15-451/651 Algorithm Design & Analysis

Fall 2022, Recitation #12

Objectives

- Provide practice reducing problems to polynomial multiplication.

Recitation Problems

1. **(Random Relationship)**

   Let X and Y be discrete random variables with a natural ($\mathbb{N}$), finite ranges. X and Y are also independent. Recall that X and Y are independent random variables if and only if...

   $$\mathbb{P}(X = x \cap Y = y) = \mathbb{P}(X = x) \cdot \mathbb{P}(Y = y)$$

   Let $Z = X + Y$. Derive a expression for $\mathbb{P}(Z = z)$ for an arbitrary value z.

   (*Hint:* express a random variable as a polynomial)

2. **(Evenly Spaced Ones)**

   Given a binary string S of length n. We wish to determine whether there exists three evenly spaced ones within S. For example, 11100000, 110110010 both have three evenly spaced 1s, while 1011 does not.

   (a) Derive a brute-force algorithm solving this problem with $O(n^2)$ complexity.

   (b) Derive an algorithm with $O(n \log n)$ complexity that uses polynomial multiplication and convolutions.

3. **(Message Validation with Binary Polynomial Division)**

In lecture, you saw a method for error correction that encodes $d+1$ numbers you wish to send into a polynomial $P(x)$ and sending $d+k+1$ outputs from $P(x)$. Here, we present another error correction algorithm that is actually quite common in the real world.

Consider the problem of sending n bits $b_{n-1}, b_{n-2}, \ldots, b_0$ from a sender to a receiver, where each bit has some nonzero chance of being corrupted and flipped. To do this, both the sender and the receiver agree on some "divisor polynomial" P of degree d. The polynomial P itself is a binary polynomial, which means that each coefficient is either 0 or 1.

The sender follows this protocol to generate a message:

- Interpret the n bit message as its own polynomial Q of degree $n-1$, such that

$$Q(x) = b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + \cdots + b_1 x + b_0$$

- Compute $x^d Q(x)/P(x)$, using binary mathematics (mod 2). This generates some quotient polynomial and, more importantly, a binary remainder polynomial $R(x)$ of degree $d-1$. $R(x)$ has d coefficients and is of the form

$$R(x) = r_{d-1}x^{d-1} + r_{d-2}x^{d-2} + \cdots + r_1 x + r_0$$

- Send the $n+d$ bit-message $b_{n-1}b_{n-2}\ldots b_1 b_0 r_{d-1} r_{d-2} \ldots r_1 r_0$.

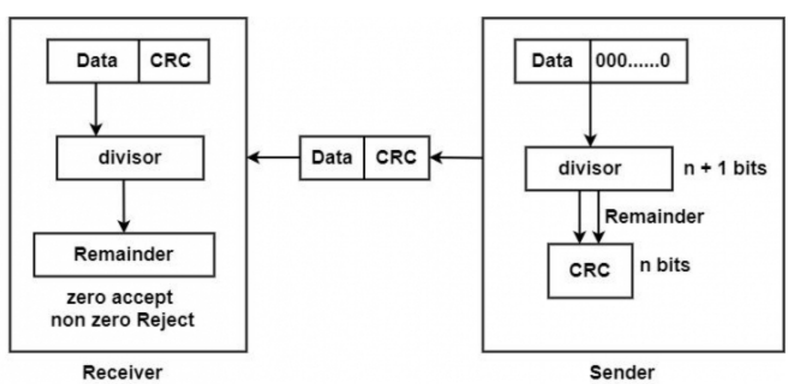
The receiver, knowing the divisor polynomial P, interprets the $n+d$-bit message as $c_{n+d-1}c_{n+d-1}\ldots c_0$ and does the following check:

- Interpret the bit message as a polynomial S of degree $n+d-1$:

$$S(x) = c_{n+d-1}x^{n+d-1} + c_{n+d-2}x^{n+d-2} + \cdots + c_1 x + c_0$$

- Compute $S(x)/P(x)$, again in modulo 2. If the remainder is zero, ACCEPT the message. Otherwise, REJECT the message.

This diagram outlines this message protocol's infrastructure. Here, CRC is that remainder polynomial $R(x)$.



(a) Warm up your polynomial division by computing the quotient and remainder of

$$\frac{x^5 + x^4 + x^3 + x + 1}{x^3 + x^2 + 1}$$

in modulo 2. If you were to interpret every step of your long division in terms of bit-logic, what binary operation does it seem similar to?

(b) Confirm that if there are no bit corruptions during sending, the receiver will always ACCEPT the message.

(c) Show that there is a possibility for false positives: that is there is a non-zero number of corruptions, but the recipient still accepts. How few corruptions are needed for general n, d, and choices of divisor polynomial?

(d) Suppose a horrible bug meant instead of sending the $n + d$ bit message for this protocol, you instead sent $n + d$ bits from elsewhere on the machine, which are essentially random! What's the probability, in terms of n and d, that the recipient will REJECT this message?

Further Review

1. **(Short answer / multiple choice)**

   (a) Given some set of d unique numbers $r_1, r_2, ..., r_d$, what is a d-degree polynomial $P(x)$ with these as roots?

   (b) What is the expression for the i^{th} root of unity of degree n?

   (c) Perform the convolution of the two vectors $\langle 3, 4, -1 \rangle$ and $\langle 2, 5, 0, 1 \rangle$.

2. **(5-SUM via Polynomial Multiplication)**

   In lecture, you saw an $O(n \log n)$ time algorithm for 2-SUM, using polynomial multiplication and FFT. Extend this concept to come up with an $O(n \log n)$ time algorithm for 5-SUM, which as its name suggests, asks whether there exists 5 elements of an array that sum to some target t, where repetitions are allowed.