2024 Columbia Training Camp
Day 5: Fast Fourier Transforms

Christian Yongwhan Lim

Friday, August 23, 2024

Please fill out the following survey.

- https://bit.ly/2024-columbia-competitive-programming-training-camp

Fast Fourier Transform [Cooley and Tukey, 1965]

- **Multiply two polynomials of length n** in `O(n log n)` time, which is better than the trivial multiplication which takes $O(n^2)$ time.

Discrete Fourier Transform (DFT)

$$w_{n,k} = e^{\frac{2k\pi i}{n}} \qquad w_{n,k} = (w_n)^k.$$

$$\begin{aligned}
\mathrm{DFT}(a_0, a_1, \ldots, a_{n-1}) &= (y_0, y_1, \ldots, y_{n-1}) \\
&= (A(w_{n,0}), A(w_{n,1}), \ldots, A(w_{n,n-1})) \\
&= (A(w_n^0), A(w_n^1), \ldots, A(w_n^{n-1}))
\end{aligned}$$

Inverse Discrete Fourier Transform (Inverse DFT)

$$\text{InverseDFT}(y_0, y_1, \ldots, y_{n-1}) = (a_0, a_1, \ldots, a_{n-1})$$

Discrete Fourier Transform (DFT)

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

$$\mathrm{DFT}(A \cdot B) = \mathrm{DFT}(A) \cdot \mathrm{DFT}(B)$$

$$A \cdot B = \mathrm{InverseDFT}(\mathrm{DFT}(A) \cdot \mathrm{DFT}(B))$$

Fast Fourier Transform (FFT)

$$A(x) = a_0 x^0 + a_1 x^1 + \cdots + a_{n-1} x^{n-1}$$

$$A_0(x) = a_0 x^0 + a_2 x^1 + \cdots + a_{n-2} x^{\frac{n}{2}-1}$$
$$A_1(x) = a_1 x^0 + a_3 x^1 + \cdots + a_{n-1} x^{\frac{n}{2}-1}$$

$$A(x) = A_0(x^2) + x A_1(x^2).$$

Fast Fourier Transform (FFT)

$$\left(y_k^0\right)_{k=0}^{n/2-1} = \mathrm{DFT}(A_0) \qquad \left(y_k^1\right)_{k=0}^{n/2-1} = \mathrm{DFT}(A_1)$$

$$y_k = y_k^0 + w_n^k y_k^1, \quad k = 0 \ldots \frac{n}{2} - 1.$$

Fast Fourier Transform (FFT)

$$\begin{aligned}
y_{k+n/2} &= A\left(w_n^{k+n/2}\right) \\
&= A_0\left(w_n^{2k+n}\right) + w_n^{k+n/2} A_1\left(w_n^{2k+n}\right) \\
&= A_0\left(w_n^{2k} w_n^{n}\right) + w_n^{k} w_n^{n/2} A_1\left(w_n^{2k} w_n^{n}\right) \\
&= A_0\left(w_n^{2k}\right) - w_n^{k} A_1\left(w_n^{2k}\right) \\
&= y_k^0 - w_n^k y_k^1
\end{aligned}$$

Fast Fourier Transform (FFT)

$$y_k = y_k^0 + w_n^k y_k^1, \qquad k = 0 \ldots \frac{n}{2} - 1,$$

$$y_{k+n/2} = y_k^0 - w_n^k y_k^1, \qquad k = 0 \ldots \frac{n}{2} - 1.$$

Inverse Fast Fourier Transform (Inverse FFT)

$$\begin{pmatrix} w_n^0 & w_n^0 & w_n^0 & w_n^0 & \cdots & w_n^0 \\ w_n^0 & w_n^1 & w_n^2 & w_n^3 & \cdots & w_n^{n-1} \\ w_n^0 & w_n^2 & w_n^4 & w_n^6 & \cdots & w_n^{2(n-1)} \\ w_n^0 & w_n^3 & w_n^6 & w_n^9 & \cdots & w_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_n^0 & w_n^{n-1} & w_n^{2(n-1)} & w_n^{3(n-1)} & \cdots & w_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Vandermonde matrix

Inverse Fast Fourier Transform (Inverse FFT)

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} w_n^0 & w_n^0 & w_n^0 & w_n^0 & \cdots & w_n^0 \\ w_n^0 & w_n^1 & w_n^2 & w_n^3 & \cdots & w_n^{n-1} \\ w_n^0 & w_n^2 & w_n^4 & w_n^6 & \cdots & w_n^{2(n-1)} \\ w_n^0 & w_n^3 & w_n^6 & w_n^9 & \cdots & w_n^{3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_n^0 & w_n^{n-1} & w_n^{2(n-1)} & w_n^{3(n-1)} & \cdots & w_n^{(n-1)(n-1)} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

Inverse Fast Fourier Transform (Inverse FFT)

$$\frac{1}{n}\begin{pmatrix} w_n^0 & w_n^0 & w_n^0 & w_n^0 & \cdots & w_n^0 \\ w_n^0 & w_n^{-1} & w_n^{-2} & w_n^{-3} & \cdots & w_n^{-(n-1)} \\ w_n^0 & w_n^{-2} & w_n^{-4} & w_n^{-6} & \cdots & w_n^{-2(n-1)} \\ w_n^0 & w_n^{-3} & w_n^{-6} & w_n^{-9} & \cdots & w_n^{-3(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w_n^0 & w_n^{-(n-1)} & w_n^{-2(n-1)} & w_n^{-3(n-1)} & \cdots & w_n^{-(n-1)(n-1)} \end{pmatrix}$$

Inverse Fast Fourier Transform (Inverse FFT)

$$a_k = \frac{1}{n}\sum_{j=0}^{n-1} y_j w_n^{-kj}$$

$$y_k = \sum_{j=0}^{n-1} a_j w_n^{kj}$$

Fast Fourier Transform: Implementation

```
using cd = complex<double>;
const double PI = acos(-1);
void fft(vector<cd> & a, bool invert) {
    int n = a.size();
    if (n == 1) return;
    vector<cd> a0(n / 2), a1(n / 2);
    for (int i = 0; 2 * i < n; i++)
        a0[i] = a[2*i], a1[i] = a[2*i+1];
    fft(a0, invert), fft(a1, invert);
```

Fast Fourier Transform: Implementation

```
    double ang = 2 * PI / n * (invert ? -1 : 1);
    cd w(1), wn(cos(ang), sin(ang));
    for (int i = 0; 2 * i < n; i++) {
        a[i] = a0[i] + w * a1[i];
        a[i + n/2] = a0[i] - w * a1[i];
        if (invert)
            a[i] /= 2, a[i + n/2] /= 2;
        w *= wn;
    }
}
```

Fast Fourier Transform: Multiplying Two Polynomials

```
vector<int> multiply(vector<int> const& a,
                     vector<int> const& b) {
    vector<cd> fa(a.begin(), a.end()),
               fb(b.begin(), b.end());
    int n = 1;
    while (n < a.size() + b.size()) n <<= 1;
    fa.resize(n), fb.resize(n);
    fft(fa, false), fft(fb, false);
    for (int i = 0; i < n; i++) fa[i] *= fb[i];
    fft(fa, true);
```

Fast Fourier Transform: Multiplying Two Polynomials

```
    vector<int> result(n);
    for (int i = 0; i < n; i++)
        result[i] = round(fa[i].real());
    return result;
}
```

Number Theoretic Transform (NTT)

- The NTT is a generalization of the classic DFT to finite fields (e.g., a polynomial with coefficients modulo a prime P).

- A fast convolutions on integer sequences can be performed **without any round-off errors**, guaranteed.
  - Useful for multiplying large numbers or long polynomials
  - NTT is faster than Karatsuba.

Number Theoretic Transform (NTT)

- Suppose the input vector is a sequence of n non-negative integers.
- Choose a minimum working modulus M such that $1 \le n < M$ and every input value is in the range $[0,M)$.
- Select some integer $k \ge 1$ and define $P = kn+1$ as the working modulus. We require P is a prime number at least M.
  - **Dirichlet's theorem** guarantees that for any n and M, there exists some choice of k to make P a prime.
  - Sometimes, for convenience, we pick $P := 2^k c + 1$ where P is a prime with some positive integers k and c.

Number Theoretic Transform (NTT)

- Because P is a prime, the multiplicative group of $\mathbb{Z}_P$ has size $\varphi(P)=P-1=kn$. Also, the group must have at least one generator g, which is also a primitive $(P-1)^{th}$ root of unity.
- Define $\omega \equiv g^k$ mod P. We have $\omega^n=g^{kn}=g^{P-1}=g^{\varphi(P)}\equiv 1$ mod P due to **Euler's theorem**. Also, because g is a generator, we know that $\omega^i=g^{ik}\not\equiv 1$ for $1\leq i<n$ because $ik<nk=N-1$. Hence ω is a primitive n^{th} root of unity, as required by the DFT of length n.

Number Theoretic Transform (NTT)

- The rest of the procedure for the forward and inverse transforms is identical to the complex DFT.
  - Moreover, the NTT can be modified to implement a FFT algorithm such as Cooley–Tukey.

FFT + Generating Functions

- Often, the power of FFT comes from coupling it with generating functions.

- In particular, the i^{th} coefficients of the generating function encodes the information about the i^{th} term in a combinatorial object.

Generating Function

- a way of encoding an infinite sequence of numbers (a_n) by treating them as the **coefficients** of a formal **power series**.

Ordinary Generating Function (OGF)

$$G(a_n; x) = \sum_{n=0}^{\infty} a_n x^n.$$

$$G(a_{m,n}; x, y) = \sum_{m,n=0}^{\infty} a_{m,n} x^m y^n$$

Exponential Generating Function (EGF)

$$\mathrm{EG}(a_n; x) = \sum_{n=0}^{\infty} a_n \frac{x^n}{n!}$$

Generating Function: Example: Geometric Series

$$\sum_{n=0}^{\infty} x^n = \frac{1}{1-x}$$

1, 1, 1, 1, 1, …

$$\sum_{n=0}^{\infty} (ax)^n = \frac{1}{1-ax}$$

1, a, a^2, a^3, a^4, a^5, …

Generating Function: Common Series

$$\frac{1}{1-x} = 1 + x + x^2 + \ldots = \sum_{n \geq 0} x^n$$

$$-\ln(1-x) = x + \frac{x^2}{2} + \frac{x^3}{3} + \ldots = \sum_{n \geq 1} \frac{x^n}{n}$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots = \sum_{n \geq 0} \frac{x^n}{n!}$$

$$(1-x)^{-k} = \binom{k-1}{0} x^0 + \binom{k}{1} x^1 + \binom{k+1}{2} x^2 + \ldots = \sum_n \binom{n+k-1}{n} x^n$$

Trick #1: Multiplication by n

- For both OGF and EGF, $C(x)=xC'(x)$ generates the sequence $c_n=na_n$.

Trick #2: Left/Right Shifting

- For OGF, $C(x)=x^kA(x)$ generates the sequence $c_n=a_{n-k}$ where $a_i=0$ for $i<0$.
- For EGF, you need to integrate the series $A(x)$ k times to get the same effect.

- For OGF, $C(x)=(A(x)-(a_0+a_1x+a_2x^2+\ldots+a_{k-1}x^{k-1}))\ /\ x^k$ generates the sequence $c_n=a_{n+k}$.
- For EGF, $C(x)=A^{(k)}(x)$ generates the sequence $c_n=a_{n+k}$, where $A^{(k)}(x)$ denotes A differentiated k times.

Trick #3: Convolution

- For OGF, $C(x)=A(x)B(x)$ generates the sequence $c_n=\sum a_k b_{n-k}$.

- For EGF, $C(x)=A(x)B(x)$ generates the sequence $c_n=\sum c(n,k) a_k b_{n-k}$.
  - EGF is useful for recurrences with binomial coefficients or factorials.

Trick #4: Power of Generating Function

- For OGF, $C(x)=A(x)^k$ generates the sequence

$$c_n=\sum_{i[1]+i[2]+\ldots+i[k]=n} a_{i[1]}a_{i[2]}\ldots a_{i[k]}$$

- For EGF, $C(x)=A(x)^k$ generates the sequence

$$c_n=\sum_{i[1]+i[2]+\ldots+i[k]=n} n!/(i[1]!i[2]!\ldots i[k]!)\ a_{i[1]}a_{i[2]}\ldots a_{i[k]}$$

Trick #5: Prefix Sum Trick

- This only works for OGF.
- Suppose want to generate the sequence $c_n = a_0 + a_1 + \ldots + a_n$.
- We can take $C(x) = 1/(1-x)\ A(x)$.
  - If we expand, we get $(1 + x + x^2 + \ldots)A(x)$.
  - To obtain the coefficient of x_n, c_n, we need to choose x^i from the first bracket and $a_{n-i}x^{n-i}$ from $A(x)$.
  - Summing over all i gives us, $c_n = a_0 + a_1 + \ldots + a_n$.

Exercise #0 (Warm-Up): A+B Problem

Given N integers in the range $[-50\,000, 50\,000]$, how many ways are there to pick three integers a_i, a_j, a_k, such that i, j, k are pairwise distinct and $a_i + a_j = a_k$? Two ways are different if their ordered triples (i, j, k) of indices are different.

Input

The first line of input consists of a single integer N ($1 \leq N \leq 200\,000$). The next line consists of N space-separated integers $a_1, a_2, \ldots, a_N$.

Output

Output an integer representing the number of ways.

Exercise #0 (Warm-Up): A+B Problem (con't)

Sample Input 1

```
4
1 2 3 4
```

Sample Output 1

```
4
```

Sample Input 2

```
6
1 1 3 3 4 6
```

Sample Output 2

```
10
```

Exercise #0 (Warm-Up): A+B Problem (con’t)

- **Any idea?**

Exercise #0: Solution Idea

- Write a generating function with a solution for n encoded in the nth term.

Exercise #0: Solution Idea

- Write a generating function with a solution for n encoded in the nth term.

- Square the polynomial (using Fast Fourier Transform)!

Exercise #0: Solution Idea

- Write a generating function with a solution for n encoded in the nth term.
- Square the polynomial (using Fast Fourier Transform)!
- Since i and j should be different, subtract the terms that have them same.

Exercise #0: Solution Idea

- Write a generating function with a solution for n encoded in the nth term.
- Square the polynomial (using Fast Fourier Transform)!
- Since i and j should be different, subtract the terms that have them same.
- Read off the correct coefficients; to remove negatives, use offset!

Exercise #1

- Count the number of permutations of length n with k cycles.

Exercise #1: Solution Idea

- **Stirling numbers of the first kind**

- Let $c_n=(n-1)!$ be the number of permutations of length n which is a cycle.

- Let $C(x)=\sum(c_n/n!)x_n$ denote the EGF of c.

- Let f_n be our answer and $F(x)$ be its EGF.

Exercise #1: Solution Idea

- The key observation here is that $F(x)=(1/k!)C(x)^k$.
  - Suppose for a moment our cycles are labelled from 1 to k.
  - For every permutation, label each element with the label of the cycle it is in. Let's fix the length of cycle i to be a_i (so $\sum a_i=n$).
  - Then, there are c_{ai} ways to permute the elements in the i^{th} cycle and $n!/(a_1!a_2!\ldots a_k!)$ ways to assign cycle labels to the elements of the permutation.
  - Finally, in our actual problem, the order of cycles doesn't matter, so we need to divide by $k!$ in the end.

Exercise #1: Solution Idea

- The answer is:

$$\frac{n!}{k!} \sum_{a_1+a_2+\ldots+a_k=n} \frac{c_{a_1} c_{a_2} \cdots c_{a_k}}{a_1!\,a_2!\ldots a_k!}$$

Exercise #1: Solution Idea

- Now, you can check: $[x^n]C(x)^k$ is:

$$\sum_{a_1+a_2+\ldots+a_k=n} \frac{c_{a_1} c_{a_2} \ldots c_{a_k}}{a_1!a_2!\ldots a_k!}$$

- So, we get: $F(x)=(1/k!)C(x)^k$

Exercise #1: Solution Idea

- Now, for a CP problem with (n,k), we can do it in O(n log n).

Exercise #1: Solution Idea

- Now, for a CP problem with (n,k), we can do it in O(n log n).

- All you need to do is to use $P(x)^k = \exp(k \ln(P(x)))$!

Exercise #2

- Count the number of permutations of length n such that all cycle lengths are in a fixed set of positive integers S.

Exercise #2: Solution Idea

- We use the same trick as the previous problem, but let c_i=0 if i is not in S.

- Because we need to sum over all values of k (number of cycles):

$$[x^n] \sum_{k \geq 0} \frac{1}{k!} C(x)^k = [x^n] \exp(C(x))$$

Exercise #3

- Find the expected number of cycles of a permutation of length n.

Exercise #3: Solution Idea

- To compute the expected number of cycles, we count the sum of number of cycles over all permutations of length n.

- Let g_n denote the sum of number of cycles over all permutations of length n and $G(x)$ as the EGF of g.

- Using the same function C in the previous problems, we need to find:

Exercise #3: Solution Idea

$$[x^n]G(x) = [x^n]\sum_{k\geq 0}\frac{k}{k!}C(x)^k = [x^n]C(x)\sum_{k\geq 1}\frac{1}{(k-1)!}C(x)^{k-1} = [x^n]C(x)\exp(C(x))$$

Exercise #3: Solution Idea

$$[x^n]G(x) = [x^n]\sum_{k\geq 0}\frac{k}{k!}C(x)^k = [x^n]C(x)\sum_{k\geq 1}\frac{1}{(k-1)!}C(x)^{k-1} = [x^n]C(x)\exp(C(x))$$

- but,

$$C(x) = \sum_{k\geq 1}\frac{(k-1)!}{k!}x^k = \sum_{k\geq 1}\frac{x^k}{k} = -\ln(1-x)$$

Exercise #3: Solution Idea

- So,

$$C(x)\exp(C(x)) = -\frac{\ln(1-x)}{(1-x)}$$

- Now,

$$[x^n](-\ln(1-x)) = \frac{1}{n}$$

Exercise #3: Solution Idea

- By prefix sum trick, we have:

$$[x^n]\frac{-\ln(1-x)}{1-x} = 1 + \frac{1}{2} + \ldots + \frac{1}{n}$$

- So,

$$[x^n]G(x) = 1 + \frac{1}{2} + \ldots + \frac{1}{n}$$

Exercise #3: Solution Idea

- Since $g_n/n!$ is the expected number of cycles of a permutation of length n, the answer is 1+1/2+...+1/n, the n^{th} Harmonic number!

References

- **Fast Fourier Transform (FFT)**
  - https://cp-algorithms.com/algebra/fft.html
- **Number Theoretic Transform (NTT)**
  - https://www.nayuki.io/page/number-theoretic-transform-integer-dft
  - https://codeforces.com/blog/entry/48798
- **Generating Function**
  - https://codeforces.com/blog/entry/77468
  - https://codeforces.com/blog/entry/77551

Now, where can you find more problems like these?

- **CodeForces Problemset:** https://codeforces.com/problemset
- **Kattis Problems:** https://open.kattis.com/problem-sources
- **solved.ac:** https://solved.ac/problems/tags
- **acmicpc.net:** https://www.acmicpc.net/category/1

ICPC U

- https://u.icpc.global/training

NAC Problem Sets: solved.ac problem difficulty tiers

- **2024**: 2 Ruby; 4 Diamond; 5 Platinum; 1 Gold; 1 Silver;
- **2023**: 1 Ruby; 3 Diamond; 5 Platinum; 2 Gold; 1 Silver; 1 Unknown;
- **2022**: 3 Diamond; 8 Platinum; 2 Gold;
- **2021**: 3 Ruby; 3 Diamond; 3 Platinum; 3 Gold; 1 Silver;
- **2020**: 1 Ruby; 6 Diamond; 3 Platinum; 2 Gold;

Qualifying to ICPC World Finals

- **2024**: 2 Ruby; 4 Diamond; **5 Platinum; 1 Gold; 1 Silver;**
- **2023:** 1 Ruby; 3 Diamond; **5 Platinum; 2 Gold; 1 Silver; 1 Unknown;**
- **2022:** 3 Diamond; **8 Platinum; 2 Gold;**
- **2021:** 3 Ruby; 3 Diamond; **3 Platinum; 3 Gold; 1 Silver;**
- **2020:** 1 Ruby; **6 Diamond; 3 Platinum; 2 Gold;**

Therefore, to TRAIN for ICPC World Finals:

- Focus on Platinum ~ Diamond problems in solved.ac.

Contact Information

- **Email**: yongwhan.lim@columbia.edu
- **Personal Website**: https://www.yongwhan.io/
- **LinkedIn Profile**: https://www.linkedin.com/in/yongwhan/
  - Feel free to send me a connection request!
  - Always happy to make connections with awesome people like yourself! 🙂