Polynomial Multiplication : FFT

$$A(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_{n-1} x^{n-1}$$

$$= \sum_{k=0}^{n-1} a_k x^k$$

$$= \langle a_0\ a_1\ a_2 \dots a_{n-1} \rangle$$

Operations :—

① **Evaluation** : $A(x)$ at x_0

If we have computed x^k then we can compute x^{k+1} with constant time operation.

So $A(x_0)$ can be done in $O(n)$ time

Horner's Rule :—

$$A(x) = a_0 + x(a_1 + x(a_2 + \dots x(a_{n-1})$$

② **Addition** : $A(x)$ & $B(x)$

$C(x) = A(x) + B(x)$ $\forall x$

$O(n)$

③ **Multiplication** : $A(x)$ & $B(x)$

$C(x) = A(x) \cdot B(x)$

$O(n^2)$

By FFT we can do polynomial multiplication in $O(n \log n)$

④ Convolution of vectors A and B (reverse (B))

Inner product of all possible shifts

* Representation of polynomial :—

Ⓐ Coeff. vector

Ⓑ roots

Ⓒ Samples (x_k, y_k) $\forall$ distinct k's

Algs	Ⓐ Coeffs	Ⓑ Roots	Ⓒ Samples
① Eval	$O(n)$	$O(n)$	$O(n^2)$
② Add	$O(n)$	∞	$O(n)$
③ Mul	$O(n^2)$	$O(n)$	$O(n)$

$n \log n$

Discrete Fourier Transform

Matrix View :—

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \vdots & & & & \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

$\hookrightarrow V$

$V_{jk} = x_j^k$

V is called Vandermonde matrix

— Coeff $\Rightarrow$ samples $= V \cdot A$ — $O(n^2)$

— Samples $\rightarrow$ coeffs $= V^{-1} A$

- By Gaussian Elimination $O(n^3)$
- By $V^{-1} A$ — $O(n^2)$

Divide and Conquer Algo :—

Goal: To compute $A(x)$ $\forall x \in X$

① Divide into even and odd coefficients

$$A_{even}(x) = \sum_{k=0}^{n/2-1} a_{2k} x^k$$

$$= \langle a_0\ a_2\ a_4 \dots \rangle$$

$$A_{odd}(x) = \langle a_1\ a_3\ a_5 \dots \rangle$$

$$= \sum_{k=0}^{n/2-1} a_{2k+1} x^k$$

② Conquer: Recursively compute

$A_{even}(y)$ & $A_{odd}(y)$

for $y \in X^2 = \{x^2 \mid x \in X\}$

③ Combine $A(x) = A_{even}(x^2)$

for $x \in X$ $+$ $x \cdot A_{odd}(x^2)$

$T(n, |X|) = 2 \cdot T(n/2, |X|) + O(n+|X|)$

Total n's in last level $= 2^{\log n} = n$

Note that :-
After dividing matrix b/w even and odd submatrices the total elements got reduced by n/2 but $|X|$ — remained the same. That's why it's n in every level of the tree.

∴ when we convert our set X to X^2 we want our X to get smaller

Now :-
If $|X|$ is just one it can be $\{1\}$.

If $|X|$ is 2 then we need 2 values in X but when we square them we get 1 value.

These values can be $\{-1, 1\}$!!

$-1, -1$ are square roots of 1

So if I take square roots and I square them then it collapses by factors of 2.

#Collapsing – Set x if $|X^2| = |X|/2$ be recursively x^2 is collapsing

or $|X| = 1$

So,

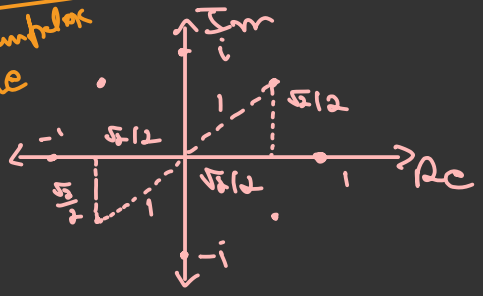
$|X| = 1$: $X = \{1\}$

$|X| = 2$: $X = \{-1, 1\}$

$|X| = 4$: $X = \{i, -i, -1, 1\}$

$|X| = 8$: $X = \{\pm\frac{\sqrt{2}}{2}(1+i), \pm\frac{\sqrt{2}}{2}(1-i), i, -i, -1, 1\}$

Geometrically on complex plane



we are get a unit's circle and the points are called with roots of unity.

The points on plane — $(\cos\theta, \sin\theta)$

Geometrical representation :-

$\cos\theta + i\sin\theta$

for $\theta = 0, \tau/n, 2\tau/n, \ldots, \frac{(n-1)\tau}{n}$

$\tau(\tau a\theta) = 2\pi$

Also as for Euler's formula :-

$\cos\theta + i\sin\theta = e^{i\theta}$

Lets see how the squares are coming :-

$(e^{i\theta})^2 = e^{i(2\theta)} = e^{i(2\theta \bmod 2\pi)}$

$e^{i 12\pi/n} = e^{i(2k)\pi/n}$ $\quad e^{i\pi} = 1$

∴ If n is a power of 2 then n^{th} roots of unity will be collapsing.

∴ $x^{12} = e^{ik\pi/n}$

Fast Fourier Transform (FFT)

= Divide & Conquer Algo for DFT

Discrete Fourier Transform:

= V.A for $x_k = e^{ik\pi/n}$

$V_{jk} = x_j^{12} = e^{ijk\pi/n}$

Fast Polynomial Multiplication :-

$A^* = FFT(A)$

$B^* = FFT(B)$

$C^*_{12} = A^*_{12} \cdot B^*_{12} \quad \forall k$

$C = Invest\ FFT(C^*)$

At this stage we know how to calculate V.A. We now have to calculate $\bar{V}.A$

Claim: $= V^{-1} = \bar{V}/n$, $\overline{a+ib} = a-ib$

∴ For Inverse

$x'_k = e^{-ik\pi/n}$ $\hookrightarrow V' = nV^{-1}$

$nV^{-1}.A^* = nA$

Proving our claim:-

claim : $V^{-1} = \bar{V}/n$

$P = V.\bar{V} = nI$

$P_{jk} = (\text{row } j \text{ of } V).(\text{col } k \text{ of } \bar{V})$

$= \sum_{m=0}^{n-1} e^{i\pi jm/n} \cdot e^{-i\pi mk/n}$

$= \sum_{m=0}^{n-1} e^{i\pi \frac{m}{n}(j-k)}$

$= n$ if $j = k$

If $j \neq k$:

$\sum_{m=0}^{n-1} \left(e^{i\pi \frac{j-12}{n}}\right)^m$ — Geometric series

$\sum_{k=0}^{n-1} 2^k = \frac{2^n - 1}{n-1}$

$= \frac{\left(e^{i\pi \frac{j-k}{n}}\right)^n - 1}{e^{i\pi \frac{j-12}{n}} - 1} = 0$

Questions discussed in class:-

* Amortised analysis:-
  - 17.1.1
  - 17.2.1
  - 17.3.1
  - 17.3.3

* Graphs:-
  - 23.2.1
  - 23.2.7
  - 23.1.1
  - 23.1.2
  - 23.1.3

- 4-5 child testing
  - Divide & Conquer

Coreman

- Cake Flipping
- Some other question

Erickson

1) Aggregate Method:-

$$\frac{\text{Total cost of all operations}}{\text{Total no. of operations}}$$

2) Amortized bounds:-

- assign cost for each operation such that it preserved the sum

$$\sum_{op.} \text{amortized} \geq \sum_{op.} \text{Actual cost}$$

Example:-

2-3 trees achieve:-
- O(1) worst case per create - empty
- O(log n*) amortized insertion
- O(1) amortized per delete

Let's say we do c creates, i insert and d delete ops.
Then:-
Total cost = O(c + i log n + d 1)

* Accounting method:-

- Allocate an operation to store credit in bank account.
- Allocate operations to take coins out of bank.
- Allow an operation to pay for time using credit in bank.
- balance ≥ 0

Claim: O(log n) per insert, 0 per delete amortized.

- Put 1 coin worth θ(log n) per insert.
- Delete consumes 1 coin
amortized cost = actual cost + deposits - withdrawals

* Table doubling (Accounting method view)

insertions only
- when insert, add coin on that item worth c' = O(1)

- when we double the last n/2 columns / cells have coins

⇒ Amortized cost

$$= \theta(n) - c\frac{n}{2} = 0 \text{ if } c \text{ is large}$$

* Potential Method:-

→ Define potential function Φ mapping data structure configuration

→ Non negative integer

→ Amortized cost = Actual cost + ΔΦ

* Binary Counter Example:-

Φ = # 1 bits

- Increment destroys t trailing 1 bits and increments one bit from 0 to 1

∴ Amortized cost

$$= \underbrace{(1+t)}_{\text{actual}} \underbrace{-t+1}_{\hookrightarrow \Delta\Phi} = 2$$

Counter is growing linearly

∴ Φ = c*(# 1 bits)

∴ Amortized cost = O[(1+t) − t + 1]

= O(1)

Master Theorem :-

$T(n) = aT(n/b) + f(n)$

1) If $f(n) = O(n^{\log_b a - \epsilon})$, then $T(n) = \theta(n^{\log_b a})$

2) If $f(n) = \theta(n^{\log_b a})$, then $T(n) = \theta(n^{\log_b a} * \log n)$

3) If $f(n) = \Omega(n^{\log_b a + \epsilon})$, then $T(n) = \theta(f(n))$

---

* Chip testing question :-

a) Assuming that in all the chips available, $(\frac{n}{2} - 1)$ are good and $(\frac{n}{2} + 1)$ are bad.
-> If good ones = g
-> Bad ones = g+1

Algorithm :-
1) Keep one extra chip from (g+1) aside.
2) From the 2g chips present now, pick 2 chips repeatedly. If testing result says that chips are good, discard one and put other back.
If result is bad, discard both.

This ensures that # bad chips > # good chips after every test.
And the chip we have kept aside is also bad.
-> After 2g steps all good chips will be removed and we will have only bad ones.
∴ We can figure out the ans.

---

* Prove that $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Proving by induction :-

Claim : $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Base case : $i = 1$

$$\frac{n(n+1)}{2} = \frac{1(1+1)}{2} = \frac{1(2)}{2} = 1$$

∴ Base case is valid.

Induction hypothesis : For every $k \geq 2$

$$\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$$

Induction step :-

$$\sum_{i=1}^{K} i = \sum_{i=1}^{K-1} i + K$$

$$= \frac{(K-1)K}{2} + K$$

$$= \frac{(K-1)K + 2K}{2} = \frac{K^2 + K}{2}$$

$$= \frac{K(K+1)}{2}$$

Hence proved.

---

* Cake flipping :-

a) Worst case will take $2n-3$ flips

b)

c) $2n - 3 + n = 3n - 3 = 3(n-1)$
$\sim O(n)$

---

* Bit flipping :-

If n = odd
Algo -> 1-2-1-2---
1 -> Flip right most bit
2 -> Flip 2^{nd} bit (2 = total 0s the ends with)

If n = even
Algo -> 2-1-2-1-2---

Example :-

1) 111 $\xrightarrow{1}$ 110 $\xrightarrow{2}$ 010 $\xrightarrow{1}$ 011 $\xrightarrow{2}$ 001 $\xrightarrow{1}$ 000

2) 1111 $\xrightarrow{2}$ 1101 $\xrightarrow{1}$ 1100 $\xrightarrow{2}$ 0100 $\xrightarrow{1}$ 0101 $\xrightarrow{2}$ 0111 $\xrightarrow{1}$ 0110 $\xrightarrow{2}$ 0010 $\xrightarrow{1}$ 0011 $\xrightarrow{2}$ 0001 $\xrightarrow{1}$ 0000

FFTMultiply(P[0...n-1]Q[0...m-1])
{
$l = \lceil \lg(n+m) \rceil$

for $j = n$ to $2^l - 1$
$P[j] = 0$

for $j = m$ to $2^l - 1$
$Q[j] = 0$

$P^* = FFT(P)$
$Q^* = FFT(Q)$

for $j = 0$ to $2^l - 1$
$A[j] = P^*[j] \cdot Q^*[j]$

return InverseFFT(A^*)

InverseFFT:

if $n = 1$
return P
for $j = 0$ to $\frac{n}{2} - 1$
$U^*[j] = P^*[2j]$
$V^*[j] = P^*[2j+1]$

$U = InverseFFT(U[0 \dots \frac{n}{2} - 1])$

$V = InverseFFT(V[0 \dots \frac{n}{2} - 1])$

$\omega_n = \cos(\frac{2\pi}{n}) - i\sin(\frac{2\pi}{n})$

$\omega = 1$

for $j = 0$ to $\frac{n}{2} - 1$
$P[j] = 2(U[j] + \omega \cdot V[j])$
$P[j + n/2] = 2(U[j] - \omega \cdot V[j])$
$\omega = \omega \cdot \omega_n$

return $P[0 \dots n-1]$

FFT$[0 \dots n-1]$:

if $n = 1$
return P
for $j = 0$ to $\frac{n}{2} - 1$
$U[j] = P[2j]$
$V[j] = P[2j+1]$

$U^* = FFT(U[0 \dots n/2 - 1])$

$V^* = FFT(V[0 \dots n/2 - 1])$

$\omega_n = \cos(\frac{2\pi}{n}) + i\sin(\frac{2\pi}{n})$

$\omega = 1$

for $j = 0$ to $\frac{n}{2} - 1$
$P^*[j] = U^*[j] + \omega \cdot V^*[j]$
$P^*[j + n/2] = U^*[j] - \omega \cdot V^*[j]$

$\omega = \omega \cdot \omega_n$

return $P^*[0 \dots n-1]$

Finding Tailpalet :-

```
int n = B.length;

int[]      P    = new int[n];

for(int i=0; i<n; i++){
        P[i]=B[i];
}
int[] squared= multiply(P,P);

for(int j=0; j<n; j++)
{
  if(j*2<n && squared[j+2]>=3
        && B[j] == B[j*2]){
        return true;
  }
}

return false;
}
}
```

* Discrete Hartley Transformation :-

$$x_j = \sum_{i=0}^{n-1} x_i \left(\cos\left(\frac{2\pi}{n} ij\right) + \sin\left(\frac{2\pi}{n} ij\right)\right)$$

Algo:-

```
computeDHT(double[] x) {
    int n = x.length;

    if(n==1) {
        return x;
    }

    double[] x1 = new double[n/2];
    double[] x2 = new double[n/2];

    for(int i=0; i<n/2; i++) {
        x1[i] = x[i];
        x2[i] = x[i+n/2];
    }

    double[] x1 = computeDHT(x1);
    "        x2 = "     (x2);

    double[] x = new double[n];
    for(int j=0; j<n/2; j++)
    {
        x[j] = x1[j] + x2[j];
        x[j+n/2] = x1[j] - x2[j];
    }
    return x;
}
```

* Minkowski Sum:-

Number of elements in X+Y in MlogM

1 -> Find largest absolute value in X and Y — calling it M.

2 -> Create 2 arrays A and B of size [2M+1]

a) For each element x in X, increment A[x+M] by 1.

b) For each element y in Y, inc. B[M-y] by 1

3) Perform circular convolution of A and B using FFT based multiplication

a) This is done by converting A and B to frequency domain, multiplying them element wise, and then transforming the result back to time domain.

4) Count the no. of non-zero elements in resulting array — This will be the number of unique sums.

Its O(M log M) because the dominant operation is FFT based convolution which is O(M log M)

Convolution of arrays A and B:-

Assuming A and B are of size N

Circular convolution = $A \circledast B$

$$C[i] = \sum_{j=0}^{N-1} A[j] \cdot B[(i-j) \bmod N]$$

Fibonacci Heaps

→ Insertion :-

- Add node to root list
- Check if node->key < FH(min)
- If yes, FH(min) = node

→ Union :-

- Create a new root list FH
- Set min(FH) = min(FH1)
- Join root list of FH_2 to FH
- Check if min->FH1 is null, $min->FH_2$ is null, and if $min->FH_2$ is smaller than the $min->FH_1$ or not.
- If yes, set min(FH) = min(FH_2)
- n(FH) = n(FH_1) + n(FH_2)

→ Extract-min :-

- Add all children of node to be deleted in root list
- Delete the node
- Call consolidate(FH)
- n(FH) = n(FH) - 1

→ Consolidate :-