Problem Solving: Homework 3.1

Name: Chen Shaoyuan Student ID: 161240004

December 27, 2017

1 [TC] Problem 32.1-2

NAIVE-STRING-MATCHER'(T,P)

```
1  n = T.length
2  m = P.legnth
3  c = 0
4  for i = 1 to n
5      if T[i] == P[c+1]
6          c = c+1
7      else
8          c = 0
9      if c == m
10         print "Pattern occurs with shift" i-m+1
11         c = 0
```

2 [TC] Problem 32.1-3

For shift s, the expected number of character-to-character comparisons is the expected length of longest common prefix of $T[s..s+m]$ and P plus 1, which is

$$E[X] = d^{-1} + d^{-2} + \cdots + d^{-m+1} + d^{-m} + 1 = \frac{1-d^{-m}}{1-d^{-1}}$$

By the linearity of expectation, the total expected comparisons is the sum of all the values above over all substrings of length m of T, which is

$$(n-m+1)\frac{1-d^{-m}}{1-d^{-1}}$$

Since $d \geq 2$, we have

$$\frac{1-d^{-m}}{1-d^{-1}} \leq 2$$

so

$$(n-m+1)\frac{1-d^{-m}}{1-d^{-1}} \leq 2(n-m+1)$$

which shows that the algorithm is efficient on random strings.

3 [TC] Problem 32.1-4

STRING-WITH-GAP-MATCHER'(T,P)

```
1  let Q be the substrings of P split by ⋄
2  q = Q.size
3  t = 1
4  for i = 1 to q
5      if t + Q[i].length - 1 > T.length
6          return FALSE
7      while T[t..t+Q[i].length-1] ≠ Q[i]
8          t = t+1
9          if t + Q[i].length - 1 > T.length
10             return FALSE
11     t = t + Q[i].length
12 return TRUE
```

This algorithm runs in $O(mn)$ time, where n is the length of text, m is the length of pattern.

4 [TC] Problem 32.2-1

i	0	1	2	3	4	5	6	7	8
T		3	1	4	1	5	9	2	6
t	9	3	8	4	4	4	4	10	9
				$\times$	$\times$	$\times$	$\surd$		
i	9	10	11	12	13	14	15	16	
T	5	3	5	8	9	7	9	3	
t	2	3	1	9	2	5			

There are 3 spurious hits.

5 [TC] Problem 32.2-2

If all k patterns have the same length, we calculate the numbers they represent modulo q when preprocessing. When matching, if the value of current substring modulo q equals to any of the values of the patterns, then we verify whether current substring matches the pattern(s).

If the patterns have different lengths, we must calculate the t_s values of substrings of all possible lengths modulo q.

6 [TC] Problem 32.2-3

We calculate the values of the rows of the pattern modulo q, obtaining an $n \times 1$ array. Then, we treat this array as a string and calculate the p value according to the aforementioned method. Likewise we calculate the $t_{s,t}$ values for every shift in the $n \times n$ array of characters. We only have to verify the shifts whose $t_{s,t}$ values equal to the p value of the pattern.

7 [TC] Problem 32.2-4

If $A \neq B$, consider the difference of $A(x)$ and $B(x)$

$$\begin{aligned} C(x) &= A(x) - B(x) \\ &= \left(\sum_{i=0}^{n-1} (a_i - b_i) x^i \right) \bmod q \end{aligned}$$

By Exercise 31.4-4, there are at most n zero points out of all q possible values of x. Since x is randomly chosen from $\mathbb{Z}_q$ with $q > 1000n$, there is at most one chance in 1000 that $A(x) = B(x)$, whereas $A(x)$ is always the same as $B(x)$ if $A = B$.

8 [TC] Problem 32.3-2

Since there are too many states, it is inconvenient to draw the transition diagram of the automaton, we show the transition table below. State 0 is the start state and state 21 is the accepting state.

state	0	1	2	3	4	5	6	7
$\delta(a)$	1	1	3	1	3	6	1	3
$\delta(b)$	0	2	0	4	5	0	7	8
state	8	9	10	11	12	13	14	15
$\delta(a)$	9	1	11	1	3	14	1	16
$\delta(b)$	0	10	0	12	13	0	15	8
state	16	17	18	19	20	21		
$\delta(a)$	1	3	19	1	3	1		
$\delta(b)$	17	18	0	20	21	0		

9 [TC] Problem 32.3-3

Let n denote length of pattern P, then the string-matching automaton for such a nonoverlappable pattern consists of $n+1$ states, and we number them from 0 to n. State 0 is the start state and state n is the accepting state. When the automaton is in state i, if the input character equals to $P[i+1]$, then it transits to state $i+1$; otherwise it transits to state 0 if it does not equal to $P[1]$, or 1 if it does.

10 [TC] Problem 32.3-5

Firstly, we build finite automata for substrings of P split by gap characters. Then we "concatenate" the automata by replacing each automaton's (except the last one's) accepting state with the next automaton's starting state, namely, all transitions to its accepting state should be modified to the next automaton's starting state. The first automaton's starting state is the new automaton's staring state, and the last one's accepting state is the accepting state, respectively. The new automaton accepts all strings of pattern P, for it never transits to a state backward across a gap character whenever fails.