CS 473: Algorithms

Ruta Mehta

University of Illinois, Urbana-Champaign

Spring 2018

Polynomials, Convolutions and FFT

Lecture 2

Jan 16/18, 2018

Most slides are courtesy Prof. Chekuri

Outline

Discrete Fourier Transfor (DFT) and Fast Fourier Transform (FFT) have many applications and are connected to important mathematics.

"One of top 10 Algorithms of 20th Century" according to IEEE.
Gilbert Strang: "The most important numerical algorithm of our lifetime".

Our goal:

- Multiplication of two degree n polynomials in $O(n \log n)$ time. Surprising and non-obvious.
- Algorithmic ideas
  - change in representation
  - mathematical properties of polynomials
  - divide and conquer

Part I

Polynomials, Convolutions and FFT

Polynomials

Definition

A polynomial is a function of one variable built from additions, subtractions and multiplications (but no divisions).

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

The numbers $a_0, a_1, \ldots, a_n$ are the coefficients of the polynomial. The degree is the highest power of x with a non-zero coefficient.

Example

$$p(x) = 3 - 4x + 5x^3$$

$a_0 = 3, a_1 = -4, a_2 = 0, a_3 = 5$ and $deg(p) = 3$

Polynomials

Definition

A polynomial is a function of one variable built from additions, subtractions and multiplications (but no divisions).

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

The numbers $a_0, a_1, \ldots, a_n$ are the coefficients of the polynomial. The degree is the highest power of x with a non-zero coefficient.

Coefficient Representation

Polynomials represented by vector $a = (a_0, a_1, \ldots a_{n-1})$ of coefficients.

Operations on Polynomials

Evaluate Given a polynomial p and a value α, compute $p(\alpha)$

Add Given (representations of) polynomials p, q, compute (reprsentation of) polynomial $p + q$

Multiply Given (representation of) polynomials p, q, compute (representation of) polynomial $p \cdot q$.

Roots Given p find all *roots* of p.

Evaluation

Compute value of polynomial $a = (a_0, a_1, \ldots a_{n-1})$ at α

```
power = 1
value = 0
for j = 0 to n − 1
    // invariant:  power = α^j
    value = value + a_j · power
    power = power · α
end for
return value
```

How many additions?

Evaluation

Compute value of polynomial $a = (a_0, a_1, \ldots a_{n-1})$ at α

```
power = 1
value = 0
for j = 0 to n − 1
    // invariant:  power = α^j
    value = value + a_j · power
    power = power · α
end for
return value
```

How many additions? n

Evaluation

Compute value of polynomial $a = (a_0, a_1, \ldots a_{n-1})$ at α

```
power = 1
value = 0
for j = 0 to n − 1
    // invariant:  power = α^j
    value = value + a_j · power
    power = power · α
end for
return value
```

How many additions? n

How many multiplications?

Evaluation

Compute value of polynomial $a = (a_0, a_1, \ldots a_{n-1})$ at α

```
power = 1
value = 0
for j = 0 to n − 1
    // invariant:  power = α^j
    value = value + a_j · power
    power = power · α
end for
return value
```

How many additions? n

How many multiplications? $2n$

Evaluation

Compute value of polynomial $a = (a_0, a_1, \ldots a_{n-1})$ at α

```
power = 1
value = 0
for j = 0 to n − 1
    // invariant:  power = α^j
    value = value + a_j · power
    power = power · α
end for
return value
```

How many additions? n

How many multiplications? $2n$

Horner's rule can be used to cut the multiplications in half

$$a(x) = a_0 + x(a_1 + x(a_2 + x(\cdots + xa_{n-1})\cdots))$$

Evaluation: Numerical Issues

Question

How long does evaluation really take? $O(n)$ time?

Bits to represent α^n is $n \log \alpha$ while bits to represent α is only $\log \alpha$. Thus, need to pay attention to size of numbers and multiplication complexity.

Ignore this issue for now. Can get around it for applications of interest where one typically wants to compute $p(\alpha) \mod m$ for some number m.

Addition

Compute the sum of polynomials
$a = (a_0, a_1, \ldots a_{n-1})$ and $b = (b_0, b_1, \ldots b_{n-1})$

Addition

Compute the sum of polynomials
$a = (a_0, a_1, \ldots a_{n-1})$ and $b = (b_0, b_1, \ldots b_{n-1})$
$a + b = (a_0 + b_0, a_1 + b_1, \ldots a_{n-1} + b_{n-1})$. Takes $O(n)$ time.

Multiplication

Compute the product of polynomials
$a = (a_0, a_1, \ldots a_n)$ and $b = (b_0, b_1, \ldots b_m)$
Recall $a \cdot b = (c_0, c_1, \ldots c_{n+m})$ where

$$c_k = \sum_{i,j:\, i+j=k} a_i \cdot b_j$$

Takes $\Theta(nm)$ time; $\Theta(n^2)$ when $n = m$.

Multiplication

Compute the product of polynomials
$a = (a_0, a_1, \ldots a_n)$ and $b = (b_0, b_1, \ldots b_m)$
Recall $a \cdot b = (c_0, c_1, \ldots c_{n+m})$ where

$$c_k = \sum_{i,j:\, i+j=k} a_i \cdot b_j$$

Takes $\Theta(nm)$ time; $\Theta(n^2)$ when $n = m$.
We will obtain a better algorithm!

Multiplication

Compute the product of polynomials
$a = (a_0, a_1, \ldots a_n)$ and $b = (b_0, b_1, \ldots b_m)$
Recall $a \cdot b = (c_0, c_1, \ldots c_{n+m})$ where

$$c_k = \sum_{i,j:\ i+j=k} a_i \cdot b_j$$

Takes $\Theta(nm)$ time; $\Theta(n^2)$ when $n = m$.
We will obtain a better algorithm!

Better/Efficient/Easy (today's lecture): preferably $O(n + m)$, but $O(n \log n)$ is also okay.

Convolutions

Definition

The convolution of vectors $a = (a_0, a_1, \ldots a_n)$ and $b = (b_0, b_1, \ldots b_m)$ is the vector $c = (c_0, c_1, \ldots c_{n+m})$ where

$$c_k = \sum_{i,j:\, i+j=k} a_i \cdot b_j$$

Convolution of vectors a and b is denoted by $a * b$. In other words, the convolution is the coefficients of the product of the two polynomials.

Revisiting Polynomial Representations

Representation

Polynomials represented by vector $a = (a_0, a_1, \ldots a_{n-1})$ of coefficients.

Revisiting Polynomial Representations

Representation

Polynomials represented by vector $a = (a_0, a_1, \ldots a_{n-1})$ of coefficients.

Question

Are there other useful ways to represent polynomials?

Representing Polynomials by Roots

Root of a polynomial $p(x)$: r such that $p(r) = 0$. If $r_1, r_2, \ldots, r_{n-1}$ are roots then $p(x) = a_{n-1}(x - r_1)(x - r_2)\ldots(x - r_{n-1})$.

Valid representation because of:

Theorem (Fundamental Theorem of Algebra)

Every polynomial $p(x)$ of degree d has exactly d roots $r_1, r_2, \ldots, r_d$ where the roots can be complex numbers and can be repeated.

Representing Polynomials by Roots

Representation

Polynomials represented by vector scale factor a_{n-1} and roots $r_1, r_2, \ldots, r_{n-1}$.

Representing Polynomials by Roots

Representation

Polynomials represented by vector scale factor a_{n-1} and roots $r_1, r_2, \ldots, r_{n-1}$.

- Evaluating p at a given x is easy. Why?

Representing Polynomials by Roots

Representation

Polynomials represented by vector scale factor a_{n-1} and roots $r_1, r_2, \ldots, r_{n-1}$.

- Evaluating p at a given x is easy. Why?
- Multiplication: given p, q with roots $r_1, \ldots, r_{n-1}$ and $s_1, \ldots, s_{m-1}$ the product $p \cdot q$ has roots $r_1, \ldots, r_{n-1}, s_1, \ldots, s_{m-1}$. Easy! $O(n + m)$ time.

Representing Polynomials by Roots

Representation

Polynomials represented by vector scale factor a_{n-1} and roots $r_1, r_2, \ldots, r_{n-1}$.

- Evaluating p at a given x is easy. Why?
- Multiplication: given p, q with roots $r_1, \ldots, r_{n-1}$ and $s_1, \ldots, s_{m-1}$ the product $p \cdot q$ has roots $r_1, \ldots, r_{n-1}, s_1, \ldots, s_{m-1}$. Easy! $O(n + m)$ time.
- Addition: requires $\Omega(nm)$ time?

Representing Polynomials by Roots

Representation

Polynomials represented by vector scale factor a_{n-1} and roots $r_1, r_2, \ldots, r_{n-1}$.

- Evaluating p at a given x is easy. Why?
- Multiplication: given p, q with roots $r_1, \ldots, r_{n-1}$ and $s_1, \ldots, s_{m-1}$ the product $p \cdot q$ has roots $r_1, \ldots, r_{n-1}, s_1, \ldots, s_{m-1}$. Easy! $O(n + m)$ time.
- Addition: requires $\Omega(nm)$ time?
- Given coefficient representation, how do we go to root representation? No finite algorithm because of potential for irrational roots.

Representing Polynomials by Samples

Let p be a polynomial of degree $n-1$.
Pick n *distinct samples* $x_0, x_1, x_2, \ldots, x_{n-1}$
Let $y_0 = p(x_0), y_1 = p(x_1), \ldots, y_{n-1} = p(x_{n-1})$.

Representation

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

Representing Polynomials by Samples

Let p be a polynomial of degree $n - 1$.
Pick n *distinct samples* $x_0, x_1, x_2, \ldots, x_{n-1}$
Let $y_0 = p(x_0), y_1 = p(x_1), \ldots, y_{n-1} = p(x_{n-1})$.

Representation

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

Is the above a valid representation?

Representing Polynomials by Samples

Let p be a polynomial of degree $n - 1$.
Pick n *distinct samples* $x_0, x_1, x_2, \ldots, x_{n-1}$
Let $y_0 = p(x_0), y_1 = p(x_1), \ldots, y_{n-1} = p(x_{n-1})$.

Representation

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

Is the above a valid representation? Why do we use $2n$ numbers instead of n numbers for coefficient and root representation?

Sample Representation

Theorem

Given a list $\{(x_0, y_0), (x_1, y_1), \ldots (x_{n-1}, y_{n-1})\}$ *there is exactly one polynomial* p *of degree* $n-1$ *such that* $p(x_j) = y_j$ *for* $j = 0, 1, \ldots, n-1$.

Sample Representation

Theorem

Given a list $\{(x_0, y_0), (x_1, y_1), \dots (x_{n-1}, y_{n-1})\}$ *there is exactly one polynomial* p *of degree* $n-1$ *such that* $p(x_j) = y_j$ *for* $j = 0, 1, \dots, n-1$.

So representation is valid.

Sample Representation

Theorem

Given a list $\{(x_0, y_0), (x_1, y_1), \ldots (x_{n-1}, y_{n-1})\}$ *there is exactly one polynomial* p *of degree* $n-1$ *such that* $p(x_j) = y_j$ *for* $j = 0, 1, \ldots, n-1$.

So representation is valid.
Can use same $x_0, x_1, \ldots, x_{n-1}$ for all polynomials of degree $n-1$.
No need to store them explicitly and hence need only n numbers $y_0, y_1, \ldots, y_{n-1}$.
(

Lagrange Interpolation

Given $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ the following polynomial p satisfies the property that $p(x_j) = y_j$ for $j = 0, 1, 2, \ldots, n-1$.

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

Lagrange Interpolation

Given $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ the following polynomial p satisfies the property that $p(x_j) = y_j$ for $j = 0, 1, 2, \ldots, n-1$.

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

For $n = 3$, $p(x) =$

$$y_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + y_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + y_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

Lagrange Interpolation

Given $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ the following polynomial p satisfies the property that $p(x_j) = y_j$ for $j = 0, 1, 2, \ldots, n-1$.

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

For $n = 3$, $p(x) =$

$$y_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + y_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + y_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

Easy to verify that $p(x_j) = y_j$! Thus there exists one polynomial of degree $n - 1$ that interpolates the values $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$.

Lagrange Interpolation

Given $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ there is a polynomial $p(x)$ such that $p(x_i) = y_i$ for $0 \leq i < n$. Can there be two distinct polynomials?

Lagrange Interpolation

Given $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ there is a polynomial $p(x)$ such that $p(x_i) = y_i$ for $0 \leq i < n$. Can there be two distinct polynomials?

No! Use Fundamental Theorem of Algebra to prove it — exercise.

Addition and Multiplication with Sample Representation

- Let $a = \{(x_0, y_0), (x_1, y_1), \ldots (x_{n-1}, y_{n-1})\}$ and $b = \{(x_0, y'_0), (x_1, y'_1), \ldots (x_{n-1}, y'_{n-1})\}$ be two polynomials of degree $n - 1$ in sample representation.

Addition and Multiplication with Sample Representation

- Let $a = \{(x_0, y_0), (x_1, y_1), \ldots (x_{n-1}, y_{n-1})\}$ and $b = \{(x_0, y'_0), (x_1, y'_1), \ldots (x_{n-1}, y'_{n-1})\}$ be two polynomials of degree $n - 1$ in sample representation.
- $a + b$ can be represented by

Addition and Multiplication with Sample Representation

- Let $a = \{(x_0, y_0), (x_1, y_1), \ldots (x_{n-1}, y_{n-1})\}$ and $b = \{(x_0, y'_0), (x_1, y'_1), \ldots (x_{n-1}, y'_{n-1})\}$ be two polynomials of degree $n-1$ in sample representation.
- $a + b$ can be represented by $\{(x_0, (y_0 + y'_0)), (x_1, (y_1 + y'_1)), \ldots (x_{n-1}, (y_{n-1} + y'_{n-1}))\}$
  - Thus, can be computed in $O(n)$ time

Addition and Multiplication with Sample Representation

- Let $a = \{(x_0, y_0), (x_1, y_1), \ldots (x_{n-1}, y_{n-1})\}$ and $b = \{(x_0, y'_0), (x_1, y'_1), \ldots (x_{n-1}, y'_{n-1})\}$ be two polynomials of degree $n - 1$ in sample representation.
- $a + b$ can be represented by $\{(x_0, (y_0 + y'_0)), (x_1, (y_1 + y'_1)), \ldots (x_{n-1}, (y_{n-1} + y'_{n-1}))\}$
    - Thus, can be computed in $O(n)$ time
- $a \cdot b$ can be evaluated at n samples

Addition and Multiplication with Sample Representation

- Let $a = \{(x_0, y_0), (x_1, y_1), \ldots (x_{n-1}, y_{n-1})\}$ and $b = \{(x_0, y'_0), (x_1, y'_1), \ldots (x_{n-1}, y'_{n-1})\}$ be two polynomials of degree $n-1$ in sample representation.
- $a + b$ can be represented by $\{(x_0, (y_0 + y'_0)), (x_1, (y_1 + y'_1)), \ldots (x_{n-1}, (y_{n-1} + y'_{n-1}))\}$
  - Thus, can be computed in $O(n)$ time
- $a \cdot b$ can be evaluated at n samples $\{(x_0, (y_0 \cdot y'_0)), (x_1, (y_1 \cdot y'_1)), \ldots (x_{n-1}, (y_{n-1} \cdot y'_{n-1}))\}$
  - Can be computed in $O(n)$ time.

Addition and Multiplication with Sample Representation

- Let $a = \{(x_0, y_0), (x_1, y_1), \ldots (x_{n-1}, y_{n-1})\}$ and $b = \{(x_0, y'_0), (x_1, y'_1), \ldots (x_{n-1}, y'_{n-1})\}$ be two polynomials of degree $n-1$ in sample representation.
- $a + b$ can be represented by $\{(x_0, (y_0 + y'_0)), (x_1, (y_1 + y'_1)), \ldots (x_{n-1}, (y_{n-1} + y'_{n-1}))\}$
  - Thus, can be computed in $O(n)$ time
- $a \cdot b$ can be evaluated at n samples $\{(x_0, (y_0 \cdot y'_0)), (x_1, (y_1 \cdot y'_1)), \ldots (x_{n-1}, (y_{n-1} \cdot y'_{n-1}))\}$
  - Can be computed in $O(n)$ time.

But what if p, q are given in coefficient form? Convolution requires p, q to be in coefficient form.

Recall

Goal: given polynomials $a = (a_0, \ldots, a_{n-1})$ and $b = (b_0, \ldots, b_{n-1})$ in coefficient representation, compute $a \cdot b$ in coefficient form **(convolution)**.

Recall

Goal: given polynomials $a = (a_0, \ldots, a_{n-1})$ and $b = (b_0, \ldots, b_{n-1})$ in coefficient representation, compute $a \cdot b$ in coefficient form **(convolution)**.

Sample representation:

- Fix $x_0, \ldots, x_{n-1}$.
- $a' = (x_0, a(x_0)), \ldots, (x_{n-1}, a(x_{n-1}))$, similarly b' from b.
  - **Theorem.** Unique degree $(n-1)$ polynomial corresponding to any given n samples.

Recall

Goal: given polynomials $a = (a_0, \ldots, a_{n-1})$ and $b = (b_0, \ldots, b_{n-1})$ in coefficient representation, compute $a \cdot b$ in coefficient form **(convolution)**.

Sample representation:

- Fix $x_0, \ldots, x_{n-1}$.
- $a' = (x_0, a(x_0)), \ldots, (x_{n-1}, a(x_{n-1}))$, similarly b' from b.
  - **Theorem.** Unique degree $(n-1)$ polynomial corresponding to any given n samples. a' is a *valid* representation of a.
- $a' \cdot b'$ requires $O(n)$ multiplications.

Recall

Goal: given polynomials $a = (a_0, \ldots, a_{n-1})$ and $b = (b_0, \ldots, b_{n-1})$ in coefficient representation, compute $a \cdot b$ in coefficient form **(convolution)**.

Sample representation:

- Fix $x_0, \ldots, x_{n-1}$.
- $a' = (x_0, a(x_0)), \ldots, (x_{n-1}, a(x_{n-1}))$, similarly b' from b.
  - **Theorem.** Unique degree $(n-1)$ polynomial corresponding to any given n samples. a' is a *valid* representation of a.
- $a' \cdot b'$ requires $O(n)$ multiplications.

Plan. Convert to sample representation. Multiply. Convert back to coefficient representation.

Coefficient representation to Sample representation

Given a polynomial a as $(a_0, a_1, \ldots, a_{n-1})$ can we obtain a sample representation $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ quickly? Also can we *invert* the representation quickly?

Coefficient representation to Sample representation

Given a polynomial a as $(a_0, a_1, \ldots, a_{n-1})$ can we obtain a sample representation $(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})$ quickly? Also can we *invert* the representation quickly?

- Suppose we choose $x_0, x_1, \ldots, x_{n-1}$ arbitrarily.
- Take $O(n)$ time to evaluate $y_j = a(x_j)$ given $(a_0, \ldots, a_{n-1})$.
- Total time is $\Omega(n^2)$
- Inversion via Lagrange interpolation also $\Omega(n^2)$

Key Idea

Can choose $x_0, x_1, \ldots, x_{n-1}$ carefully!

Total time to evaluate $a(x_0), a(x_1), \ldots, a(x_{n-1})$ should be better than evaluating each separately.

Key Idea

Can choose $x_0, x_1, \ldots, x_{n-1}$ carefully!

Total time to evaluate $a(x_0), a(x_1), \ldots, a(x_{n-1})$ should be better than evaluating each separately.

How do we choose $x_0, x_1, \ldots, x_{n-1}$ to save work?

A Simple Start

$$a(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{n-1} x^{n-1}$$

Assume n is a power of 2 for rest of the discussion.

Observation: $(-x)^{2j} = x^{2j}$. Can we exploit this?

A Simple Start

$$a(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{n-1} x^{n-1}$$

Assume n is a power of 2 for rest of the discussion.

Observation: $(-x)^{2j} = x^{2j}$. Can we exploit this?

Example

$3+4x+6x^2+2x^3+x^4+10x^5 = (3+6x^2+x^4)+x(4+2x^2+10x^4)$

A Simple Start

$$a(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{n-1} x^{n-1}$$

Assume n is a power of 2 for rest of the discussion.

Observation: $(-x)^{2j} = x^{2j}$. Can we exploit this?

Example

$$3+4x+6x^2+2x^3+x^4+10x^5 = (3+6x^2+x^4)+x(4+2x^2+10x^4)$$

$$a(c) = (3 + 6c^2 + c^4) + c(4 + 2c^2 + 10c^4)$$

A Simple Start

$$a(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{n-1} x^{n-1}$$

Assume n is a power of 2 for rest of the discussion.

Observation: $(-x)^{2j} = x^{2j}$. Can we exploit this?

Example

$$3+4x+6x^2+2x^3+x^4+10x^5 = (3+6x^2+x^4)+x(4+2x^2+10x^4)$$

$$a(c) = (3 + 6c^2 + c^4) + c(4 + 2c^2 + 10c^4)$$
$$a(-c) =$$

A Simple Start

$$a(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_{n-1} x^{n-1}$$

Assume n is a power of 2 for rest of the discussion.

Observation: $(-x)^{2j} = x^{2j}$. Can we exploit this?

Example

$$3+4x+6x^2+2x^3+x^4+10x^5 = (3+6x^2+x^4)+x(4+2x^2+10x^4)$$

$$a(c) = (3+6c^2+c^4) + c(4+2c^2+10c^4)$$
$$a(-c) = (3+6c^2+c^4) - c(4+2c^2+10c^4)$$

Odd and Even Decomposition

- Let $a = (a_0, a_1, \ldots a_{n-1})$ be a polynomial.
- Let $a_{\mathrm{odd}} = (a_1, a_3, a_5, \ldots)$ be the $(n/2 - 1)$ degree polynomial defined by the odd coefficients; so

$$a_{\mathrm{odd}}(x) = a_1 + a_3 x + a_5 x^2 + \cdots$$

Odd and Even Decomposition

- Let $a = (a_0, a_1, \ldots a_{n-1})$ be a polynomial.
- Let $a_{\mathrm{odd}} = (a_1, a_3, a_5, \ldots)$ be the $(n/2 - 1)$ degree polynomial defined by the odd coefficients; so

$$a_{\mathrm{odd}}(x) = a_1 + a_3 x + a_5 x^2 + \cdots$$

- Similarly, let $a_{\mathrm{even}}(x) = a_0 + a_2 x + \ldots$ be the $(n/2 - 1)$ degree polynomial defined by the even coefficients.

Odd and Even Decomposition

- Let $a = (a_0, a_1, \ldots a_{n-1})$ be a polynomial.
- Let $a_{\mathrm{odd}} = (a_1, a_3, a_5, \ldots)$ be the $(n/2 - 1)$ degree polynomial defined by the odd coefficients; so

$$a_{\mathrm{odd}}(x) = a_1 + a_3 x + a_5 x^2 + \cdots$$

- Similarly, let $a_{\mathrm{even}}(x) = a_0 + a_2 x + \ldots$ be the $(n/2 - 1)$ degree polynomial defined by the even coefficients.
- Observe

$$a(x) = a_{\mathrm{even}}(x^2) + x a_{\mathrm{odd}}(x^2)$$

- Thus, evaluating a at x can be reduced to evaluating lower degree polynomials plus constantly many arithmetic operations.

Exploiting Odd-Even Decomposition

$$a(x) = a_{\mathrm{even}}(x^2) + x a_{\mathrm{odd}}(x^2)$$

- Choose n samples
  $x_0, x_1, x_2, \ldots, x_{n/2-1}, -x_0, -x_1, \ldots, -x_{n/2-1}$
- Evaluate a_{even} and a_{odd} at $x_0^2, x_1^2, x_2^2, \ldots, x_{n/2-1}^2$.

Exploiting Odd-Even Decomposition

$$a(x) = a_{\text{even}}(x^2) + x a_{\text{odd}}(x^2)$$

- Choose n samples
  $x_0, x_1, x_2, \ldots, x_{n/2-1}, -x_0, -x_1, \ldots, -x_{n/2-1}$
- Evaluate a_{even} and a_{odd} at $x_0^2, x_1^2, x_2^2, \ldots, x_{n/2-1}^2$.
- For each $i = 0$ to $(n/2 - 1)$, evaluate

$$a(x_i) = a_{\text{even}}(x_i^2) + x_i a_{\text{odd}}(x_i^2)$$
$$a(-x_i) = a_{\text{even}}(x_i^2) - x_i a_{\text{odd}}(x_i^2)$$

Exploiting Odd-Even Decomposition

$$a(x) = a_{\text{even}}(x^2) + x a_{\text{odd}}(x^2)$$

- Choose n samples
  $x_0, x_1, x_2, \ldots, x_{n/2-1}, -x_0, -x_1, \ldots, -x_{n/2-1}$
- Evaluate a_{even} and a_{odd} at $x_0^2, x_1^2, x_2^2, \ldots, x_{n/2-1}^2$.
- For each $i = 0$ to $(n/2 - 1)$, evaluate

$$a(x_i) = a_{\text{even}}(x_i^2) + x_i a_{\text{odd}}(x_i^2)$$
$$a(-x_i) = a_{\text{even}}(x_i^2) - x_i a_{\text{odd}}(x_i^2)$$

  Total of $O(n)$ work!

Exploiting Odd-Even Decomposition

$$a(x) = a_{\text{even}}(x^2) + x a_{\text{odd}}(x^2)$$

- Choose n samples
  $x_0, x_1, x_2, \ldots, x_{n/2-1}, -x_0, -x_1, \ldots, -x_{n/2-1}$
- Evaluate a_{even} and a_{odd} at $x_0^2, x_1^2, x_2^2, \ldots, x_{n/2-1}^2$.
- For each $i = 0$ to $(n/2 - 1)$, evaluate
  $$a(x_i) = a_{\text{even}}(x_i^2) + x_i a_{\text{odd}}(x_i^2)$$
  $$a(-x_i) = a_{\text{even}}(x_i^2) - x_i a_{\text{odd}}(x_i^2)$$
  Total of $O(n)$ work!
- Suppose we can make this work recursively. Then

$$T(n) = 2T(n/2) + O(n) \text{ which implies } T(n) = O(n \log n)$$

Collapsible sets

Definition

Given a set X of numbers square(X) (for square of X) is the set $\{x^2 \mid x \in X\}$.

Collapsible sets

Definition

Given a set X of numbers square(X) (for square of X) is the set $\{x^2 \mid x \in X\}$.

Definition

A set X of n numbers is *collapsible* if square$(X) \subset X$ and $|\text{square}(X)| = n/2$.

Collapsible sets

Definition

Given a set X of numbers $\text{square}(X)$ (for square of X) is the set $\{x^2 \mid x \in X\}$.

Definition

A set X of n numbers is *collapsible* if $\text{square}(X) \subset X$ and $|\text{square}(X)| = n/2$.

Definition

A set X of n numbers (for n a power of 2) is *recursively collapsible* if $n = 1$ or if X is collapsible and $\text{square}(X)$ is recursively collapsible.

Divide and Conquer assuming collapsible set

Given a *recursively collapsible* set X of size n, compute sample representation of polynomial a of degree $(n-1)$ as follows:

SampleRepresentation(a, X, n)
    If $n = 1$ return $a(x_0)$ where $X = \{x_0\}$
    Compute square(X) in $O(n)$ time %note:$|\text{square}(X)| = n/2$

Divide and Conquer assuming collapsible set

Given a *recursively collapsible* set X of size n, compute sample representation of polynomial a of degree $(n-1)$ as follows:

SampleRepresentation(a, X, n)
    If $n = 1$ return $a(x_0)$ where $X = \{x_0\}$
    Compute square(X) in $O(n)$ time %note: $|\text{square}(X)| = n/2$
    $\{y_0, y_1, \ldots, y_{n/2-1}\}$ = **SampleRepresentation**$(a_{odd}, \text{square}(X), n/2)$
    $\{y'_0, y'_1, \ldots, y'_{n/2-1}\}$ = **SampleRepresentation**$(a_{even}, \text{square}(X), n/2)$

Divide and Conquer assuming collapsible set

Given a *recursively collapsible* set X of size n, compute sample representation of polynomial a of degree $(n-1)$ as follows:

SampleRepresentation$(a,\ X,\ n)$
```
If n = 1 return a(x_0) where X = {x_0}
Compute square(X) in O(n) time %note:|square(X)| = n/2
{y_0, y_1, ..., y_{n/2-1}} = SampleRepresentation(a_odd, square(X), n/2)
{y'_0, y'_1, ..., y'_{n/2-1}} = SampleRepresentation(a_even, square(X), n/2)

For each i from 0 to (n-1) compute
    z_i = a_even(x_i^2) + x_i a_odd(x_i^2)

Return {z_0, z_1, ..., z_{n-1}}
```

Divide and Conquer assuming collapsible set

Given a *recursively collapsible* set X of size n, compute sample representation of polynomial a of degree $(n-1)$ as follows:

```
SampleRepresentation(a, X, n)
    If n = 1 return a(x_0) where X = {x_0}
    Compute square(X) in O(n) time %note:|square(X)| = n/2
    {y_0, y_1, ..., y_{n/2-1}} = SampleRepresentation(a_odd, square(X), n/2)
    {y'_0, y'_1, ..., y'_{n/2-1}} = SampleRepresentation(a_even, square(X), n/2)

    For each i from 0 to (n-1) compute
        z_i = a_even(x_i^2) + x_i a_odd(x_i^2)

    Return {z_0, z_1, ..., z_{n-1}}
```

Exercise: show that algorithm runs in $O(n \log n)$ time

Are there collapsible sets?

- n samples $x_0, x_1, x_2, \ldots, x_{n/2-1}, -x_0, -x_1, \ldots, -x_{n/2-1}$
- Next step in recursion $x_0^2, x_1^2, \ldots, x_{n/2-1}^2$

Are there collapsible sets?

- n samples $x_0, x_1, x_2, \ldots, x_{n/2-1}, -x_0, -x_1, \ldots, -x_{n/2-1}$
- Next step in recursion $x_0^2, x_1^2, \ldots, x_{n/2-1}^2$
- To continue recursion, we need

$$\{x_0^2, x_1^2, \ldots, x_{\frac{n}{2}-1}^2\} = \{z_0, z_1, \ldots, z_{\frac{n}{4}-1}, -z_0, -z_1, \ldots, -z_{\frac{n}{4}-1}\}$$

Are there collapsible sets?

- n samples $x_0, x_1, x_2, \ldots, x_{n/2-1}, -x_0, -x_1, \ldots, -x_{n/2-1}$
- Next step in recursion $x_0^2, x_1^2, \ldots, x_{n/2-1}^2$
- To continue recursion, we need

$$\{x_0^2, x_1^2, \ldots, x_{\frac{n}{2}-1}^2\} = \{z_0, z_1, \ldots, z_{\frac{n}{4}-1}, -z_0, -z_1, \ldots, -z_{\frac{n}{4}-1}\}$$

- If $z_0 = x_0^2$ and $-z_0 = x_{n/4}^2$ then $x_0 = \sqrt{-1}x_{n/4}$ That is $x_0 = ix_{n/4}$ where i is the imaginary number.

Are there collapsible sets?

- n samples $x_0, x_1, x_2, \ldots, x_{n/2-1}, -x_0, -x_1, \ldots, -x_{n/2-1}$
- Next step in recursion $x_0^2, x_1^2, \ldots, x_{n/2-1}^2$
- To continue recursion, we need

$$\{x_0^2, x_1^2, \ldots, x_{\frac{n}{2}-1}^2\} = \{z_0, z_1, \ldots, z_{\frac{n}{4}-1}, -z_0, -z_1, \ldots, -z_{\frac{n}{4}-1}\}$$

- If $z_0 = x_0^2$ and $-z_0 = x_{n/4}^2$ then $x_0 = \sqrt{-1}x_{n/4}$ That is $x_0 = ix_{n/4}$ where i is the imaginary number.
- Can continue recursion but need to go to *complex* numbers.

Complex Numbers

Notation

For the rest of lecture, i stands for $\sqrt{-1}$

Definition

Complex numbers are points lying in the complex plane represented as

Cartesian $a + ib = \sqrt{a^2 + b^2} e^{(\arctan(b/a))i}$

Polar $re^{\theta i} = r(\cos\theta + i\sin\theta)$

Thus, $e^{\pi i} = -1$ and $e^{2\pi i} = 1$.

Power Series for Functions (Recall)

What is e^z when z is a real number? When z is a complex number?

$$e^z = 1 + z/1! + z^2/2! + \ldots + z^j/j! + \ldots$$

Therefore

$$\begin{aligned} e^{i\theta} &= 1 + i\theta/1! + (i\theta)^2/2! + (i\theta)^3/3! + \ldots \\ &= (1 - \theta^2/2! + \theta^4/4! - \ldots +) + i(\theta - \theta^3/3! + \ldots +) \\ &= \cos\theta + i\sin\theta \end{aligned}$$

Complex Roots of Unity

What are the roots of the polynomial $x^k - 1$? $(e^{2\pi i} = 1)$

- Clearly 1 is a root.

Complex Roots of Unity

What are the roots of the polynomial $x^k - 1$? $(e^{2\pi i} = 1)$

- Clearly 1 is a root.
- Suppose $re^{\theta i}$ is a root then $r^k e^{k\theta i} = 1$ which implies that $r = 1$ and $k\theta = 2\pi \Rightarrow \theta = 2\pi/k$

Complex Roots of Unity

What are the roots of the polynomial $x^k - 1$? $(e^{2\pi i} = 1)$

- Clearly 1 is a root.
- Suppose $re^{\theta i}$ is a root then $r^k e^{k\theta i} = 1$ which implies that $r = 1$ and $k\theta = 2\pi \Rightarrow \theta = 2\pi/k$
- Let $\omega_k = e^{2\pi i/k}$. The roots are $1 = \omega_k^0, \omega_k^2, \ldots, \omega_k^{k-1}$ where $\omega_k^j = e^{2\pi ji/k}$.

Complex Roots of Unity

What are the roots of the polynomial $x^k - 1$? $(e^{2\pi i} = 1)$

- Clearly 1 is a root.
- Suppose $re^{\theta i}$ is a root then $r^k e^{k\theta i} = 1$ which implies that $r = 1$ and $k\theta = 2\pi \Rightarrow \theta = 2\pi/k$
- Let $\omega_k = e^{2\pi i/k}$. The roots are $1 = \omega_k^0, \omega_k^2, \ldots, \omega_k^{k-1}$ where $\omega_k^j = e^{2\pi j i/k}$.

Proposition

Let ω_k be $e^{2\pi i/k}$. The equation $x^k = 1$ has k distinct complex roots given by $\omega_k^j = e^{(2\pi j)i/k}$ for $j = 0, 1, \ldots k-1$

Proof.

$(\omega_k^j)^k = (e^{2\pi j i/k})^k = e^{2\pi j i} = (e^{2\pi i})^j = (1)^j = 1$ □

Roots of unity form a collapsible set

Observation 1: $\omega_k^j = \omega_k^{j \bmod k}$

$j = 3k + 4$

$\omega_k^j = \omega_k^{3k} \cdot \omega_k^4 = \omega_k^4$

$\omega_k^{3k} = 1$

Roots of unity form a collapsible set

Observation 1: $\omega_k^j = \omega_k^{j \bmod k}$

Lemma

Assume n is a power of 2. The n'th roots of unity are a recursively collapsible set.

Proof.

Let $X_n = \{1, \omega_n, \omega_n^2, \ldots, \omega_n^{n-1}\}$.

Roots of unity form a collapsible set

Observation 1: $\omega_k^j = \omega_k^{j \bmod k}$

Lemma

Assume n is a power of 2. The n'th roots of unity are a recursively collapsible set.

Proof.

Let $X_n = \{1, \omega_n, \omega_n^2, \ldots, \omega_n^{n-1}\}$. $(\omega_n^{n/2+j})^2 = \omega_n^{n+2j} = \omega_n^{2j}$,

Roots of unity form a collapsible set

Observation 1: $\omega_k^j = \omega_k^{j \bmod k}$

Lemma

Assume n is a power of 2. The n'th roots of unity are a recursively collapsible set.

Proof.

Let $X_n = \{1, \omega_n, \omega_n^2, \ldots, \omega_n^{n-1}\}$. $(\omega_n^{n/2+j})^2 = \omega_n^{n+2j} = \omega_n^{2j}$, for each $j < n/2$. □

Roots of unity form a collapsible set

Observation 1: $\omega_k^j = \omega_k^{j \bmod k}$

Lemma

Assume n is a power of 2. The n'th roots of unity are a recursively collapsible set.

Proof.

Let $X_n = \{1, \omega_n, \omega_n^2, \ldots, \omega_n^{n-1}\}$. $(\omega_n^{n/2+j})^2 = \omega_n^{n+2j} = \omega_n^{2j}$, for each $j < n/2$. □

- $X_1 = \{1\}$, $X_2 = \{1, -1\}$
- $X_4 = \{1, -1, i, -i\}$
- $X_8 = \{1, -1, i, -i, \frac{1}{\sqrt{2}}(\pm 1 \pm i)\}$

Discrete Fourier Transform

Definition

Given vector $a = (a_0, a_1, \ldots, a_{n-1})$ the *Discrete Fourier Transform* (DFT) of a is the vector $a' = (a'_0, a'_1, \ldots, a'_{n-1})$ where $a'_j = a(\omega_n^j)$ for $0 \leq j < n$.

a' is a sample representation of polynomial with coefficient reprentation a at n'th roots of unity.

Discrete Fourier Transform

Definition

Given vector $a = (a_0, a_1, \ldots, a_{n-1})$ the *Discrete Fourier Transform* (DFT) of a is the vector $a' = (a'_0, a'_1, \ldots, a'_{n-1})$ where $a'_j = a(\omega_n^j)$ for $0 \leq j < n$.

a' is a sample representation of polynomial with coefficient reprentation a at n'th roots of unity.

We have shown that a' can be computed from a in $O(n \log n)$ time. This divide and conquer *algorithm* is called the *Fast Fourier Transform* (FFT).

Back to Convolutions and Polynomial Multiplication

Convolutions (products)

Compute convolution $c = (c_0, c_1, \ldots, c_{2n-2})$ of $a = (a_0, a_1, \ldots a_{n-1})$ and $b = (b_0, b_1, \ldots b_{n-1})$

1. Evaluate a and b at some n sample points.
2. Compute sample representation of product. That is $c' = (a'_0 b'_0, a'_1 b'_1, \ldots, a'_{n-1} b'_{n-1})$.
3. Compute coefficients of unique polynomial associated with sample representation of product. That is compute c from c'.

Back to Convolutions and Polynomial Multiplication

Convolutions (products)

Compute convolution $c = (c_0, c_1, \ldots, c_{2n-2})$ of $a = (a_0, a_1, \ldots a_{n-1})$ and $b = (b_0, b_1, \ldots b_{n-1})$

1. Evaluate a and b at the nth roots of unity.
2. Compute sample representation of product. That is $c' = (a'_0 b'_0, a'_1 b'_1, \ldots, a'_{n-1} b'_{n-1})$.
3. Compute coefficients of unique polynomial associated with sample representation of product. That is compute c from c'.

Can we really compute c from c'?

Back to Convolutions and Polynomial Multiplication

Convolutions (products)

Compute convolution $c = (c_0, c_1, \ldots, c_{2n-2})$ of $a = (a_0, a_1, \ldots a_{n-1})$ and $b = (b_0, b_1, \ldots b_{n-1})$

1. Evaluate a and b at the nth roots of unity.
2. Compute sample representation of product. That is $c' = (a'_0 b'_0, a'_1 b'_1, \ldots, a'_{n-1} b'_{n-1})$.
3. Compute coefficients of unique polynomial associated with sample representation of product. That is compute c from c'.

Can we really compute c from c'? We only have n sample points and c' has $2n-1$ coefficients!

Convolutions and Polynomial Multiplication

Convolutions

Compute convolution $c = (c_0, c_1, \ldots, c_{2n-2})$ of $a = (a_0, a_1, \ldots a_{n-1})$ and $b = (b_0, b_1, \ldots b_{n-1})$

1. Pad a with n zeroes to make it a $(2n-1)$ degree polynomial $a = (a_0, a_1, \ldots, a_{n-1}, a_n, a_{n+1}, \ldots, a_{2n-1})$. Similarly for b.

Convolutions and Polynomial Multiplication

Convolutions

Compute convolution $c = (c_0, c_1, \ldots, c_{2n-2})$ of $a = (a_0, a_1, \ldots a_{n-1})$ and $b = (b_0, b_1, \ldots b_{n-1})$

1. Pad a with n zeroes to make it a $(2n - 1)$ degree polynomial $a = (a_0, a_1, \ldots, a_{n-1}, a_n, a_{n+1}, \ldots, a_{2n-1})$. Similarly for b.
2. Compute values of a and b at the $2n$th roots of unity.
3. Compute sample representation of product. That is $c' = (a'_0 b'_0, a'_1 b'_1, \ldots, a'_{n-1} b'_{n-1}, \ldots, a'_{2n-1} b'_{2n-1})$.
4. Compute coefficients of unique polynomial associated with sample representation of product. That is compute c from c'.

Convolutions and Polynomial Multiplication

Convolutions

Compute convolution $c = (c_0, c_1, \ldots, c_{2n-2})$ of $a = (a_0, a_1, \ldots a_{n-1})$ and $b = (b_0, b_1, \ldots b_{n-1})$

1. Pad a with n zeroes to make it a $(2n-1)$ degree polynomial $a = (a_0, a_1, \ldots, a_{n-1}, a_n, a_{n+1}, \ldots, a_{2n-1})$. Similarly for b.
2. Compute values of a and b at the $2n$th roots of unity.
3. Compute sample representation of product. That is $c' = (a'_0 b'_0, a'_1 b'_1, \ldots, a'_{n-1} b'_{n-1}, \ldots, a'_{2n-1} b'_{2n-1})$.
4. Compute coefficients of unique polynomial associated with sample representation of product. That is compute c from c'.

- Step 2 takes $O(n \log n)$ using divide and conquer algorithm
- Step 3 takes $O(n)$ time
- Step 4?

Part II

Inverse Fourier Transform

Inverse Fourier Transform

Input Given the evaluation of a $n-1$-degree polynomial a on the nth roots of unity specified by vector a'

Goal Compute the coefficients of a

We saw that a' can be computed from a in $O(n \log n)$ time. Can we compute a from a' in $O(n \log n)$ time?

A Matrix Point of View

$$a(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$$

$a'_0 = a(x_0), a'_1 = a(x_1), \ldots, a'_{n-1} = a(x_{n-1})$.

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_j & x_j^2 & \ldots & x_j^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \ldots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_j \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} a'_0 \\ a'_1 \\ \vdots \\ a'_j \\ \vdots \\ a'_{n-1} \end{bmatrix}$$

A Matrix Point of View

$$a(x) = a_0 + a_1 x + \cdots + a_{n-1}x^{n-1}$$

$x^n - 1 = 0$

Denote $\omega = \omega_n^1 = e^{2\pi/n}$. Let $x_j = \omega^j$

$a'_0 = a(1), a'_1 = a(\omega), \ldots, a'_{n-1} = a(\omega^{n-1})$.

$1, \omega, \omega^2, \ldots, \omega^{n-1}$

n^{th} roots of unity

$$\begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega & \omega^2 & \ldots & \omega^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^j & \omega^{2j} & \ldots & \omega^{j(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \ldots & \omega^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_j \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} a'_0 \\ a'_1 \\ \vdots \\ a'_j \\ \vdots \\ a'_{n-1} \end{bmatrix}$$

Inverting the Matrix

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_j \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^j & \omega^{2j} & \dots & \omega^{j(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{bmatrix}^{-1} \begin{bmatrix} a'_0 \\ a'_1 \\ \vdots \\ a'_j \\ \vdots \\ a'_{n-1} \end{bmatrix}$$

Inverting the Matrix

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^j & \omega^{2j} & \dots & \omega^{j(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)(n-1)} \end{bmatrix}^{-1} = \frac{1}{n} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \dots & \omega^{-(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-j} & \omega^{-2j} & \dots & \omega^{-j(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \dots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$

Replace ω by ω^{-1} which is also a root of unity!
Since $\omega^j = \omega^{j \mod n}$, we get $\omega^{-j} = e^{-j2\pi/n} = \omega^{(n-j)2\pi/n}$.

Inverting the Matrix

$$
\begin{bmatrix}
1 & 1 & 1 & \dots & 1 \\
1 & \omega & \omega^2 & \dots & \omega^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^j & \omega^{2j} & \dots & \omega^{j(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{n-1} & \omega^{2(n-1)} & \dots & \omega^{(n-1)(n-1)}
\end{bmatrix}^{-1}
= \frac{1}{n}
\begin{bmatrix}
1 & 1 & 1 & \dots & 1 \\
1 & \omega^{-1} & \omega^{-2} & \dots & \omega^{-(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{-j} & \omega^{-2j} & \dots & \omega^{-j(n-1)} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \omega^{-(n-1)} & \omega^{-2(n-1)} & \dots & \omega^{-(n-1)(n-1)}
\end{bmatrix}
$$

Replace ω by ω^{-1} which is also a root of unity!
Since $\omega^j = \omega^{j \bmod n}$, we get $\omega^{-j} = e^{-j2\pi/n} = \omega^{(n-j)2\pi/n}$.

Inverse matrix is simply a permutation of the original matrix modulo scale factor $1/n$.

Why does it work?

Check $VV^{-1} = I$ where I is the $n \times n$ identity matrix.

Why does it work?

Check $VV^{-1} = I$ where I is the $n \times n$ identity matrix.

Observation: $\sum_{s=0}^{n-1}(\omega^j)^s = (1 + \omega^j + \omega^{2j} + \ldots + \omega^{(n-1)j}) = 0$, $j \neq 0$

Why does it work?

Check $VV^{-1} = I$ where I is the $n \times n$ identity matrix.

Observation: $\sum_{s=0}^{n-1} (\omega^j)^s = (1 + \omega^j + \omega^{2j} + \ldots + \omega^{(n-1)j}) = 0$, $j \neq 0$

- ω^j is root of $x^n - 1 = (x - 1)(x^{n-1} + x^{n-2} + \ldots + 1)$
- Thus, ω^j is root of $(x^{n-1} + x^{n-2} + \ldots + 1)$

Why does it work?

Check $VV^{-1} = I$ where I is the $n \times n$ identity matrix.

Observation: $\sum_{s=0}^{n-1} (\omega^j)^s = (1 + \omega^j + \omega^{2j} + \ldots + \omega^{(n-1)j}) = 0$, $j \neq 0$

- ω^j is root of $x^n - 1 = (x-1)(x^{n-1} + x^{n-2} + \ldots + 1)$
- Thus, ω^j is root of $(x^{n-1} + x^{n-2} + \ldots + 1)$

$$(1, \omega^j, \omega^{2j}, \ldots, \omega^{j(n-1)}) \cdot (1, \omega^{-k}, \omega^{-2k}, \ldots, \omega^{-k(n-1)}) = \sum_{s=0}^{n-1} \omega^{s(j-k)}$$

Note that ω^{j-k} is a n'th root of unity. If $j = k$ then sum is n, otherwise by previous observation sum is 0.

Why does it work?

Check $VV^{-1} = I$ where I is the $n \times n$ identity matrix.

Observation: $\sum_{s=0}^{n-1} (\omega^j)^s = (1 + \omega^j + \omega^{2j} + \ldots + \omega^{(n-1)j}) = 0$, $j \neq 0$

- ω^j is root of $x^n - 1 = (x - 1)(x^{n-1} + x^{n-2} + \ldots + 1)$
- Thus, ω^j is root of $(x^{n-1} + x^{n-2} + \ldots + 1)$

$$(1, \omega^j, \omega^{2j}, \ldots, \omega^{j(n-1)}) \cdot (1, \omega^{-k}, \omega^{-2k}, \ldots, \omega^{-k(n-1)}) = \sum_{s=0}^{n-1} \omega^{s(j-k)}$$

Note that ω^{j-k} is a n'th root of unity. If $j = k$ then sum is n, otherwise by previous observation sum is 0.

Rows of matrix V (and hence also those of V^{-1}) are *orthogonal*. Thus $a' = Va$ can be thought of transforming the vector a into a new Fourier basis with basis vectors corresponding to rows of V.

Inverse Fourier Transform

Input Given the evaluation of a $n-1$-degree polynomial a on the nth roots of unity specified by vector a'

Goal Compute the coefficients of a

We saw that a' can be computed from a in $O(n \log n)$ time. Can we compute a from a' in $O(n \log n)$ time?

Yes! $a = V^{-1}a'$ which is simply a permuted and scaled version of DFT. Hence can be computed in $O(n \log n)$ time.

Convolutions Once More

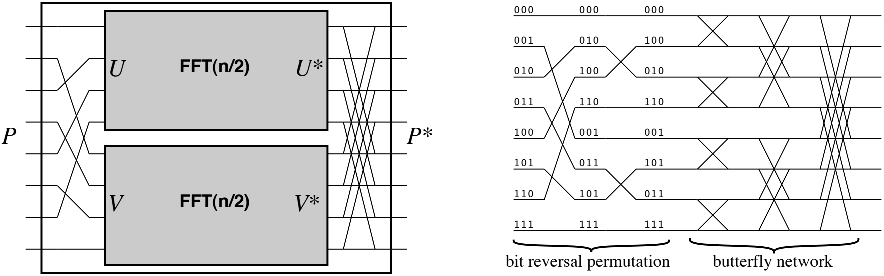
Convolutions

Compute convolution of $a = (a_0, a_1, \ldots a_{n-1})$ and $b = (b_0, b_1, \ldots b_{n-1})$

1. Compute values of a and b at the $2n$th roots of unity
2. Compute sample representation c' of product $c = a \cdot b$
3. Compute c from c' using inverse Fourier transform.

- Step 1 takes $O(n \log n)$ using two FFTs
- Step 2 takes $O(n)$ time
- Step 3 takes $O(n \log n)$ using one FFT

FFT Circuit

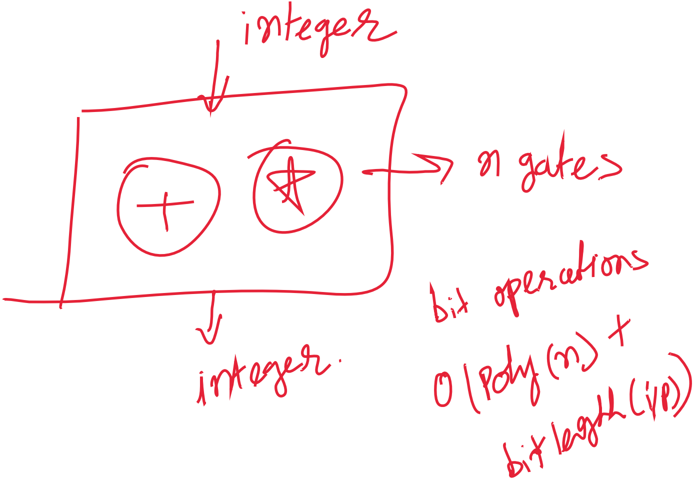


The recursive structure of the FFT algorithm.

Numerical Issues

- As noted earlier evaluating a polynomial p at a point x makes numbers big
- Are we cheating when we say $O(n \log n)$ algorithm for convolution?
- Can get around numerical issues — work in finite fields and avoid numbers growing too big.
- Outside the scope of lecture
- We will assume for reductions that convolution can be done in $O(n \log n)$ time.

Numerical Issues: Puzzle



Part III

Application to String Matching

Basic string matching problem:

Input: Given a pattern string P on length m and a text string T of length n over a fixed alphabet Σ

Goal: Does P occur as a substring of T? Find all "matches" of P in T.

Basic string matching problem:

- **Input** Given a pattern string P on length m and a text string T of length n over a fixed alphabet Σ
- **Goal** Does P occur as a substring of T? Find all "matches" of P in T.

Several generalizations. Matching with don't cares.

- **Input** Given a pattern string P on length m over $\Sigma \cup \{*\}$ ($*$ is a don't care) and a text string T of length n over Σ
- **Goal** Find all "matches" of P in T. $*$ matches with any character of Σ

Basic string matching problem:

Input Given a pattern string P on length m and a text string T of length n over a fixed alphabet Σ

Goal Does P occur as a substring of T? Find all "matches" of P in T.

Several generalizations. Matching with don't cares.

Input Given a pattern string P on length m over $\Sigma \cup \{*\}$ ($*$ is a don't care) and a text string T of length n over Σ

Goal Find all "matches" of P in T. $*$ matches with any character of Σ

Example: $P = a**$, $T = aardvark$

Matches?

Shifted products via Convolution

Given two arrays A and B with say with $A[0..m-1]$ and $B[0..n-1]$ with $m \leq n$

Input Two arrays: $A[0..(m-1)]$ and $B[0..(n-1)]$.

Goal Compute all shifted products in array $C[0..(n-m-1)]$ where $C[i] = \sum_{j=0}^{m-1} A[j]B[i+j]$.

Shifted products via Convolution

Given two arrays A and B with say with $A[0..m-1]$ and $B[0..n-1]$ with $m \leq n$

Input Two arrays: $A[0..(m-1)]$ and $B[0..(n-1)]$.

Goal Compute all shifted products in array $C[0..(n-m-1)]$ where $C[i] = \sum_{j=0}^{m-1} A[j]B[i+j]$.

Example: $A = [0, 1, 1, 0]$, $B = [0, 0, 1, 1, 1, 0, 1]$
$C =$

Shifted products via Convolution

Given two arrays A and B with say with $A[0..m-1]$ and $B[0..n-1]$ with $m \leq n$

Input Two arrays: $A[0..(m-1)]$ and $B[0..(n-1)]$.

Goal Compute all shifted products in array $C[0..(n-m-1)]$ where $C[i] = \sum_{j=0}^{m-1} A[j]B[i+j]$.

Example: $A = [0,1,1,0]$, $B = [0,0,1,1,1,0,1]$
$C =$

Lemma

Reverse of C is the convolution of the vectors A and reverse of B.

Proof.

Exercise. □

Reduction of pattern matching to shifted products

Assume first that $\Sigma = \{0, 1\}$

Goal:

- Convert $P = a_0 a_1 \ldots a_{m-1}$ to binary array A of size m.
- Convert $T = b_0 b_1 \ldots b_{n-1}$ to binary array B of size n.
- So that we can use shifted product C of A and B to count "mismatches".

Reduction of pattern matching to shifted products

Assume first that $\Sigma = \{0, 1\}$

Goal:

- Convert $P = a_0 a_1 \ldots a_{m-1}$ to binary array A of size m.
- Convert $T = b_0 b_1 \ldots b_{n-1}$ to binary array B of size n.
- So that we can use shifted product C of A and B to count "mismatches".
- **Type 1** mismatches: $C[i]$ counts # j's where $P[j] = 0$ and $T[i + j] = 1$, when P is aligned with T at $T[i]$.

Reduction of pattern matching to shifted products

Assume first that $\Sigma = \{0, 1\}$

Goal:

- Convert $P = a_0 a_1 \ldots a_{m-1}$ to binary array A of size m.
- Convert $T = b_0 b_1 \ldots b_{n-1}$ to binary array B of size n.
- So that we can use shifted product C of A and B to count "mismatches".
- **Type 1** mismatches: $C[i]$ counts # j's where $P[j] = 0$ and $T[i + j] = 1$, when P is aligned with T at $T[i]$.

Example:

$$T = 10110010\ldots$$
$$P = 010$$

Reduction of pattern matching to shifted products

Assume first that $\Sigma = \{0, 1\}$

Goal:

- Convert $P = a_0 a_1 \ldots a_{m-1}$ to binary array A of size m.
- Convert $T = b_0 b_1 \ldots b_{n-1}$ to binary array B of size n.
- So that we can use shifted product C of A and B to count "mismatches".
- **Type 1** mismatches: $C[i]$ counts # j's where $P[j] = 0$ and $T[i + j] = 1$, when P is aligned with T at $T[i]$.

Example:
$$\begin{aligned} T &= 10110010\ldots \\ P &= 010 \end{aligned}$$

- Finding Type 1 mismatches:
  - $B[j] = T[j]$
  - If $P[j] = 0$ set $A[j] = 1$, if $P[j] = 1$ or $*$ set $A[j] = 0$.

Reduction of pattern matching to shifted products

- **Type 2** mismatches: $C[i]$ counts $\#$ j's where $P[j] = 1$ and $T[i+j] = 0$, when P is aligned with T at $T[i]$.

Reduction of pattern matching to shifted products

- **Type 2** mismatches: $C[i]$ counts # j's where $P[j] = 1$ and $T[i + j] = 0$, when P is aligned with T at $T[i]$.

Example:
$$T = 10110010\ldots$$
$$P = 010$$

Reduction of pattern matching to shifted products

- **Type 2** mismatches: $C[i]$ counts # j's where $P[j] = 1$ and $T[i + j] = 0$, when P is aligned with T at $T[i]$.

Example: $$T = 10110010\ldots$$ $$P = 010$$

- Finding Type 2 mismatches:
  - $B[j] = (1 - T[j])$ (flip the bits)
  - If $P[j] = 0$ or $*$ set $A[j] = 0$, if $P[j] = 1$ set $A[j] = 1$.

Reduction of pattern matching to shifted products

- **Type 2** mismatches: $C[i]$ counts $\#$ j's where $P[j] = 1$ and $T[i+j] = 0$, when P is aligned with T at $T[i]$.

Example: $$\begin{aligned} T &= 10110010\ldots \\ P &= 010 \end{aligned}$$

- Finding Type 2 mismatches:
  - $B[j] = (1 - T[j])$ (flip the bits)
  - If $P[j] = 0$ or $*$ set $A[j] = 0$, if $P[j] = 1$ set $A[j] = 1$.

There is a match at position i of T iff both types of mismatches are 0.

Running time analysis

- Reducing to shift product is $O(n)$.
- Need to compute two convolutions with polynomials of size n and m. Total run time is $O(n \log n)$ (here we assume $m \leq n$).

Running time analysis

- Reducing to shift product is $O(n)$.
- Need to compute two convolutions with polynomials of size n and m. Total run time is $O(n \log n)$ (here we assume $m \leq n$).
- Can reduce to $O(n \log m)$ as follows. Break text T into $O(n/m)$ *overlapping* substrings of length $2m$ each and compute matches of P with these substrings. Total time is $O(n \log m)$.

Exercise: work out the details of this improvement.

General Alphabet

If Σ is not binary replace each character $\alpha \in \Sigma$ by its binary representation. Need $s = \lceil \log |\Sigma| \rceil$ bits. Running time increases to $O(n \log m \log s)$.

Can remove dependence on s and obtain $O(n \log m)$ time where $m = |P|$ using more advanced ideas and/or randomization.

Trivia

FFT algorithm is used billions of times everyday: image/sound processing – jpeg, mp3, MRI scans, etc.

Even your brain is running FFT!

A fun video on FFT applications:
https://www.youtube.com/watch?v=aqa6vyGSdos