Assignment Project Exam Help

Algorithms

COMP3121/9101

https://powcoder.com

Aleks Ignjatović

Add WeChat powcoder

School of Computer Science and Engineering
University of New South Wales

5. THE FAST FOURIER TRANSFORM
(not examinable material)

Our strategy to multiply polynomials fast:

- Given two polynomials of degree at most n,

$$P_A(x) = A_n x^n + \ldots + A_0; \qquad P_B(x) = B_n x^n + \ldots + B_0$$

Assignment Project Exam Help

1. convert them into value representation at $2n+1$ distinct points $x_0, x_1, \ldots, x_{2n}$:

$$P_A(x) \leftrightarrow \{(x_0, P_A(x_0)), (x_1, P_A(x_1)), \ldots, (x_{2n}, P_A(x_{2n}))\}$$
$$P_B(x) \leftrightarrow \{(x_0, P_B(x_0)), (x_1, P_B(x_1)), \ldots, (x_{2n}, P_B(x_{2n}))\}$$

https://powcoder.com

2. multiply them point by point using $2n+1$ multiplications:

$$\Big\{(x_0, \underbrace{P_A(x_0)P_B(x_0)}_{P_C(x_0)}), (x_1, \underbrace{P_A(x_1)P_B(x_1)}_{P_C(x_1)}), \ldots, (x_{2n}, \underbrace{P_A(x_{2n})P_B(x_{2n})}_{P_C(x_{2n})})\Big\}$$

Add WeChat powcoder

3. Convert such value representation of $P_C(x)$ to its coefficient form

$$P_C(x) = C_{2n}x^{2n} + C_{2n-1}x^{2n-1} + \ldots + C_1 x + C_0;$$

- So, we need $2n+1$ values of $P_A(x_i)$ and $P_B(x_i)$, $0 \leq i \leq 2n$.
- If we use $2n+1$ integers which are the smallest by their absolute value, i.e.,

$$-n, -(n-1), \ldots, -1, 0, 1, \ldots, n-1, n$$

among the values which we need to compute is
$P_A(n) = A_0 + A_1 n + \ldots + A_{n-1} n^{n-1} + A_n n^n$

- We saw that the trouble is that, as the degree n of the polynomials $P_A(x)$ and $P_B(x)$ increases, the value of n^n increases very fast and causes rapid increase of the computational complexity of the algorithm for polynomial multiplication which we used in the generalised Karatsuba algorithm.
- **Key Question:** What values should we take for $x_0, \ldots, x_{2n}$ to avoid "explosion" of size when we evaluate x_i^n while computing
$P_A(x_i) = A_0 + A_1 x + \ldots + A_n x_i^n$?

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Complex numbers revisited

Complex numbers $z = a + ib$ can be represented using their *modulus* $|z| = \sqrt{a^2 + b^2}$ and their *argument*, $\arg(z)$, which is an angle taking values in $(-\pi, \pi]$ and satisfying:

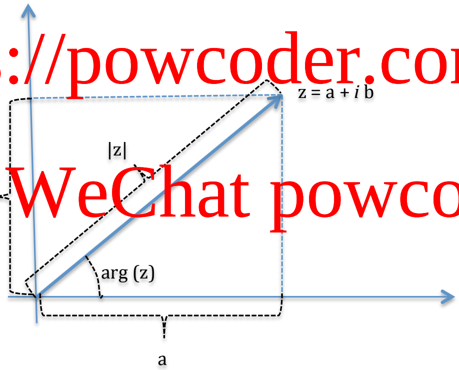
$$z = |z| e^{i \arg(z)} = |z|(\cos \arg(z) + i \sin \arg(z)),$$

see figure below

Assignment Project Exam Help

https://powcoder.com

Add WeChat powcoder



Complex numbers revisited

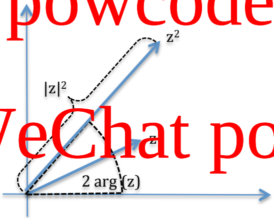
Recall that

$$z^n = \left(|z|e^{i\arg(z)}\right)^n = |z|^n e^{i\,n\arg(z)} = |z|^n\left(\cos(n\arg(z)) + i\sin(n\arg(z))\right),$$

Assignment Project Exam Help

see the figure.

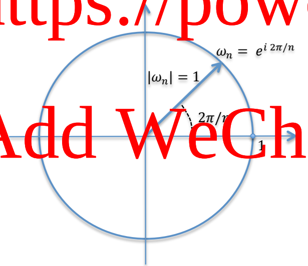
https://powcoder.com

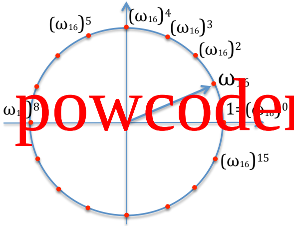


Add WeChat powcoder

- *Roots of unity of order* n are complex numbers which satisfy $z^n = 1$.
- If $z^n = |z|^n(\cos(n \arg(z)) + i \sin(n \arg(z))) = 1$ then $|z| = 1$ and $n \arg(z)$ is an integer multiple of 2π;
- Thus, $n \arg(z) = 2\pi\, k$, i.e., $\arg(z) = \dfrac{2\pi\, k}{n}$
- We denote $\omega_n = e^{i\, 2\pi/n}$; such ω_n is called a *primitive root of unity of order* n.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder





Roots of unity of order 16

- A root of unity ω of order n is *primitive* if all other roots of unity of the same order can be obtained as its powers ω^k.

Complex roots of unity

- For $\omega_n = e^{i\,2\pi/n}$ and for all k such that $0 \leq k \leq n-1$,

Assignment Project Exam Help

$$(\omega_n{}^k)^n = (\omega_n)^{n\,k} = (\omega_n{}^n)^k = 1^k = 1.$$

https://powcoder.com

Add WeChat powcoder

- Thus, $\omega_n^k = (\omega_n)^k$ is also a root of unity (and it can be shown that it is primitive just in case k is relatively prime with n).
- Since ω_n^k are roots of unity for $k = 0, 1, \ldots, n-1$ and there are at most n distinct roots of unity of order n (i.e., solutions to the equation $x^n - 1 = 0$) we conclude that every root of unity of order n must be of the form ω_n^k.
- For the product of any two roots of unity ω_n^k and ω_n^m of the same order we have $\omega_n^k \omega_n^m = \omega_n^{k+m}$.
- If $k + m \geq n$ then $k + m = n + l$ for $l = (k + m) \bmod n$ and we have $\omega_n^k \omega_n^m = \omega_n^{k+m} = \omega_n^{n+l} = \omega_n^n \omega_n^l = 1 \cdot \omega_n^l = \omega_n^l$ where $0 \leq l < n$.
- Thus, the product of any two roots of unity of the same order is just another root of unity of the same order.

Complex roots of unity

- So in the set of all roots of unity of order n, i.e., $\{1, \omega_n, \omega_n^2, \ldots, \omega_n^{n-1}\}$ we can multiply any two elements or raise an element to any power without going out of this set.

Assignment Project Exam Help

- Note that this is not true for addition, i.e., the sum of two roots of unity is NOT another root of unity!

- A most important property of the roots of unity is:

https://powcoder.com

The Cancelation Lemma: $\omega_{kn}^{km} = \omega_n^m$ for all integers k, m, n.

Proof:

Add WeChat powcoder

$$\omega_{kn}^{km} = (\omega_{kn})^{km} = (e^{i\frac{2\pi}{kn}})^{km} = e^{i\frac{2\pi km}{kn}} = e^{i\frac{2\pi m}{n}} = (e^{i\frac{2\pi}{n}})^m = \omega_n^m$$

- Thus, in particular, $(\omega_{2n}^k)^2 = \omega_{2n}^{2k} = (\omega_{2n}^2)^k = \omega_n^k$.

- So the squares of the roots of unity of order $2n$ are just the roots of unity of order n.

- Let $A = \langle A_0, A_1, \ldots, A_{n-1}\rangle$ be a sequence of n real or complex numbers.

Assignment Project Exam Help

- We can form the corresponding polynomial $P_A(x) = \sum_{j=0}^{n-1} A_j x^j$.
- We can evaluate it at all complex roots of unity of order n, i.e., we compute $P_A(\omega_n^k)$ for all $0 \leq k \leq n-1$.

https://powcoder.com

- The sequence of values $\langle P_A(1), P_A(\omega_n), P_A(\omega_n^2), \ldots, P_A(\omega_n^{n-1})\rangle$, is called **the Discrete Fourier Transform (DFT)** of the sequence $A = \langle A_0, A_1, \ldots, A_{n-1}\rangle$.

Add WeChat powcoder

- The value $P_A(\omega_n^k)$ is usually denoted by $\widehat{A}_k$ and the sequence of values $\langle P_A(1), P_A(\omega_n), P_A(\omega_n^2), \ldots, P_A(\omega_n^{n-1})\rangle$ is usually denoted by $\widehat{A} = \langle \widehat{A}_0, \widehat{A}_1, \ldots, \widehat{A}_{n-1}\rangle$.
- The DFT $\widehat{A}$ of a sequence A can be computed VERY FAST using a divide-and-conquer algorithm called the **Fast Fourier Transform**.

New way for fast multiplication of polynomials

- If we multiply a polynomial

$$P_A(x) = A_0 + \ldots + A_{n-1}x^{n-1}$$

Assignment Project Exam Help

of degree $n-1$ with a polynomial

$$P_B(x) = B_0 + \ldots + B_{m-1}x^{m-1}$$

https://powcoder.com

of degree $m-1$ we get a polynomial

$$C(x) = P_A(x)P_B(x) = C_0 + \ldots + C_{m+n-2}x^{m+n-2}$$

of degree $n-1+m-1 = m+n-2$ with $m+n-1$ coefficients.

Add WeChat powcoder

- To uniquely determine such a polynomial $C(x)$ of degree $m+n-2$ we need $m+n-1$ many values.

- Thus, we will evaluate both $P_A(x)$ and $P_B(x)$ at all the roots of unity of order $n+m-1$ (instead of at $-(n-1), \ldots, -1, 0, 1, \ldots, m-1$ as we would in Karatsuba's method!)

Assignment Project Exam Help

- Note that we defined the DFT of a sequence of length n as the values of the corresponding polynomial of degree $n-1$ at the n roots of unity of order n, i.e., ω_n^k $(0 \le k \le n-1)$.
- So the DFT of a sequence A is another sequence $\widehat{A}$ of exactly the same length; we do not have an operation which would evaluate a polynomial of degree $n-1$ at m roots of unity of order $m \neq n$.

https://powcoder.com

- For that reason, since we need $n+m-1$ values of $P_C(x) = P_A(x)P_B(x)$, we pad A with $m-1$ zeros at the end, $(A_0, A_1, \ldots, A_{n-1}, \underbrace{0, \ldots, 0}_{m-1})$ to make it of length $m+n-1$, and similarly we pad B with $n-1$ zeros at the end, $(B_0, B_1, \ldots, B_{m-1}, \underbrace{0, \ldots, 0}_{n-1})$ to also obtain a sequence of length $n+m-1$.

Add WeChat powcoder

- Note that this does not change the associated polynomials because the added higher powers have the corresponding coefficients equal to zero.

New way for fast multiplication of polynomials

- We can now compute the DFTs of the two (0 padded) sequences:

$$DFT(\langle A_0, A_1, \dots, A_{n-1}, \underbrace{0, \dots, 0}_{m-1}\rangle) = \langle \widehat{A}_0, \widehat{A}_1, \dots, \widehat{A}_{n+m-2}\rangle$$

Assignment Project Exam Help

and

$$DFT(\langle B_0, B_1, \dots, B_{m-1}, \underbrace{0, \dots, 0}_{n-1}\rangle) = \langle \widehat{B}_0, \widehat{B}_1, \dots, \widehat{B}_{n+m-2}\rangle$$

https://powcoder.com

- For each k we multiply the corresponding values $\widehat{A}_k = P_A(\omega_{n+m-1}^k)$ and $\widehat{B}_k = P_B(\omega_{n+m-1}^k)$, thus obtaining

Add WeChat powcoder

$$\widehat{C}_k = \widehat{A}_k \widehat{B}_k = P_A(\omega_{n+m-1}^k) P_B(\omega_{n+m-1}^k) = P_C(\omega_{n+m-1}^k)$$

- We then use the inverse transformation for DFT, called IDFT, to recover the coefficients $\langle C_0, C_1, \dots, C_{n+m-1}\rangle$ of the product polynomial $P_C(x)$ from the sequence $\langle \widehat{C}_0, \widehat{C}_1, \dots, \widehat{C}_{n+m-1}\rangle$ of its values $C_k = P_C(\omega_{n+m-1}^k)$ at the roots of unity of order $n + m - 1$.

$$P_A(x) = A_0 + \ldots + A_{n-1}x^{n-1} + 0 \cdot x^n + \ldots + 0 \cdot x^{n+m-2};$$

$$P_B(x) = B_0 + \ldots + B_{m-1}x^{m-1} + 0 \cdot x^m + \ldots + 0 \cdot x^{n+m-2}$$

Assignment Project Exam Help

$\Downarrow$ DFT $\qquad\qquad$ $\Downarrow$ DFT

$$\{P_A(1), P_A(\omega_{n+m-1}), \ldots, P_A(\omega_{n+m-1}^{n+m-2})\}; \quad \{P_B(1), P_B(\omega_{n+m-1}), \ldots, P_B(\omega_{n+m-1}^{n+m-2})\}$$

https://powcoder.com

$\Downarrow$ multiplication

$$\left\{ \underbrace{P_A(1)P_B(1)}_{P_C(1)}, \underbrace{P_A(\omega_{n+m-1})P_B(\omega_{n+m-1})}_{P_C(\omega_{n+m-1})}, \ldots, \underbrace{P_A(\omega_{n+m-1}^{n+m-2})P_B(\omega_{n+m-1}^{n+m-2})}_{P_C(\omega_{n+m-1}^{n+m-2})} \right\}$$

Add WeChat powcoder

$\Downarrow$ IDFT

$$P_C(x) = P_A(x) \cdot P_B(x) = \sum_{j=0}^{n+m-2} C_j x^j = \sum_{j=0}^{n+m-2} \underbrace{\left(\sum_{i=0}^{j} A_i B_{j-i} \right)}_{C_j} x^j$$

The Fast Fourier Transform (FFT)

- Crucial fact: the values $P_A(\omega_n^k)$ for all k such that $0 \leq k < n$ can be computed in $\mathbf{O(n \log n)}$ time!

Assignment Project Exam Help

- Note that a direct evaluation of a polynomial of degree $n-1$ at n roots of unity of order n would take n^2 many multiplications, even if we precompute all powers $\omega_n^{k\,m}$, because we have to perform multiplications $A_m \cdot (\omega_n^k)^m$ for all $0 \leq m \leq n-1$ and all $0 \leq k \leq n-1$.

https://powcoder.com

  - We can assume that n is a power of 2 - otherwise we can pad $P_A(x)$ with zero coefficients until its number of coefficients becomes equal to the nearest power of 2.

Add WeChat powcoder

  - <u>Exercise:</u> Show that for every n which is not a power of two the smallest power of 2 larger or equal to n is smaller than $2n$.
  - *Hint:* consider n in binary. How many bits does the nearest power of two have?

The Fast Fourier Transform (FFT)

- **Problem:** Given a sequence $A = \langle A_0, A_1, \ldots, A_n \rangle$ compute its DFT.
- This amounts to finding values of $P_A(x)$ for all $x = \omega_n^k$, $0 \leq k \leq n-1$.
- **The main idea of the FFT algorithm:** divide-and-conquer by splitting the polynomial $P_A(x)$ into the even powers and the odd powers:

Assignment Project Exam Help

$$
\begin{aligned}
P_A(x) &= (A_0 + A_2x^2 + A_4x^4 + \ldots + A_{n-2}x^{n-2}) + (A_1x + A_3x^3 + \ldots + A_{n-1}x^{n-1}) \\
&= A_0 + A_2x^2 + A_4(x^2)^2 + \ldots + A_{n-2}(x^2)^{n/2-1} \\
&\quad + x\left(A_1 + A_3x^2 + A_5(x^2)^2 + \ldots + A_{n-1}(x^2)^{n/2-1}\right)
\end{aligned}
$$

https://powcoder.com

- Let us define $A^{[0]} = \langle A_0, A_2, A_4, \ldots A_{n-2} \rangle$ and $A^{[1]} = \langle A_1, A_3, A_5, \ldots A_{n-1} \rangle$; then

Add WeChat powcoder

$$P_{A^{[0]}}(y) = A_0 + A_2y + A_4y^2 + \ldots + A_{n-2}y^{n/2-1}$$

$$P_{A^{[1]}}(y) = A_1 + A_3y + A_5y^2 + \ldots + A_{n-1}y^{n/2-1}$$

$$P_A(x) = P_{A^{[0]}}(x^2) + x\,P_{A^{[1]}}(x^2)$$

- Note that the number of coefficients of the polynomials $P_{A^{[0]}}(y)$ and $P_{A^{[1]}}(y)$ is $n/2$ each, while the number of coefficients of the polynomial $P_A(x)$ is n.

The Fast Fourier Transform (FFT)

- **Problem of size n:**
  Evaluate a polynomial with n coefficients at n many roots of unity.

- **Problem of size $n/2$:**
  Evaluate a polynomial with $n/2$ coefficients at $n/2$ many roots of unity.

Assignment Project Exam Help

- We reduced evaluation of our polynomial $P_A(x)$ with n coefficients at inputs $x = \omega_n^0,\ x = \omega_n^1,\ x = \omega_n^2, \ldots, x = \omega_n^{n-1}$ to evaluation of two polynomials $P_{A^{[0]}}(y)$ and $P_{A^{[1]}}(y)$ each with $n/2$ coefficients, at points $y = x^2$ for the same values of inputs x.

https://powcoder.com

- However, as x ranges through values $\{\omega_n^0, \omega_n^1, \omega_n^2, \ldots, \omega_n^{n-1}\}$, the value of $y = x^2$ ranges through $\{\omega_{n/2}^0, \omega_{n/2}^1, \omega_{n/2}^2, \ldots, \omega_{n/2}^{n-1}\}$, and **there are only $n/2$ distinct such values.**

Add WeChat powcoder

- Once we get these $n/2$ values of $A^{[0]}(x^2)$ and $A^{[1]}(x^2)$ we need n additional multiplications with numbers ω_n^k to obtain the values of

$$
\begin{aligned}
P_A(\omega_n^k) &= P_{A^{[0]}}((\omega_n^k)^2) + \omega_n^k \cdot P_{A^{[1]}}((\omega_n^k)^2) \\
&= P_{A^{[0]}}(\omega_{n/2}^k) + \omega_n^k \cdot P_{A^{[1]}}(\omega_{n/2}^k).
\end{aligned}
$$

- Thus, we have reduced a problem of size n to two such problems of size $n/2$, plus a linear overhead.

- Note that by the Cancelation Lemma $\omega_n^{n/2} = \omega_{2n/2}^{n/2} = \omega_2 = -1$.
- Thus,

$$\omega_n^{k+n/2} = \omega_n^{n/2}\omega_n^k = \omega_2\omega_n^k = -\omega_n^k;$$

Assignment Project Exam Help

- We can now simplify evaluation of

$$P_A(\omega_n^k) = P_{A^{[0]}}(\omega_{n/2}^k) + \omega_n^k P_{A^{[1]}}(\omega_{n/2}^k)$$

https://powcoder.com

for $n/2 \leq k < n$ as follows:
let $k = n/2 + m$ where $0 \leq m < n/2$; then

Add WeChat powcoder

$$\begin{aligned}
P_A(\omega_n^{n/2+m}) &= P_{A^{[0]}}(\omega_{n/2}^{n/2+m}) + \omega_n^{n/2+m} P_{A^{[1]}}(\omega_{n/2}^{n/2+m}) \\
&= P_{A^{[0]}}(\omega_{n/2}^{n/2}\omega_{n/2}^m) + \omega_n^{n/2}\omega_n^m P_{A^{[1]}}(\omega_{n/2}^{n/2}\omega_{n/2}^m) \\
&= P_{A^{[0]}}(\omega_{n/2}^m) - \omega_n^m P_{A^{[1]}}(\omega_{n/2}^m)
\end{aligned}$$

- Compare this with $P_A(\omega_n^m) = P_{A^{[0]}}(\omega_{n/2}^m) + \omega_n^m P_{A^{[1]}}(\omega_{n/2}^m)$ for $0 \leq m < n/2$.

The Fast Fourier Transform (FFT) - a simplification

- So we can replace evaluations of

Assignment Project Exam Help

$$P_A(\omega_n^k) = P_{A^{[0]}}(\omega_{n/2}^k) + \omega_n^k P_{A^{[1]}}(\omega_{n/2}^k)$$

for $k = 0$ to $k = n - 1$

https://powcoder.com

with such evaluations only for $k = 0$ to $k = n/2 - 1$

and just let for $k = 0$ to $k = n/2 - 1$

Add WeChat powcoder

$$P_A(\omega_n^k) = P_{A^{[0]}}(\omega_{n/2}^k) + \omega_n^k P_{A^{[1]}}(\omega_{n/2}^k)$$
$$P_A(\omega_n^{n/2+k}) = P_{A^{[0]}}(\omega_{n/2}^k) - \omega_n^k P_{A^{[1]}}(\omega_{n/2}^k)$$

- We can now write a pseudo-code for our FFT algorithm:

function FFT(A)
  $n \leftarrow \text{length}[A]$
  if $n = 1$ **then return** A
  else
    $A^{[0]} \leftarrow (A_0, A_2, \dots A_{n-2})$;
    $A^{[1]} \leftarrow (A_1, A_3, \dots A_{n-1})$;
    $y^{[0]} \leftarrow FFT(A^{[0]})$;
    $y^{[1]} \leftarrow FFT(A^{[1]})$;
    $\omega_n \leftarrow e^{i\frac{2\pi}{n}}$;
    $\omega \leftarrow 1$;   % a variable to hold powers of ω_n
    for $k = 0$ to $k = n/2 - 1$ **do**:   % $\underbrace{P_A(\omega_n^k)}_{y_k} = \underbrace{P_{A^{[0]}}(\omega_{n/2}^k)}_{y_k^{[0]}} + \omega_n^k \underbrace{P_{A^{[1]}}(\omega_{n/2}^k)}_{y_k^{[1]}}$
      $y_k \leftarrow y_k^{[0]} + \omega \cdot y_k^{[1]}$;
      $y_{n/2+k} \leftarrow y_k^{[0]} - \omega \cdot y_k^{[1]}$;   % $\underbrace{P_A(\omega_n^{n/2+k})}_{y_{n/2+k}} = \underbrace{P_{A^{[0]}}(\omega_{n/2}^k)}_{y_k^{[0]}} - \omega_n^k \underbrace{P_{A^{[1]}}(\omega_{n/2}^k)}_{y_k^{[1]}}$
      $\omega \leftarrow \omega \cdot \omega_n$;
    end for
    return y
  end if
end function

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

How fast is the Fast Fourier Transform?

- We have recursively reduced evaluation of a polynomial $P_A(x)$ with n coefficients at n roots of unity of order n to evaluations of two polynomials $P_{A^{[0]}}(y)$ and $P_{A^{[1]}}(y)$, each with $n/2$ coefficients, at $n/2$ many roots of unity of order $n/2$.

Assignment Project Exam Help

- Once we get these $n/2$ values of $P_{A^{[0]}}(y)$ and $P_{A^{[1]}}(y)$ we need $n/2$ additional multiplications to obtain the values of

$$\underbrace{P_A(\omega_n^k)}_{y_k} = \underbrace{P_{A^{[0]}}(\omega_{n/2}^k)}_{y_k^{[0]}} + \omega_n^k \underbrace{P_{A^{[1]}}(\omega_{n/2}^k)}_{y_k^{[1]}} \tag{1}$$

https://powcoder.com

and

$$\underbrace{P_A(\omega_n^{n/2+k})}_{y_{n/2+k}} = \underbrace{A^{[0]}(\omega_{n/2}^k)}_{y_k^{[0]}} - \omega_n^k \underbrace{A^{[1]}(\omega_{n/2}^k)}_{y_k^{[1]}} \tag{2}$$

Add WeChat powcoder

for all $0 \leq k < n/2$.

- Thus, we have reduced a problem of size n to two such problems of size $n/2$, plus a linear overhead.
- Consequently, our algorithm's run time satisfies the recurrence

$$T(n) = 2\,T\,(n/2) + c\,n$$

- The Master Theorem gives $T(n) = \Theta(n \log n)$.

Matrix representation of polynomial evaluation

- Evaluation of a polynomial $P_A(x) = A_0 + A_1 x + \ldots + A_{n-1}x^{n-1}$ at roots of unity ω_n^k of order n can be represented in the matrix form as follows:

Assignment Project Exam Help

$$\begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_n & \omega_n^2 & \ldots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \ldots & \omega_n^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \ldots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{n-1} \end{pmatrix} = \begin{pmatrix} P_A(1) \\ P_A(\omega_n) \\ P_A(\omega_n^2) \\ \vdots \\ P_A(\omega_n^{n-1}) \end{pmatrix} = \begin{pmatrix} \widehat{A}_0 \\ \widehat{A}_1 \\ \widehat{A}_2 \\ \vdots \\ \widehat{A}_{n-1} \end{pmatrix}$$

https://powcoder.com

- The FFT is just a method of replacing this matrix-vector multiplication taking n^2 many multiplications with an $n \log n$ procedure.
- From $P_A(1) = P_A(\omega_n^0), P_A(\omega_n), P_A(\omega_n^2), \ldots, P_A(\omega_n^{n-1})$, we get the coefficients from

Add WeChat powcoder

$$\begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \omega_n & \omega_n^2 & \ldots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \ldots & \omega_n^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \ldots & \omega_n^{(n-1)(n-1)} \end{pmatrix}^{-1} \begin{pmatrix} \widehat{A}_0 \\ \widehat{A}_1 \\ \widehat{A}_2 \\ \vdots \\ \widehat{A}_{n-1} \end{pmatrix} \tag{3}$$

- To obtain the inverse of the above matrix, all we have to do is just change the signs of the exponents and divide everything by n:

Assignment Project Exam Help

$$
\begin{pmatrix}
1 & 1 & 1 & \dots & 1 \\
1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\
1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2\cdot(n-1)} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)}
\end{pmatrix}^{-1} =
$$

https://powcoder.com

$$
\frac{1}{n}
\begin{pmatrix}
1 & 1 & 1 & \dots & 1 \\
1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\
1 & \omega_n^{-2} & \omega_n^{-2\cdot 2} & \dots & \omega_n^{-2\cdot(n-1)} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)}
\end{pmatrix}
$$

Add WeChat powcoder

To see this, note that if we compute the product

$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-2\cdot 2} & \dots & \omega_n^{-2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{pmatrix}$$

the (i,j) entry in the product matrix is equal to a product of i^{th} row and j^{th} column:

$$\begin{pmatrix} 1 & \omega_n^i & \omega_n^{2\cdot i} & \dots & \omega_n^{i(n-1)} \end{pmatrix} \begin{pmatrix} 1 \\ \omega_n^{-j} \\ \omega_n^{-2j} \\ \vdots \\ \omega_n^{-(n-1)j} \end{pmatrix} = \sum_{k=0}^{n-1} \omega_n^{ik}\omega_n^{-jk} = \sum_{k=0}^{n-1} \omega_n^{(i-j)k}$$

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

We now have two possibilities:

1. $i = j$: then
$$\sum_{k=0}^{n-1} \omega_n^{(i-j)k} = \sum_{k=0}^{n-1} \omega_n^0 = \sum_{k=0}^{n-1} 1 = n;$$

Assignment Project Exam Help

2. $i \neq j$: then $\sum_{k=0}^{n-1} \omega_n^{(i-j)k}$ represents a sum of a geometric progression with the ratio ω_n^{i-j} and thus
$$\sum_{k=0}^{n-1} \omega_n^{(i-j)k} = \frac{1-\omega_n^{(i-j)n}}{1-\omega_n^{i-j}} = \frac{1-(\omega_n^n)^{i-j}}{1-\omega_n^{i-j}} = \frac{1-1}{1-\omega_n^{i-j}} = 0$$

https://powcoder.com

So,

Add WeChat powcoder

$$\begin{pmatrix} 1 & \omega_n^i & \omega_n^{2\cdot i} & \dots & \omega_n^{i\cdot(n-1)} \end{pmatrix} \begin{pmatrix} 1 \\ \omega_n^{-j} \\ \omega_n^{-2j} \\ \vdots \\ \omega_n^{-(n-1)j} \end{pmatrix} = \sum_{k=0}^{n-1} \omega_n^{(i-j)k} = \begin{cases} n & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases} \tag{4}$$

So we get:

$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-2\cdot 2} & \dots & \omega_n^{-2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{pmatrix}$$

$$= \begin{pmatrix} n & 0 & 0 & \dots & 0 \\ 0 & n & 0 & \dots & 0 \\ 0 & 0 & n & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & n \end{pmatrix} = n \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}$$

Assignment Project Exam Help

https://powcoder.com

i.e.,

Add WeChat powcoder

$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix}^{-1} = \frac{1}{n} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-2\cdot 2} & \dots & \omega_n^{-2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{pmatrix}$$

- We now have

$$\begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n & \omega_n^2 & \dots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^{2\cdot 2} & \dots & \omega_n^{2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \dots & \omega_n^{(n-1)(n-1)} \end{pmatrix}^{-1} \begin{pmatrix} P_A(1) \\ P_A(\omega_n) \\ P_A(\omega_n^2) \\ \vdots \\ P_A(\omega_n^{n-1}) \end{pmatrix} =$$

$$= \frac{1}{n} \begin{pmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega_n^{-1} & \omega_n^{-2} & \dots & \omega_n^{-(n-1)} \\ 1 & \omega_n^{-2} & \omega_n^{-2\cdot 2} & \dots & \omega_n^{-2\cdot(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \omega_n^{-(n-1)} & \omega_n^{-2(n-1)} & \dots & \omega_n^{-(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} P_A(1) \\ P_A(\omega_n) \\ P_A(\omega_n^2) \\ \vdots \\ P_A(\omega_n^{n-1}) \end{pmatrix}$$

- This means that to covert from the values

$$\langle P_A(1), P_A(\omega_n), P_A(\omega_n^2), \dots, P_A(\omega_n^{n-1}) \rangle$$

which we denoted by $\langle \widehat{A}_0, \widehat{A}_1, \widehat{A}_2, \dots, \widehat{A}_{n-1} \rangle$ back to the coefficient form

$$P_A(x) = A_0 + A_1 x + A_2 x^2 + A_{n-1} x^{n-1}$$

we can use **the same** FFT algorithm with the only change that:

1. the root of unity ω_n is replaced by $\overline{\omega_n} = e^{-i\frac{2\pi}{n}}$,
2. the resulting output values are divided by n.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

function IFFT$^*(\widehat{A})$
  $n \leftarrow \text{length}(\widehat{A})$
  if $n = 1$ **then return** $\widehat{A}$
  else
    $\widehat{A}^{[0]} \leftarrow (\widehat{A}_0, \widehat{A}_2, \dots \widehat{A}_{n-2})$;
    $\widehat{A}^{[1]} \leftarrow (\widehat{A}_1, \widehat{A}_3, \dots \widehat{A}_{n-1})$;
    $y^{[0]} \leftarrow IFFT^*(\widehat{A}^{[0]})$;
    $y^{[1]} \leftarrow IFFT^*(\widehat{A}^{[1]})$;
    $\omega_n \leftarrow e^{-i\frac{2\pi}{n}}$; $\Leftarrow$ different from FFT
    $\omega \leftarrow 1$;
    for $k = 0$ to $k = n/2 - 1$ **do**;
      $y_k \leftarrow y_k^{[0]} + \omega \cdot y_k^{[1]}$;
      $y_{n/2+k} \leftarrow y_k^{[0]} - \omega \cdot y_k^{[1]}$;
      $\omega \leftarrow \omega \cdot \omega_n$;
    end for
    return y;
  end if
end function

function IFFT$(\widehat{A})$ $\Leftarrow$ different from FFT
  return $IFFT^*(\widehat{A})/\text{length}(\widehat{A})$
end function

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

Important note:

Computer science books take the forward DFT operation to be the evaluation of the corresponding polynomial at all roots of unity $\omega_n^k = \cos\frac{2\pi k}{n} + i\sin\frac{2\pi k}{n}$ and the InverseDFT to be the evaluation of the polynomial at the complex conjugates of the roots of unity, i.e., at $\omega_n^{-k} = \cos\frac{2\pi k}{n} - i\sin\frac{2\pi k}{n}$.

Assignment Project Exam Help

However, Electrical engineering books do it just opposite, the direct DFT evaluates the polynomial at ω_n^{-k} and the InverseDFT at ω_n^k!

While for the purpose of multiplying polynomials both choices are equally good, the choice made by the electrical engineers is much better for all other purposes. We will explain this in the Advanced Algorithms 4121 when we do the JPEG.

https://powcoder.com

We did here only multiplication of polynomials, and did not apply it to multiplication of large integers. This is possible to do but one has to be careful because roots of unity are represented by floating point numbers so you have to show that if you do FFT with sufficient precision you can round off the results and obtain correct integer values, but all of this is tricky.

Add WeChat powcoder

Earlier results along this line produced algorithms for multiplication of large integers which operate in time $n \log n \log(\log n)$ but very recently David Harvey of the School of Mathematics at UNSW came up with an algorithm to multiply large integers which runs in time $n \log n$.

Back to fast multiplication of polynomials

$P_A(x) = A_0 + A_1x + \ldots + A_{n-1}x^{n-1}$ $\qquad$ $P_B(x) = B_0 + B_1x + \ldots + B_{n-1}x^{n-1}$

$\Downarrow$ DFT $\quad O(n \log n)$ $\qquad$ $\Downarrow$ DFT $\quad O(n \log n)$

Assignment Project Exam Help

$\{P_A(1), P_A(\omega_{2n-1}), P_A(\omega_{2n-1}^2), \ldots, P_A(\omega_{2n-1}^{2n-2})\}$; $\{P_B(1), P_B(\omega_{2n-1}), P_B(\omega_{2n-1}^2), \ldots, P_B(\omega_{2n-1}^{2n-2})\}$

$\Downarrow$ multiplication $\quad O(n)$

https://powcoder.com

$\{P_A(1)P_B(1), \quad P_A(\omega_{2n-1})P_B(\omega_{2n-1}), \ldots, P_A(\omega_{2n-1}^{2n-2})P_B(\omega_{2n-1}^{2n-2})\}$

$\Downarrow$ IDFT $\quad O(n \log n)$

Add WeChat powcoder

$$P_C(x) = \sum_{j=0}^{2n-2} \underbrace{\left(\sum_{i=0}^{j} A_i B_{j-i}\right)}_{C_j} x^j = \sum_{j=0}^{2n-2} C_j x^j = P_A(x) \cdot P_B(x)$$

Thus, the product $P_C(x) = P_A(x)\,P_B(x)$ of two polynomials $P_A(x)$ and $P_B(x)$ can be computed in time $O(n \log n)$.

Computing the convolution $C = A * B$

$$A = \langle A_0, A_1, \ldots, A_{n-1} \rangle \qquad\qquad B = \langle B_0, B_1, \ldots, B_{n-1} \rangle$$

$$\Downarrow \quad O(n) \qquad\qquad \Downarrow \quad O(n)$$

$$P_A(x) = A_0 + A_1 x + \ldots + A_{n-1} x^{n-1} \qquad P_B(x) = B_0 + B_1 x + \ldots + B_{n-1} x^{n-1}$$

Assignment Project Exam Help

$$\Downarrow \text{ DFT } \quad O(n \log n) \qquad\qquad \Downarrow \text{ DFT } \quad O(n \log n)$$

$$\{P_A(1), P_A(\omega_{2n-1}), P_A(\omega_{2n-1}^2), \ldots, P_A(\omega_{2n-1}^{2n-2})\}; \quad \{P_B(1), P_B(\omega_{2n-1}), P_B(\omega_{2n-1}^2), \ldots, P_B(\omega_{2n-1}^{2n-2})\}$$

$$\Downarrow \text{ multiplication } \quad O(n)$$

$$\{P_A(1)P_B(1), \; P_A(\omega_{2n-1})P_B(\omega_{2n-1}), \; \ldots, P_A(\omega_{2n-1}^{2n-2})P_B(\omega_{2n-1}^{2n-2})\}$$

https://powcoder.com

$$\Downarrow \text{ IDFT } \quad O(n \log n)$$

$$P_C(x) = \sum_{j=0}^{2n-2} \underbrace{\left(\sum_{i=0}^{j} A_i B_{j-i} \right)}_{C_j} x^j$$

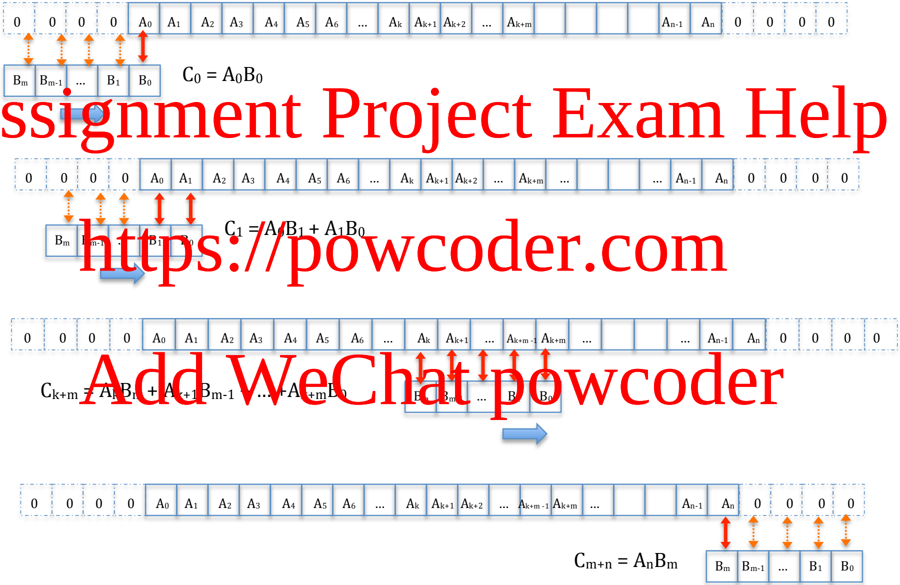
Add WeChat powcoder

$$\Downarrow$$

$$C = \left\langle \sum_{i=0}^{j} A_i B_{j-i} \right\rangle_{j=0}^{j=2n-2}$$

Convolution $C = A * B$ of sequences A and B is computed in time $O(n \log n)$.

Visualizing Convolution $C = A * B$

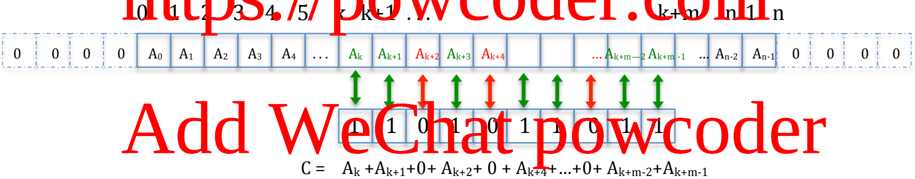


Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder

An Exercise

- Assume you are given a map of a straight sea shore of length $100n$ meters as a sequence on $100n$ numbers such that A_i is the number of fish between i^{th} meter of the shore and $(i+1)^{th}$ meter, $0 \leq i \leq 100n - 1$. You also have a net of length n meters but unfortunately it has holes in it. Such a net is described as a sequence N of n ones and zeros, where 0's denote where the holes are. If you throw such a net starting at meter k and ending at meter $k + n$, then you will catch only the fish in one meter stretches of the shore where the corresponding bit of the net is 1; see the figure.

Assignment Project Exam Help
https://powcoder.com
Add WeChat powcoder



Find the spot where you should place the left end of your net in order to catch the largest possible number of fish using an algorithm which runs in time $O(n \log n)$.
Hint: Let N' be the net sequence N in the reverse order; Compute $A * B'$ and look for the peak of that sequence.

Assignment Project Exam Help

- On a circular highway there are n petrol stations, unevenly spaced, each containing a different quantity of petrol. It is known that the total quantity of petrol on all stations is enough to go around the highway once, and that the tank of your car can hold enough fuel to make a trip around the highway. Prove that there always exists a station among all of the stations on the highway, such that if you take it as a starting point and take the fuel from that station, you can continue to make a complete round trip around the highway, never emptying your tank before reaching the next station to refuel.

https://powcoder.com

Add WeChat powcoder