30 Polynomials and the FFT

30.1 Representing polynomials

30.1-1

> Multiply the polynomials $A(x) = 7x^3 - x^2 + x - 10$ and $B(x) = 8x^3 - 6x + 3$ using equations (30.1) and (30.2).

$$\begin{aligned} & 56x^6 - 8x^5 + (8-42)x^4 + (-80+6+21)x^3 + (-3-6)x^2 + (60+3)x - 30 \\ = \quad & 56x^6 - 8x^5 - 34x^4 - 53x^3 - 9x^2 + 63x - 30. \end{aligned}$$

30.1-2

> Another way to evaluate a polynomial $A(x)$ of degree-bound n at a given point x_0 is to divide $A(x)$ by the polynomial $(x - x_0)$, obtaining a quotient polynomial $q(x)$ of degree-bound $n - 1$ and a remainder r, such that
>
> $$A(x) = q(x)(x - x_0) + r.$$
>
> Clearly, $A(x_0) = r$. Show how to compute the remainder r and the coefficients of $q(x)$ in time $\Theta(n)$ from x_0 and the coefficients of A.

Let A be the matrix with 1's on the diagonal, $-x_0$'s on the super diagonal, and 0's everywhere else. Let q be the vector $(r, q_0, q_1, \ldots, x_{n-2})$. If $a = (a_0, a_1, \ldots, a_{n-1})$ then we need to solve the matrix equation $Aq = a$ to compute the remainder and coefficients. Since A is tridiagonal, Problem 28-1 (e) tells us how to solve this equation in linear time.

30.1-3

> Derive a point-value representation for $A^{rev}(x) = \sum_{j=0}^{n-1} a_{n-1-j} x^j$ from a point-value representation for $A(x) = \sum_{j=0}^{n-1} a_j x^j$, assuming that none of the points is 0.

For each pair of points, $(p, A(p))$, we can compute the pair $(\frac{1}{p}, A^{rev}(\frac{1}{p}))$. To do this, we note that

$$\begin{aligned} A^{rev}(\frac{1}{p}) &= \sum_{j=0}^{n-1} a_{n-1-j} (\frac{1}{p})^j \\ &= \sum_{j=0}^{n-1} a_j (\frac{1}{p})^{n-1-j} \\ &= p^{1-n} \sum_{j=0}^{n-1} a_j p^j \\ &= p^{1-n} A(p), \end{aligned}$$

since we know what $A(p)$ is, we can compute $A^{rev}(\frac{1}{p})$ of course, we are using the fact that $p \neq 0$ because we are dividing by it. Also, we know that each of these points are distinct, because $\frac{1}{p} = \frac{1}{p'}$ implies that $p = p'$ by cross multiplication. So, since all the x values were distinct in the point value representation of A, they will be distinct in this point value representation of A^{rev} that we have made.

30.1-4

> Prove that n distinct point-value pairs are necessary to uniquely specify a polynomial of degree-bound n, that is, if fewer than n distinct point-value pairs are given, they fail to specify a unique polynomial of degree-bound n. (Hint: Using Theorem 30.1, what can you say about a set of $n - 1$ point-value pairs to which you add one more arbitrarily chosen point-value pair?)

Suppose that just $n-1$ point-value pairs uniquely determine a polynomial P which satisfies them. Append the point value pair (x_{n-1}, y_{n-1}) to them, and let P' be the unique polynomial which agrees with the n pairs, given by Theorem 30.1. Now append instead (x_{n-1}, y'_{n-1}) where $y_{n-1} \neq y'_{n-1}$, and let P'' be the polynomial obtained from these points via Theorem 30.1. Since polynomials coming from n pairs are unique, $P' \neq P''$. However, P' and P'' agree on the original $n-1$ point-value pairs, contradicting the fact that P was determined uniquely.

30.1-5

> Show how to use equation (30.5) to interpolate in time $\Theta(n^2)$. (Hint: First compute the coefficient representation of the polynomial $\prod_j (x - x_j)$ and then divide by $(x - x_k)$ as necessary for the numerator of each term; see Exercise 30.1-2. You can compute each of the n denominators in time $O(n)$.)

First, we show that we can compute the coefficient representation of $\prod_j (x - x_j)$ in time $\Theta(n^2)$. We will do it by recursion, showing that multiplying $\prod_{j<k} (x - x_j)$ by $(x - x_k)$ only takes time $O(n)$, since this only needs to be done n times, this gets is total runtime of $O(n)$. Suppose that $\sum_{i=0}^{k-1} k_i x^i$ is a coefficient representation of $\prod_{j<k} (x - x_j)$. To multiply this by $(x - x_k)$, we just set $(k+1)_i = k_{i-1} - x_k k_i$ for $i = 1, \ldots, k$ and $(k+1)_0 = -x_k \cdot k_0$. Each of these coefficients can be computed in constant time, since there are only linearly many coefficients, then, the time to compute the next partial product is just $O(n)$.

Now that we have a coefficient representation of $\prod_j (x - x_j)$, we need to compute, for each k $\prod_{j-k} (x - x_j)$, each of which can be computed in time $\Theta(n)$ by problem 30.1-2. Since the polynomial is defined as a product of things containing the thing we are dividing by, we have that the remainder in each case is equal to 0. Lets call these polynomials f_k. Then, we need only compute the sum $\sum_k y_k \frac{f_k(x)}{f_k(x_k)}$. That is, we compute $f(x_k)$ each in time $\Theta(n)$, so all told, only $\Theta(n^2)$ time is spent computing all the $f(x_k)$ values. For each of the terms in the sum, dividing the polynomial $f_k(x)$ by the number $f_k(x_k)$ and multiplying by y_k only takes time $\Theta(n)$, so total it takes time $\Theta(n^2)$. Lastly, we are adding up n polynomials, each of degree bound $n-1$, so the total time taken there is $\Theta(n^2)$.

30.1-6

> Explain what is wrong with the "obvious" approach to polynomial division using a point-value representation, i.e., dividing the corresponding y values. Discuss separately the case in which the division comes out exactly and the case in which it doesn't.

If we wish to compute P/Q but Q takes on the value zero at some of these points, then we can't carry out the "obvious" method. However, as long as all point value pairs (x_i, y_i) we choose for Q are such that $y_i \neq 0$, then the approach comes out exactly as we would like.

30.1-7

> Consider two sets A and B, each having n integers in the range from 0 to $10n$. We wish to compute the ***Cartesian sum*** of A and B, defined by
>
> $$C = \{x + y : x \in A \text{ and } y \in B\}.$$
>
> Note that the integers in C are in the range from 0 to $20n$. We want to find the elements of C and the number of times each element of C is realized as a sum of elements in A and B. Show how to solve the problem in $O(n \lg n)$ time. (Hint: Represent A and B as polynomials of degree at most $10n$.)

For the set A, we define the polynomial f_A to have a coefficient representation that has a_i equal zero if $i \notin A$ and equal to 1 if $i \in A$. Similarly define f_B. Then, we claim that looking at $f_C := f_A \cdot f_B$ in coefficient form, we have that the ith coefficient, c_i is exactly equal to the number of times that i is realized as a sum of elements from A and B. Since we can perform the polynomial multiplication in time $O(n \lg n)$ by the methods of this chapter, we can get the final answer in time $O(n \lg n)$. To see that f_C has the nice property described, we'll look at the ways that we could end up having a term of x^i appear. Each contribution to that coefficient must come from there being some k so that $a_k \neq 0$ and $b_{i-k} \neq 0$, because the powers of x attached to each are additive when we multiply. Since each of these contributions is only ever 1, the final coefficient is counting the total number of such contributions, therefore counting the number of $k \in A$ such that $-k \in B$, which is exactly what we claimed f_C was counting.

30.2 The DFT and FFT

30.2-1

> Prove Corollary 30.4.

(Omit!)

30.2-2

> Compute the DFT of the vector $(0, 1, 2, 3)$.

(Omit!)

30.2-3

> Do Exercise 30.1-1 by using the $\Theta(n \lg n)$-time scheme.

(Omit!)

30.2-4

> Write pseudocode to compute DFT_n^{-1} in $\Theta(n \lg n)$ time.

(Omit!)

30.2-5

> Describe the generalization of the FFT procedure to the case in which n is a power of 3. Give a recurrence for the running time, and solve the recurrence.

(Omit!)

30.2-6 ⋆

> Suppose that instead of performing an n-element FFT over the field of complex numbers (where n is even), we use the ring $\mathbb{Z}_m$ of integers modulo m, where $m = 2^{tn/2} + 1$ and t is an arbitrary positive integer. Use $\omega = 2^t$ instead of ω_n as a principal nth root of unity, modulo m. Prove that the DFT and the inverse DFT are well defined in this system.

(Omit!)

30.2-7

> Given a list of values $z_0, z_1, \ldots, z_{n-1}$ (possibly with repetitions), show how to find the coefficients of a polynomial $P(x)$ of degree-bound $n + 1$ that has zeros only at $z_0, z_1, \ldots, z_{n-1}$ (possibly with repetitions). Your procedure should run in time $O(n \lg^2 n)$. (Hint: The polynomial $P(x)$ has a zero at z_j if and only if $P(x)$ is a multiple of $(x - z_j)$.)

(Omit!)

30.2-8 ⋆

> The ***chirp transform*** of a vector $a = (a_0, a_1, \ldots, a_{n-1})$ is the vector $y = (y_0, y_1, \ldots, y_{n-1})$, where $y_k = \sum_{j=0}^{n-1} a_j z^{kj}$ and z is any complex number. The DFT is therefore a special case of the chirp transform, obtained by taking $z = \omega_n$. Show how to evaluate the chirp transform in time $O(n \lg n)$ for any complex number z. (Hint: Use the equation

> $$y_k = z^{k^2/2} \sum_{j=0}^{n-1} \left(a_j z^{j^2/2}\right)\left(z^{-(k-j)^2/2}\right)$$
>
> to view the chirp transform as a convolution.)

(Omit!)

30.3 Efficient FFT implementations

30.3-1

> Show how ITERATIVE-FFT computes the DFT of the input vector $(0, 2, 3, -1, 4, 5, 7, 9)$.

(Omit!)

30.3-2

> Show how to implement an FFT algorithm with the bit-reversal permutation occurring at the end, rather than at the beginning, of the computation. (Hint: Consider the inverse DFT.)

(Omit!)

30.3-3

> How many times does ITERATIVE-FFT compute twiddle factors in each stage? Rewrite ITERATIVE-FFT to compute twiddle factors only 2^{s-1} times in stage s.

(Omit!)

30.3-4 ⋆

> Suppose that the adders within the butterfly operations of the FFT circuit sometimes fail in such a manner that they always produce a zero output, independent of their inputs. Suppose that exactly one adder has failed, but that you don't know which one. Describe how you can identify the failed adder by supplying inputs to the overall FFT circuit and observing the outputs. How efficient is your method?

(Omit!)

Problem 30-1 Divide-and-conquer multiplication

> **a.** Show how to multiply two linear polynomials $ax + b$ and $cx + d$ using only three multiplications. (Hint: One of the multiplications is $(a + b) \cdot (c + d)$.)
>
> **b.** Give two divide-and-conquer algorithms for multiplying two polynomials of degree-bound n in $\Theta(n^{\lg 3})$ time. The first algorithm should divide the input polynomial coefficients into a high half and a low half, and the second algorithm should divide them according to whether their index is odd or even.
>
> **c.** Show how to multiply two n-bit integers in $O(n^{\lg 3})$ steps, where each step operates on at most a constant number of 1-bit values.

(Omit!)

Problem 30-2 Toeplitz matrices

> A ***Toeplitz matrix*** is an $n \times n$ matrix $A = (a_{ij})$ such that $a_{ij} = a_{i-1,j-1}$ for $i = 2, 3, \ldots, n$ and $j = 2, 3, \ldots, n$.
>
> **a.** Is the sum of two Toeplitz matrices necessarily Toeplitz? What about the product?

b. Describe how to represent a Toeplitz matrix so that you can add two $n \times n$ Toeplitz matrices in $O(n)$ time.

c. Give an $O(n \lg n)$-time algorithm for multiplying an $n \times n$ Toeplitz matrix by a vector of length n. Use your representation from part (b).

d. Give an efficient algorithm for multiplying two $n \times n$ Toeplitz matrices. Analyze its running time.

(Omit!)

Problem 30-3 Multidimensional fast Fourier transform

We can generalize the 1-dimensional discrete Fourier transform defined by equation (30.8) to d dimensions. The input is a d-dimensional array $A = (a_{j_1, j_2, \ldots, j_d})$ whose dimensions are $n_1, n_2, \ldots, n_d$, where $n_1 n_2 \cdots n_d = n$. We define the d-dimensional discrete Fourier transform by the equation

$$y_{k_1, k_2, \ldots, k_d} = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \cdots \sum_{j_d=0}^{n_d-1} a_{j_1, j_2, \cdots, j_d} \omega_{n_1}^{j_1 k_1} \omega_{n_2}^{j_2 k_2} \cdots \omega_{n_d}^{j_d k_d}$$

for $0 \le k_1 < n_1, 0 \le k_2 < n_2, \ldots, 0 \le k_d < n_d$.

a. Show that we can compute a d-dimensional DFT by computing 1-dimensional DFTs on each dimension in turn. That is, we first compute n/n_1 separate 1-dimensional DFTs along dimension 1. Then, using the result of the DFTs along dimension 1 as the input, we compute n/n_2 separate 1-dimensional DFTs along dimension 2. Using this result as the input, we compute n/n_3 separate 1-dimensional DFTs along dimension 3, and so on, through dimension d.

b. Show that the ordering of dimensions does not matter, so that we can compute a d-dimensional DFT by computing the 1-dimensional DFTs in any order of the d dimensions.

c. Show that if we compute each 1-dimensional DFT by computing the fast Fourier transform, the total time to compute a d-dimensional DFT is $O(n \lg n)$, independent of d.

(Omit!)

Problem 30-4 Evaluating all derivatives of a polynomial at a point

Given a polynomial $A(x)$ of degree-bound n, we define its tth derivative by

$$A^{(t)}(x) = \begin{cases} A(x) & \text{if } t = 0, \\ \frac{d}{dx} A^{(t-1)}(x) & \text{if } 1 \le t \le n-1, \\ 0 & \text{if } t \ge n. \end{cases}$$

From the coefficient representation $(a_0, a_1, \ldots, a_{n-1})$ of $A(x)$ and a given point x_0, we wish to determine $A^{(t)}(x_0)$ for $t = 0, 1, \ldots, n-1$.

a. Given coefficients $b_0, b_1, \ldots, b_{n-1}$ such that

$$A(x) = \sum_{j=0}^{n-1} b_j (x - x_0)^j,$$

show how to compute $A^{(t)}(x_0)$ for $t = 0, 1, \ldots, n-1$, in $O(n)$ time.

b. Explain how to find $b_0, b_1, \ldots, b_{n-1}$ in $O(n \lg n)$ time, given $A(x_0 + \omega_n^k)$ for $k = 0, 1, \ldots, n-1$.

c. Prove that

$$A(x_0 + \omega_n^k) = \sum_{r=0}^{n-1} \left(\frac{\omega_n^{kr}}{r!} \sum_{j=0}^{n-1} f(j)g(r-j) \right),$$

where $f(j) = a_j \cdot j!$ and

$$g(l) = \begin{cases} x_0^{-l}/(-l)! & \text{if } -(n-1) \le l \le 0, \\ 0 & \text{if } 1 \le l \le n-1. \end{cases}$$

d. Explain how to evaluate $A(x_0 + \omega_n^k)$ for $k = 0, 1, \ldots, n-1$ in $O(n \lg n)$ time. Conclude that we can evaluate all nontrivial derivatives of $A(x)$ at x_0 in $O(n \lg n)$ time.

(Omit!)

Problem 30-5 Polynomial evaluation at multiple points

We have seen how to evaluate a polynomial of degree-bound n at a single point in $O(n)$ time using Horner's rule. We have also discovered how to evaluate such a polynomial at all n complex roots of unity in $O(n \lg n)$ time using the FFT. We shall now show how to evaluate a polynomial of degree-bound n at n arbitrary points in $O(n \lg^2 n)$ time.

To do so, we shall assume that we can compute the polynomial remainder when one such polynomial is divided by another in $O(n \lg n)$ time, a result that we state without proof. For example, the remainder of $3x^3 + x^2 - 3x + 1$ when divided by $x^2 + x + 2$ is

$$(3x^3 + x^2 - 3x + 1) \mod (x^2 + x + 2) = -7x + 5.$$

Given the coefficient representation of a polynomial $A(x) = \sum_{k=0}^{n-1} a_k x^k$ and n points $x_0, x_1, \ldots, x_{n-1}$, we wish to compute the n values $A(x_0), A(x_1), \ldots, A(x_{n-1})$. For $0 \le i \le j \le n-1$, define the polynomials $P_{ij}(x) = \prod_{k=i}^{j} (x - x_k)$ and $Q_{ij}(x) = A(x) \mod P_{ij}(x)$. Note that $Q_{ij}(x)$ has degree at most $j - i$.

a. Prove that $A(x) \mod (x - z) = A(z)$ for any point z.

b. Prove that $Q_{kk}(x) = A(x_k)$ and that $Q_{0,n-1}(x) = A(x)$.

c. Prove that for $i \le k \le j$, we have $Q_{ik}(x) = Q_{ij}(x) \mod P_{ik}(x)$ and $Q_{kj}(x) = Q_{ij}(x) \mod P_{kj}(x)$.

d. Give an $O(n \lg^2 n)$-time algorithm to evaluate $A(x_0), A(x_1), \ldots, A(x_{n-1})$.

(Omit!)

Problem 30-6 FFT using modular arithmetic

As defined, the discrete Fourier transform requires us to compute with complex numbers, which can result in a loss of precision due to round-off errors. For some problems, the answer is known to contain only integers, and by using a variant of the FFT based on modular arithmetic, we can guarantee that the answer is calculated exactly. An example of such a problem is that of multiplying two polynomials with integer coefficients. Exercise 30.2-6 gives one approach, using a modulus of length $\Omega(n)$ bits to handle a DFT on n points. This problem gives another approach, which uses a modulus of the more reasonable length $O(\lg n)$; it requires that you understand the material of Chapter 31. Let n be a power of 2.

a. Suppose that we search for the smallest k such that $p = kn + 1$ is prime. Give a simple heuristic argument why we might expect k to be approximately $\ln n$. (The value of k might be much larger or smaller, but we can reasonably expect to examine $O(\lg n)$ candidate values of k on average.) How does the expected length of p compare to the length of n?

Let g be a generator of $\mathbb{Z}_p^*$, and let $w = g^k \mod p$.

> **b.** Argue that the DFT and the inverse DFT are well-defined inverse operations modulo p, where w is used as a principal nth root of unity.
>
> **c.** Show how to make the FFT and its inverse work modulo p in time $O(n \lg n)$, where operations on words of $O(\lg n)$ bits take unit time. Assume that the algorithm is given p and w.
>
> **d.** Compute the DFT modulo $p = 17$ of the vector $(0, 5, 3, 7, 7, 2, 1, 6)$. Note that $g = 3$ is a generator of $\mathbb{Z}_{17}^*$.

(Omit!)