Szybka transformacja Fouriera

Michał Dobranowski

wrzesień 2022
v1.0

Niniejszy skrypt przedstawia algorytm szybkiej transformacji Fouriera (FFT), który pozwala na szybsze mnożenie wielomianów. Nie opisujemy tutaj innych zastosowań transformacji Fouriera, takich jak analiza sygnałów czy kompresja danych. Aby w pełni zrozumieć ten skrypt, Czytelnik powinien znać podstawowe algorytmy i techniki algorytmiczne oraz swobodnie używać notacji określających asymptotyczne tempo wzrostu. Ponadto powinien mieć podstawową wiedzę na temat wielomianów, liczb zespolonych (w tym znać wzór Eulera) oraz macierzy.

Spis treści

§1 Reprezentacja i mnożenie wielomianów

Definicja 1.1 (Mnożenie wielomianów)**.** Iloczynem wielomianów $A(x)$ oraz $B(x)$ jest wielomian $C(x)$ taki, że dla każdego x zachodzi $C(x) = A(x) \cdot B(x)$.

Weźmy wielomiany $A(x) = a_0 + a_1 x + \ldots + a_n x^n$ oraz $B(x) = b_0 + b_1 x + \ldots + b_n x^n$.

Fakt 1.2. Możemy obliczyć współczynniki wielomianu $C(x) = A(x) \cdot B(x)$ jako

$$c_j = \sum_{k=0}^{j} a_k b_{j-k}.$$

Na podstawie tego wzoru możemy z łatwością skonstruować algorytm mnożenia dwóch wielomianów, który będzie działać w czasie $\Theta(n^2)$. Niestety, taka złożoność nas nie zadowala; będziemy szukać algorytmu podkwadratowego.

Zauważmy, że zwykle reprezentujemy wielomian jako wektor współczynników – taką reprezentację nazwiemy *współczynnikową*. Wielomian n-tego stopnia f można jednak reprezentować również za pomocą zbioru $n+1$ parami różnych punktów $(x, f(x))$ – tę reprezentację nazwiemy z kolei *punktową*. Mając dwa wielomiany n-tego stopnia w postaci punktowej, możemy po prostu pomnożyć odpowiednie wartości i otrzymać ich iloczyn (również w postaci punktowej) w czasie $\Theta(n)$.

Proces zamiany postaci współczynnikowej na punktową nazywa się *ewaluacją*, a proces odwrotny – *interpolacją*. Zanim przejdziemy dalej, powinniśmy jednak wykazać, że wielomian w postaci punktowej jest jednoznacznie opisany.

Twierdzenie 1.3 (Jednoznaczność wielomianu interpolacyjnego)

Dla zbioru $n+1$ różnych punktów $\{(x_1, y_1), (x_2, y_2), \ldots, (x_{n+1}, y_{n+1})\}$ istnieje dokładnie jeden wielomian $A(x)$ co najwyżej n-tego stopnia, który dla każdego k spełnia $A(x_k) = y_k$.

Dowód. Ponieważ punkty x_k są parami różne, to wyznacznik macierzy Vandermonde'a stworzonej z punktów $(x_1, x_2, \ldots, x_{n+1})$ jest różny od zera, więc macierz jest odwracalna. Niech V będzię wspomnianą macierzą.

$$V = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n+1} & x_{n+1}^2 & \cdots & x_{n+1}^{n-1} \end{pmatrix}$$

Rozwiązania układu równań

$$(a_0, a_1, a_2, \ldots, a_n)^T = V^{-1}(y_1, y_2, \ldots, y_{n+1})^T$$

pozwala jednoznacznie wyznaczyć współczynniki wielomianu. □

Chociaż powyższy dowód jest zupełnie w porządku, możemy zauważyć, że do postaci punktowej przechodzimy z postaci wielomianowej, więc w każdym przypadku wiemy, że dany wielomian istnieje. W takim razie potrzebujemy wykazać jedynie, że taki wielomian jest jednoznacznie określony, na co można przedstawić bardzo ładny dowód nie wprost.

Dowód. Załóżmy przeciwnie, że istnieją dwa wielomiany co najwyżej n-tego stopnia, $f(x)$ oraz $g(x)$, spełniające dany układ równań. W takim razie wielomian $h(x) = f(x) - g(x)$ jest stopnia $\deg(h) \leqslant n$. Ten wielomian ma również $n+1$ miejsc zerowych, co prowadzi do sprzeczności z zasadniczym twierdzeniem algebry. □

Uwaga 1.4

Przy dodawaniu lub mnożeniu wielomianów w postaci punktowej, punkty, w których znamy wartości dwóch danych wielomianów, powinny być oczywiście takie same dla obydwu z nich. Ponadto, o ile przy dodawaniu wystarczyłoby $n+1$ punktów, o tyle przy mnożeniu nie wystarczy. Wielomian wynikowy będzie stopnia $2n$, więc potrzebujemy $2n+1$ punktów również dla wielomianów wejściowych.

§1.1 Zmiana reprezentacji

Skoro wiemy już, że możemy mnożyć dwa wielomiany w postaci punktowej w czasie liniowym, warto się zastanowić, jak szybko możemy dokonać ewaluacji i interpolacji. Jeśli osiągniemy czas podkwadratowy, mamy gotowy algorytm mnożenie o złożoności $o(n^2)$.

Ewaluacja

Niestety, naiwny algorytm ewaluacji $2n+1$ punktów ma złożoność $\Theta(n^3)$, co możemy ulepszyć do $\Theta(n^2)$, jeśli zastosujemy schemat Hornera.

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

Interpolacja

Podobnie sytuacja wygląda z interpolacją – korzystając z eliminacji Gaussa, otrzymamy algorytm o złożoności $\Theta(n^3)$, co, korzystając ze wzoru Lagrange'a[1], możemy ulepszyć jedynie do $\Theta(n^2)$.

§2 Dziel i zwyciężaj

Niezniechęceni niepowodzeniami z poprzednich akapitów spróbujemy wykorzystać technikę z powyższego tytułu. Problem ewaluacji oraz interpolacji wielomianu

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

będziemy chcieli rekurencyjnie rozbijać na dwa podproblemy, więc na potrzeby dalszej dyskusji załóżmy, że n będzie potęgą dwójki. Zauważmy, że dla wygody zmieniliśmy nieznacznie oznaczenia — teraz wielomian A jest n-mianem o stopniu równym $n-1$.

Zdefiniujmy wielomiany $A_{[0]}(x)$ oraz $A_{[1]}(x)$ o współczynnikach z odpowiednio parzystymi i nieparzystymi indeksami:

$$A_{[0]}(x) = a_0 + a_2 x + a_4 x^2 + \ldots + a_{n-2} x^{n/2-1},$$

$$A_{[1]}(x) = a_1 + a_3 x + a_5 x^2 + \ldots + a_{n-1} x^{n/2-1}.$$

Oczywiście zachodzi równość

$$A(x) = A_{[0]}(x^2) + x A_{[1]}(x^2). \tag{1}$$

Chcielibyśmy, żeby teraz problem ewaluacji $A(x)$ w n punktach sprowadzał się do ewaluacji $A_{[0]}(x)$ i $A_{[1]}(x)$ w $\frac{n}{2}$ punktach[2]. Okazuje się, że potrzebną nam własność ma zbiór pierwiastków n-tego stopnia z jedności.

[1] który jest poza zakresem tego artykułu, ale można o nim przeczytać w `https://en.wikipedia.org/wiki/Lagrange_polynomial`

[2] własność oczywiście powinna być rekurencyjna

§2.1 Pierwiastki jedności

Definicja 2.1. Pierwiastek n-tego stopnia z jedynki to liczba $\omega \in \mathbb{C}$ spełniająca równanie $\omega^n = 1$.

Pierwiastki n-tego stopnia z jedności będziemy oznaczać $\omega_n^k = e^{2\pi ik/n} = \operatorname{cis}\left(\frac{2\pi}{n}k\right)$, gdzie $\operatorname{cis} x = \cos x + i \sin x$.

Lemat 2.2

Niech $\omega \neq 1$ będzie pierwiastkiem n-tego stopnia z jedynki. Wtedy

$$\omega + \omega^2 + \ldots + \omega^n = 0.$$

Dowód. Z równości $\omega^n = 1$ mamy

$$(1-\omega)(\omega + \omega^2 + \ldots + \omega^n) = \omega - \omega^{n+1} = 0.$$

Z założenia $(1-\omega) \neq 0$, więc otrzymujemy tezę. □

Lemat 2.3 (o redukcji)

Jeśli n jest parzyste, to kwadraty pierwiastków n-tego stopnia z jedności są pierwiastkami jedności stopnia $\frac{n}{2}$.

Dowód. Wystarczy zauważyć, że dla dowolnego k zachodzi

$$\left(\omega_n^k\right)^2 = \operatorname{cis}\left(\frac{2\pi}{n}2k\right) = \operatorname{cis}\left(\frac{2\pi}{n/2}k\right) = \omega_{n/2}^k.$$

□

Ponadto można dowieść, że każdy pierwiastek jedności $\frac{n}{2}$ stopnia uzyskamy, podnosząc dokładnie dwa różne pierwiastki jedności stopnia n. Jeśli n jest liczbą parzystą, to $\omega_n^{n/2} = -1$, więc

$$\omega_n^{k+n/2} = -\omega_n^k,$$
$$\therefore (\omega_n^{k+n/2})^2 = (\omega_n^k)^2.$$

Łatwo zauważyć, że na mocy lematu o redukcji (2.3) zbiór pierwiastków n-tego stopnia z jedności spełnia wymagania postawione w poprzedniej sekcji.

§2.2 Szybka transformacja Fouriera

Jeśli oznaczymy wektor współczynników wielomianu A jako $a = (a_0, a_1, \ldots, a_{n-1})$ oraz weźmiemy $y_k = A(\omega_n^k)$, to wektor $y = (y_0, y_1, \ldots, y_{n-1})$ nazwiemy *dyskretną transformatą Fouriera*[3] wektora współczynników a i oznaczymy $y = \mathrm{DFT}_n(a)$.

Algorytm, którego główne założenia opisaliśmy wyżej, służy do obliczania DFT (czyli ewaluacji wielomianu w pierwiastkach jedności) w czasie $O(n \log n)$ i jest nazywany *szybką transformacją Fouriera*[4].

[3] ang. *Discrete Fourier Transform*, DFT
[4] ang. *Fast Fourier Transform*, FFT

§2.2.1 Implementacja

Poniżej przedstawiony jest algorytm FFT, który dla danego wektora współczynników a liczy $\mathrm{DFT}(a)$. Obliczenia wykonywane są w miejscu, a więc funkcja `fft()` nie zwraca wektora $\mathrm{DFT}(a)$, tylko zmienia wektor a.

```cpp
#include <bits/stdc++.h>
using namespace std;

typedef complex<double> cd;
const double PI = acos(-1);

void fft(vector<cd>& a) {
    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);
    fft(a1);

    double ang = 2 * PI / n;
    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];
        w *= wn;
    }
}
```

§2.3 Transformacja odwrotna

Aby osiągnąć liniowo-logarytmiczną złożoność obliczeniową dla problemu mnożenia wielomianów, potrzebujemy już jedynie szybkiego sposobu na interpolację wielomianu w pierwiastkach jedności.

Wróćmy do wspomnianego w dowodzie twierdzenia 1.3 układu równań. Mamy

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix},$$

czyli

$$y_k = \sum_{j=0}^{n-1} a_k \omega_n^{jk}. \tag{2}$$

Nie wiemy jeszcze, jak szybko policzyć wektor $a = \mathrm{DFT}_n^{-1}(y)$ (czyli transformatę odwrotną do DFT), ale możemy ten wektor wyznaczyć za pomocą

$$\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{pmatrix}^{-1} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{pmatrix}.$$

Macierz z omegami będziemy oznaczać V, ponieważ jest ona macierzą Vandermonde'a. Szukamy więc macierzy V^{-1}, aby otrzymać wzór na a_k.

Lemat 2.4

Elementem na pozycji (j, j') w macierzy V^{-1} jest liczba $\omega_n^{-jj'}/n$.

Dowód. Rozważmy element na pozycji (j, j') w macierzy VV^{-1}.

$$\begin{aligned}[VV^{-1}]_{j,j'} &= \sum_{k=0}^{n-1}(\omega_n^{-kj}/n)(\omega_n^{kj'}) \\ &= \sum_{k=0}^{n-1}(\omega_n^{k(j'-j)}/n)\end{aligned}$$

Powyższa suma jest oczywiście równa 1, jeśli $j = j'$. W przeciwnym przypadku, na mocy lematu 2.2, jest równa 0. Z tego wynika, że VV^{-1} jest macierzą jednostkową, co kończy dowód. $\square$

Uzyskaliśmy więc wzór na a_k:

$$a_k = \frac{1}{n}\sum_{j=0}^{n-1} y_k \omega_n^{-jk}. \tag{3}$$

Zauważając podobieństwo między wzorami 2 i 3 możemy stwierdzić, że aby interpolować wielomian w pierwiastkach jedności, wystarczy użyć FFT, tylko zamiast ω wiąć ω^{-1} oraz wynik na końcu podzielić przez n. Tak zmodyfikowany algorytm oczywiście dalej ma złożoność obliczeniową $O(n \log n)$.

§3 Implementacja

Podsumowując, aby pomnożyć wielomiany $A(x), B(x)$ przez siebie w czasie $O(n \log n)$, należy najpierw zamienić je oba na postać punktową (ewaluację robimy za pomocą FFT w czasie liniowo-logarytmicznym), następnie pomnożyć punkty przez siebie (w czasie liniowym) i z powrotem zamienić na postać współczynnikową (interpolację robimy również w czasie liniowo-logarytmicznym[5]). Cały algorytm wykonywany jest w czasie $O(n \log n)$.

Z powodu podobieństwa algorytmów FFT oraz IFFT, zaimplementujemy je w jednej funkcji (z dodatkowym argumentem).

```
#include <bits/stdc++.h>
using namespace std;

typedef complex<double> cd;
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];
```

[5] algorytmem nazywanym IFFT, ang. *Inverse Fast Fourier Transform*

```
    }
    fft(a0, invert);
    fft(a1, invert);

    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;
    }
}

vector<int> multiply(vector<int>& a, vector<int>& b) {
    vector<cd> fa(a.begin(), a.end()), fb(b.begin(), b.end());
    int n = 1 << (__lg(a.size() + b.size() - 1) + 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);

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

§4 Zadania

Problem 4.1 (*Thief in a Shop*, Codeforces 632E)**.**

§5 Rozwiązania

4.1 Weźmy ciąg binarny taki, że jego m-ty wyraz jest 1 wtedy i tylko wtedy, gdy $m \in a$. Funkcją tworzącą tego ciągu jest wielomian

$$\mathcal{A}(x) = \sum_{i=1}^{n} x^{a_i}.$$

Rozważmy teraz taki ciąg binarny, że jego m-ty wyraz jest 1 wtedy i tylko wtedy, gdy m jest sumą pewnego k-elementowego podzbioru z powtórzeniami zbioru a. Będziemy wypisywać wszystkie takie m. Łatwo zauważyć, że funkcją tworzącą takiego ciągu będzie wielomian

$$\mathcal{A}(x)^k.$$

Stosując szybkie potęgowanie oraz FFT otrzymujemy złożoność $O(w \log w \log k)$, gdzie w jest iloczynem maksymalnej ceny i liczby k, a więc jest rzędu 10^6. Warto zauważyć, że w trakcie potęgowania współczynniki wielomianu $\mathcal{A}(x)$ szybko mogą stać się bardzo duże, a skoro nikt nas nie pyta nas o liczbę możliwości uzyskania danej wartości plecaka, to możemy po każdym mnożeniu ujednolicić współczynniki wielomianu funkcją signum.

Implementacja: `https://codeforces.com/contest/632/submission/178896692`.

Literatura

[1] **Wprowadzenie do algorytmów**, str. 920-940, Thomas H. Cormen et al.

[2] **Współczynniki wielomianów na okręgu jednostkowym kręcą się, kręcą się**, Delta, sierpień 2022, Radosław Kujawa. `https://www.deltami.edu.pl/2022a/08/2022-08-delta-art-07-kujawa.pdf`

[3] **cp-algorithms**. `https://cp-algorithms.com/algebra/fft.html`