Assignment 5 – Selected Model Answers

Exercise 1. For the KMP algorithm design an algorithm to compute the sequence $NEXT$ in time in $O(m)$, where m is the length of the pattern sequence.

Solution. Let the pattern string be P with length m. Consider a copy of P, which we denote as P'. We must always have $NEXT(1) = 0$.

Then we scan P from position $j = 2$ to $j = m$. Each time for increasing values $i = 0, 1, \dots$ we compare $P(j+i)$ with $P'(i+1)$. As long as we encounter equality, we set $NEXT(j+i) = 1$ in case $P(j+i) \neq P'(1)$, and $NEXT(j+i) = 0$ in case $P(j+i) = P'(1)$. Once we get $P(j+i) \neq P'(i+1)$, we set $NEXT(j+i) = i+1$ and continue with position $j+i+1$ in P and reset i to 0.

For each position j in P we make at most 2 comparisons to determine $NEXT(j)$, so the algorithm works in linear time.

If P is used as pattern string in the KMP algorithm with a target string S and we encounter a mismatch at position j of P, say $S(i+j-1) \neq P(j)$, then we know that the substring $S(i) \dots S(i+j-2)$ coincides with $P(1) \dots P(j-1)$, so $P(1)$ has to be re-aligned with $S(i+k)$ such that $S(i+k) \dots S(i+j-2)$ coincides with $P(1) \dots P(j-1-k)$. The former of these sequences coincides with $P(k+1) \dots P(j-1)$. According to our algorithm we have $NEXT(j) = j-k$, hence $j - NEXT(j) = k$ is the correct number of positions, by which P has to be shifted in the KMP algorithm.

Exercise 2. Modify the KMP algorithm such that it will find all occurrences of a pattern sequence P in target sequence S in time in $O(n)$, where n is the length of S.

Solution. We extend the algorithm in Exercise 1 letting j run from 2 to $m+1$. In order to compute $NEXT(m+1)$ we let the result of comparing $P(m+1)$ (which is undefined) with $P'(i+1)$ for the current value of i be inequality.

Then we modify the KMP algorithm as follows. If KMP is successful, when $P(1)$ is aligned with $S(k)$, the position k is returned as output. As $P(m)$ is aligned with $S(k+m-1)$, the pattern string P is shifted by $m+1-NEXT(m+1)$ positions, and the search is continued with $S(k+m)$, which is aligned with $P(NEXT(m+1))$. In this way each element of S is compared at most once, so the complexity remains in $O(n)$.

As we assume success of the KMP algorithm, the substring $S(k) \dots S(k+m-1)$ is equal to P. By construction the last $NEXT(m+1)-1$ positions of P coincide with the substring $P(1) \dots P(NEXT(m+1)-1)$, hence the last $NEXT(m+1)-1$ positions of $S(k) \dots S(k+m-1)$ coincide with $P(1) \dots P(NEXT(m+1)-1)$. So there is no need to check again these positions, and the algorithm is correct.

Exercise 3.

(i) Represent a polynomial $P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_d x^d$ of degree d by an array P of length $d+1$ containing the coefficient $a_0, \dots, a_d$. If $a_d = 1$, such a polynomial is called *monic*.

Assume you are given already an algorithm to multiply a polynomial of degree i with a polynomial of degree 1 in time in $O(i)$. Assume further to be given another algorithm to multiply two polynomials of degree i in time in $O(i \log i)$.

Let $n_1, \dots, n_d$ be integers. Design an efficient divide-and-conquer algorithm to find the unique monic polynomial $P(x)$ of degree d such that $P(n_1) = P(n_2) = \cdots = P(n_d) = 0$ holds, and analyse the complexity of your algorithm.

(ii) Let $x_1, \dots, x_n$ be pairwise distinct values. Design an efficient algorithm to compute the coefficients of the unique monic polynomial $P(x)$ of degree n such that $P(x_1) = P(x_2) = \cdots = P(x_n) = 0$ holds. The algorithm is to require time in $O(n \log^2 n)$, provided that all necessary operations are elementary.

SOLUTION.

(i) For $d = 1$ the polynomial $x - n_i$ is the unique monic polynomial P_i of degree 1 with root n_i. It is represented by the pair $(-n_i, 1)$ and can thus be computed in constant time.

For arbitrary d we either have $d = 2\lfloor d/2 \rfloor + 1$ or $d = 2\lfloor d/2 \rfloor$. For $d' = \lfloor d/2 \rfloor$ we can recursively compute the polynomials

$$\prod_{i=1}^{d'} P_i \quad \text{and} \quad \prod_{i=d'+1}^{2d'} P_i$$

and multiply the result with the polynomial P_d, if d is odd.

Let $f(d)$ denote the complexity to compute the requested monic polynomial of degree d. Then we have

$$f(d) = 2 \cdot c \cdot f(\lfloor d/2 \rfloor) + c \cdot \lfloor d/2 \rfloor \log(\lfloor d/2 \rfloor) + c \cdot d \quad \text{or} \quad f(d) = 2 \cdot c \cdot f(\lfloor d/2 \rfloor) + c \cdot \lfloor d/2 \rfloor \log(\lfloor d/2 \rfloor) \ ,$$

depending on whether d is odd or even. Here $c > 0$ is some constant. Replacing d by 2^n we get a recurrence equation $a_n = 2a_{n-1} + 2^{n-1}(n-1)2^n$ for $a_n = f(2^n)$. The characteristic polynomial is $(x-2)^3$, which gives rise to $a_n = c_1 \cdot 2^n + c_2 \cdot n \cdot 2^n + c_3 \cdot n(n-1)2^n$ and further $f(d) = c_1 \cdot d + c_2 \cdot d \cdot \log_2 d + c_3 \cdot d \log_2^2 d + c_4$. Hence $f(d) \in O(d \log^2 d \mid \varphi)$ with the condition φ that d is a power of 2. The function f is almost everywhere monotone increasing and $d \log^2 d$ is smooth, which imply $f(d) \in O(d \log^2 d)$.

(ii) In order to multiply $P(x) = a_0 + a_1 x + \cdots + a_d x^d$ with $Q(x) = b_0 + b_1 x$, we only need to compute coefficients $c_0 = a_0 b_0$, $c_i = a_i b_0 + a_{i-1} b_1$ for $1 \le i \le d$ and $c_{d+1} = a_d b_1$, which requires $3d + 4$ elementary multiplications and additions. That is, to multiply a polynomial of degree d with a polynomial of degree 1 we need time in $O(d)$.

For the multiplication of two monic polynomials of the same degree d we can apply the FFT domain transformation, which requires time in $O(d \log d)$, then multiply the values and apply the inverse transform, which again requires time in $O(d \log d)$. In summary, we obtain an algorithm with tiome complexity in $O(d \log d)$ for the multiplication of two polynomials of degree d.

Finally, we can exploit the method in (i) to obtain an algorithm for the multiplication of all monomials $P_i(X) = x - x_i$ with time in $O(n \log n)$.

EXERCISE 4. Implement the KMP algorithm on arbitrary lists.

SOLUTION. See the C++ header and program files in `Ass5_Ex4solution.zip`.