The Dihedral FFT over a Prime Finite Field

1 Setup

Let

$$D_{2n} = \langle r, s \mid r^n = s^2 = e,\ srs^{-1} = r^{-1} \rangle$$

be the dihedral group of order $2n$. We work over a prime field $\mathbb{F}_p$, where $p \nmid 2n$, and assume that $n \mid p-1$. This ensures that $\mathbb{F}_p^\times$ contains a primitive n-th root of unity, denoted ω.

A function $f : D_{2n} \to \mathbb{F}_p$, or equivalently an element of the group algebra $\mathbb{F}_p[D_{2n}]$, is represented by two arrays

$$a_k = f(r^k), \qquad b_k = f(sr^k), \qquad 0 \le k < n.$$

The array a records the coefficients on the rotations, and the array b records the coefficients on the reflections.

We use the normalized finite-group Fourier transform convention

$$\widehat{f}(\rho) = \frac{1}{|D_{2n}|} \sum_{g \in D_{2n}} f(g)\rho(g) = \frac{1}{2n} \sum_{g \in D_{2n}} f(g)\rho(g).$$

All arithmetic is performed in $\mathbb{F}_p$, so the factor $1/(2n)$ means the multiplicative inverse of $2n$ modulo p.

2 Irreducible Representations

The one-dimensional representations are computed directly. For all n there are two:

$$\chi_{++}(r) = 1, \quad \chi_{++}(s) = 1, \qquad \chi_{+-}(r) = 1, \quad \chi_{+-}(s) = -1.$$

If n is even, there are two more:

$$\chi_{-+}(r) = -1, \quad \chi_{-+}(s) = 1, \qquad \chi_{--}(r) = -1, \quad \chi_{--}(s) = -1.$$

For a character with $\chi(r) = \epsilon$ and $\chi(s) = \delta$, the Fourier coefficient is

$$\widehat{f}(\chi) = \frac{1}{2n} \sum_{k=0}^{n-1} \left(a_k \epsilon^k + b_k \delta \epsilon^k \right).$$

The higher-dimensional irreducible representations are two-dimensional. For

$$1 \le j < \frac{n}{2} \quad \text{if } n \text{ is even,}$$

and

$$1 \le j \le \frac{n-1}{2} \quad \text{if } n \text{ is odd,}$$

we use

$$\rho_j(r^k) = \begin{pmatrix} \omega^{jk} & 0 \\ 0 & \omega^{-jk} \end{pmatrix}, \qquad \rho_j(sr^k) = \begin{pmatrix} 0 & \omega^{-jk} \\ \omega^{jk} & 0 \end{pmatrix}.$$

3 The Forward FFT

Define four cyclic transforms

$$A_j^+ = \sum_{k=0}^{n-1} a_k \omega^{jk}, \qquad A_j^- = \sum_{k=0}^{n-1} a_k \omega^{-jk},$$

$$B_j^+ = \sum_{k=0}^{n-1} b_k \omega^{jk}, \qquad B_j^- = \sum_{k=0}^{n-1} b_k \omega^{-jk}.$$

Then the two-dimensional Fourier coefficient is

$$\widehat{f}(\rho_j) = \frac{1}{2n} \begin{pmatrix} A_j^+ & B_j^- \\ B_j^+ & A_j^- \end{pmatrix}.$$

This is the main FFT reduction. Instead of computing each matrix coefficient from scratch, we compute the four full cyclic transforms

$$\mathrm{NTT}_\omega(a), \qquad \mathrm{NTT}_{\omega^{-1}}(a), \qquad \mathrm{NTT}_\omega(b), \qquad \mathrm{NTT}_{\omega^{-1}}(b).$$

The entries $A_j^+, A_j^-, B_j^+, B_j^-$ are read off from these four vectors and assembled into the matrix coefficients above.

The reason this works is that the two-dimensional representations separate the rotation and reflection data into ordinary cyclic Fourier sums. The nonabelian part only appears in how those four cyclic sums are placed into a 2×2 matrix.

4 The Inverse Transform

The inverse transform reverses this packing. The goal is to reconstruct the two ordinary cyclic spectra

$$A^+ = (A_0^+, \dots, A_{n-1}^+), \qquad B^+ = (B_0^+, \dots, B_{n-1}^+),$$

and then apply two inverse cyclic NTTs.

First consider the frequency $j = 0$. Since

$$\widehat{f}(\chi_{++}) = \frac{A_0^+ + B_0^+}{2n}, \qquad \widehat{f}(\chi_{+-}) = \frac{A_0^+ - B_0^+}{2n},$$

we recover

$$A_0^+ = n\left(\widehat{f}(\chi_{++}) + \widehat{f}(\chi_{+-})\right), \qquad B_0^+ = n\left(\widehat{f}(\chi_{++}) - \widehat{f}(\chi_{+-})\right).$$

If n is even, the two additional one-dimensional representations recover the middle frequency $j = n/2$:

$$A_{n/2}^+ = n\left(\widehat{f}(\chi_{-+}) + \widehat{f}(\chi_{--})\right), \qquad B_{n/2}^+ = n\left(\widehat{f}(\chi_{-+}) - \widehat{f}(\chi_{--})\right).$$

For the two-dimensional representations, write

$$\widehat{f}(\rho_j) = \begin{pmatrix} m_{00}^{(j)} & m_{01}^{(j)} \\ m_{10}^{(j)} & m_{11}^{(j)} \end{pmatrix}.$$

Since $A_j^- = A_{n-j}^+$ and $B_j^- = B_{n-j}^+$, the matrix coefficient recovers the paired frequencies j and $n - j$:

$$A_j^+ = 2n\, m_{00}^{(j)}, \qquad B_j^+ = 2n\, m_{10}^{(j)},$$

$$A_{n-j}^+ = 2n\, m_{11}^{(j)}, \qquad B_{n-j}^+ = 2n\, m_{01}^{(j)}.$$

After all entries of A^+ and B^+ have been reconstructed, we compute

$$a = \mathrm{NTT}_\omega^{-1}(A^+), \qquad b = \mathrm{NTT}_\omega^{-1}(B^+).$$

These are the original rotation and reflection coefficient arrays.

5 Fast Multiplication in the Group Algebra

The same transform gives a fast product in the group algebra $\mathbb{F}_p[D_{2n}]$. For comparison, direct multiplication is governed by the group law

$$r^i r^j = r^{i+j}, \qquad (sr^i)(sr^j) = r^{j-i},$$

$$r^i(sr^j) = sr^{j-i}, \qquad (sr^i)r^j = sr^{i+j},$$

with all exponents taken modulo n. If

$$x = \sum_i a_i r^i + \sum_i b_i s r^i, \qquad y = \sum_j c_j r^j + \sum_j d_j s r^j,$$

then $xy = \sum_m e_m r^m + \sum_m h_m s r^m$, where

$$e_m = \sum_{i+j\equiv m} a_i c_j + \sum_{j-i\equiv m} b_i d_j,$$

$$h_m = \sum_{j-i\equiv m} a_i d_j + \sum_{i+j\equiv m} b_i c_j.$$

This direct product costs $O(n^2)$ field operations.

Fourier transform turns this product into blockwise multiplication. If

$$z = xy,$$

then for every irreducible representation ρ,

$$\sum_g z(g)\rho(g) = \left(\sum_g x(g)\rho(g)\right)\left(\sum_g y(g)\rho(g)\right).$$

Because our transform is normalized by $1/|D_{2n}| = 1/(2n)$, this becomes

$$\widehat{z}(\rho) = |D_{2n}|\,\widehat{x}(\rho)\widehat{y}(\rho) = 2n\,\widehat{x}(\rho)\widehat{y}(\rho).$$

For one-dimensional representations this is scalar multiplication. For the two-dimensional representations this is ordinary 2×2 matrix multiplication, in the same order as the group-algebra product.

Thus the fast multiplication algorithm is:

$$x \longmapsto \widehat{x}, \qquad y \longmapsto \widehat{y},$$

multiply each Fourier block by the rule

$$\widehat{z}(\rho) = 2n\,\widehat{x}(\rho)\widehat{y}(\rho),$$

and then apply the inverse dihedral FFT to $\widehat{z}$. The cost is two forward dihedral FFTs, linear-time block multiplication, and one inverse dihedral FFT. Therefore the product costs

$$O(n \log n) = O(|D_{2n}| \log |D_{2n}|),$$

instead of $O(n^2)$.

6 Implementation Notes

The implementations use radix-2 number-theoretic transforms, so the fast path currently requires n to be a power of two. The naive dihedral DFT costs $O(n^2)$ field operations, while the fast forward transform costs four cyclic NTTs plus linear-time assembly:

$$4O(n \log n) + O(n) = O(n \log n).$$

Since $|D_{2n}| = 2n$, this is equivalently

$$O(|D_{2n}| \log |D_{2n}|).$$

The Python implementation supports arbitrary compatible primes p: it checks that p is prime, that $n \mid p-1$, and then finds a primitive root modulo p in order to construct ω. True extension fields $\mathbb{F}_{p^m}$ are not implemented yet; the current package works over prime fields $\mathbb{F}_p$. The Rust crate also supports the coefficient rings supported by its NTT backend, including some integer quotient rings $\mathbb{Z}/m\mathbb{Z}$.