Mnożenie macierzy

IIUWr. II rok informatyki. Przygotował: Krzysztof Loryś

1 Metoda Strassena

Problem:

- Dane są dwie macierze A i B o rozmiarach $n \times n$, elementów z pierścienia R.
- Chcemy obliczyć $C = A \cdot B$.

Idea.
Stosujemy strategię Dziel i Rządź:
Dzielimy macierze A, B i C na cztery podmacierze o rozmiarze $\frac{n}{2} \times \frac{n}{2}$ każda (dla prostoty zakładamy, że n jest potęgą liczby 2):

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \qquad C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix},$$

a następnie obliczamy niezależnie podmacierze C_{ij}.

Podmacierze te możemy obliczyć korzystając z oczywistych wzorów: $\forall_{i=1,2;\, j=1,2}\ C_{ij} = A_{i1} \cdot B_{1j} + A_{i2} \cdot B_{2j}$. Czyli jedno mnożenie macierzy $n \times n$ zastępujemy ośmioma mnożeniami macierzy $\frac{n}{2} \times \frac{n}{2}$. Tak otrzymany algorytm ma niestety złożoność $\Omega(n^3)$, czyli taką samą jak tradycyjny algorytm.

Kluczem do przyśpieszenia algorytmu jest zredukowanie liczby mnożeń macierzy $\frac{n}{2} \times \frac{n}{2}$ z ośmiu do siedmiu.

Algorytm *Metoda Strassena*

1. Jeśli n jest małe oblicz iloczyn $A \cdot B$ metodą tradycyjną.
   W przeciwnym przypadku:
2. Oblicz 7 pomocniczych macierzy m_i o rozmiarze $\frac{n}{2} \times \frac{n}{2}$.

$$\begin{aligned} m_1 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22}) \\ m_2 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\ m_3 &= (A_{11} - A_{21}) \cdot (B_{11} + B_{12}) \\ m_4 &= (A_{11} + A_{12}) \cdot B_{22} \\ m_5 &= A_{11} \cdot (B_{12} - B_{22}) \\ m_6 &= A_{22} \cdot (B_{21} - B_{11}) \\ m_7 &= (A_{21} + A_{22}) \cdot B_{11} \end{aligned}$$

3. Oblicz składowe podmacierze C_{ij} macierzy wynikowej C.

$$\begin{aligned} C_{11} &= m_1 + m_2 - m_4 + m_6 \\ C_{12} &= m_4 + m_5 \\ C_{21} &= m_6 + m_7 \\ C_{22} &= m_2 - m_3 + m_5 - m_7 \end{aligned}$$

Twierdzenie 1 *Powyższy algorytm oblicza iloczyn dwóch dowolnych macierzy $n \times n$ o elementach z dowolnego pierścienia za pomocą $O(n^{\log_2 7}) \approx O(n^{2.807})$ operacji arytmetycznych na elementach tego pierścienia.*

UWAGI:

- Stała skryta pod "dużym O" czyni algorytm Strassena niepraktycznym dla macierzy małych rozmiarów oraz dla macierzy specjalnych postaci, dla których opracowano szybkie algorytmy mnożenia (np. dla macierzy rzadkich).
- Nie można zmniejszyć liczby mnożeń macierzy 2×2 do 6-iu.
- Można otrzymać szybszy algorytm dokonując podziału macierzy na większą liczbę cześci. Przykładowo, Victor Pan pokazał, że dokonując podziału macierze o wymiarach 70×70 można pomnożyć wykonując 143640 mnożeń skalarnych (zamiast 70^3 wykonywanych w tradycyjnej metodzie). Oparta na tym fakcie metoda dziel i zwyciężaj daje algorytm o złożoności $O(n^{log_{70} 143640}) \approx O(n^{2.795})$
- Obecnie najszybszy asymptotycznie algorytm mnożenia macierzy ([1]) działa w czasie $O(n^{2.3716})$.
- Nieznane jest obecnie ograniczenie dolne na złożoność problemu mnożenia macierzy lepsze niż trywialne ograniczenie $\Omega(n^2)$.

2 Mnożenie macierzy logicznych

2.1 Metoda wykorzystująca metodę Strassena

Metody Strassena nie można wykorzystać bezpośrednio do mnożenia macierzy logicznych, ponieważ $\langle\{\mathbf{0}, \mathbf{1}\}; \vee, \wedge\rangle$ nie stanowi pierścienia. Istnieje jednak prosty sposób obejścia tego problemu:
Traktujemy macierze logiczne A i B jako macierze nad $\mathcal{Z}_{n+1}$ ($\mathbf{0}$ i $\mathbf{1}$ traktujemy odpowiednio jako 0 i 1 z $\mathcal{Z}_{n+1}$). Mnożymy A i B w $\mathcal{Z}_{n+1}$ (tj. zastępując $\vee$ przez sumę modulo $(n+1)$ a $\wedge$ przez iloczyn modulo $(n+1)$. Jeśli tak otrzymany iloczyn oznaczymy przez C a iloczyn logiczny przez D to mamy:

Fakt 1 $\forall_{1 \leq i,j \leq n}\ D[i,j] = \mathbf{1}$ *iff* $C[i,j] \neq 0$.

KOSZT:

- $O(n^{2.81})$ operacji arytmetycznych w $\mathcal{Z}_{n+1}$.
- $O(n^{2.81} \log(n) \log\log(n) \log\log\log(n))$ operacji na bitach.
  UZASADNIENIE: Dodawanie i odejmowanie liczb k-bitowych wymaga $O(k)$ operacji bitowych, zaś mnożenie metodą Schönchagego-Strassena - $O(k \log k \log\log k)$ operacji bitowych. My wszystkie operacje wykonujemy na elementach z $\mathcal{Z}_n$, a więc na liczbach $\log n$ bitowych.

2.2 Metoda czterech Rosjan

Metoda Czterech Rosjan daje algorytm o nieco gorszej złożoności od wyżej opisanej metody. Jest ona jednak atrakcyjna ze względu na możliwość wykorzystania operacji na wektorach bitów (realizowalnych hardware'owo), co wydatnie zwiększa jej praktyczną szybkość.

IDEA:
Macierz A dzielimy na $\frac{n}{\log n}$ podmacierzy A_i o rozmiarze $n \times \log n$ każda, a macierz B na $\frac{n}{\log n}$ podmacierzy B_i o rozmiarze $\log n \times n$ każda (zakładamy dla prostoty opisu, że $\log n$ jest liczbą całkowitą dzielącą n). Podmacierz A_i (B_i) składa się z kolejnych kolumn (wierszy) macierzy A (macierzy B) o numerach od $(\log n)(i-1)+1$ do $(\log n)i$. Łatwo sprawdzić, że $\forall_{i=1,\ldots,n/\log n}\ A_i \cdot B_i$ jest macierzą $n \times n$ oraz że

$$A \cdot B = \sum_{i=1}^{n/\log n} A_i \cdot B_i.$$

Kluczowym trickiem jest metoda obliczania iloczynów $A_i \cdot B_i$ w czasie $O(n^2)$.

SPOSTRZEŻENIE:

1. Jeśli j-ty wiersz macierzy A_i składa się z samych zer, to j-ty wiersz macierzy C_i również składa się z samych zer.
2. Jeśli wiersze j_1 i j_2 macierzy A_i różnią się tylko na pozycji k-tej, przy czym wiersz j_1 ma na tej pozycji zero, to wiersz j_2 macierzy C_i jest równy sumie logicznej wiersza j_1 macierzy C_i oraz wiersza k macierzy B_i.

Wiersze macierzy A_i są wektorami z $\{\mathbf{0},\mathbf{1}\}^{\log n}$. Różnych takich wektorów jest n. Na podstawie Spostrzeżenia, wszystkie iloczyny postaci $\mathbf{x} \cdot B_i$, gdzie $\mathbf{x} \in \{\mathbf{0},\mathbf{1}\}^{\log n}$, można obliczyć w czasie $O(n^2)$.

2.2.1 Algorytm

OZNACZENIA:

- $\mathbf{b}_s^i$ - s-ty wiersz macierzy B_i,
- $\mathbf{a}_j$ - j-ty wiersz macierzy A,
- jeśli $\mathbf{v} \in \{\mathbf{0},\mathbf{1}\}^n$, to $\tilde{\mathbf{v}}$ - odpowiadający mu wektor w $\{0,1\}^n$, a $num(\mathbf{v}) = \sum_{i=1}^{n} \tilde{\mathbf{v}}_i 2^{n-i}$.

Algorytm *CzterechRosjan*

procedure $LogMatMult(A, B : array[1..n, 1..n]\ of\ boolean)$
  for $i \leftarrow 1$ **to** $\frac{n}{\log n}$ **do**
    $Rowsum[0] \leftarrow [\underbrace{0,0,\ldots,0}_{n \text{ razy}}]$
    for $j \leftarrow 1$ **to** n **do**
      $k \leftarrow \max\{l \mid j \geq 2^l\}$
      $Rowsum[j] \leftarrow Rowsum[j-2^k] + \mathbf{b}_{k+1}^i$ {sumowanie po współrzędnych}
    niech C_i będzie macierzą, której j-ty wiersz
    jest równy $Rowsum[num(\mathbf{a_j})]$ $(j = 1,\ldots,n)$
  return $C = \sum_{i=1}^{\frac{n}{\log n}} C_i$

KOMENTARZ: Wektory z $\{\mathbf{0},\mathbf{1}\}^{\log n}$ utożsamiamy z liczbami $\log n$-bitowymi. Dzięki temu łatwo możemy takimi wektorami indeksować tablicę *Rowsum*. Dla ustalonego i, $Rowsum[j]$ będzie pamiętał iloczyn wektora odpowiadającego liczbie j oraz macierzy B_i. Jak łatwo sprawdzić, wektory odpowiadające j oraz $j - 2^k$ różnią się tylko na jednej pozycji i $Rowsum[j]$ możemy obliczyć przez dodanie k-ego wiersza macierzy B_i $Rowsum[j - 2^k]$.

Fakt 2 *Powyższy algorytm oblicza $C = A \cdot B$ w czasie $O(n^3/\log n)$. Ponadto można go zaimplementować tak, by wymagał $O(n^2/\log n)$ operacji na wektorach bitów.*

3 Inne operacje macierzowe

Dalej nie czytać!!!!

Wiele operacji macierzowych można zredukować do problemu mnożenia macierzy.

3.1 Odwracanie macierzy

3.2 Obliczanie wyznacznika

Jeśli macierz jest osobliwa, wykryje to algorytm dokonujący jej rozkładu LUP. W przeciwnym razie $det(A) = det(LUP) = det(L)det(U)det(P)$. Ponieważ L jest jednostkową macierzą tójkątna jej wyznacznik jest równy 1. Wyznacznik macierzy U jest równy iloczynowi elementów na jej przekątnej, wiec może być obliczony w czasie $\Theta(n)$. Wyznacznk macierzy permutacyjnej jest zawsze równy $\pm$, tak więc by obliczyć wartość $det(P)$ wystarczy sprawdzić, czy P reprezentuje parzystą, czy nieparzystaą permutację.

4 Zastosowanie operacji macierzowych w algorytmice

Literatura

[1] V. Vassilevska Williams, Y.Xu, Z. Xu, R. Zhou: New Bounds for Matrix Multiplication: from Alpha to Omega. w: *Proceedings of 35th SODA*, 2024, s. 3792-3835.