*. Give the table of transitions. How many backward arcs has each automaton?