Teoria współbieżności

Laboratorium 7

Teoria śladów

Wielowątkowa eliminacja Gausa

Patryk Zajdel

11.01.2024

Spis treści

1. Teoria

Zadanie polega na zaprojektowaniu współbieżnego algorytmu eliminacji Gaussa. Problem rozpatry- wany będzie dla macierzy współczynników **M** o rozmiarze **N** × **N**, **N**-elementowego wektora wyrazów wolnych **y** oraz **N**-elementowego wektora niewiadomych **x**:

$$M \cdot x = y$$

Dla N = 3, problem wygląda następująco

$$\begin{bmatrix} M_{1,1} & M_{1,2} & M_{1,3} \\ M_{2,1} & M_{2,2} & M_{2,3} \\ M_{3,1} & M_{3,2} & M_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Dla uproszczenia zapisu, przyjmijmy:

$$M = (M \mid y) = \left[\begin{array}{ccc|c} M_{1,1} & M_{1,2} & M_{1,3} & y_1 \\ M_{2,1} & M_{2,2} & M_{2,3} & y_2 \\ M_{3,1} & M_{3,2} & M_{3,3} & y_3 \end{array}\right] = \left[\begin{array}{ccc|c} M_{1,1} & M_{1,2} & M_{1,3} & M_{1,4} \\ M_{2,1} & M_{2,2} & M_{2,3} & M_{2,4} \\ M_{3,1} & M_{3,2} & M_{3,3} & M_{3,4} \end{array}\right]$$

1.1. Niepodzielne operacje

Wyróżniamy następujące rodzaje operacji na elementach macierzy:

- $A_{i,k}$ - znalezienie $\boldsymbol{m_{k,i}}$ - mnożnika dla wiersza i, do odejmowania go od k-tego wiersza.

$$M_{k,i}/M_{i,i} \rightarrow \boldsymbol{m_{k,i}}$$

- $B_{i,j,k}$ - znalezienie $\boldsymbol{n_{k,i,j}}$ – j-tego elementu wiersza i-tego pomnożonego przez $\boldsymbol{m_{k,i}}$.

$$M_{i,j} * m_{k,i} \rightarrow \boldsymbol{n_{k,i,j}}$$

- $C_{i,j,k}$ - odjęcie $\boldsymbol{n_{k,i,j}}$ od j-tego elementu wiersza k-tego.

$$M_{k,j} - n_{k,i,j} \rightarrow \boldsymbol{M_{k,j}}$$

1.2. Alfabet w sensie teorii śladów

Niech $\boldsymbol{t}$ - rozmiar macierzy (liczba wierszy)
Zdefiniujmy najpierw podzbiory alfabetu zawierające czynności tego samego typu:

$$\Sigma_A = \{A_{i,j} \mid 1 \leq i < j \leq \boldsymbol{t}\}$$

$$\Sigma_B = \{B_{i,j,k} \mid 1 \leq i < \boldsymbol{t},\ i \leq j \leq \boldsymbol{t+1},\ i < k \leq \boldsymbol{t}\}$$
$$\Sigma_C = \{C_{i,j,k} \mid 1 \leq i < \boldsymbol{t},\ i \leq j \leq \boldsymbol{t+1},\ i < k \leq \boldsymbol{t}\}$$

Alfabet w sensie teorii śladów jest sumą powyższych zbiorów:

$$\Sigma = \Sigma_A \cup \Sigma_B \cup \Sigma_C$$

1.3. Relacja niezależności i zależności

Wyznaczmy teraz relację zależności D. W tym celu patrzymy, która "zmienna" zapisywana w danej operacji jest używana w innych.

$$\delta_1 = \{(A_{i,j}, B_{l,m,n}) \in \Sigma_A \times \Sigma_B \mid i = l \wedge j = n\}$$

$$\delta_2 = \{(B_{i,j,k}, C_{l,m,n}) \in \Sigma_B \times \Sigma_C \mid i = l \wedge j = m \wedge k = n\}$$

$$\delta_3 = \{(C_{i,j,k}, A_{l,m}) \in \Sigma_C \times \Sigma_A \mid i = l - 1 \wedge j = l \wedge (k = l \vee k = m)\}$$

$$\delta_4 = \{(C_{i,j,k}, B_{l,m,n}) \in \Sigma_C \times \Sigma_B \mid l \neq m \wedge i = l - 1 \wedge j = m \wedge k = l\}$$

$$\delta_5 = \{(C_{i,j,k}, C_{l,m,n}) \in \Sigma_C \times \Sigma_C \mid l \neq m \wedge i = l - 1 \wedge j = m \wedge k = n\}$$

Wyznaczamy teraz ostateczny zbiór relacji zależności:

$$D = sym((\delta_1 \cup \delta_2 \cup \delta_3 \cup \delta_4 \cup \delta_5)^+) \cup I_\Sigma$$

Relację niezależności wyznaczamy w oparciu o relację zależności:

$$I = \Sigma^2 - D$$

2. Opis algorytmu eliminacji Gaussa

Skonstrujemy ciąg operacji odpowiadający algorytmowi eliminacji.
Niech $\boldsymbol{O_{i,k}}$ oznacza odjęcie **k**-tego wiersza od wiersza **i**-tego z pominięciem elementów na kolumnach **j**-tych, gdzie **j** < **i**. Kolumny te powinny już być odpowiednio wyzerowane.

$$O_{i,k} = (A_{i,k}, B_{i,j,k}, C_{i,i,k}, B_{i,i+1,k}, C_{i,i+1,k}, \ldots, B_{i,t+1,k}, C_{i,t+1,k})$$

Niech $\boldsymbol{Z_i}$ oznacza wyzerowanie wartości w **i**-tej kolumnie pod przekątną (może skutkować przekształceniami także poza **i**-tą kolumną:

$$Z_i = (O_{i,i+1}, O_{i,i+2}, \ldots, O_{i,t})$$

Wówczas algorytm eliminacji Gaussa możemy przedstawić jako ciąg:

$$G_t = (Z_1, Z_2, \ldots, Z_{t-1})$$

Przykładowo, dla $\mathbf{t} = 3$, algorytm wygląda następująco:

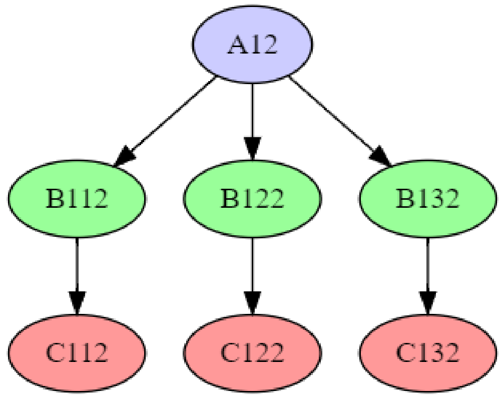
$$G_3 = (Z_1, Z_2) = (O_{1,2}, O_{1,3}, O_{2,3})$$
$$O_{1,2} = (A_{1,2}, B_{1,1,2}, C_{1,1,2}, B_{1,2,2}, C_{1,2,2}, B_{1,3,2}, C_{1,3,2}, B_{1,4,2}, C_{1,4,2})$$
$$O_{1,3} = (A_{1,3}, B_{1,1,3}, C_{1,1,3}, B_{1,2,3}, C_{1,2,3}, B_{1,3,3}, C_{1,3,3}, B_{1,4,3}, C_{1,4,3})$$
$$O_{2,3} = (A_{2,3}, B_{2,2,3}, C_{2,2,3}, B_{2,3,3}, C_{2,3,3}, B_{2,4,3}, C_{2,4,3})$$

3. Graf Diekerta

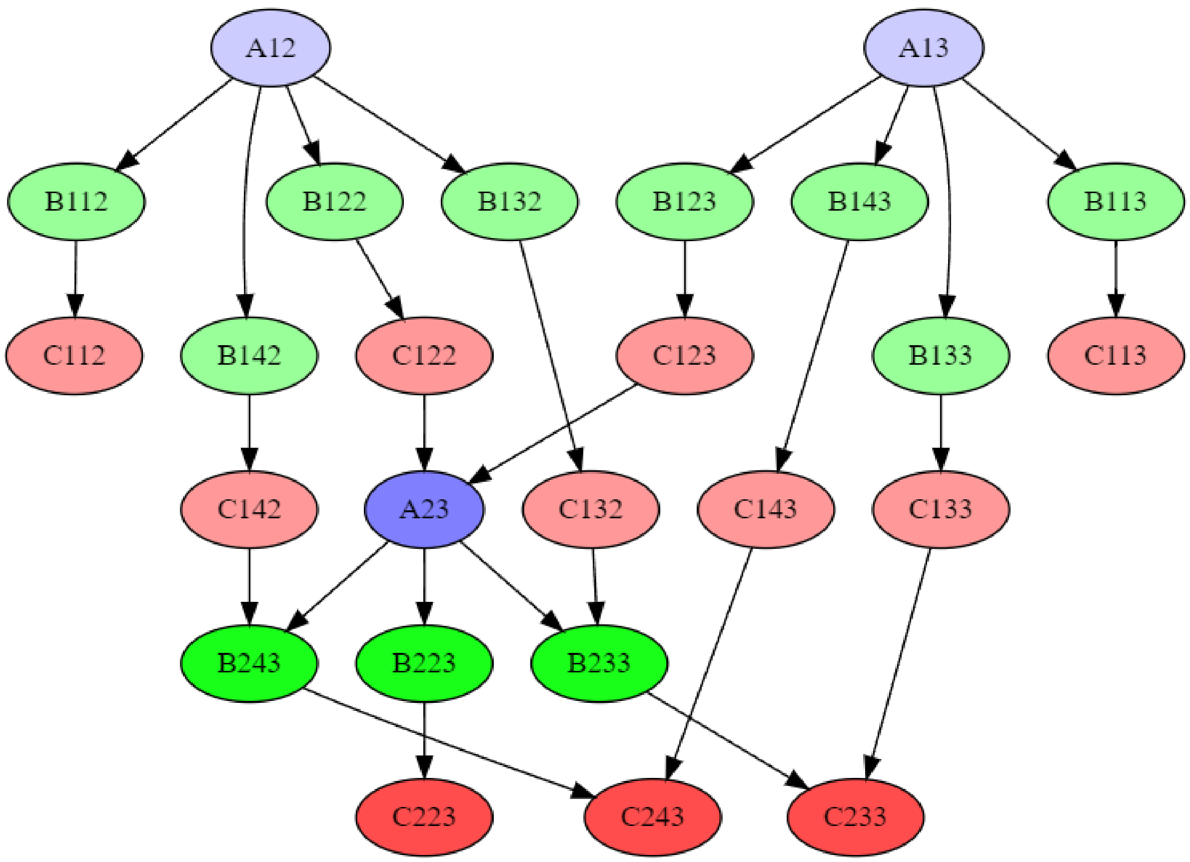
Podzbiory $D_1, \ldots, D_5$ zostały wyznaczone w taki sposób, by były ze sobą rozłączne. Oznacza to, że zawierają tylko zależności bezpośrednie. Oprócz tego, skierowane są zgodnie z postępowaniem algorytmu. Wobec tego, zbiory te wyznaczają krawędzie w grafie Diekerta:

$$E = D_1 \cup D_2 \cup D_3 \cup D_4 \cup D_5$$

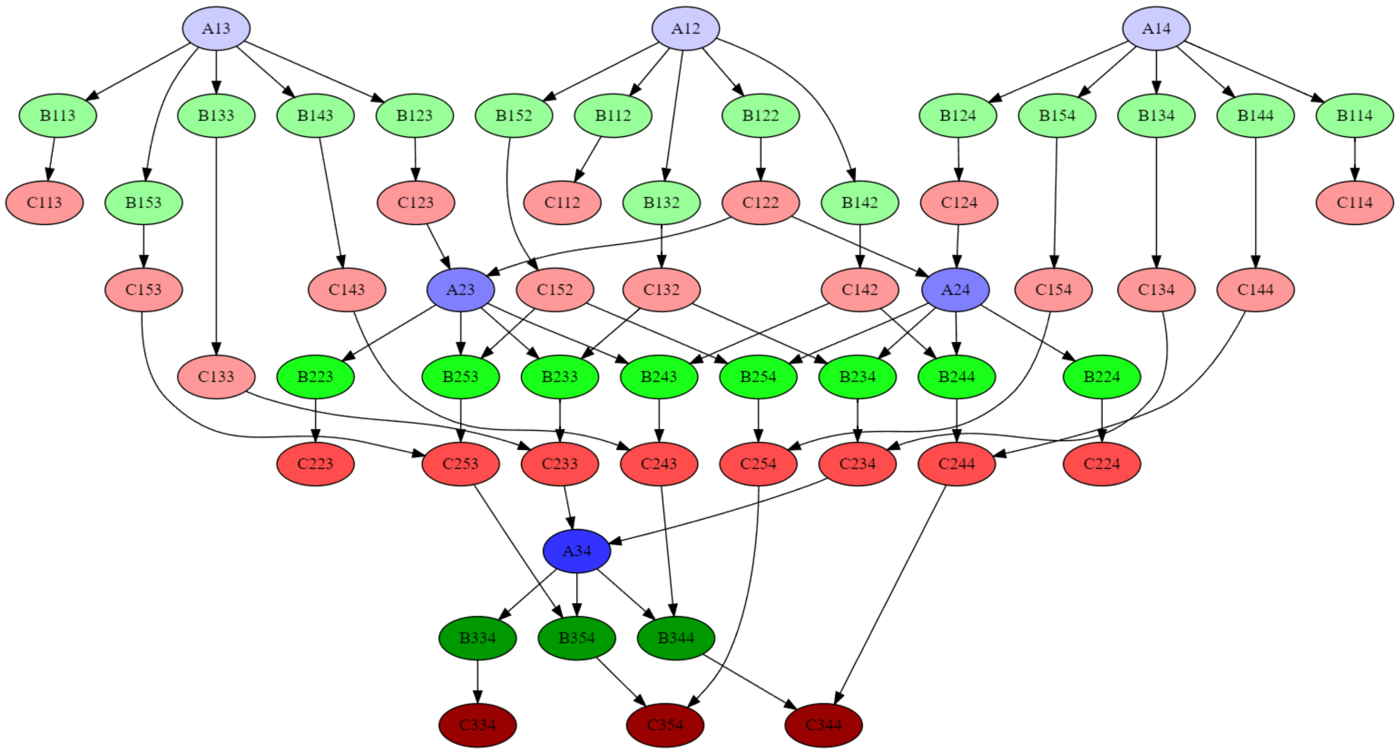
W języku Python napisano program (gauss_diekert.py) generujący graf Diekerta dla zadanego t. Wykorzystano pakiet graphviz. Graf generowany jest w formacie dot. Poniżej przedstawiono wygenerowane grafy dla $\mathbf{t} = 2$, $\mathbf{t} = 3$ i $\mathbf{t} = 4$. Grafiki zostały wygenerowane za pomocą Graphviz Online



Rysunek 1: Graf Diekerta współbieżnego algorytmu Gaussa dla t = 2. Kolorami oznaczono poszczególne klasy Foaty



Rysunek 2: Graf Diekerta współbieżnego algorytmu Gaussa dla t = 3. Kolorami oznaczono poszczególne klasy Foaty



Rysunek 3: Graf Diekerta współbieżnego algorytmu Gaussa dla t = 4. Kolorami oznaczono poszczególne klasy Foaty.

4. Klasy Foaty

Na powyższych rysunkach można łatwo zaobserwować podział na poszczególne klasy Foaty. Podział ten zilustrowano kolorami wierzchołków. Przyjmijmy oznaczenia pomocnicze:

$$F_{A,x} = \{A_{i,j} \in \Sigma_A \mid i = x\}$$
$$F_{B,x} = \{B_{i,j,k} \in \Sigma_B \mid i = x\}$$
$$F_{C,x} = \{C_{i,j,k} \in \Sigma_C \mid i = x\}$$

Postać normalną Foaty można teraz zdefiniować następująco:

$$FNF_t = [F_{A,1}][F_{B,1}][F_{C,1}][F_{A,2}][F_{B,2}][F_{C,2}]\dots[F_{A,t-1}][F_{B,t-1}][F_{C,t-1}]$$

Przykładowo, dla $\mathbf{t} = 3$, FNF wygląda następująco:

$$FNF_3 = [A_{1,2}\ A_{1,3}] \tag{1}$$
$$[B_{1,1,2}\ B_{1,2,2}\ B_{1,3,2}\ B_{1,4,2}\ B_{1,1,3}\ B_{1,2,3}\ B_{1,3,3}\ B_{1,4,3}] \tag{2}$$
$$[C_{1,1,2}\ C_{1,2,2}\ C_{1,3,2}\ C_{1,4,2}\ C_{1,1,3}\ C_{1,2,3}\ C_{1,3,3}\ C_{1,4,3}][A_{2,3}] \tag{3}$$
$$[B_{2,2,3}\ B_{2,3,3}\ B_{2,4,3}][C_{2,2,3}\ C_{2,3,3}\ C_{2,4,3}] \tag{4}$$

5. Implementacja

Współbieżny algorytm Gaussa zaimplementowano w języku C++. Algorytm działa zgodnie z wyprowadzoną w poprzednim punkcie postacią normalną Foaty. Różni się on jedynie indeksacją – w mojej implementacji indeksy rozpoczynają się od 0. Dodatkowo, jeżeli jest taka konieczność, wykonywany jest pivoting. Na koniec wykonywane jest podstawianie wstecz. Jako wynik uzyskiwana jest macierz wypełniona zerami, z jedynkami na przekątnej. Wektor wyrazów wolnych zawiera wartości poszczególnych niewiadomych. Implementacja znajduje się w katalogu *concurrent_gauss*.

6. Zawartość katalogu

Za generowanie grafu Diekerta odpowiedzialny jest plik *gauss_diekert.py*. Implementacja współbieżnego algorytmu Gaussa znajduje się w katalogu *concurrent_gauss*. W środku znajdują się następujące pliki:

- *Makefile* – służy do uruchamiania. Na Linuxie można użyć GNU Make i komendy *make*, natomiast na Windowsie – mingw i komendy *mingw32-make*. Domyślnie program uruchamiany jest z argumentami *input.txt* i *output.txt* określającymi kolejno plik wejściowy i wyjściowy. Program oczywiście można uruchomić z dowolnymi argumentami (po skompilowaniu: ./main [input] [output]).
- *input.txt* – przykładowe wejście
- *main.cpp* – główny plik projektu. Wykorzystuje klasę GaussMatrix
- *gauss_matrix.h* i *gauss_matrix.cpp* – pliki zawierające implementację klasy GaussMatrix. Znajduje się w niej główna funkcjonalność projektu. Metody actionA, actionB i actionC wykonują zdefiniowane wcześniej taski. Metoda concurrentGauss wykorzystuje powyższe akcje do wykonania współbieżnej eliminacji Gaussa.