Chapter 3

Algorithms on Polynomials

Excellent book references on this subject are [Knu1] and [GCL].

3.1 Basic Algorithms

3.1.1 Representation of Polynomials

Before studying algorithms on polynomials, we need to decide how they will be represented in an actual program. The straightforward way is to represent a polynomial

\[ P(X) = a_nX^n + a_{n-1}X^{n-1} + \cdots a_1X + a_0 \]

by an array \( a[0], a[1], \ldots a[n] \). The only difference between different implementations is that the array of coefficients can also be written in reverse order, with \( a[0] \) being the coefficient of \( X^n \). We will always use the first representation. Note that the leading coefficient \( a_n \) may be equal to 0, although usually this will not be the case.

The true degree of the polynomial \( P \) will be denoted by \( \deg(P) \), and the coefficient of \( X^{\deg(P)} \), called the leading coefficient of \( P \), will be denoted by \( \ell(P) \). In the example above, if, as is usually the case, \( a_n \neq 0 \), then \( \deg(P) = n \) and \( \ell(P) = a_n \).

The coefficients \( a_i \) may belong to any commutative ring with unit, but for many algorithms it will be necessary to specify the base ring. If this base ring is itself a ring of polynomials, we are then dealing with polynomials in several variables, and the representation given above (called the dense representation) is very inefficient, since multivariate polynomials usually have very few non-zero coefficients. In this situation, it is better to use the so-called sparse representation, where only the exponents and coefficients of the non-zero monomials are stored. The study of algorithms based on this kind of representation would however carry us too far afield, and will not be considered here. In any case, practically all the algorithms that we will need, use only polynomials in one variable.

The operations of addition, subtraction and multiplication by a scalar, i.e. the vector space operations, are completely straightforward and need not be discussed. On the other hand, it is necessary to be more specific concerning multiplication and division.
3.1.2 Multiplication of Polynomials

As far as multiplication is concerned, one can of course use the straightforward method based on the formula:

\[
(\sum_{i=0}^{m} a_i X^i)(\sum_{j=0}^{n} b_j X^j) = \sum_{k=0}^{n+m} c_k X^k ,
\]

where

\[
c_k = \sum_{i=0}^{k} a_i b_{k-i} ,
\]

where it is understood that \(a_i = 0\) if \(i > m\) and \(b_j = 0\) if \(j > n\). This method requires \((m + 1)(n + 1)\) multiplications and \(mn\) additions. Since in general multiplications are much slower than additions, especially if the coefficients are multi-precision numbers, it is reasonable to count only the multiplication time. If \(T(M)\) is the time for multiplication of elements in the base ring, the running time is thus \(O(mnT(M))\). It is possible to multiply polynomials faster than this, however. We will not study this in detail, but will give an example. Assume we want to multiply two polynomials of degree 1. The straightforward method above gives:

\[
(a_1 X + a_0)(b_1 X + b_0) = c_2 X^2 + c_1 X + c_0 ,
\]

with

\[
c_0 = a_0 b_0 , \quad c_1 = a_0 b_1 + a_1 b_0 , \quad c_2 = a_1 b_1 .
\]

As mentioned, this requires 4 multiplications and 1 addition. Consider instead the following alternate method for computing the \(c_k\):

\[
c_0 = a_0 b_0 , \quad c_2 = a_1 b_1 ,
\]

\[
d = (a_1 - a_0)(b_1 - b_0) , \quad c_1 = c_0 + (c_2 - d) .
\]

This requires only 3 multiplications, but 4 additions (subtraction and addition times are considered identical). Hence it is faster if one multiplication in the base ring is slower than 3 additions. This is almost always the case, especially if the base ring is not too simple or involves large integers. Furthermore, this method can be used for any degree, by recursively splitting the polynomials in two pieces of approximately equal degrees.

There is a generalization of the above method which is based on Lagrange’s interpolation formula. To compute \(A(X)B(X)\), which is a polynomial of degree \(m+n\), compute its value at \(m+n+1\) suitably chosen points. This involves only \(m+n+1\) multiplications. One can then recover the coefficients of \(A(X)B(X)\) (at least if the ring has characteristic zero) by using a suitable algorithmic form of Lagrange’s interpolation formula. The overhead which this implies is unfortunately quite large, and for practical implementations, the reader is advised either to stick to the straightforward method, or to use the recursive splitting procedure mentioned above.