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)

```
pref[0] = m
g = 0
f = 1
for i = 1 to m − 1 do
   if i < g and pref[i − f] ≠ g − i then
      pref[i] = min(pref[i − f], g − i)
   else
      g = max(g, i)
      f = i
      while g < m and x[g] == x[g − f] do
         g = g + 1
      end while
      pref[i] = g − f
   end if
end for
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?

Solution:

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

The algorithm runs in time $\Theta(m)$ with less than $2m$ comparisons between the letters of the string x. ■

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)

pos $= 0$
while pos $\leq n - m$ **do**
    $i = m - 1$
    while $i \geq 0$ **and** $x[i] == y[\text{pos} + i]$ **do**
        $i = i - 1$
    end while
    if $i == -1$ **then**
        output: x 'occurs in' y 'at position' pos
        pos $=$ pos $+$ $\mathit{period}(x)$
    else
        pos $=$ pos $+ \max(d[i], DA[y[\text{pos} + i]] - m + i + 1)$
    end if
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)

for all σ **in** Σ **do**
    $DA[\sigma] = m$
end for
for $i = 0$ **to** $m - 2$ **do**
    $DA[x[i]] = m - i - 1$
end for
return DA

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

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

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$[i]$	0	1	2	3	4	5	6	7	8	9
ababa										
abcacabb										
abcacababc										
abacabacab										

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$[i]$			
$d[i]$	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											

Solution:

$DA[i]$	a	b	c
ababa	2	1	5
abcacabb	2	1	3
abcacababc	2	1	5
abacabacab	1	4	2

suff[i]	0	1	2	3	4	5	6	7	8	9
$ababa$	1	0	3	0	5					
$abcacabb$	0	1	0	0	0	0	1	8		
$abcacababc$	0	0	3	0	1	0	0	0	0	10
$abacabacab$	0	2	0	0	0	6	0	0	0	10

a) The worst case running time of the algorithm when the pattern is not present in the text is $\mathcal{O}(n+m)$.
b) The worst case running time of the algorithm when the pattern is present in the text is $\mathcal{O}(mn)$.
c) Considering the pattern $x = aba$ and the text $y = abcacacabac$, fill up the values of the following table, considering the following displacement table associated to the pattern.

	0	1	2
suff[i]	1	0	3
$d[i]$	2	2	1

Furthermore, one needs also the bad-character table DA for aba.

$DA[i]$	a	b	c
aba	2	1	3

Now we can go through the algorithm and fill up the table.

pos	0	3	3	5	5	7	7	7	7	9
i	2	2	1	2	1	2	1	0	-1	-1

■

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?*

Solution: In this solution each state is represented by the corresponding prefix of the string, while the final state is underlined. An easy way to find out the number of backward arcs, is go through the transition table and count in each column how many expressesstates not longer

states	$\varepsilon/0$	a	ab	aba	$abab$	$\underline{ababa}$
a	a	a	aba	a	$ababa$	a
b	0	ab	0	$abab$	0	$abab$

The automaton has 4 backward arcs.

states	$\varepsilon/0$	a	ab	abc	$abca$	$abcac$	$abcaca$	$abcacab$	$\underline{abcacabb}$
a	a	a	a	$abca$	a	$abcaca$	a	a	0
b	0	ab	0	0	ab	0	$abcacab$	$abcacabb$	0
c	0	0	abc	0	$abcac$	0	0	abc	0

The automaton has 7 backward arcs.

states	$\varepsilon/0$	a	ab	abc	$abca$	$abcac$	$abcaca$	$abcacab$	$abcacaba$	$abcacabab$	$\underline{abcacababc}$
a	a	a	a	$abca$	a	$abcaca$	a	$abcacaba$	a	a	$abca$
b	0	ab	0	0	ab	0	$abcacab$	0	$abcacabab$	0	0
c	0	0	abc	0	$abcac$	0	0	abc	0	$abcacababc$	0

The automaton has 9 backward arcs.

states	$\varepsilon/0$	a	ab	aba	$abac$	$abaca$	$abacab$	$abacaba$	$abacabac$	$abacabaca$	$\underline{abacabacab}$
a	a	a	aba	a	$abaca$	a	$abacaba$	a	$abacabaca$	a	$abacaba$
b	0	ab	0	ab	0	$abacab$	0	ab	0	$abacabacab$	0
c	0	0	0	$abac$	0	0	0	$abacabac$	0	0	0

The automaton has 8 backward arcs. ∎