Brza Furijeova Transformacija

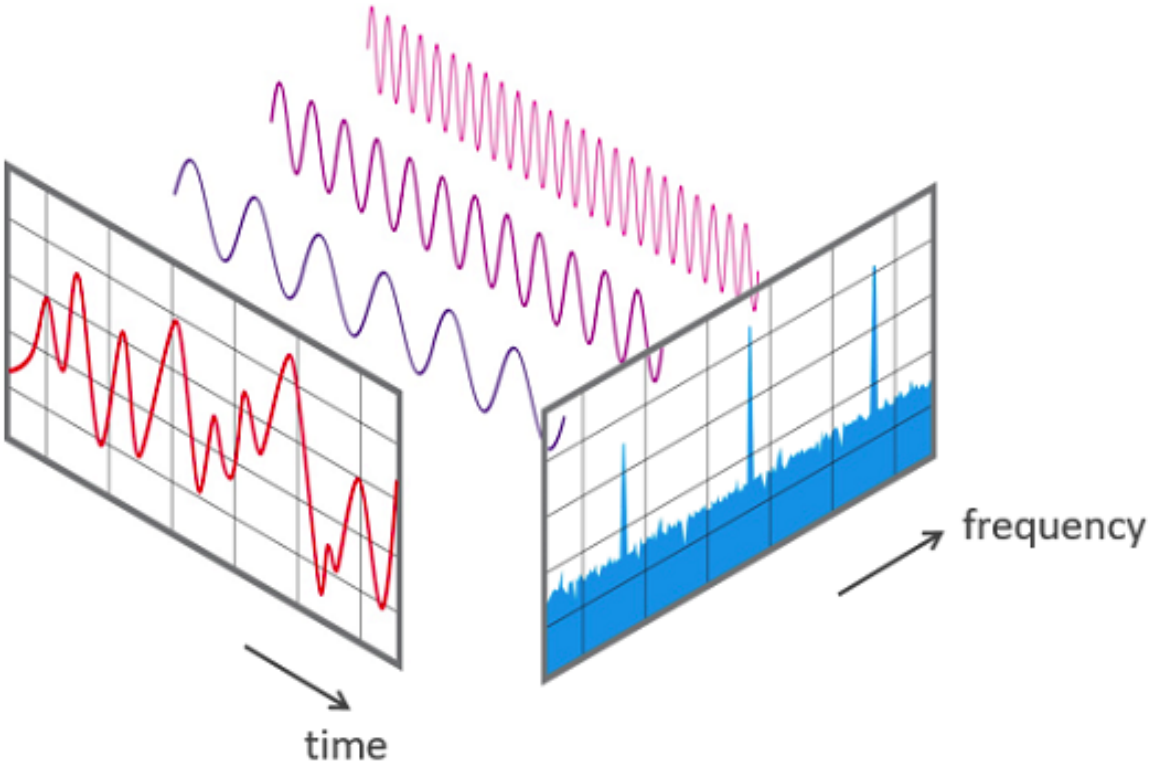
Sana Stojanović Đurđević

Matematički fakultet, Beograd, 08. Septembar, 2020.

Pregled

- Objavljen je 1965. godine Tukey, Cooley, IBM (1805. godine, Gauss)
- Jedan od 10 najbitnijih algoritama XX veka (pored Quicksort algoritma, Simpleks metoda...)
- Dosta različitih primena
  - Množenje polinoma
  - Množenje dva dugačka broja
  - Pretraga niske (kompletno poklapanje i parcijalno poklapanje)
  - Obrada signala
  - ...

Obrada signala



- Diskretan skup podataka izmeren sa fiksiranim razmakom
- Izdvajanje pojedinačnih frekvenci iz zadatog signala
- Svaka sinusioda generiše jedan ton koji ima svoju specifičnu frekvenciju

Množenje polinoma

- $P(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0$
- $st(P) = n - 1$, $st(Q) = n - 1$, $st(P * Q) = 2 * n - 2$
- Predstavljanje polinoma svojim koeficijentima:
  Množenje ima vremensku složenost $O(n^2)$
- Predstavljanje polinoma vrednostima u n različitih tačaka:
  Množenje ima vremensku složenost od $O(n)$
- Direktna i inverzna Furijeova transformacija, prelaz iz jedne u drugu reprezentaciju polinoma. Složenost $O(n \log n)$.

Diskretna Furijeova transformacija

- Koeficijenti $->$ vrednosti

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{bmatrix} * \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} P(x_0) \\ P(x_1) \\ \vdots \\ P(x_{n-1}) \end{bmatrix}$$

- Vandermondova matrica za konkretan skup od n kompleksnih tačaka $(n = 2^k)$, $x^n = 1$
- Primitivan n-ti koren iz jedinice w: $w^n = 1$, $w^j \neq 1$ $(0 < j < n)$
- Biramo brojeve $1, w, w^2, \dots, w^{n-1}$,
  odnosno $w_{n,k} = e^{\frac{2k\pi i}{n}}$, $k \in \{0, 1, \dots, n-1\}$
- $w = w_{n,1} = e^{\frac{2\pi i}{n}}$, $w_k = w^k$.

Podeli pa vladaj

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & w & w^2 & \dots & w^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w^{n-1} & w^{(n-1)*2} & \dots & w^{n-1*n-1} \end{bmatrix} * \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} P(1) \\ P(w) \\ \vdots \\ P(w^{n-1}) \end{bmatrix}$$

- $P(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1x + a_0$
  parne pozicije: $P_0(x) = a_0x^0 + a_2x^1 + \ldots + a_{n-2}x^{n/2-1}$
  neparne pozicije: $P_1(x) = a_1x^0 + a_3x^1 + \ldots + a_{n-1}x^{n/2-1}$
- $P(x) = P_0(x^2) + x * P_1(x^2)$
- Zbog pogodno odabranog skupa vrednosti, veličina skupa vrednosti se smanjuje na pola u rekurzivnim pozivima
- Složenost $T(n) = 2T(n/2) + O(n)$, $O(n \log n)$

Inverzna Furijeova transformacija

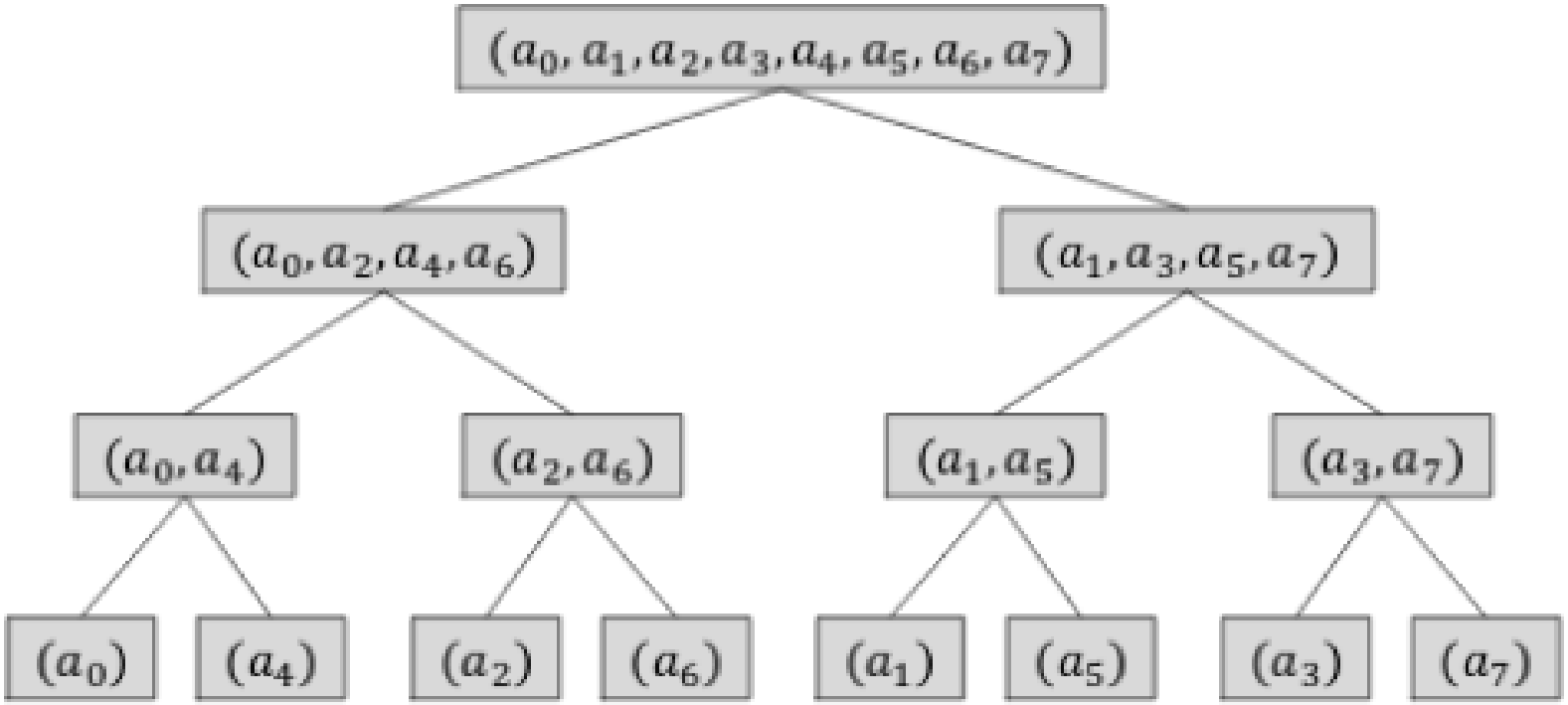
- Vrednosti $->$ koeficijenti
- $V(w) * \vec{a} = \vec{v}$, sistem linearnih jednačina: $\vec{a} = V(w)^{-1} * \vec{v}$ (u opštem slučaju složenost $O(n^3)$)
- $\vec{a} = [a_0, a_1, \ldots, a_{n-1}]^T$, $\vec{v} = [P(1), P(w), \ldots, P(w^{n-1})]^T$
- $V(w) * V(w^{-1}) = n * I$ pa je otuda $V(w)^{-1} = \frac{1}{n} V(w^{-1})$
- $\vec{a} = \frac{1}{n} V(w^{-1}) \vec{v}$
- w^{-1} je takođe primitivni n-ti koren iz jedinice

Jednostavna implementacija u jeziku C++

```
#include <complex> //zaglavlje za rad sa kompleksnim brojevima
typedef complex<double> Complex;        //skraceni zapis
typedef vector<Complex> ComplexVector; //vektor kompleksnih brojeva
ComplexVector fft(const ComplexVector& a, bool inverzna){
...
  // izdvajamo koeficijente na parnim i na neparnim pozicijama
  ComplexVector A(n / 2), B(n / 2);
  for(int i = 0; i < n / 2; i++) {
    A[i] = a[2 * i];
    B[i] = a[2 * i + 1];
  }
  // rekurzivni pozivi
  ComplexVector fftA = fft(A, inverzna), fftB = fft(B, inverzna);

  // objedinjavanje rezultata
  for(int k = 0; k < n; k++) {
    // odredjujemo primitivni n-ti koren iz jedinice
    double coeff = inverzna ? -1.0 : 1.0;
    complex<double> w = exp((coeff * 2 * k * M_PI / n) * 1i);
    // racunamo vrednost polinoma u toj tacki
    rez[k] = fftA[k % (n / 2)] + w * fftB[k % (n / 2)];
  }
...
```

Primer razlaganja rekurzivnih poziva



$P_{0,1,2,3,4,5,6,7}(1, w, w^2, \ldots, w^7) => P_{0,2,4,6}(1, w^2, w^4, w^6)$ i $P_{1,3,5,7}(1, w^2, w^4, w^6)$

$P_{0,2,4,6}(1, w^2, w^4, w^6) => P_{0,4}(1, w^4)$ i $P_{2,6}(1, w^4)$

$P_{0,4}(1, w^4) => P_0(1) = 0$ i $P_4(1) = 4$

Poboljšanje implementacije

- Vremensko - svi primitivni koreni se mogu izračunati unapred
  - Ako je broj k-ti primitivni koren iz jedinice, onda je on i $2k$-ti primitivni koren
- Prostorno - nema potrebe za dodatnom alokacijom memorije
  - Pamtimo poziciju početka (u originalnom vektoru) i pomeraj

Primena FFT - Sve moguće sume

- Sve moguće sume jednog elementa prvog niza i jednog elementa drugog niza
- $a = [1, 3, 4]$, $b = [2, 2, 3]$
- Vrednost 3 se može dobiti na 2 načina,
  vrednost 4 na 1 način,
  vrednost 5 na 2 načina,
  vrednost 6 na 3 načina,
  vrednost 7 na 1 način
- Transformacija: $a[i] => x^{a[i]}$, $count(a[i]) = coef(x^{a[i]})$
- $A(x) = 1x^1 + 1x^3 + 1x^4$, $B(x) = 2x^2 + 1x^3$
- $C(x) = A(x) * B(x) = 2x^3 + 1x^4 + 2x^5 + 3x^6 + 1x^7$
- Složenost $O(n + m \log m)$, n - dužina, m - najveći broj koji se javlja

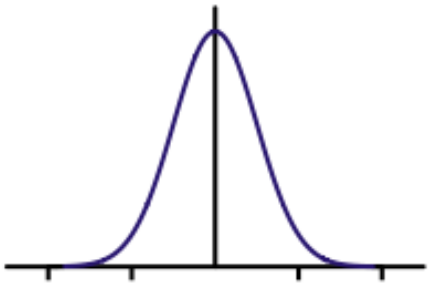
Primena FFT - Svi mogući skalarni proizvodi

- Za zadata dva niza dužine n, izračunati skalarni proizvod prvog vektora sa svim cikličnim rotacijama drugog vektora
  $[1,2]x[3,4] = 1*3+2*4$
  $[1,2]x[4,3] = 1*4+2*3$
- Transformacija: generišemo dva vektora dužine $2n$, prvi niz rotiramo i dopišemo n nula na kraj, drugi niz dopišemo sam na sebe
- $A(x) = 2x^0 + 1x^1 + 0x^2 + 0x^3$, $B(x) = 3x^0 + 4x^1 + 3x^2 + 4x^3$

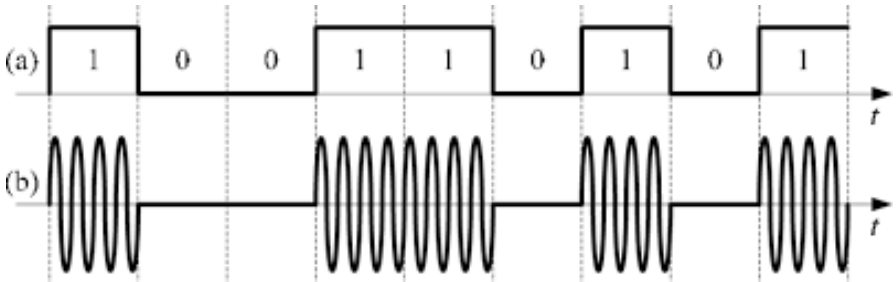
$$A(x) * B(x) = 2*3x^0 + 2*4x^1 + |2*3x^2| + |2*4x^3| + 0*x^4 \\ + 1*3x^1 + |1*4x^2| + |1*3x^3| + 1*4x^4$$

Primena cikličnih skalarnih proizvoda

- Primena filtera na dati signal



- Problem poravnavanja binarnih signala tako da im se ne poklapaju logički tačne vrednosti



Pretraga stringova

- Ilustracija množenja polinoma sa šiftovanjem:
  $(a + bx) * (c + dx) = ac + (ad + bc)x + bdx^2$

```
      a + bx    |      a + bx     |     a + bx
dx + c          |     dx + c      |          dx + c
```

- Binarna azbuka, pretražujemo string ab u stringu bab
  $P(x) = b + ax$ (rotiramo zbog ponovnog rotiranja) i
  $T(x) = b + ax + bx^2$
  $P(x) * T(x) = b^2 + (ab + ba)x + (a^2 + b^2)x^2 + bax^2$
- Kodiramo $a = 1$, $b = -1$, i tada je poklapanje (stringa dužine 2) označeno sa $a^2 + b^2 = 2$ (za poklapanje stringa dužine k će biti jednako k)

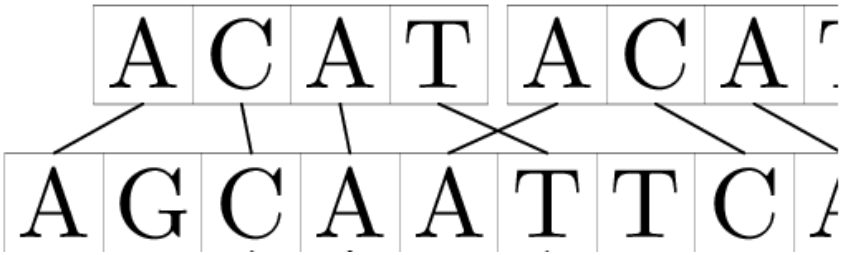
```
    b + ax + bx^2  |   b + ax + bx^2  |  b + ax + bx^2  |  b + ax + bx^2
ax + b             |  ax + b          |      ax + b     |           ax  + b
```

Parcijalno poklapanje stringova

- Pretraga stringa oblika `a_b`, gde je sa `_` označen proizvoljan karakter
- Kodiranje $a = 1$, $b = -1$, _=0. Poklapanje dobijamo kada u zbiru dobijemo vrednost k koja je jednaka broju navedenih karaktera (a ne dužini niske koja se traži)
- Složenost $O((m+n)\log(m+n))$ (m i n dužine teksta i stringa koji se traži)

Uopštenje azbuke

- Pretraga genoma (slova A, T, G, C) u kome mogu biti prisutne sitne mutacije. Dodatno, gornje ograničenje na broj mutacija.



- Uopštenje azbuke na mala slova engleske abecede:
  Kodiranje u tekstu: $e^{\frac{2\pi i}{26} * k}$. Kodiranje u stringu: $e^{\frac{-2\pi i}{26} * k}$
  Poklapanje pojedinačnog karaktera daje jedinicu, odnosno poklapanje celog stringa daje dužinu stringa koji se traži.