Multiplying polynomials quickly

**Definition**

polynomial \( p(x) \) of degree \( n \):
a function

\[
p(x) = \sum_{j=0}^{n} a_j x^j = a_0 + x(a_1 + x(a_2 + \ldots + x a_n)).
\]

\( x_0 \): \( p(x_0) \) can be computed in \( O(n) \) time.

“dual” (and equivalent) representation...

**Theorem**

For any set \( \{(x_0, y_0), (x_1, y_1), \ldots, (x_{n-1}, y_{n-1})\} \) of \( n \) point-value pairs such that all the \( x_k \) values are distinct, there is a unique polynomial \( p(x) \) of degree \( n - 1 \), such that \( y_k = p(x_k) \), for \( k = 0, \ldots, n - 1 \).

Polynomial via point-value

\( \{(x_0, y_0), (x_1, y_1), (x_2, y_2)\} \): polynomial through points:

\[
p(x) = y_0 \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_0-x_0)(x_0-x_1)(x_0-x_2)} + y_1 \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_1-x_0)(x_1-x_1)(x_1-x_2)} + y_2 \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_2-x_0)(x_2-x_1)(x_2-x_2)}
\]

\( i \)th is zero for \( x = x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{n-1} \), and is equal to \( y_i \) for \( x = x_i \).
Polynomials: regular vs. point-value pair representation

Just because.

1. Given $n$ point-value pairs. Can compute $p(x)$ in $O(n^2)$ time.
2. Point-value pairs representation: Multiply polynomials quickly!
3. $p, q$ polynomial of degree $n - 1$, both represented by $2n$ point-value pairs
   $\{(x_0, y_0), (x_1, y_1), \ldots, (x_{2n-1}, y_{2n-1})\}$ for $p(x)$, and
   $\{(x_0', y_0'), (x_1', y_1'), \ldots, (x_{2n-1}', y_{2n-1}')\}$ for $q(x)$.
4. $r(x) = p(x)q(x)$: product.
5. In point-value representation representation of $r(x)$ is
   $\{(x_0, r(x_0)), \ldots, (x_{2n-1}, r(x_{2n-1}))\}$
   $= \{(x_0, p(x_0)q(x_0)), \ldots, (x_{2n-1}, p(x_{2n-1})q(x_{2n-1}))\}$.

Which implies...

- $p(x)$ and $q(x)$: point-value pairs $\implies$ compute $r(x) = p(x)q(x)$ in linear time!
- $r(x)$ is in point-value representation. Bummer.
- $r(x)$ is in point-value representation. Bummer.
- Purpose: Translate quickly (i.e., $O(n \log n)$ time) from the standard $r$ to point-value pairs representation of polynomials.
- ...and back!
- $\implies$ computing product of two polynomials in $O(n \log n)$ time.
- Fast Fourier Transform is a way to do this.
- choosing the $x_i$ values carefully, and using divide and conquer.

Part I

Computing a polynomial quickly on $n$ values

Let’s just use some magic.

- Assume: polynomials have degree $n - 1$, where $n = 2^k$.
- ... pad polynomials with terms having zero coefficients.
- Magic set of numbers: $\Psi = \{x_1, \ldots, x_n\}$.
  Property: $|\text{SQ}(\Psi)| = n/2$, where $\text{SQ}(\Psi) = \{x^2 \mid x \in \Psi\}$.
- $|\text{square}(\Psi)| = |\Psi| / 2$.
- Easy to find such set...
- Magic: Have this property repeatedly...
  $\text{SQ}(\text{SQ}(\Psi))$ has $n/4$ distinct values.
  $\text{SQ}(\text{SQ}(\text{SQ}(\Psi)))$ has $n/8$ values.
  $\text{SQ}^i(\Psi)$ has $n/2^i$ distinct values.
- Oops: No such set of real numbers.
- NO SUCH SET.
Collapsible sets

Assume magic...

Let us for the time being ignore this technicality, and fly, for a moment, into the land of fantasy, and assume that we do have such a set of numbers, so that \(|\text{SQ}'(\Psi)| = n/2^i\) numbers, for \(i = 0, \ldots, k\). Let us call such a set of numbers collapsible.

**FFT: The dividing stage**

- \(p(x) = \sum_{i=0}^{n-1} a_i x^i\) as \(p(x) = u(x^2) + x \cdot v(x^2)\).
- \(\Psi\): collapsible set of size \(n\).
- \(p(\Psi)\): compute polynomial of degree \(n - 1\) on \(n\) values.

Decompose:

\[
\begin{align*}
u(y) &= \sum_{i=0}^{n/2-1} a_{2i} y^i \quad \text{and} \quad v(y) &= \sum_{i=0}^{n/2-1} a_{1+2i} y^i.
\end{align*}
\]

Need to compute \(u(x^2)\), for all \(x \in \Psi\).

Need to compute \(v(x^2)\), for all \(x \in \Psi\).

\(\text{SQ}(\Psi) = \{x^2 \mid x \in \Psi\}\).

\(\implies\) Need to compute \(u(\text{SQ}(\Psi)), v(\text{SQ}(\Psi))\).

\(u(\text{SQ}(\Psi)), v(\text{SQ}(\Psi))\): comp. poly. degree \(\frac{n}{2} - 1\) on \(\frac{n}{2}\) values.

**Breaking the input polynomial into two polynomials of half the degree**

- For a set \(\mathcal{X} = \{x_0, \ldots, x_n\}\) and polynomial \(p(x)\), let

\[
p(\mathcal{X}) = \left\langle (x_0, p(x_0)), \ldots, (x_n, p(x_n)) \right\rangle.
\]

- \(p(x) = \sum_{i=0}^{n-1} a_i x^i\) as \(p(x) = u(x^2) + x \cdot v(x^2)\), where

\[
\begin{align*}
u(y) &= \sum_{i=0}^{n/2-1} a_{2i} y^i \quad \text{and} \quad v(y) &= \sum_{i=0}^{n/2-1} a_{1+2i} y^i.
\end{align*}
\]

All even degree terms in \(u(\cdot)\), all odd degree terms in \(v(\cdot)\).

Maximum degree of \(u(y)\), \(v(y)\) is \(n/2\).

**FFT: The conquering stage**

- \(\Psi\): Collapsible set of size \(n\).
- \(p(x) = \sum_{i=0}^{n-1} a_i x^i\) as \(p(x) = u(x^2) + x \cdot v(x^2)\).

\(u(y) = \sum_{i=0}^{n/2-1} a_{2i} y^i\) and \(v(y) = \sum_{i=0}^{n/2-1} a_{1+2i} y^i\).

\(u(\text{SQ}(\Psi)), v(\text{SQ}(\Psi))\): Computed recursively.

Need to compute \(p(\Psi)\).

For \(x \in \Psi\): Compute \(p(x) = u(x^2) + x \cdot v(x^2)\).

Takes constant time per single element \(x \in \Psi\).

Takes \(O(n)\) time overall.
FFT algorithm

\texttt{FFTAlg}(p, X)

\begin{itemize}
\item \textbf{input:} \(p(x):\) A polynomial of degree \(n:\)
\[ p(x) = \sum_{i=0}^{n-1} a_i x^i \]
\item \textbf{output:} \(p(X):\) A collapsible set of \(n\) elements.
\end{itemize}

\begin{itemize}
\item \(u(y) = \sum_{i=0}^{n/2-1} a_2i y^i\)
\item \(v(y) = \sum_{i=0}^{n/2-1} a_{1+2i} y^i.\)
\item \(Y = \text{SQ}(X) = \{ x^2 \mid x \in X \}.\)
\item \(U = \text{FFTAlg}(u, Y)\) /* \(U = u(\textbf{Y})*/
\item \(V = \text{FFTAlg}(v, Y)\) /* \(V = v(\textbf{Y})*/
\end{itemize}

\texttt{Out} \leftarrow \emptyset

\textbf{for} \(x \in X\) \textbf{do}

\begin{itemize}
\item \(\texttt{Out} \leftarrow \texttt{Out} \cup \{(x, p(x))\}\)
\end{itemize}

\textbf{return} \texttt{Out}

Generating Collapsible Sets

- **How to generate collapsible sets?**
- **Trick:** Use complex numbers!

Complex numbers – a quick reminder

- **Complex number:** pair \((\alpha, \beta)\) of real numbers. Written as \(\tau = \alpha + \beta i\).
  - \(\alpha:\) \textit{real} part,
  - \(\beta:\) \textit{imaginary} part.
- \(i\) is the root of \(-1\).
- Geometrically: a point in the complex plane:

  \[
  \tau = \alpha + \beta i
  \]

  ![Complex Plane Diagram]

  \[
  \tau = r \cos \phi + ir \sin \phi = r (\cos \phi + i \sin \phi)
  \]
A useful formula: \( \cos \phi + i \sin \phi = e^{i \phi} \)

- By Taylor’s expansion:
  \[
  \sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots ,
  \]
  \[
  \cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots ,
  \]
  and \( e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots . \)
- Since \( i^2 = -1 \):
  \[
  e^{ix} = 1 + i \frac{x}{1!} - \frac{x^2}{2!} - i \frac{x^3}{3!} + \frac{x^4}{4!} + i \frac{x^5}{5!} - \frac{x^6}{6!} \cdots 
  \]
  \[
  = \cos x + i \sin x .
  \]

Roots of unity
The desire to avoid war?

For \( j = 0, \ldots, n - 1 \), we get the \( n \) distinct roots of unity.

\[
\gamma_j(n) = \cos((2\pi j)/n) + i \sin((2\pi j)/n) = \gamma^j .
\]

Let \( \mathcal{A}(n) = \{ \gamma_0(n), \ldots, \gamma_{n-1}(n) \} \).

- \(|\text{SQ}(\mathcal{A}(n))| \) has \( n/2 \) entries.
- \( \text{SQ}(\mathcal{A}(n)) = \mathcal{A}(n/2) \)
- \( n \) to be a power of 2, then \( \mathcal{A}(n) \) is the required collapsible set.
Theorem
Given polynomial \( p(x) \) of degree \( n \), where \( n \) is a power of two, then we can compute \( p(X) \) in \( O(n \log n) \) time, where \( X = \mathcal{A}(n) \) is the set of \( n \) different powers of the \( n \)th root of unity over the complex numbers.

Recovering the polynomial
Think about FFT as a matrix multiplication operator.
\[
p(x) = \sum_{i=0}^{n-1} a_i x^i.
\]
Evaluating \( p(\cdot) \) on \( \mathcal{A}(n) \):

\[
\begin{pmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{pmatrix}
= \begin{pmatrix}
1 & \gamma_0 & \gamma_0^2 & \gamma_0^3 & \cdots & \gamma_0^{n-1} \\
1 & \gamma_1 & \gamma_1^2 & \gamma_1^3 & \cdots & \gamma_1^{n-1} \\
1 & \gamma_2 & \gamma_2^2 & \gamma_2^3 & \cdots & \gamma_2^{n-1} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \gamma_{n-1} & \gamma_{n-1}^2 & \gamma_{n-1}^3 & \cdots & \gamma_{n-1}^{n-1}
\end{pmatrix}
\begin{pmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{pmatrix},
\]

where \( \gamma_j = \gamma_j(n) = (\gamma_1(n))^j \) is the \( j \)th power of the \( n \)th root of unity, and \( y_j = p(\gamma_j) \).

The Vandermonde matrix
Because every matrix needs a name

\( V \) is the Vandermonde matrix.
\( V^{-1} \): inverse matrix of \( V \)
Vandermonde matrix. And let multiply the above formula from the left. We get:

\[
\begin{pmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{pmatrix}
= V
\begin{pmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{pmatrix}
\implies
\begin{pmatrix}
a_0 \\
a_1 \\
a_2 \\
\vdots \\
a_{n-1}
\end{pmatrix}
= V^{-1}
\begin{pmatrix}
y_0 \\
y_1 \\
y_2 \\
\vdots \\
y_{n-1}
\end{pmatrix}.
\]
The inverse Vandermonde matrix
..for the rescue

1. Recover the polynomial \( p(x) \) from the point-value pairs
   \[ \{ (\gamma_0, p(\gamma_0)), (\gamma_1, p(\gamma_1)), \ldots, (\gamma_{n-1}, p(\gamma_{n-1})) \} \]
   by doing a single matrix multiplication of \( V^{-1} \) by the vector \([y_0, y_1, \ldots, y_{n-1}]\).
2. Multiplying a vector with \( n \) entries with \( n \times n \) matrix takes \( O(n^2) \) time.

   No benefit so far...

Proof
Consider the \((u, v)\) entry in the matrix \( C = V^{-1}V \). We have
\[
C_{u,v} = \sum_{j=0}^{n-1} \left( \beta_u \right)^j (\gamma_j)^v.
\]

As \( \gamma_j = (\gamma_1)^j \). Thus,
\[
C_{u,v} = \sum_{j=0}^{n-1} \left( \frac{\beta_u}{n} \right)^j (\gamma_1)^j = \sum_{j=0}^{n-1} \left( \frac{\beta_u \gamma_v}{n} \right)^j.
\]

Clearly, if \( u = v \) then
\[
C_{u,u} = \frac{1}{n} \sum_{j=0}^{n-1} (\beta_u \gamma_u)^j = \frac{1}{n} \sum_{j=0}^{n-1} (1)^j = \frac{n}{n} = 1.
\]

Proof continued...
If \( u \neq v \) then,
\[
\beta_u \gamma_v = (\gamma_u)^{-1} \gamma_v = (\gamma_1)^{-u} \gamma_1^v = (\gamma_1)^{v-u} = \gamma_{v-u}.
\]

And
\[
C_{u,v} = \frac{1}{n} \sum_{j=0}^{n-1} (\gamma_{v-u})^j = \frac{1}{n} \gamma_{v-u}^n - 1 = \frac{1}{n} \cdot \frac{1 - 1}{\gamma_{v-u} - 1} = 0,
\]

Proved that the matrix \( C \) have ones on the diagonal and zero everywhere else.

What is the inverse of the Vandermonde matrix

Vandermonde matrix is famous, beautiful and well known -- a celebrity matrix

Claim
\[
V^{-1} = \frac{1}{n} \begin{pmatrix}
1 & \beta_0 & \beta_0^2 & \ldots & \beta_0^{n-1} \\
1 & \beta_1 & \beta_1^2 & \ldots & \beta_1^{n-1} \\
1 & \beta_2 & \beta_2^2 & \ldots & \beta_2^{n-1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & \beta_{n-1} & \beta_{n-1}^2 & \ldots & \beta_{n-1}^{n-1}
\end{pmatrix},
\]

where \( \beta_j = (\gamma_j(n))^{-1} \).
Recap...

- **n** point-value pairs \(\{(\gamma_0, y_0), \ldots, (\gamma_{n-1}, y_{n-1})\}\) of a polynomial \(p(x) = \sum_{i=0}^{n-1} a_i x^i\) over the set of \(n\)th roots of unity.
- Can recover coefficients of the polynomial by multiplying the vector \([y_0, y_1, \ldots, y_n]\) by the matrix \(V^{-1}\). Namely,

\[
\begin{pmatrix}
    a_0 \\
    a_1 \\
    \vdots \\
    a_{n-1}
\end{pmatrix} = \frac{1}{n} \begin{pmatrix}
    1 & \beta_0 & \beta_0^2 & \beta_0^3 & \cdots & \beta_0^{n-1} \\
    1 & \beta_1 & \beta_1^2 & \beta_1^3 & \cdots & \beta_1^{n-1} \\
    \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
    1 & \beta_{n-1} & \beta_{n-1}^2 & \beta_{n-1}^3 & \cdots & \beta_{n-1}^{n-1}
\end{pmatrix}
\begin{pmatrix}
    y_0 \\
    y_1 \\
    \vdots \\
    y_{n-1}
\end{pmatrix}.
\]

- \(W(x) = \sum_{i=0}^{n-1} (y_i/n)x^i\): \(a_i = W(\beta_i)\).

Recovering continued...

- recover coefficients of \(p(\cdot)\)...
- ... compute \(W(\cdot)\) on \(n\) values: \(\beta_0, \ldots, \beta_{n-1}\).
- \(\{\beta_0, \ldots, \beta_{n-1}\} = \{\gamma_0, \ldots, \gamma_{n-1}\}\).
- Indeed \(\beta_i^n = (\gamma_i^{-1})^n = (\gamma_i^n)^{-1} = 1^{-1} = 1\).
- Apply the \texttt{FFTAlg} algorithm on \(W(x)\) to compute \(a_0, \ldots, a_{n-1}\).

Result

**Theorem**

Given \(n\) point-value pairs of a polynomial \(p(x)\) of degree \(n - 1\) over the set of \(n\) powers of the \(n\)th roots of unity, we can recover the polynomial \(p(x)\) in \(O(n \log n)\) time.

**Theorem**

Given two polynomials of degree \(n\), they can be multiplied in \(O(n \log n)\) time.
**Convolutions**

- Two vectors: $A = [a_0, a_1, \ldots, a_n]$ and $B = [b_0, \ldots, b_n]$.
- *dot product* $A \cdot B = \langle A, B \rangle = \sum_{i=0}^{n} a_i b_i$.
- $A_r$: shifting of $A$ by $n - r$ locations to the left
  - Padded with zeros: $a_j = 0$ for $j \not\in \{0, \ldots, n\}$.
- $A_r = [a_{n-r}, a_{n+1-r}, a_{n+2-r}, \ldots, a_{2n-r}]$
  - where $a_j = 0$ if $j \not\in \{0, \ldots, n\}$.
- Observation: $A_n = A$.

**Example of shifting**

*Example*

For $A = [3, 7, 9, 15]$, $n = 3$
$A_2 = [7, 9, 15, 0]$
$A_5 = [0, 0, 3, 7]$.

**Definition**

**Definition**

Let $c_i = A_i \cdot B = \sum_{j=n-i}^{2n-i} a_j b_{j-n+i}$, for $i = 0, \ldots, 2n$. The vector $[c_0, \ldots, c_{2n}]$ is the *convolution* of $A$ and $B$.

**Question**

How to compute the convolution of two vectors of length $n$?

**Convolution via multiplication polynomials**

- $p(x) = \sum_{i=0}^{n} \alpha_i x^i$, and $q(x) = \sum_{i=0}^{n} \beta_i x^i$.
- Coefficient of $x^i$ in $r(x) = p(x)q(x)$ is $d_i = \sum_{j=0}^{i} \alpha_j \beta_{i-j}$.
- Want to compute $c_i = A_i \cdot B = \sum_{j=n-i}^{2n-i} a_j b_{j-n+i}$.
- Set $\alpha_i = a_i$ and $\beta_i = b_{n-i-1}$.
Convolution by example

- Consider coefficient of $x^2$ in product of $p(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ and $q(x) = b_0 + b_1x + b_2x^2 + b_3x^3$.
- Sum of the entries on the anti diagonal:

<table>
<thead>
<tr>
<th></th>
<th>$b_0$</th>
<th>$a_1x$</th>
<th>$a_2b_0x^2$</th>
<th>$a_3x^3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$b_0$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$+b_1x$</td>
<td>$a_1b_1x^2$</td>
<td>$a_2b_0x^2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$+b_2x^2$</td>
<td>$a_0b_2x^2$</td>
<td>$a_1b_1x^2$</td>
<td>$a_3x^3$</td>
<td></td>
</tr>
<tr>
<td>$+b_3x^3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Entry in the $i$th row and $j$th column is $a_i b_j$.

Convolution

**Theorem**

Given two vectors $A = [a_0, a_1, \ldots, a_n]$, $B = [b_0, \ldots, b_n]$ one can compute their convolution in $O(n \log n)$ time.

**Proof.**

Let $p(x) = \sum_{i=0}^{n} a_{n-i}x^i$ and let $q(x) = \sum_{i=0}^{n} b_i x^i$. Compute $r(x) = p(x)q(x)$ in $O(n \log n)$ time using the convolution theorem. Let $c_0, \ldots, c_{2n}$ be the coefficients of $r(x)$. It is easy to verify, as described above, that $[c_0, \ldots, c_{2n}]$ is the convolution of $A$ and $B$. \qed