Rensselaer Polytechnic Institute
Computer Science Department

Fall 2000

Computer Science 2300

Data Structures and Algorithms
Project 3

Handed Out: October 16, 2000 Due: November 9, 2000

The following project concerns efficient multiplication of polynomials. Suppose that we wish to mutiply the polynomials $A(x) = 5x^2 + 7x + 1$ and $B(x) = 2x + 3$; the answer is

$$C(x) = A(x)B(x) = 10x^3 + 29x^2 + 23x + 3$$

You can verify this result using the “algorithm” that you were given in high-school algebra for polynomial mutliplication. The goal of this project is to be able to obtain results like the above one in an efficient way. In order to do so, we will initially analyze the “high-school algorithm” for polynomial multiplication, and then slowly move towards the more efficient method. Along the way, many side issues will arise.

Before posing any problems, it is worth reviewing some notation.

Polynomial Definition

A **polynomial** in the variable x is a function $A(x)$ that can be represented as

$$A(x) = \sum_{j=0}^{n-1} a_j x^j$$

We call $n - 1$ the **degree** of the polynomial, and we call the values $a_0, \ldots, a_{n-1}$ the **coefficients** of the polynomial. It is assumed that the highest coefficient, a_{n-1}, is nonzero; otherwise, the polynomial’s degree would be lower. The coefficients are taken to be real numbers.

Coefficient Representation

There are two representations of polynomials which will be useful to us: the coefficient representation and the point-value representation. The **coefficient representation** of a polynomial $A(x) = \sum_{j=0}^{n-1} a_j x^j$ of degree $n-1$ is a vector of n coefficients

$$a = (a_0, a_1, \ldots, a_{n-1})$$

A polynomial has a unique coefficient representation.

As an example, take the polynomial $A(x) = 5x^2 + 7x + 1$; the coefficient representation of A is the vector $a = (1, 7, 5)$.

Point-Value Representation

The **point-value representation** of a polynomial $A(x)$ with degree $n-1$ is a set of n point-value pairs

$$\{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\}$$

such that all of the x_k are distinct and

$$y_k = A(x_k)$$

for $k = 0, 1, \ldots, n-1$. A polynomial has many different point-value representations, since any set of n distinct points $x_0, x_1, \ldots, x_{n-1}$ can be used as a basis for the representation.

As an example, once again take the polynomial $A(x) = 5x^2 + 7x + 1$. Then noting that $A(0) = 1$, $A(1) = 13$, and $A(2) = 35$, one possible point-value representation for $A(x)$ is

$$\{(0,1), (1,13), (2,35)\}$$

However, as was explained above, this choice is *not* unique. For another possibility, use the fact that $A(-1) = -1$, $A(\sqrt{2}) = 11 + 7\sqrt{2}$, and again that $A(0) = 1$. A second point-value representation is thus

$$\{(-1,-1), (\sqrt{2}, 11 + 7\sqrt{2}), (0,1)\}$$

Obviously, there are an infinite number of choices.

Roots of Unity

This material will not be used until problem 6. Suppose we wish to solve the following equation for q:

$$q^n = 1$$

given a particular value of n. Obviously, one solution is $q = 1$. If n is even, then another solution will be $q = -1$. In general, however, there are n distinct solutions $q_{n,0}, \ldots, q_{n,n-1}$, which may be written

$$q_{n,k} = e^{2\pi i k/n}, \quad k = 0, \ldots, n-1$$

where $i = \sqrt{-1}$. These are referred to as the $\mathbf{n^{th}}$ **roots of unity**.

Do not panic! You may be worried about such things as $i = \sqrt{-1}$; i is a complex number, and you may not be familiar with complex numbers. Even if you are familiar with complex numbers, you may not be familiar with the operation of exponentiating complex numbers, as is necessary to calculate $q_{n,k}$. **Don't worry. For this project, all you need to know is:**

1. Complex exponentiation has the following two properties, which are also valid for ordinary (real) exponentiation:

$$e^{ix}e^{iy} = e^{i(x+y)} \quad \text{and} \quad (e^{ix})^y = e^{ixy}$$

2. For any integer m (positive or negative),

$$e^{2\pi im} = 1$$

Using the second property, you can verify that the $q_{n,k}^n = 1$, that is, the $q_{n,k}$ are really the n^{th} roots of unity.

Outline of the Project

We are interested in finding algorithms for polynomial multiplication. In particular, given the coefficient representation of two polynomials, we would like the coefficient representation of their product. The overall idea is that efficient algorithms can be designed by dividing the multiplication process into three parts:

1. Convert the two polynomials from coefficient representation to point-value representation.

2. Multiply the point-value representations of the two polynomials. The result is the point-value representation of their product.

3. Convert back: take the point-value representation of the product, and find the coefficient representation of the product.

There are 7 problems. Problem 1 asks you to formalize the "high school algorithm" for multiplying the coefficient representation of polynomials, and to find its complexity. This is the naive approach to polynomial multiplication, and its complexity serves as a good benchmark by which we can judge the efficient algorithm. Problem 2 requires an analysis of multiplication in the point-value domain. In particular, how can the point-value representations of two polynomials be multiplied, and what is the complexity? Problem 3 examines a scheme for converting from coefficient representation to point-value representation. You are asked to calculate the complexity of the relevant algorithms. Problem 4 looks briefly at a scheme for converting back from point-value representation to coefficient representaion. Problem 5 takes a second stab at an algorithm for polynomial mutliplication. Problem 6 describes a more efficient algorithm for conversion from coefficient to point-value representation. Problem 7 takes a third stab at polynomial multiplication. The complexity here may be compared to that derived in problem 1.

Problem 1: Basic Multiplication

Given are three polynomials: $A(x)$, $B(x)$, and their product $C(x) = A(x)B(x)$. A and B are of degree $n-1$. Let the coefficient representations of A and B be $a = (a_0, \ldots, a_{n-1})$ and $b = (b_0, \ldots, b_{n-1})$ respectively. Let the coefficient representation of C be $c = (c_0, \ldots, c_{r-1})$.

1. What is $r-1$, i.e., what is the degree of C?
2. What is c in terms of a and b? That is, find the expression for each c_j, $j = 0, \ldots, r-1$, in terms of $a_0, \ldots, a_{n-1}$ and $b_0, \ldots, b_{n-1}$.
3. What is the complexity of the operation to find c, as a function of n?

Problem 2: Multiplication with Point-Value Representations

Given are three polynomials: $A(x)$, $B(x)$, and their product $C(x) = A(x)B(x)$. A and B are of degree $n-1$. Let the point-value representations of A, B be $\{(x_{a,0}, y_{a,0}), \ldots, (x_{a,r-1}, y_{a,r-1})\}$ and $\{(x_{b,0}, y_{b,0}), \ldots, (x_{b,r-1}, y_{b,r-1})\}$, respectively, where r is the value from problem 1.1. Note that while it is only necessary to specify n values, since A and B are degree $n-1$, it is necessary to specify r values for C, since it is of degree $r-1$. Furthermore, assume that $x_{a,j} = x_{b,j}$ for all $j = 0, \ldots, r-1$.

1. Express the point-value representation for C, $\{(x_{c,0}, y_{c,0}), \ldots, (x_{c,r-1}, y_{c,r-1})\}$, in terms of the point-value representations of a and b.
2. What is the complexity of this operation as a function of n?

Problem 3: Conversion from Coefficient Representation to Point-Value Representation

In order to convert a particular polynomial A from coefficient representation to point-value representation, we must calculate

$$y_k = \sum_{j=0}^{n-1} a_j x_k^j$$

for $k = 0, \ldots, n-1$.

1. The following algorithm is used:

```
CONVERT($\{a_j\}$, $\{x_k\}$)
for $k = 0$ to $n-1$
  $y_k = 0$
  for $j = 0$ to $n-1$
    $y_k = y_k + a_j$ * POWER($x_k$, $j$)
return $\{y_k\}$

POWER($x$, $n$)
$p = 1$
for $i$ = 1 to $n$
  $p$ = $p$ * $x$
return $p$
```

   Suppose that each addition and each multiplication are $O(1)$. What is the complexity of this conversion as a function of n? Explain.

2. You can lower the complexity of this algorithm simply by avoiding redundant multiplications. Write pseudo-code for an algorithm which implements this faster conversion, but only requires $O(1)$ more storage (that is, it will not need another n-element array).

3. What is the complexity of this algorithm?

Problem 4: Conversion from Point-Value Representation to Coefficient Representation

In order to convert from point-value representation to coefficient representation, *Lagrange's formula* may be used:

$$A(x) = \sum_{k=0}^{n-1} y_k \prod_{j \neq k} \left(\frac{x - x_j}{x_k - x_j} \right)$$

Show that Lagrange's formula satisfies $A(x_k) = y_k$ for all $k = 0, \ldots, n-1$. The complexity of finding the coefficient representation using Lagrange's formula is $O(n^2)$ (you do not need to show this).

Problem 5: Polynomial Multiplication, Part II

Suppose that you are given the following scheme for multiplying two polynomials in coefficient representation:

(a) Convert the two polynomials from coefficient representation to point-value representation.
(b) Multiply the point-value representations of the two polynomials. The result is the point-value representation of their product
(c) Convert back: take the point-value representation of the product, and find the coefficient representation of the product.

Using the methods outlined in problems 2, 3, and 4, what is best running time that can be achieved? How does this compare with the running time of problem 1?

Problem 6: Conversion from Coefficient Representation to Point-Value Representation, Part II

The goal is to design a fast divide-and-conquer method for converting from coefficient representation to point-value representation. First, assume that n is even and let

$$A^0(x) = a_0 + a_2 x + a_4 x^2 + \cdots + a_{n-2} x^{n/2-1}$$
$$A^1(x) = a_1 + a_3 x + a_5 x^2 + \cdots + a_{n-1} x^{n/2-1}$$

Note that A^0 contains all the even-index coefficients of A and A^1 contains all the odd-index coefficients. It follows that

$$A(x) = A^0(x^2) + xA^1(x^2) \tag{1}$$

So the problem of evaluating $A(x)$ at x_k for $k = 0, \ldots, n-1$ reduces to

(a) evaluating A^0 and A^1 at x_k^2 for $k = 0, \dots, n-1$
(b) combining the results according to equation (1).

Recall that there are many point-set representations for a given polynomial; any set of n distinct points $\{x_k\}$ may be used. The idea is to pick a clever choice of the $\{x_k\}$ in order to enable the divide and conquer approach to work. In particular, we will use the roots of unity: let $x_k = q_{n,k}$.

Througout this problem, you may assume that n is a power of 2, i.e. $n = 2^p$ for some p. This is useful for the divide and conquer algorithm, which will constantly be dividing into 2.

1. Using the properties of roots of unity given in the preamble, prove that $q_{n,k+n/2}^2 = q_{n,k}^2$.
2. Using the properties of roots of unity given in the preamble, prove that $q_{n,k}^2 = q_{n/2,k}$.
3. Based on your answer in (6.1) and (6.2), how many distinct numbers are there amongst the set $\{q_{n,0}^2, q_{n,1}^2, \dots, q_{n,n-1}^2\}$?
4. Using your results from (6.2) and (6.3), and the method for calculating $A(x)$ using $A^0(x)$ and $A^1(x)$ given above, write pseudo-code for an efficient divide and conquer algorithm to convert from coefficient representation to point-value representation.
5. Write a recursion for the running time of the algorithm, and solve it. What is the complexity?

Problem 7: Polynomial Multiplication, Part III

Suppose that we use the algorithm outlined in problem 5, but this time we use the method in problem 6 for converting from coefficient representation to point-value representation. Furthermore, suppose that there is an equally efficient method for converting in the other direction, from point-value representation to coefficient representation. What is the complexity of multiplying two degree $n-1$ polynomials with this method?