アルゴリズム及び演習 第 12 回演習解答

小野 孝男*

2007 年 7 月 9 日

1. 1 の原始 8 乗根であるから, 「8 乗根」かつ「8 乗しないと 1 にならない」ものを探せばよい. まず 8 乗根を考えると, これは $x^8 = 1$ の解である. $x = r \exp i\theta$ ($r > 0$, $0 \le \theta < 2\pi$) と極座標表示すると

$$r^8 = 1 \text{ かつ } 8\theta = 2k\pi \ (k = 0, \ldots, 7)$$

である. つまり $r = 1$ かつ $\theta = k\pi/4$ ($k = 0, \ldots, 7$) である. ここで, 偶数の k に対しては高々 4 乗すると 1 となり, 一方で奇数の k に対しては 8 乗しない限り 1 となることはない. 従って, 原始 8 乗根は k を奇数にとればよい. $\cos \pi/4 = \sin \pi/4 = 1/\sqrt{2}$ なので 1 の原始 8 乗根は

$$\omega_8 = \frac{\pm 1 \pm i}{\sqrt{2}}$$

(複号任意) の 4 個である.

2. ω_n は 1 の原始 n 乗根であるから, 次の 2 つの性質を満たす:
   - $\omega_n^n = 1$,
   - $0 < k < n$ なら $\omega_n^k \neq 1$.

   このことに注意すると

   (a) $x = \omega_n^2$ とおく. まず, $x^{n/2} = (\omega_n^2)^{n/2} = \omega_n^n = 1$ である. また, $0 < k < n/2$ に対しては ($0 < 2k < n$ より) $x^k = (\omega_n^2)^k = \omega_n^{2k} \neq 1$ となる. 従って, $x = \omega_n^2$ は 1 の原始 $n/2$ 乗根である.

   (b) $x = \omega_n^{n/2}$ とおく. $0 < n/2 < n$ より $x^2 = 1$ かつ $x \neq 1$ である. 従って $x^2 - 1 = (x+1)(x-1) = 0$ だが, $x - 1 \neq 0$ で割ることができて $x + 1 = 0$. つまり $x = -1$ である.

* ono@is.nagoya-u.ac.jp

3. 教科書の FFT アルゴリズムでは, 関数 `fft` を呼び出すごとにワークとしてそれぞれ大きさ `AMAX` の配列 `y`, `z` を確保している. ところが, n 個のデータに対する FFT を行うにはそれぞれ $n/2$ 以上の大きさが必要である. つまり, 1 回 `fft` を呼び出すごとに $O(n)$ の領域を必要とする.
   また, `fft` は再帰呼び出しで FFT を行っており, (n 個のデータが与えられたときに $n/2$ 個のデータとして再帰的に実行するので) $\log n$ 段の再帰呼び出しが行われる. つまり, $\log n$ 段の再帰呼び出しのそれぞれで $O(n)$ の領域を必要とするので全体では $O(n)\log n = O(n\log n)$ の領域計算量となる.
   領域計算量を $O(n)$ に減らすために最も簡単な方法は,「ワークとして確保している配列を固定長でとるのではなく必要な量だけに制限する」というものである. つまり, 今のアルゴリズムで

```
complex y[AMAX], z[AMAX];
```

   となっているのを

```
complex *y = malloc(sizeof(complex) * n/2), *z = malloc(sizeof(complex) * n/2);
```

   とすればよい. 深さ k の再帰では処理するデータが $n/2^k$ 個なので必要なワークの大きさも $n/2^k$ である. こうすれば, 再帰全体でも必要な領域は

$$\sum_{k=0}^{\log n} \frac{n}{2^k} \leq \sum_{k=0}^{\infty} \frac{n}{2^k} = 2n$$

   となる.
4. 逆 FFT は, 用いる 1 の原始 n 乗根が ω_n から ω_n^{-1} になることと「最後に全体を n で割る」ことを除いて FFT と同じである. 従って, 基本的には pp. 126–127 のアルゴリズムをそのまま使えばよい. 変更点は次の通りである:
   - 関数 `butterfly` で使う `w(n, k)` を `w(n, n-k)` に変更する.
   - `main` 中, `ifft` を呼び出したあとで $x[\cdot]$ を全て n で割る.

   前者が ω_n^{-1} を使うことに, 後者が「最後に全体を n で割る」ことに対応する. 実行時間や領域計算量に影響する変更はなく, どちらも元の FFT と同じである.