CS 161: Design and Analysis of Algorithms

Divide & Conquer IV: FFT

- Recap
- Choosing a good set of points
- The FFT
- The DFT

Multiplying Polynomials

$$P(z) = \sum_{i=0}^{d} a_i z^i$$

$$Q(z) = \sum_{i=0}^{d} b_i z^i$$

$$a_i = b_i = 0 \forall i > d$$

$$R(z) = P(z)Q(z) = \sum_{i=0}^{2d} \left(\sum_{j=0}^{i} a_j b_{i-j} \right) z^i$$

Representing Polynomials

- Generally, polynomials represented by coefficients a_i
- Theorem: Let Z be a set of n > d inputs, and let P(z) be a polynomial of degree d. Then P(z) is completely determined by the values $P(z_0)$, $P(z_1)$, …, $P(z_d)$

Multiplying Polynomials

- To multiply polynomials P and Q:
  - Pick a set Z of at least 2d+1 inputs
  - Compute $P(z_i)$, $Q(z_i)$
  - Compute $R(z_i) = P(z_i)Q(z_i)$
  - Compute coefficients of R(z)

Changing Representation

- Say d = 2k+1

$$P(z) = a_{2k+1}z^{2k+1} + \ldots + a_0$$

$$= (a_{2k}z^{2k} + a_{2k-2}z^{2k-2} + \ldots + a_0) + (a_{2k+1}z^{2k+1} + a_{2k-1}z^{2k-1} + \ldots + a_1 z)$$

$$= P_{even}(z^2) + zP_{odd}(z^2)$$

$$P_{even}(z) = a_{2k}z^{k} + a_{2k-2}z^{k-1} + \ldots + a_0$$

$$P_{odd}(z) = a_{2k+1}z^{k} + a_{2k-1}z^{k-1} + \ldots + a_1 z$$

Divide and Conquer

- Let Z = $\{z_0, -z_0, z_1, -z_1, \dots, -z_k, z_k\}$
- Let Z' = $\{z_0^2, z_1^2, \dots, z_k^2\}$
- To evaluate P on all the points in Z:
  - Evaluate P_{even} and P_{odd} on all the points in Z'

$$P(z) = P_{even}(z^2) + zP_{odd}(z^2)$$

$$P(\pm z_i) = P_{even}(z_i^2) \pm z_i P_{odd}(z_i^2)$$

Complex Numbers

- Imaginary number i: $i^2 = 1$
- Complex numbers have the form: $a + b\, i$
- $(a + b\, i) + (c + d\, i) = (a + c)+(b + d)\, i$
- $(a + b\, i)(c + d\, i) = ac + bc\, i + ad\, i + bd\, i^2$

  $= (ac-bd) + (bc+ad)\, i$

Complex Numbers

- Fact: $e^{i\theta} = \cos(\theta) + i \sin(\theta)$
- $e^{i2\pi} = 1$
- Alternative representation of complex numbers:
  - $Re^{i\theta}$ where R and θ are real numbers
  - Same representation if we use $\theta+2\pi k$ for any integer k
  - $(Re^{i\theta})\ (Se^{i\varphi})=(RS)e^{i(\theta+\varphi)}$

Complex Numbers

- Roots of unity:
  - $e^{i(2\pi/n)k}$ = k (2π/n) for some integer k
  - i.e., $z = e^{i(2\pi/n)}$

Complex Numbers

- Primitive nth root of unity:
  - $z^n = 1$
  - $z^k \neq 1$ for $0 \leq k < n$
  - Example: $e^{i2\pi/n}$
  - Fact: Let ω be a primitive nth root of unity. Then $\{1, \omega, \omega^2, \ldots, \omega^{n-1}\}$ all nth roots of unity, and are all distinct

Choosing a Good Set of Points

- Want a set $\{z_0, \ldots, z_{n-1}\}$, $n > d$, such that:
  - All z_i are distinct
  - The z_i can be grouped off into pairs $\pm z$
    - $\{z_0, z_1, \ldots, z_{n/2-1}, -z_0, \ldots, -z_{n/2-1}\}$
  - The set of squares of these numbers has size n/2 and satisfies the same properties
  - Impossible over real numbers. Need complex numbers

Choosing a Good Set of Points

- Let n be the lowest power of two such that $n \geq d+1$
- Consider the nth roots of unity:
  - There are n of them: $e^{i2\pi k/n}$ for k in [0,n-1]
  - If ω is a nth root of 1, so is $-\omega$:
    - $(-\omega)^n = (-1)^n\omega^n = (-1)^n = 1$
  - Set of squares: $(\omega^2)^{n/2} = \omega^n = 1$
    - (n/2)-roots of unity
    - In fact, ω^2 is primitive (n/2)-root

Choosing a Good Set of Points

- Good set: $\{1, \omega^1, \omega^2, \ldots, \omega^{n-1}\}$ for some primitive nth root of unity
- Convention: $\omega = e^{-i2\pi/n}$

Divide and Conquer Algorithm: FFT

- Let $P(z) = a_0 + a_1 z + \dots + a_d z^d$
- To compute $\{P(1), P(\omega), P(\omega^2), \dots, P(\omega^{n-1})\}$:
  - Let $P_{even}(z) = a_0 + a_2 z + \dots + a_{d-1} z^{(d-1)/2}$
  - Let $P_{odd}(z) = a_1 + a_3 z + \dots + a_d z^{(d-1)/2}$
  - Let $\lambda = \omega^2$, and recursively compute
    - $\{P_{even}(1), P_{even}(\lambda), \dots, P_{even}(\lambda^{n/2-1})\}$
    - $\{P_{odd}(1), P_{odd}(\lambda), \dots, P_{odd}(\lambda^{(n/2-1)})\}$
  - Compute $P(\omega^k) = P_{even}(\omega^{2k}) + \omega^k P_{odd}(\omega^{2k})$

Running Time

- T(n):
  - 2 recursive calls of size n/2: 2 T(n/2)
  - O(n) work to get P_{even} and P_{odd}
  - O(1) work per computation of $P(\omega^k) = P_{even}(\omega^{2k}) + \omega^k P_{odd}(\omega^{2k})$
  - Total: 2 T(n/2) + O(n)
- Solved by T(n) = O(n log n)

The Fast Fourier Transform (FFT)

- What are we computing?
  - Given $P(z) = a_0 + \ldots + a_d z^d$
  - Compute $P(\omega^k)$ for $k = 0,\ldots, n-1$

$$P(\omega^k) = \sum_{t=0}^{n-1} a_t \omega^{kt} = \sum_{t=0}^{n-1} a_t e^{-i\frac{2\pi}{n}kt} = A_k$$

The Discrete Fourier Transform (DFT)

- Input: $(a_0, a_1, \ldots, a_{n-1})$
- Output: $(A_0, A_1, \ldots, A_{n-1})$
- Where:

$$A_k = \sum_{t=0}^{n-1} a_t e^{-i\frac{2\pi}{n}kt}$$

- FFT: O(n log n) algorithm for computing the DFT

The FFT

- FFT$[(a_0, a_1, \ldots, a_{n-1}), \omega]$
  - Recursively perform 2 FFTs:
    - $(A_0^{even}, A_1^{even}, \ldots, A_{n/2-1}^{even}) =$ FFT$[(a_0, a_2, \ldots, a_{n-2}), \omega^2]$
    - $(A_0^{odd}, A_1^{odd}, \ldots, A_{n/2-1}^{odd}) =$ FFT$[(a_1, a_3, \ldots, a_{n-1}), \omega^2]$
  - Output the sequence $(A_0, A_1, \ldots, A_{n-1})$ where

$$A_k = A_{k \bmod (n/2)}^{even} + \omega^k A_{k \bmod (n/2)}^{odd}$$

Correctness

- Base case: The DFT of (a_0) is just (a_0)
- Claim: For an sequence $(b_0, \ldots, b_{n'-1})$, we can extend the DFT to all integers k, with the property that $B_{k+n'} = B_k$ for all k.

$$B_{k+n'} = \sum_{t=0}^{n'-1} b_t \omega^{(k+n')t} = \sum_{t=0}^{n'-1} b_t \omega^{kt} \left(\omega^{n'}\right)^t$$

$$= \sum_{t=0}^{n'-1} b_t \omega^{kt} = B_k$$

Correctness

- Base case: The DFT of (a_0) is just (a_0)
- Otherwise,

$$A_k = \sum_{t=0}^{n-1} a_t \omega^{kt} = \sum_{s=0}^{n/2-1} a_{2s} \omega^{k(2s)} + \sum_{s=0}^{n/2-1} a_{2s+1} \omega^{k(2s+1)}$$

$$= \sum_{s=0}^{n/2-1} a_{2s} \left(\omega^2\right)^{ks} + \omega^k \sum_{s=0}^{n/2-1} a_{2s+1} \left(\omega^2\right)^{ks}$$

$$= A_{k \bmod n/2}^{even} + \omega^k A_{k \bmod n/2}^{odd}$$

How do we undo the DFT?

$$A_k = \sum_{t=0}^{n-1} a_t e^{-i\frac{2\pi}{n}kt}$$

$$a_t = \frac{1}{n} \sum_{k=0}^{n-1} A_k e^{i\frac{2\pi}{n}kt}$$

Proof

$$a_t \stackrel{?}{=} \frac{1}{n}\sum_{k=0}^{n-1}\left(\sum_{s=0}^{n-1} a_s e^{-i\frac{2\pi}{n}ks}\right)e^{i\frac{2\pi}{n}kt}$$

Proof

$$\frac{1}{n}\sum_{k=0}^{n-1}\left(\sum_{s=0}^{n-1}a_s e^{-i\frac{2\pi}{n}ks}\right)e^{i\frac{2\pi}{n}kt} = \frac{1}{n}\sum_{k=0}^{n-1}\sum_{s=0}^{n-1}a_s e^{-i\frac{2\pi}{n}k(s-t)}$$

$$= \frac{1}{n}\sum_{s=0}^{n-1}a_s\left(\sum_{k=0}^{n-1}e^{-i\frac{2\pi}{n}(s-t)k}\right)$$

Proof

$$\sum_{k=0}^{n-1} e^{-i\frac{2\pi}{n}(s-t)k} = \sum_{k=0}^{n-1} \left(e^{-i\frac{2\pi}{n}(s-t)} \right)^k = \sum_{k=0}^{n-1} \alpha_{s,t}^k$$

$$= \begin{cases} n & \text{if } \alpha_{s,t} = 1 \\ \dfrac{1-\alpha_{s,t}^n}{1-\alpha_{s,t}} & \text{otherwise} \end{cases}$$

Proof

$$\alpha_{s,t} = e^{-i\frac{2\pi}{n}(s-t)}$$

$$\alpha_{s,t}^{n} = 1$$

$$\alpha_{t,t} = 1$$

Proof

$$\sum_{k=0}^{n-1} e^{-i\frac{2\pi}{n}(s-t)k} = \begin{cases} n & \text{if } s = t \\ 0 & \text{otherwise} \end{cases}$$

Proof

$$\frac{1}{n}\sum_{s=0}^{n-1} a_s \left(\sum_{k=0}^{n-1} e^{-i\frac{2\pi}{n}(s-t)k} \right) = \frac{1}{n}(na_t) = a_t$$

Computing the Inverse DFT

$$A_k = \sum_{t=0}^{n-1} a_t e^{-i\frac{2\pi}{n}kt}$$

$$a_t = \frac{1}{n}\sum_{k=0}^{n-1} A_k e^{i\frac{2\pi}{n}kt}$$

- Compute inverse with FFT($(A_0, \ldots, A_{n-1})$, $e^{i2\pi/n}$)/n
- $e^{i2\pi/n} = (e^{-i2\pi/n})^{-1} = \omega^{-1}$

The DFT

- Useful in signal processing
  - If a_t represents the values of some signal (say sound), then A_k represent the amount of each frequency in the signal

The DFT

- Linear Time-Invariant Systems:
  - Transform discrete function to another discrete function
  - If we add two input functions together, outputs are added
  - If we multiply an input function by a constant, output multiplied by same constant
  - If we shift an input function by a constant amount, we shift the output by the same amount

The DFT

- Linear Time-Invariant Systems:
  - Impulse response: output of system on input (1,0,0,…,0)
  - Turns out that impulse response completely determines LTI systems.
  - If we pass a signal $(a_0,\ldots,a_{n-1})$ system corresponds to multiplying $(A_0, \ldots, A_{n-1})$ by the DFT of the impulse response

The FFT

- The DFT is a transformation of sequences of length n to sequences of length n
- Naïve implementation requires $O(n^2)$ arithmetic operations
- FFT: Algorithm for computing DFT using $O(n \log n)$ arithmetic operations

The FFT

- Important for efficient signal processing
- Also important in quantum computing:
  - Quantum version called QFT
  - Allows quantum computers to solve some difficult problems, including factoring integers

How to multiply Polynomials

- To multiply $P(z) = a_d\, z^d + \ldots + a_0$ with
  $Q(z) = b_{d'}\, z^{d'} + \ldots + b_0$:
  - Choose n a power of 2 such that $n \geq d+d'+1$.
  - Write P and Q as sequences of n values:
    - $P = (a_0, a_1, \ldots, a_d, 0, 0, \ldots, 0)$
    - $Q = (b_0, b_1, \ldots, b_{d'}, 0, 0, \ldots, 0)$
  - DFT the sequences
  - Pointwise multiply the sequences
  - DFT back, obtaining $(c_0, c_1, \ldots, c_{n-1})$
  - $P(z)Q(z) = c_{n-1}z^{n-1} + \ldots + c_0$

Polynomial Multiplication Example

- Let $P(z) = x + 2$, $Q(z) = 2x – 3$
  - $d = d' = 1$, so $n = 4$ will do
  - $P = (2,1,0,0)$, $Q = (-3,2,0,0)$
  - $\omega = e^{-i2\pi/4} = -i$

Polynomial Multiplication Example

- DFT of P?
  - P(1) = 3
  - $P(\omega) = P(-i) = 2 - i$
  - $P(\omega^2) = P(-1) = 1$
  - $P(\omega^3) = P(i) = 2 + i$
  - DFT of P = (3,2-i,1,2+i)

Polynomial Multiplication Example

- DFT of Q?
  - $Q(1) = -1$
  - $Q(\omega) = Q(-i) = -3 - 2i$
  - $Q(\omega^2) = Q(-1) = -5$
  - $Q(\omega^3) = Q(i) = -3 + 2i$
  - DFT of Q = $(-1, -3-2i, -5, -3+2i)$

Polynomial Multiplication Example

- Pointwise multiply:
  - DFT of P = (3, 2-i, 1, 2+i)
  - DFT of Q = (-1, -3-2i, -5, -3+2i)
  - DFT of PQ = (-3, -8-i, -5, -8+i)

Polynomial Multiplication Example

- Inverse DFT
  - $R_r = (-3 + (-8-i)(i)^r + (-5)(-1)^r + (-8+i)(-i)^r)/4$
  - (-6,1,2,0)
  - Therefore, $P(z)Q(z) = 2x^2 + x - 6$

FFT to Multiply Integers

- We reduced multiplying integers to multiplying polynomials
- We reduced multiplying polynomials to computing DFTs
- Can compute DFTs using FFT in O(n log n) time
- So can we multiply n-bit integers in O(n log n time)?

FFT to Multiply Integers

- Problem:
  - $\omega = e^{i2\pi/n} = \cos(2\pi/n) + i \sin(2\pi/n)$
  - Real irrational numbers
  - To represent accurately, many bits required
  - Adding/multiplying not O(1)
  - Using clever tricks, can get $O(n \log n \log \log n)$
  - Even better: $O(n \log n\ c^{\log^*(n)})$