Szybkie mnożenie wielomianów i macierzy

Łukasz Kowalik

Instytut Informatyki, Uniwersytet Warszawski

Szybka Transformata Fouriera

Wielomiany

Wielomian reprezentowany jako ciąg współczynników

Niech R będzie pierścieniem (np. $R = \mathbb{C}, \mathbb{R}, \mathbb{Z}, \ldots$).
Wielomian stopnia $n-1$ to funkcja $A : \mathbb{C} \to \mathbb{C}$ postaci

$$A(x) = \sum_{i=0}^{n-1} a_i x^i,$$

gdzie $a_i \in R$.
Wielomian A reprezentujemy jako ciąg $(a_0, \ldots, a_{n-1})$.

Dodawanie

Dane dwa wielomiany $A = (a_0, \ldots, a_{n-1})$ i $B = (b_0, \ldots, b_{n-1})$.
Obliczyć wielomian C, $C(x) = A(x) + B(x)$.
Rozwiązanie: $C = (a_0 + b_0, \ldots, a_{n-1} + b_{n-1})$.
Czas: $O(n)$ operacji arytmetycznych (na elementach R).

Mnożenie wielomianów w $O(n^2)$

Problem

Dane dwa wielomiany $A = (a_0, \ldots, a_{n-1})$ i $B = (b_0, \ldots, b_{n-1})$.
Obliczyć wielomian C, $C(x) = A(x) \cdot B(x)$.

Rozwiązanie

Rozwiązanie: $C = (c_0, \ldots, c_{2n-2})$, gdzie

$$c_k = \sum_{i=0}^{k} a_i \cdot b_{k-i}.$$

Czas: $O(n^2)$ operacji arytmetycznych (na elementach R).

Mnożenie wielomianów w $O(n)$

A gdyby tak...

Wyobraźmy sobie nowy lepszy świat, w którym wielomiany mnoży się równo łatwo jak dodaje. **(Co to za świat?)**

Mnożenie wielomianów w $O(n)$

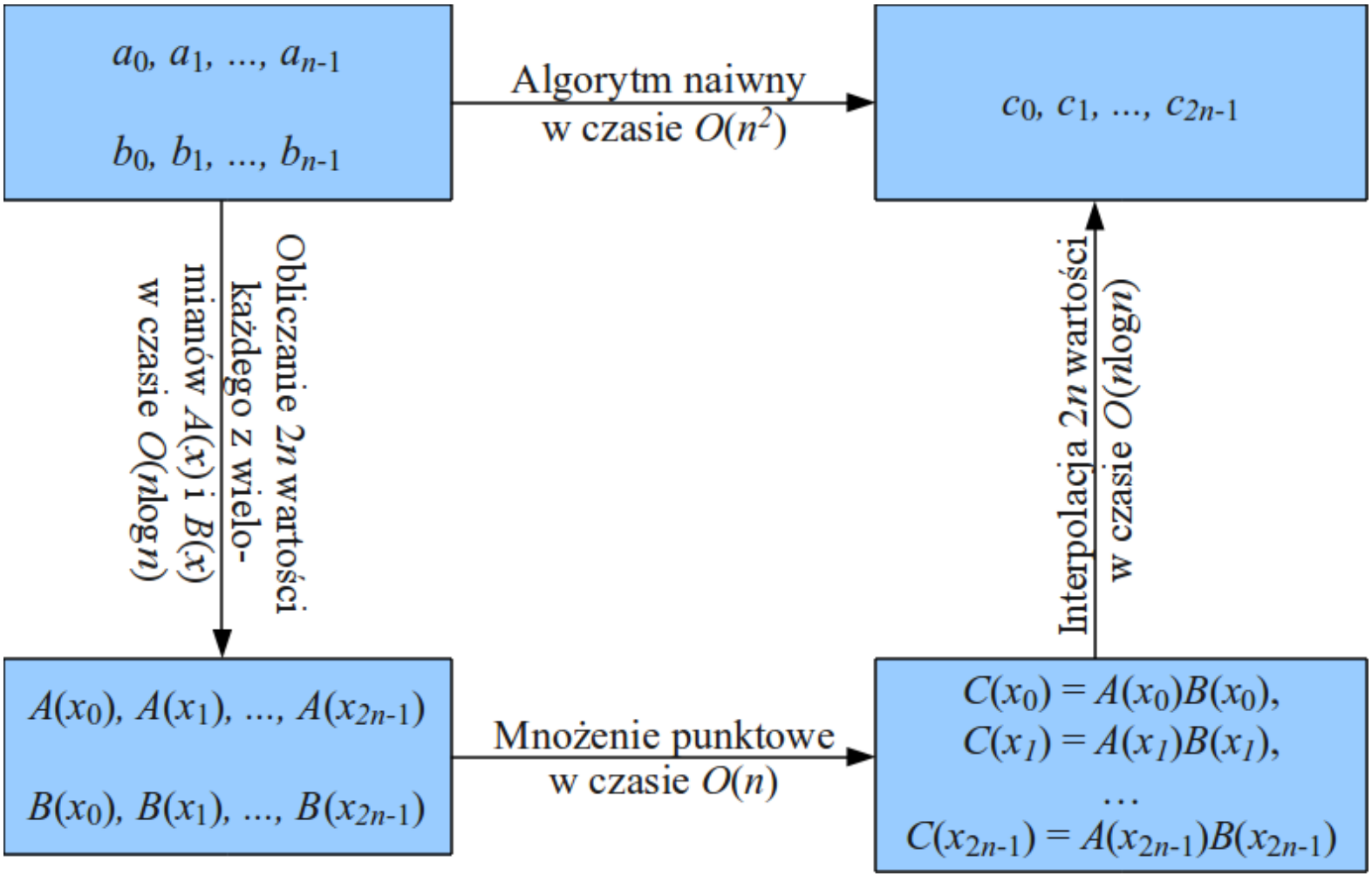
Twierdzenie (O Interpolacji)

Dla dowolnego zbioru $\{(x_0, y_0), \ldots, (x_{n-1}, y_{n-1})\}$ *takiego, że* x_i *są parami różne, istnieje dokładnie jeden wielomian* P *stopnia mniejszego niż* n *taki, że* $P(x_i) = y_i$ *dla każdego* $i = 0, \ldots, n-1$.

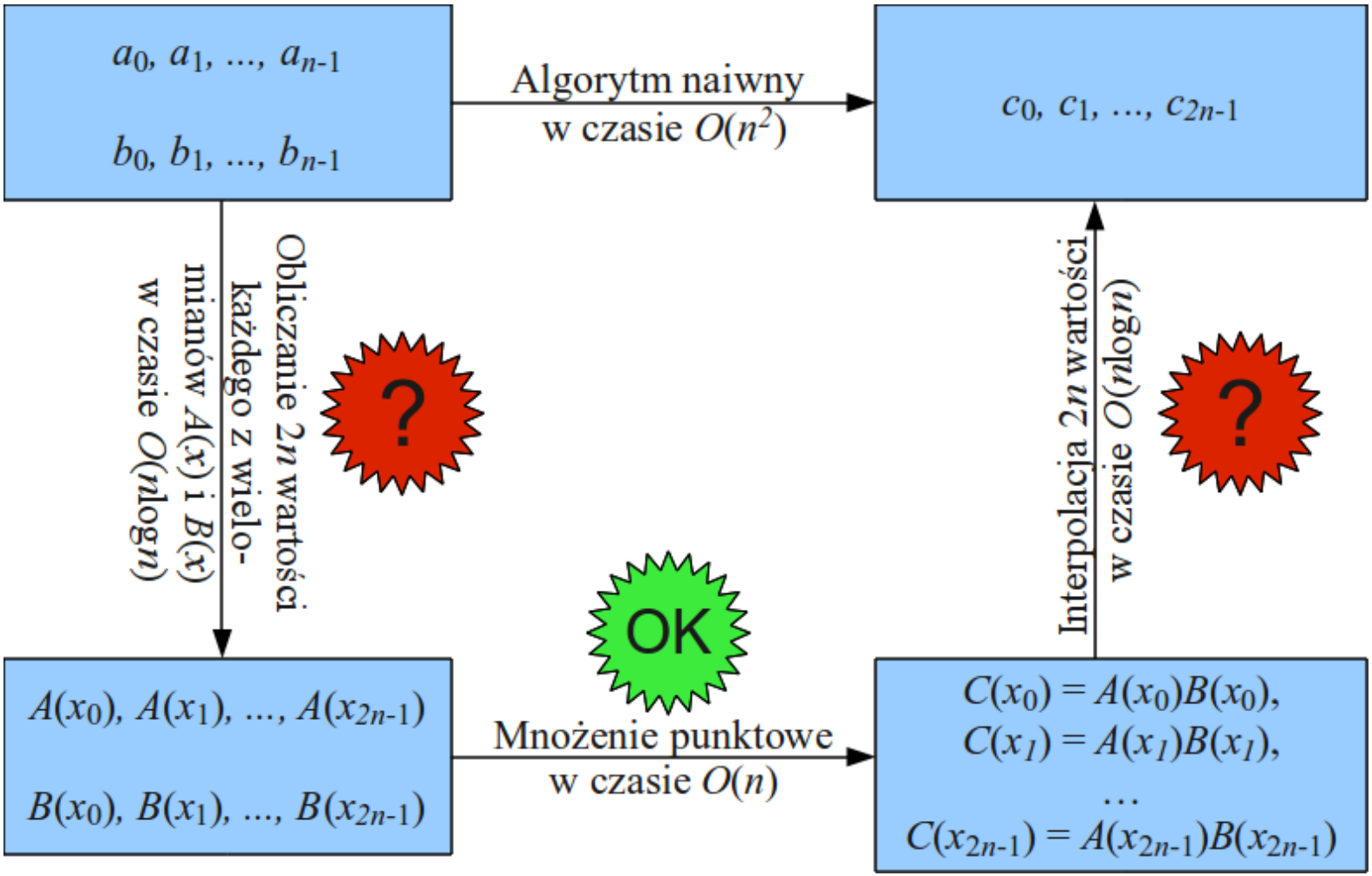
Reprezentacja przez wartości w $2n$ punktach

- n par (x_i, y_i) jednoznacznie wyznacza wielomian stopnia $< n$.
- $2n$ par tym bardziej.
- Niech A i B będą stopnia $< n$, dane jako $A = \{(x_0, y_0), \ldots, (x_{2n-1}, y_{2n-1})\}$ oraz $B = \{(x_0, y'_0), \ldots, (x_{2n-1}, y'_{2n-1})\}$.
- Wtedy $C = A \cdot B$ jest reprezentowany przez $\{(x_0, y_0 \cdot y'_0), \ldots, (x_{2n-1}, y_{2n-1} \cdot y'_{2n-1})\}$
- Obliczyliśmy C w czasie $O(n)$.

(Prawdziwe) mnożenie wielomianów w $O(n \log n)$



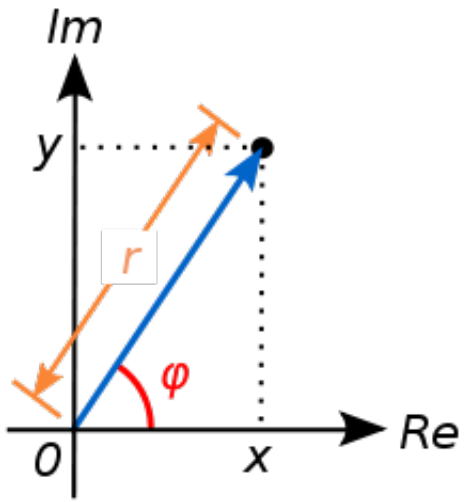
(Prawdziwe) mnożenie wielomianów w $O(n \log n)$

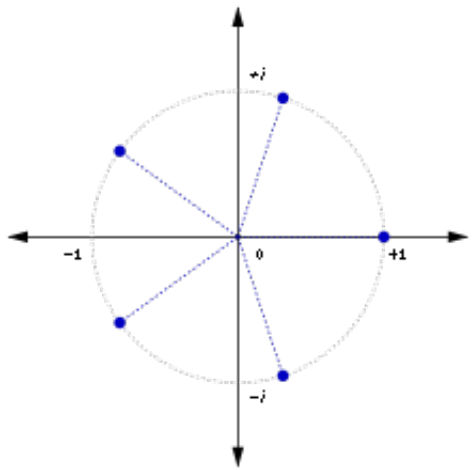


Liczby zepolone, pierwiastki z 1: przypomnienie

Liczby zespolone przedstawiamy jako:

- $z = x + yi = r(\cos\varphi + i\sin\varphi)$
- $z = r \cdot e^{\varphi i}$





- $\omega_n = e^{\frac{2\pi}{n}}$ jest pierwiastkiem stopnia n z 1.
- wszystkie pierwiastki stopnia n z 1: $\omega_n^0, \omega_n^1, \ldots \omega_n^{n-1}$.

DFT i FFT

Dyskretna Transformata Fouriera, DFT

Dany ciąg $(a_0, \ldots, a_{n-1})$ reprezentujący wielomian $A(x) = \sum_{i=0}^{n-1} a_i x^i$.
Obliczyć wartości $A(x_0), \ldots, A(x_{n-1})$, gdzie $x_j = \omega_n^j$ dla $j = 0, \ldots, n$.

Uwaga

W naszym zastosowaniu do mnożenia wielomianów stopnia $< d$, $n = 2d$ oraz $a_d = a_{d+1} = \ldots = a_{2d-1} = 0$.

- Pokażemy algorytm, który oblicza DFT używając $O(n \log n)$ operacji arytmetycznych (na liczbach zespolonych).
- Ten algorytm to szybka transformata Fouriera (FFT), odkryty przez Cooleya i Tuckeya w 1965 (właściwie wyważyli oni drzwi otwarte w 1805 przez Gaussa).

FFT: dziel i zwyciężaj

Załóżmy, że $n = 2k$ (wpp. dodajemy $a_n = 0$).
Określmy 2 wielomiany stopnia $< n/2$:

- $A^{[0]}(x) = a_0 + a_2 x + a_4 x^2 + \ldots + a_{n-2} x^{n/2-1}$,
- $A^{[1]}(x) = a_1 + a_3 x + a_5 x^2 + \ldots + a_{n-1} x^{n/2-1}$.

Wtedy:

- $A(x) = A^{[0]}(x^2) + xA^{[1]}(x^2)$,
- (tzn. wystarczy obliczyć n wartości 2 wielomianów stopnia $< n/2$.)
- fartownie, $(\omega_n)^2 = (e^{\frac{2\pi}{n} i})^2 = e^{\frac{2\pi}{n/2} i} = \omega_{n/2}$.
- stąd, dla $k = 0, \ldots, n/2$, mamy:
  $(\omega_n^k)^2 = \omega_{n/2}^k$ oraz $(\omega_n^{k+n/2})^2 = \omega_{n/2}^k \cdot \omega_{n/2}^{n/2} = \omega_{n/2}^k$.
- czyli $\{(\omega_n^0)^2, (\omega_n^1)^2, \ldots, (\omega_n^{n-1})^2\} = \{\omega_{n/2}^0, \omega_{n/2}^1, \ldots, \omega_{n/2}^{n/2-1}\}$
- **Rekurencyjnie** obliczamy $n/2$ wartości 2 wielomianów stopnia $< n/2$

FFT: dziel i zwyciężaj

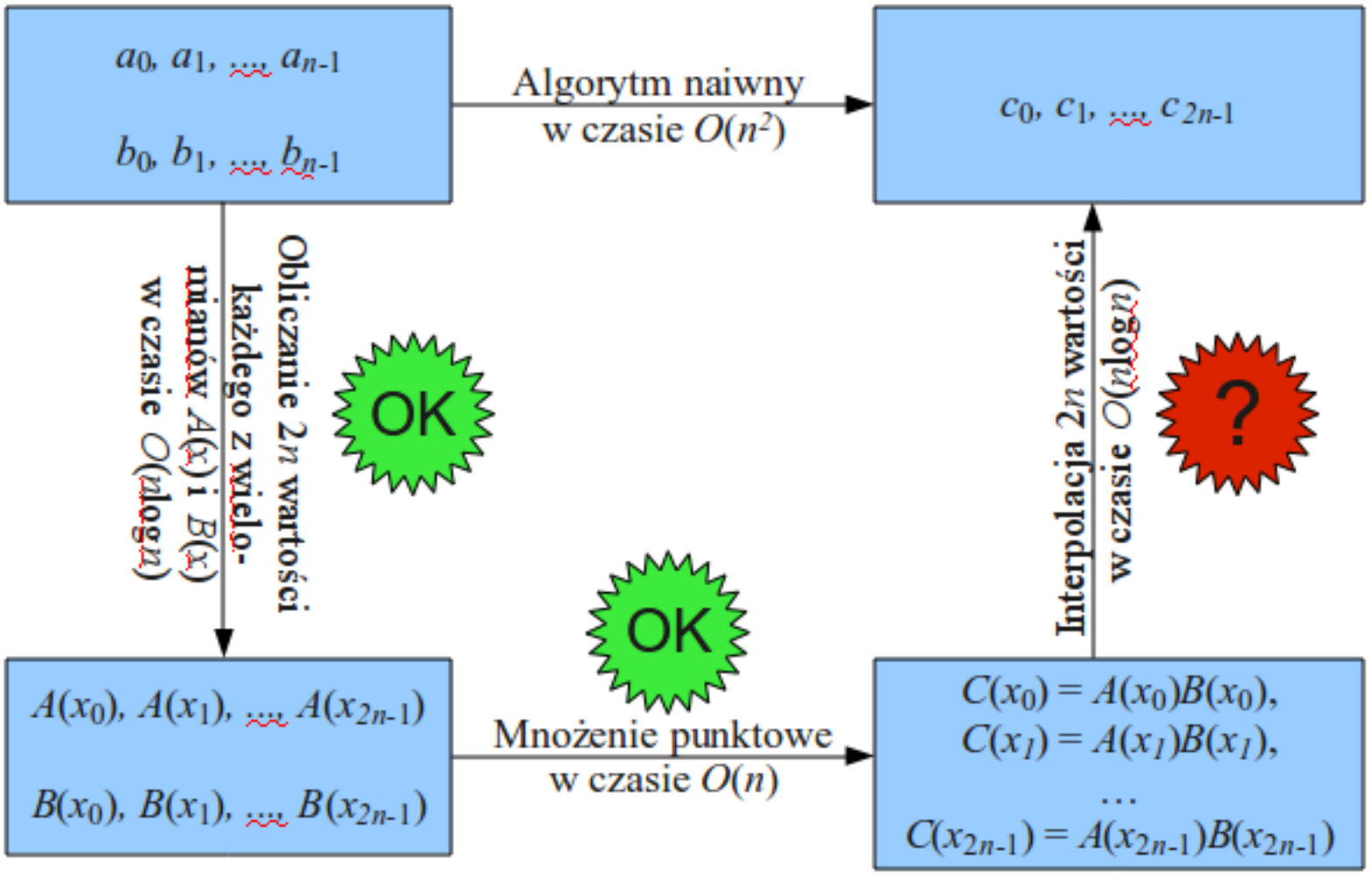
Wniosek

Algorytm FFT działa w czasie $O(n \log n)$.

Dowód

Bo $T(n) = 2T(n/2) + O(n)$.

Mnożenie wielomianów w $O(n \log n)$: Czego jeszcze brakuje?



Odwrotna Dyskretna Transformata Fouriera (IDFT)

Odwrotna Dyskretna Transformata Fouriera, IDFT

Dany ciąg $(y_0, \ldots, y_{n-1})$ reprezentujący wartości wielomianu $A(x) = \sum_{i=0}^{n-1} a_i x^i$ odpowiednio w punktach $1, \omega_n^1, \ldots, \omega_n^{n-1}$.
Wykonać interpolację, tzn. obliczyć wartości $a_0, \ldots, a_{n-1}$.

Chcemy znaleźć $a_0, \ldots, a_{n-1}$ takie, że:

$$
\begin{array}{rcccccccc}
y_0 & = & a_0 + & a_1 & + & a_2 & + \ldots + & a_{n-1}, \\
y_1 & = & a_0 + & a_1\omega_n & + & a_2\omega_n^2 & + \ldots + & a_{n-1}\omega_n^{n-1}, \\
y_2 & = & a_0 + & a_1\omega_n^2 & + & a_2\omega_n^4 & + \ldots + & a_{n-1}\omega_n^{2(n-1)}, \\
\vdots & & \vdots & & & & & \\
y_{n-1} & = & a_0 + & a_1\omega_n^{n-1} & + & a_2\omega_n^{2(n-1)} & + \ldots + & a_{n-1}\omega_n^{(n-1)(n-1)}.
\end{array}
$$

Odwrotna Dyskretna Transformata Fouriera (IDFT)

Odwrotna Dyskretna Transformata Fouriera, IDFT

Dany ciąg $(y_0, \ldots, y_{n-1})$ reprezentujący wartości wielomianu $A(x) = \sum_{i=0}^{n-1} a_i x^i$ odpowiednio w punktach $1, \omega_n^1, \ldots, \omega_n^{n-1}$.
Wykonać interpolację, tzn. obliczyć wartości $a_0, \ldots, a_{n-1}$.

Chcemy znaleźć $a_0, \ldots, a_{n-1}$ takie, że:

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n & \omega_n^2 & \omega_n^3 & \cdots & \omega_n^{n-1} \\ 1 & \omega_n^2 & \omega_n^4 & \omega_n^6 & \cdots & \omega_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{n-1} & \omega_n^{2(n-1)} & \omega_n^{3(n-1)} & \cdots & \omega_n^{(n-1)(n-1)} \end{bmatrix} \cdot \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

czyli $\mathbf{y} = V_n \cdot \mathbf{a}$, gdzie V_n jest macierzą Vandermonde'a
$\det V_n = \prod_{0 \le i < j \le n-1} (\omega_n^j - \omega_n^i) \neq 0$. Stąd, $\mathbf{a} = V_n^{-1}\mathbf{y}$.

Odwrotna Dyskretna Transformata Fouriera (IDFT), cd

Chcemy obliczyć $\mathbf{a} = V_n^{-1}\mathbf{y}$,
(V_n jest macierzą Vandermonde'a dla wartości $1, \omega_n, \omega_n^2, \ldots, \omega_n^{n-1}$.)

Fakt (dowód: obliczenie $V_n^{-1}V_n$, $V_nV_n^{-1}$, nuda.)

$(V_n^{-1})_{j,k} = \frac{1}{n}\,\omega_n^{-jk}$

Wniosek

$$a_j = \sum_{k=0}^{n-1} \frac{1}{n}\,\omega_n^{-jk} \cdot y_k = \frac{1}{n}\sum_{k=0}^{n-1} y_k(\omega_n^{-j})^k = \frac{1}{n}Y(\omega_n^{n-j}),$$

gdzie Y jest wielomianem o współczynnikach $(y_0, \ldots, y_{n-1})$.

Wniosek z wniosku

$\mathbf{a} = \frac{1}{n}(DFT(y_0, \ldots, y_{n-1}))^R$, gdzie operacja v^R odwraca wektor v, oprócz pierwszej składowej.
Czyli IDFT liczymy za pomocą FFT w czasie $O(n \log n)$.

Uwagi

- W algorytmie zakładaliśmy dokładną arytmetykę liczb rzeczywistych (zespolonych). W praktyce korzystamy z liczb o ograniczonej precyzji. Nie prowadzi to do dużych błedów (algorytm FFT ma bardzo dobre własności numeryczne)
- Jeśli współczynniki na wejściu są całkowite (więc na wyjściu chcemy również mieć całkowite) to możemy zaokrąglać/obcinać, ale należy przeprowadzić analizę jak dokładnej arytmetyki liczb rzeczywistych potrzebujemy.
- Algorytm FFT bardzo łatwo się zrównolegla: można go zaimplementować jako obwód arytmetyczny o głębokości $\log_2 n$.

Zastosowanie 1: Mnożenie

Powiedzmy, że chcemy pomnożyć przez siebie 2 liczby n-bitowe $a = \sum_{i=0}^{n-1} a_i 2^i$ i $b = \sum_{i=0}^{n-1} b_i 2^i$.

1. a i b reprezentujemy jako wielomiany $A(x)$, $B(x)$:
   - można po prostu jako $A(x) = \sum_{i=0}^{n-1} a_i x^i$, gdzie $a_i \in \{0, 1\}$
   - ale zwykle bardziej się opłaca pokroić liczby na grubsze bloki, ℓ-bitowe. Wtedy $A(x) = \sum_{i=0}^{\lceil n/\ell \rceil - 1} a_i x_i$, gdzie $a_i \in \{0, \ldots, 2^\ell - 1\}$.

Zastosowanie 1: Mnożenie

Powiedzmy, że chcemy pomnożyć przez siebie 2 liczby n-bitowe $a = \sum_{i=0}^{n-1} a_i 2^i$ i $b = \sum_{i=0}^{n-1} b_i 2^i$.

1. a i b reprezentujemy jako wielomiany $A(x)$, $B(x)$:
   - można po prostu jako $A(x) = \sum_{i=0}^{n-1} a_i x^i$, gdzie $a_i \in \{0, 1\}$
   - ~~ale zwykle bardziej się opłaca pokroić liczby na grubsze bloki, ℓ-bitowe. Wtedy $A(x) = \sum_{i=0}^{\lceil n/\ell \rceil - 1} a_i x_i$, gdzie $a_i \in \{0, \ldots, 2^\ell - 1\}$.~~
2. za pomocą FFT obliczamy $C(x) = A(x)B(x)$ (wówczas $c_j \in \{0, \ldots, n\}$).

Zastosowanie 1: Mnożenie

Powiedzmy, że chcemy pomnożyć przez siebie 2 liczby n-bitowe $a = \sum_{i=0}^{n-1} a_i 2^i$ i $b = \sum_{i=0}^{n-1} b_i 2^i$.

1. a i b reprezentujemy jako wielomiany $A(x)$, $B(x)$:
   - można po prostu jako $A(x) = \sum_{i=0}^{n-1} a_i x^i$, gdzie $a_i \in \{0, 1\}$
   - ~~ale zwykle bardziej się opłaca pokroić liczby na grubsze bloki, ℓ-bitowe. Wtedy $A(x) = \sum_{i=0}^{\lceil n/\ell \rceil - 1} a_i x_i$, gdzie $a_i \in \{0, \ldots, 2^\ell - 1\}$.~~
2. za pomocą FFT obliczamy $C(x) = A(x)B(x)$ (wówczas $c_j \in \{0, \ldots, n\}$).
3. wykonujemy przeniesienie za pomocą $O(n \log n)$ operacji na bitach
   - Przeniesienie p_i zawsze ma co najwyżej wartość n:
     - Na początku $\lfloor c_0/2 \rfloor \leq n$, czyli OK
     - Jeśli $p_i \leq n$ to $p_{i+1} = \lfloor (p_i + c_i)/2 \rfloor \leq n$.
   - Stąd, obliczenie i-tej cyfry wyniku wymaga $O(\log n)$ operacji bitowych.

Zastosowanie 1: Mnożenie, cd

Analiza

- Powiedzmy, że n bitów podzieliliśmy na $\approx 2^k$ bloków długości ℓ.
- Wówczas algorytm wykona się w czasie $O(n + k \cdot Mn/\ell)$, gdzie M to czas mnożenia dwóch liczb zespolonych.
- Powiedzmy, że liczby zespolone reprezentujemy na m bitach.
  Jak duże powinno być m żeby wynik mnożenia był poprawny?
- $m \geq 4k + 2\ell$ (patrz Knuth, t. II)

Wnioski – sytuacja praktyczna

- W praktyce dla „rozsądnych" danych (np. $n = 10^9$) wystarczy liczby zespolone reprezentować jako parę liczb typu double.
- Wówczas mnożenie dwóch liczb m-bitowych działa w czasie $O(1)$.
  Biorąc $\ell = k$ dostajemy algorytm w czasie $O(n)$.

Zastosowanie 1: Mnożenie, cd

Analiza

- Powiedzmy, że n bitów podzieliliśmy na $\approx 2^k$ bloków długości ℓ.
- Wówczas algorytm wykona się w czasie $O(n + k \cdot Mn/\ell)$, gdzie M to czas mnożenia dwóch liczb zespolonych.
- Powiedzmy, że liczby zespolone reprezentujemy na m bitach. Jak duże powinno być m żeby wynik mnożenia był poprawny?
- $m \geq 4k + 2\ell$ (patrz Knuth, t. II)

Teoria

- (Schönhage-Strassen 1971) W powyższej analizie weźmy $\ell = k$. Mamy rekurencję $T(n) = O(nT(\log n))$, stąd $T(n) = O(n \log n \log\log n \log\log\log n \cdots)$
- (Schönhage-Strassen 1971) Zamiast ciała $\mathbb{C}$ bierzemy pierścień $\mathbb{Z}_{2^e+1}$ dla pewnego e. Prowadzi to do czasu $O(n \log n \log\log n)$.
- (Fürer 2007) algorytm $O(n \log n 2^{\log^* n})$.

Zastosowanie 2: Dodawanie zbiorów

Problem

Dane dwa zbiory $A, B \subset \{0, \ldots, n\}$,
Znaleźć $C = \{a + b \ : \ a \in A, b \in B\}$.

Rozwiązanie w czasie $O(n \log n)$

- Weźmy $A(x) = \sum_{a \in A} x^a$, $B(x) = \sum_{b \in B} x^b$.
- Obliczamy $C(x) = A(x)B(x)$.
- Jeśli $C(x) = \sum_{j=0}^{2n} c_j x^j$ to $C = \{j \ : \ c_j \neq 0\}$.

Szybkie mnożenie macierzy

Mnożenie macierzy (kwadratowych)

Problem

Dane macierze $n \times n$: A i B.
Znaleźć macierz $C = A \cdot B$.

Algorytm naiwny (wg zwykłego wzoru)

$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$.
Czas: $O(n^3)$ operacji arytmetycznych.

Mnożenie macierzy: Dziel i zwyciężaj (1)

Bez straty ogólności $n = 2^k$.
Podzielmy $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ na podmacierze o wymiarach $(n/2) \times (n/2)$:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}$$

Wówczas

$$\mathbf{C} = \left[\begin{array}{c|c} \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1} & \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2} \\ \hline \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1} & \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2} \end{array}\right]$$

Mamy rekurencję $T(n) = 8\,T(n/2) + O(n^2)$ czyli $T(n) = O(n^3)$.
(Dominujący jest ostatni poziom, gdzie jest $8^{\log_2 n} = n^3$ węzłów.)

Mnożenie macierzy: Dziel i zwyciężaj (2)

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}$$

Drugie podejście (Strassen 1969):

$$\mathbf{M}_1 := (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2}) \qquad \mathbf{M}_2 := (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1}$$
$$\mathbf{M}_3 := \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2}) \qquad \mathbf{M}_4 := \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1})$$
$$\mathbf{M}_5 := (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2} \qquad \mathbf{M}_6 := (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2})$$
$$\mathbf{M}_7 := (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2}).$$

Wtedy:

$$\mathbf{C} = \left[\begin{array}{c|c} \mathbf{A}_{1,1}\mathbf{B}_{1,1} + \mathbf{A}_{1,2}\mathbf{B}_{2,1} & \mathbf{A}_{1,1}\mathbf{B}_{1,2} + \mathbf{A}_{1,2}\mathbf{B}_{2,2} \\ \hline \mathbf{A}_{2,1}\mathbf{B}_{1,1} + \mathbf{A}_{2,2}\mathbf{B}_{2,1} & \mathbf{A}_{2,1}\mathbf{B}_{1,2} + \mathbf{A}_{2,2}\mathbf{B}_{2,2} \end{array}\right]$$
$$= \left[\begin{array}{c|c} \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7 & \mathbf{M}_3 + \mathbf{M}_5 \\ \hline \mathbf{M}_2 + \mathbf{M}_4 & \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6 \end{array}\right]$$

Mamy rekurencję $T(n) = 7T(n/2) + O(n^2)$ czyli
$T(n) = O(7^{\log_2 n}) = O(n^{\log_2 7}) = O(n^{2.81})$.

Kilka faktów podanych bez dowodu

- Najszybszy znany algorytm mnożenia macierzy Coppersmitha i Winograda (1990) działa w czasie $O(n^{2.38})$ (jest kompletnie niepraktyczny).
- Najlepsze znane dolne ograniczenie to $\Omega(n^2)$.
- Ciekawe wyniki dla macierzy prostokątnych, np. jeśli $r \leq 0.294$ to macierz $n \times n^r$ można pomnożyć przez macierz $n^r \times n$ w czasie $O(n^{2+o(1)})$.

Niech $M(n)$ to czas mnożenia macierzy $n \times n$.
Wiemy, że $M(n) = O(n^\omega)$, gdzie $\omega < 2.38$.

- Można znaleźć odwrotność macierzy w czasie $O(M(n))$.
- Można obliczyć wyznacznik macierzy w czasie $O(M(n))$.

Dowody dwóch ostatnich faktów można znaleźć w podręczniku Cormena.

Zastosowanie 1: liczba marszrut

Lemat

Niech $\mathbf{A}$ będzie macierzą sąsiedztwa n-wierzchołkowego grafu G (skierowanego lub nieskierowanego). Dla $k \in \mathbb{N}_{>0}$. Wówczas dla dowolnego $i, j = 1, \ldots, n$ element $\mathbf{A}^k_{i,j}$ zawiera liczbę marszrut długości k od wierzchołka i do wierzchołka j

Dowód

Indukcja. Dla $k = 1$ OK.

Dla $k > 1$ mamy:

- $\mathbf{A}^{k-1}_{i,\ell} \cdot \mathbf{A}_{\ell,j}$ jest liczbą marszrut długości k od i do j, w których przedostatni wierzchołek to ℓ.
- Stąd, $\mathbf{A}^k_{i,j} = \sum_{\ell=1}^{n} \mathbf{A}^{k-1}_{i,\ell} \cdot \mathbf{A}_{\ell,j}$ jest liczbą wszystkich marszrut długości k od i do j.

Wniosek Liczbę marszrut długości k między wszystkimi parami wierzchołków możemy policzyć w czasie $O(n^\omega \log k)$.

Szybkie mnożenie macierzy: zastosowania

Oznaczmy przez $O(n^\omega)$ najlepszy znany czas mnożenia macierzy $n \times n$.

1. Rozwiązywanie układów równań w $O(n^\omega)$
2. Wyszukiwanie trójkątów w grafie w $O(n^\omega)$
3. Domknięcie przechodnie w $O(n^\omega \log n)$
4. Sprawdzenie (ew. znajdowanie) czy graf zawiera skojarzenie doskonałe w czasie (n^ω).

MAX-SAT (Williams 2004)

Problem MAX-SAT

Dana formuła ϕ w postaci 2-CNF, zawierająca n zmiennych. Znaleźć wartościowanie zmiennych, które maksymalizuje liczbę spełnionych klauzul.

Złożoność

Odpowiedni problem decyzyjny jest NP-zupełny.
Algorytm naiwny ma złożoność $O(2^n)$
Pytanie: Czy można szybciej? Np. $O(1.9^n)$?

MAX-SAT (Williams 2004)

Problem MAX-SAT

Dana formuła ϕ w postaci 2-CNF, zawierająca n zmiennych. Znaleźć wartościowanie zmiennych, które maksymalizuje liczbę spełnionych klauzul.

Złożoność

Odpowiedni problem decyzyjny jest NP-zupełny.
Algorytm naiwny ma złożoność $O(2^n)$
Pytanie: Czy można szybciej? Np. $O(1.9^n)$?

Będziemy się zajmować równoważnym (z dokładnością do czynnika log(#klauzul)) problemem:

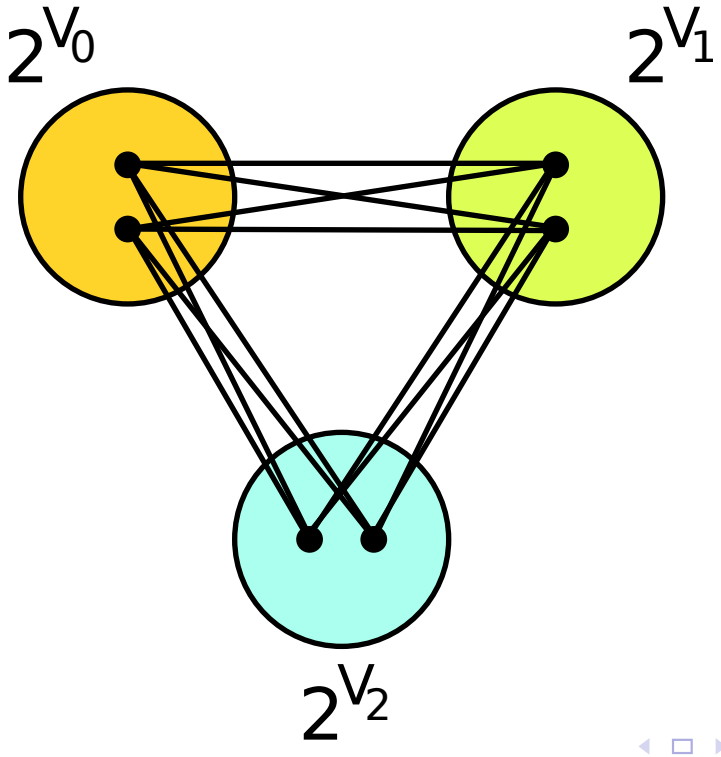
Problem MAX-SAT, wersja testująca

Dana formuła ϕ w postaci 2-CNF, zawierająca n zmiennych oraz $k \in \mathbb{N}$
Czy istnieje wartościowanie zmiennych, dla którego jest dokładnie k spełnionych klauzul.

MAX-SAT (Williams 2004)

Zbudujemy pewien graf G o $O(2^{n/3})$ wierzchołkach.

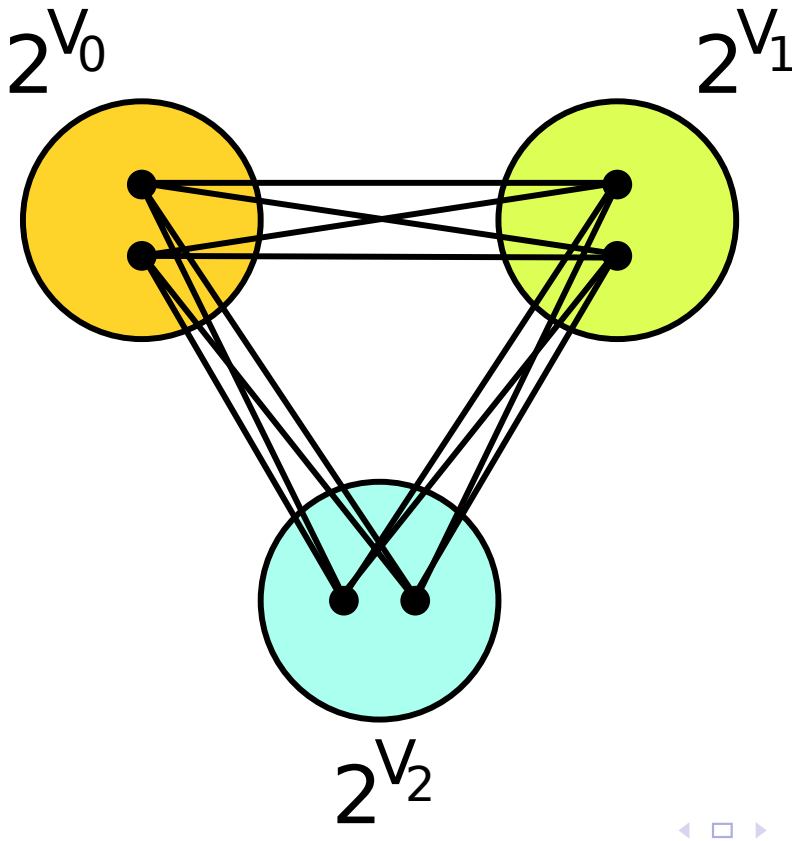
- Ustalmy dowolny podział $V = V_0 \cup V_1 \cup V_2$ na trzy równe części (tak równe jak się da).
- Wierzchołkami G są wszystkie wartościowania $v_i : V_i \to \{0, 1\}$ dla $i = 0, 1, 2$.
- Dla dowolnych $v \in V_i$, $w \in V_{(i+1) \bmod 3}$ graf G zawiera krawędź vw.



MAX-SAT (Williams 2004)

Idea rozwiązania

- Dobierzemy tak wagi na krawędziach, że waga trójkąta vwu w G będzie równa liczbie spełnionych klauzul przy wartościowaniu (v, w, u).
- Wtedy wystarczy sprawdzić, czy istnieje trójkąt o wadze k w G.



MAX-SAT (Williams 2004)

Idea rozwiązania

- Dobierzemy tak wagi na krawędziach, że waga trójkąta vwu w G będzie równa liczbie spełnionych klauzul przy wartościowaniu (v, w, u).
- Wtedy wystarczy sprawdzić, czy istnieje trójkąt o wadze k w G.

Problem 1 Jak dobrać wagi?
Niech $c(v) =$ wszystkie klauzule, które są spełnione przy wartościowaniu v.
Wtedy liczba spełnionych klauzul przy wartościowaniu (v, w, u) wynosi:

$$\begin{aligned}|c(v) \cup c(w) \cup c(u)| = |c(v)| + |c(w)| + |c(u)| \\ - |c(v) \cap c(w)| - |c(v) \cap c(u)| - |c(w) \cap c(u)| \\ + |c(v) \cap c(w) \cap c(u)|.\end{aligned}$$

MAX-SAT (Williams 2004)

Idea rozwiązania

- Dobierzemy tak wagi na krawędziach, że waga trójkąta vwu w G będzie równa liczbie spełnionych klauzul przy wartościowaniu (v, w, u).
- Wtedy wystarczy sprawdzić, czy istnieje trójkąt o wadze k w G.

Problem 1 Jak dobrać wagi?
Niech $c(v) =$ wszystkie klauzule, które są spełnione przy wartościowaniu v.
Wtedy liczba spełnionych klauzul przy wartościowaniu (v, w, u) wynosi:

$$\begin{aligned}|c(v) \cup c(w) \cup c(u)| = {} & |c(v)| + |c(w)| + |c(u)| \\ & - |c(v) \cap c(w)| - |c(v) \cap c(u)| - |c(w) \cap c(u)| \\ & + \underbrace{|c(v) \cap c(w) \cap c(u)|}_{0}.\end{aligned}$$

MAX-SAT (Williams 2004)

Idea rozwiązania

- Dobierzemy tak wagi na krawędziach, że waga trójkąta vwu w G będzie równa liczbie spełnionych klauzul przy wartościowaniu (v, w, u).
- Wtedy wystarczy sprawdzić, czy istnieje trójkąt o wadze k w G.

Problem 1 Jak dobrać wagi?
Niech $c(v) =$ wszystkie klauzule, które są spełnione przy wartościowaniu v.
Wtedy liczba spełnionych klauzul przy wartościowaniu (v, w, u) wynosi:

$$\begin{aligned}|c(v) \cup c(w) \cup c(u)| = {} & |c(v)| + |c(w)| + |c(u)| \\ & - |c(v) \cap c(w)| - |c(w) \cap c(u)| - |c(u) \cap c(v)| \\ & + \underbrace{|c(v) \cap c(w) \cap c(u)|}_{0}.\end{aligned}$$

Czyli dajemy $\text{waga}(xy) = |c(x)| - |c(x) \cap c(y)|$.

MAX-SAT (Williams 2004)

Pozostało sprawdzić, czy istnieje trójkąt o wadze k w G.

Trick

Rozważamy wszystkie $O(m^2) = O(n^4)$ podziałów ($m =$ liczba klauzul) $k = k_0 + k_1 + k_2$. Dla każdego podziału budujemy graf G_{k_0,k_1,k_2} złożony tylko z:

- krawędzi o wadze k_0 między 2^{V_0} a 2^{V_1},
- krawędzi o wadze k_1 między 2^{V_1} a 2^{V_2},
- krawędzi o wadze k_2 między 2^{V_2} a 2^{V_0}.

Wtedy wystarczy...

MAX-SAT (Williams 2004)

Pozostało sprawdzić, czy istnieje trójkąt o wadze k w G.

Trick

Rozważamy wszystkie $O(m^2) = O(n^4)$ podziałów ($m =$ liczba klauzul) $k = k_0 + k_1 + k_2$. Dla każdego podziału budujemy graf G_{k_0,k_1,k_2} złożony tylko z:

- krawędzi o wadze k_0 między 2^{V_0} a 2^{V_1},
- krawędzi o wadze k_1 między 2^{V_1} a 2^{V_2},
- krawędzi o wadze k_2 między 2^{V_2} a 2^{V_0}.

Wtedy wystarczy... sprawdzić, czy istnieje dowolny trójkąt.

Sprawdzanie, czy G_{k_0,k_1,k_2} zawiera trójąt

Wniosek

- Graf G_{k_0,k_1,k_2} ma $3 \cdot 2^{n/3}$ wierzchołków.
- Możemy sprawdzić, czy G_{k_0,k_1,k_2} zawiera trójąt w czasie $O(2^{\omega n/3}) = O(1.732^n)$
- Czyli możemy sprawdzić, czy G zawiera trójąt o wadze k w czasie $O(n^4 \cdot 2^{\omega n/3}) = O(n^4 \cdot 1.732^n) = O(1.733^n)$

MAX-SAT (Williams 2004): Podsumowanie

Wniosek

Możemy rozwiązać MAX-SAT w czasie i pamięci $O(1.733^n)$.

MAX-SAT (Williams 2004): Podsumowanie

Wniosek

Możemy rozwiązać MAX-SAT w czasie i pamięci $O(1.733^n)$.

Łatwo przerobić nasz algorytm (jak?) żeby dostać

Wniosek

Możemy zliczyć wszystkie rozwiązania optymalne MAX-SAT w czasie i pamięci $O(1.733^n)$.