Exercises, chapter 15, solutions

---

1. **Algorithm:**

```
DISTINCT-CHARS-PATTERN-MATCHER(T, P)
 1  n = T.length
 2  m = P.length
 3  s = 0
 4  while s ≤ n − m
 5       i = 1
 6       while i ≤ m and P[i] = T[s + i]
 7            i = i + 1
 8       if i = m + 1
 9            print "Pattern occurs with shift" s
10       s = max(s + 1, s + i − 1)
```

Explanation:

The algorithm proceeds like the brute-force string matching algorithm, except that it may increase the shift position counter by more than one when a mismatch occurs. More precisely, when reaching line 10 in any iteration of the main loop, if a mismatch had occurred between $P[1]$ and $T[s+1]$ then it increases s by one as in the brute-force string matching algorithm. However, if the first $i-1$ characters of P matched $T[(s+1)..(s+i-1)]$ where $i \geq 2$, then the next possible occurrence of $P[1]$ in T is at $T[s+i]$ due to all characters of P being different, so the algorithm sets s to $s+i-1$. Line 10 combines the two cases by letting s be the maximum of $s+1$ and $s+i-1$.

Time complexity analysis:

First note that every symbol comparison done by the algorithm is between a character from P and a character from T and that every character of T participates in at most two symbol comparisons. The reason is that when the algorithm reaches line 10 in any iteration, s is increased so that either none or only the last one of the characters of T that was compared to P on line 6 will be compared to P again in the following iteration. Also, some characters at the end of T will participate in zero symbol comparisons if $T[(s+1)..n]$ is nonempty and shorter than P after the final iteration. Therefore, the total number of symbol comparisons is $O(n)$,

We now compute the time complexity. Lines 1–3 take $O(1)$ time. Then, each iteration of the main loop spends: (i) $O(1)$ time multiplied by the number of performed symbol comparisons [lines 6–7]; plus (ii) $O(1)$ time [lines 4–5 and 8–10]. By the above, the sum of all contributions of (i) to the overall time complexity is $O(1 \cdot n) = O(n)$, and since the main loop has $O(n)$ iterations, the sum of all contributions of (ii) is $O(1 \cdot n) = O(n)$. In total, the time complexity is $O(1+n+n) = O(n)$.

2. (a)

j	0	1	2	3	4	5	6	7
		A	B	A	B	A	C	A
b	0	0	0	1	2	3	0	1

(b)

	A	B	C	D
rt	2	3	1	7

3. See chapter 15.3 in the textbook or the slides from Lecture 12.

4. **Algorithm strategy:**
When $|R| = |S|$, construct a pattern p and a text t in such a way that R is a cyclic rotation of S if and only if p occurs as a substring somewhere in t, and then apply an algorithm for the Exact String Matching Problem to p and t. To do so, let p be equal to R and let t be two copies of S concatenated together. For example, if $R =$ `HEART` and $S =$ `EARTH` then we get $p =$ `HEART` and $t =$ `EARTHEARTH`; note that here, p is indeed a substring of t.

Pseudocode:

> 1. If $|R| \neq |S|$ then output "NO".
> 2. If $|R| = |S|$ then do the following:
> 3. Set $p = R$ and $t = SS$, and run the Knuth-Morris-Pratt algorithm on (p, t). If p has at least one occurrence in t then output "YES"; otherwise, output "NO".

Explanation:
If R is a cyclic rotation of S then $R = YX$ for some suffix Y of S and some prefix X of S. Since $|R| = |S|$, we have $S = XY$. Also, since $|R| = |S|$, the algorithm will execute line 3, where it will output "YES" because $p = R = YX$ is a substring of $t = SS = XYXY$.

Conversely, any length-$|S|$ substring of SS is a cyclic rotation of S because it has the form YX, where Y is a suffix of S and X is a prefix of S. Therefore, if R is not a cyclic rotation of S then either $|R| \neq |S|$, in which case the algorithm immediately outputs "NO" on line 1, or $|R| = |S|$ but R is not a substring of SS, in which case the algorithm outputs "NO" on line 3.

Time complexity analysis:
The time complexity of the Knuth-Morris-Pratt algorithm is $O(|p| + |t|)$. Here, $|p| = |R|$ and $|t| = 2|S|$, so the resulting time complexity is $O(|R| + |S|) + O(|R| + 2|S|) = O(|R| + |S|)$.