Współczynniki wielomianów na okręgu jednostkowym kręcą się, kręcą się

Radosław KUJAWA

Rozważmy następujący problem: jak pomnożyć dwa wielomiany? Definiując ten problem bardziej precyzyjnie: mamy dane dwa wielomiany stopnia n: $A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$ i $B(x) = b_0 + b_1x + b_2x^2 + \ldots + b_nx^n$. Chcemy znaleźć współczynniki takiego wielomianu $C(x) = c_0 + c_1x + \ldots + c_{2n}x^{2n}$, że $C(x) = A(x)B(x)$ dla każdego x.

Moglibyśmy nie zakładać, że $A(x)$ i $B(x)$ są równego stopnia, ale dużo wtedy nie zyskujemy, bo możemy niskim kosztem dołożyć „mniejszemu" wielomianowi zerowych współczynników przy brakujących potęgach.

Przyjrzyjmy się wielomianowi $C(x)$. Wygląda on z grubsza tak: $C(x) = a_0b_0 + {} + (a_0b_1 + a_1b_0)x + (a_0b_2 + a_1b_1 + a_2b_0)x^2 + \ldots + (a_{n-1}b_n + a_nb_{n-1})x^{2n-1} + a_nb_nx^{2n}$. Mamy więc jawny wzór na jego współczynniki! Matematyk powie: „wielomian C jest znaleziony, problem rozwiązany". Jednak informatyk może zakrzyknąć: „hola, hola, ale jak to zaimplementować?". No właśnie.

Naiwna implementacja obliczania współczynników wielomianu $C(x)$ pododawałaby do siebie wymnożone parami współczynniki wielomianów A i B:

```
POMNÓŻ NAIWNIE([a_0, ..., a_n], [b_0, ..., b_n])
  FOR i ← 0 TO 2n DO
    c_i ← 0
    FOR j ← 0 TO i DO
      c_i ← c_i + a_j · b_{i−j}
  RETURN [c_0, c_1, ..., c_{2n}]
```

Notację dużego O można przetłumaczyć na „co najwyżej". Przykładowo wykonanie algorytmu w czasie $O(n^2)$ będzie trwać co najwyżej proporcjonalnie do kwadratu danych wejściowych.

Powyższy pseudokod wykona $O(n^2)$ mnożeń. Jest to bardzo słaby wynik. W następnym kroku pokażę więc przepiękny algorytm, który dzięki kilku Błyskotliwym Trikom działa w czasie $O(n \log n)$, czyli znacznie szybciej.

Błyskotliwy Trik numer 1 związany jest z reprezentacją wielomianu. Powyżej $A(x)$ i $B(x)$ były zdefiniowane jako wektory współczynników. Można jednak rozważyć wielomian W w postaci $n+1$ par $(x_i, W(x_i))$. Jeśli tylko x_i będą parami różne, a wielomian będzie stopnia n, to taka reprezentacja jest jednoznaczna. Zauważmy jej bardzo obiecującą własność: jeśli $A(x)$ i $B(x)$ będą zdefiniowane w tych samych punktach, to wyliczenie $C(x)$ jest banalne – wystarczy wziąć pary $(x_i, A(x_i) \cdot B(x_i))$! Tu jest jeden drobny haczyk – musimy mieć $2n+1$ takich punktów, by $C(x)$ był jednoznaczny, wszak jest on stopnia $2n$.

Niestety! Złośliwie zdefiniowaliśmy sobie taki problem, w którym wielomiany do pomnożenia dane są jako współczynniki, a nie jako wartości w punktach. Zmiana treści zadania jest powszechnie uważana za oszustwo. Jeśli chcemy więc, by nasz Błyskotliwy Trik numer 1 był użyteczny, musimy znaleźć sposób na konwersję pomiędzy tymi dwiema reprezentacjami. Ogólny schemat wymarzonego algorytmu będzie taki:

Konwersja ze współczynników do wartości w punktach nazywa się ewaluacją (stąd `EWALUUJ`). Operacja odwrotna to interpolacja(stąd też `INTERPOLUJ` w schemacie wymarzonego algorytmu).

```
POMNÓŻ SPRYTNIE(A[0..n], B[0..n], X[0..2n])
  [a'_0, ..., a'_{2n}] ← EWALUUJ(A, X)
  [b'_0, ..., b'_{2n}] ← EWALUUJ(B, X)
  FOR i ← 0 TO 2n DO
    c'_i ← a'_i · b'_i
  C[0..2n] ← INTERPOLUJ([c'_0, ..., c'_{2n}], X)
  RETURN C
```

Od teraz (dla wygody, którą będzie widać niebawem) będę zakładał, że rozważany wielomian $W(x)$ jest stopnia $n-1$ i potrzeba go obliczyć w n punktach.

Zacznijmy od ewaluacji. Znowu spróbujmy naiwnie:

```
EWALUUJ([w_0, ..., w_{n−1}], [x_0, ..., x_{n−1}])
  FOR i ← 0 TO n − 1 DO
    y_i ← w_{n−1}
    FOR j ← n − 2 TO 0 DO
      y_i ← y_i · x_i + w_j
  RETURN [y_0, ..., y_{n−1}]
```

Ten algorytm nie jest aż tak naiwny, jak mógłby być, bo stosuje schemat Hornera, który pozwala wyliczyć wartość wielomianu w czasie $O(n)$.

Niestety widać wyraźnie, że jesteśmy w punkcie wyjścia, bo algorytm wykona $O(n^2)$ operacji. Trudno zobaczyć, co tutaj można byłoby zrobić lepiej. Jednakże słuszną nadzieję może budzić fakt, że $x_0, \ldots, x_{n-1}$ mogą być zupełnie dowolne (byle tylko parami różne) i póki co z tego nie skorzystaliśmy. Może dałoby się tak dobrać punkty do ewaluacji wielomianu, by się aż tak bardzo nie naliczyć?

Przykład: weźmy wielomian $W(x) = 2x^6 + x^2 + 7$. Jeśli policzymy, że $W(1) = 10$, $W(2) = 139$, $W(3) = 1474$, to od razu $W(-1) = 10$, $W(-2) = 139$, $W(-3) = 1474$. Trzy na siedem koniecznych wartości mamy za darmo.

Prosta obserwacja jest taka: jeśli wielomian ma iksy tylko w parzystych potęgach, to $P(x) = P(-x)$. Można więc liczyć wartości dla par liczb przeciwnych: $x_0, -x_0, x_1, -x_1, \ldots$ Dla każdej pary wartości wielomianu będą takie same.

Druga prosta obserwacja: można sprawić, by każdy wielomian miał same parzyste potęgi – wystarczy wyciągnąć jeden x przed nawias tam, gdzie to konieczne:

$$W(x) = (w_0 + w_2x^2 + w_4x^4 + \ldots) + x(w_1 + w_3x^2 + w_5x^4 + \ldots).$$

Tym razem oszustwo jest legalne i użyteczne – w obu nawiasach wyrażenia są niewrażliwe na znak argumentu. Co prawda całe $W(a)$ oczywiście nie będzie równe $W(-a)$, ale do obliczenia obu tych wartości każdy z nawiasów wystarczy obliczyć raz.

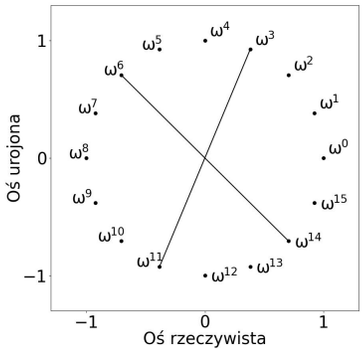
Jeśli $n \neq 2^k$, to dodanie kolejnych zerowych współczynników przy potęgach większych od n pozwoli powiększyć dane wejściowe tak, by liczba współczynników stała się najbliższą potęgą dwójki większą od n.

Bardzo obiecująco wygląda podział na dwa podwielomiany. Pójdźmy dalej tym tropem w stronę jakiegoś algorytmu „dziel i zwyciężaj”. W tym celu poczyńmy jeszcze niegroźne założenie, że $n = 2^k$, to bardzo uprości dalsze rozważania, o czym więcej na marginesie.

Żeby mieć prawidłowy algorytm „dziel i zwyciężaj”, potrzebujemy rozłożyć problem na dwa problemy o połowę mniejsze. W naszym przypadku potrzebujemy mieć dwa podwielomiany o połowę mniejszego stopnia. Możemy to zrobić o tak:

$$\begin{aligned} W_e(x) &= \left(w_0 + w_2x + w_4x^2 + \ldots + w_nx^{\frac{n}{2}}\right), \\ W_o(x) &= \left(w_1 + w_3x + w_5x^2 + \ldots + w_{n-1}x^{\frac{n}{2}-1}\right), \\ W(x) &= W_e(x^2) + xW_o(x^2). \end{aligned}$$

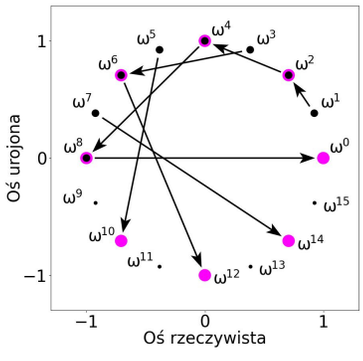
Będę używał notacji ω_n (lub krócej – ω, jeśli n będzie jasne) na oznaczenie liczby $\sqrt[n]{1} = e^{\frac{2i\pi}{n}} = \cos\frac{2\pi}{n} + i \cdot \sin\frac{2\pi}{n}$. Będę starał się myśleć o pierwiastkach z jedności jako o punktach na okręgu:



Liczby przeciwne leżą po przekątnej

Chcemy wyliczyć $W(x)$ w punktach $x_0, -x_0, x_1, -x_1, \ldots, x_{\frac{n}{2}}, -x_{\frac{n}{2}}$. Wystarczą nam w tym celu wartości W_e i W_o w punktach $x_0^2, x_1^2, \ldots, x_{\frac{n}{2}}^2$. Te moglibyśmy policzyć rekurencyjnie, mamy jednak problem: teraz wszystkie $\frac{n}{2}$ punktów to kwadraty, więc wyglądają tak dodatnio, jak się tylko da, i trudno będzie je ułożyć w pary liczb przeciwnych. Chyba że rozszerzymy dziedzinę na zbiór liczb zespolonych...

Tak odkrywamy Błyskotliwy Trik numer 2 – jako punkty ewaluacji wielomianu bierzemy n zespolonych pierwiastków z jedności!

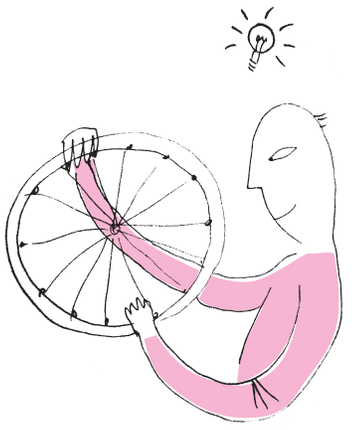


Liczby oznaczone czarnymi kropkami podniesione do kwadratu zmienią się w kolorowe kropki

Rysunki na marginesie przedstawiają pierwiastki 16 stopnia z jedności na płaszczyźnie zespolonej. Liczby z jednego półokręgu są przeciwne do liczb z drugiego półokręgu, zaś podniesione do kwadratu dadzą cały okrąg, na którym punkty rozmieszczone są dwa razy rzadziej. Czyli mamy obie cechy, na których tak nam zależało! Fantastyczna sprawa.

Możemy teraz napisać pseudokod, który zrobi szybką ewaluację wielomianu w pierwiastkach z jedności:

```
EWALUUJ SZYBKO([w_0, ..., w_{n-1}])
  IF n = 1 THEN
    RETURN [w_0]
  [(y_e)_0, ..., (y_e)_{n/2-1}] = EWALUUJ SZYBKO([w_0, w_2, ..., w_{n-2}])
  [(y_o)_0, ..., (y_o)_{n/2-1}] = EWALUUJ SZYBKO([w_1, w_3, ..., w_{n-1}])
  FOR i ← 0 TO n/2 − 1 DO
    y_i ← (y_e)_i + ω^i · (y_o)_i
    y_{i+n/2} ← (y_e)_i − ω^i · (y_o)_i
  RETURN [y_0, ..., y_{n-1}]
```



Algorytm jest zwięzły i niezwykle estetyczny, a do tego działa satysfakcjonująco szybko: składają się nań $O(n)$ operacji i dwa obliczenia rekurencyjne. Mamy więc następujące równanie na liczbę operacji:

$$T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n).$$

Rekursja ta schodzi na $\log n$ poziomów, na każdym poziomie jest do wykonania $O(n)$ operacji, więc całość działa w czasie $O(n \log n)$. Wiwat!

Nie możemy jednak spocząć na laurach, bo jesteśmy dopiero w połowie drogi. Została nam wszak do wymyślenia procedura odwrotna, czyli interpolacja. Spróbujmy wydobyć na światło dzienne jeszcze jeden Błyskotliwy Trik, który pozwoli nam wykorzystać ten sam algorytm, który już mamy.

Zapiszmy zagadnienie ewaluacji w formie macierzowej:

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega & \omega^2 & \dots & \omega^{n-1} \\ 1 & \omega^2 & \omega^{2\cdot 2} & \dots & \omega^{2(n-1)} \\ \vdots & & & \vdots & \vdots \\ 1 & \omega^{n-1} & \omega^{(n-1)\cdot 2} & \dots & \omega^{(n-1)(n-1)} \end{bmatrix} \cdot \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

I to macierz identycznościowa, czyli taka, która ma jedynki na przekątnej, a wszędzie indziej zera:

$$I = \begin{bmatrix} 1 & 0 & & \dots & 0 \\ 0 & 1 & & & \\ \vdots & & \ddots & & \vdots \\ & & & 1 & 0 \\ 0 & \dots & & 0 & 1 \end{bmatrix}$$

Równanie to mówi tyle, że $y_k = W(\omega^k)$ dla $k = 0, \dots, n-1$. Macierz z omegami jest szczególnym przypadkiem macierzy Vandermonde'a, więc nazwijmy ją V. Powyższe równanie przybiera postać $V \cdot \vec{W} = \vec{Y}$. Chcielibyśmy tak je zmodyfikować, by na podstawie $\vec{Y}$ otrzymać $\vec{W}$. Zgadnijmy takie V', że $V' \cdot V = nI$:

$$V' = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \dots & \omega^{-(n-1)} \\ 1 & \omega^{-2} & \omega^{(-2)\cdot 2} & \dots & \omega^{(-2)(n-1)} \\ \vdots & & & \vdots & \vdots \\ 1 & \omega^{-(n-1)} & \omega^{-(n-1)\cdot 2} & \dots & \omega^{-(n-1)(n-1)} \end{bmatrix}$$

V zawiera pierwiastki z jedności brane z okręgu jednostkowego przeciwnie do ruchu wskazówek zegara. V' zawiera te same pierwiastki, tylko brane zgodnie z ruchem wskazówek zegara.

Sprawdźmy, że $V' \cdot V$ rzeczywiście da nI. W i-tym wierszu i j-tej kolumnie mamy:

$$\sum_{k=0}^{n-1} \omega^{-ik} \cdot \omega^{jk} = \sum_{k=0}^{n-1} \omega^{(j-i)k}.$$

Czemu przy $j \neq i$ suma będzie 0? Interesują nas dwa przypadki: $m = j - i > 0$ i $m = j - i < 0$. Drugi łatwo sprowadzić do pierwszego, bo $\omega^{j-i} = \omega^{j-i} \cdot 1 = \omega^{j-i} \cdot \omega^n = \omega^{n-(i-j)}$, a $n - (i - j)$ już jest dodatnie. Kolejne potęgi ω^m układają się na okręgu jednostkowym: zaczynamy od $1 = \omega^0$ i skaczemy co m-ty punkt. Jeśli $NWD(m, n) = 1$, to zapełnimy cały okrąg jednostkowy. Jeśli $NWD(m, n) > 1$, to niektóre pierwiastki będą powtórzone $NWD(m, n)$ razy, a inne nie będą odwiedzone wcale. Tak czy inaczej, stworzymy jakiś nowy, kompletny zbiór pierwiastków z jedności (być może zwielokrotniony) – a taki zawsze sumuje się do zera.

Jeśli $j = i$, to suma oczywiście jest równa n. Okazuje się, że przy $j \neq i$ ta suma zawsze jest równa 0 (patrz margines). Zatem mamy $V' \cdot V \cdot \vec{W} = V' \cdot \vec{Y}$, czyli $V' \cdot \vec{Y} = n\vec{W}$. To równanie bardzo przypomina to, które opisywało ewaluację! Dotarliśmy do sedna Błyskotliwego Triku numer 3 – interpolację robimy *niemal dokładnie tak samo* jak ewaluację, tylko zamiast ω weźmiemy ω^{-1}, a wynik na końcu podzielimy przez n.

```
INTERPOLUJ SZYBKO([y_0, ..., y_{n-1}])
  IF n = 1 THEN
    RETURN [y_0]
  [(w_e)_0, ..., (w_e)_{n/2-1}] = INTERPOLUJ SZYBKO([y_0, y_2, ..., y_{n-2}])
  [(w_o)_0, ..., (w_o)_{n/2-1}] = INTERPOLUJ SZYBKO([y_1, y_3, ..., y_{n-1}])
  FOR i ← 0 TO n/2 - 1 DO
    w_i ← (w_e)_i + ω^{-i} · (w_o)_i
    w_{i+n/2} ← (w_e)_i - ω^{-i} · (w_o)_i
  RETURN [w_0/n, ..., w_{n-1}/n]
```

Teraz nasza procedura `POMNÓŻ SPRYTNIE` jest kompletna.

Algorytm ewaluacji w pierwiastkach z jedności nazywa się powszechnie FFT (*Fast Fourier Transform*), algorytm interpolacji to IFFT (*Inverse Fast Fourier Transform*). Szybkie mnożenie wielomianów jest tylko jednym z wielu zdumiewających zastosowań transformacji Fouriera (gorąco zachęcam do zajrzenia do Δ_{19}^{11} po więcej szczegółów). Bez FFT nie byłoby cyfrowego przetwarzania sygnałów ani kompresji obrazów. Fascynujące jest to, że ten algorytm jest jednocześnie ogromnie przydatny i wyjątkowo piękny. To rzadki ptak!

Bardzo ciekawe jest to, że mnożenia wielomianów w czasie $O(n \log n)$ nie umiemy uogólnić na mnożenie macierzy, a algorytmy robiące to szybciej niż w $O(n^3)$ korzystają z zupełnie innych technik (gorąco zachęcam do zajrzenia do artykułu Michała Włodarczyka *Jak proste problemy stały się trudne* (Δ_{18}^{11}) po więcej szczegółów).