# Sprawozdanie
# „Współbieżna eliminacja Gaussa”
# Teoria Współbieżności

**gr. śr 17:50**

**24.11.2021**

Wykonał: Grzegorz Legęza (401501)

## 1. Cel ćwiczenia

Zadanie polega na wykonaniu współbieżnej eliminacji Gaussa. W tym celu najpierw należy zastosować teorie śladów do stworzenia odpowiedniej kolejności wykonywania zadań. Następnie należy zaimplementować program wykonujący eliminację Gaussa współbieżnie zgodnie z postacią normalną Foaty. Eliminacje przeprowadzamy na macierzy NxN.

## 2. Notacja

Najpierw określmy jak wygląda problem do rozwiązania. Mamy macierz kwadratową o rozmiarze N oraz wektor wyrazów wolnych. Macierz wraz z wektorem zapisujemy do jednej macierzy o rozmiarze N x (N+1) oraz wprowadzamy odpowiednie oznaczenia pól tej macierzy. Pole leżące w i-tym wierszu i j-tej kolumnie będziemy oznaczać jako $M_{i,j}$. Macierz wygląda następująco:

$$\begin{pmatrix} M_{1,1} & M_{1,2} & \cdots & M_{1,N} & M_{1,N+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ M_{N,1} & M_{N,2} & \cdots & M_{N,N} & M_{N,N+1} \end{pmatrix}$$

Wektor wyrazów wolnych to < $M_{1,N+1}$, $M_{2,N+1}$, $M_{3,N+1}$, …, $M_{N,N+1}$,>.

## 3. Analiza zadań

Napiszę pseudo-kod realizujący eliminację Gaussa dla powyższej macierzy i określę zadania.

(a) for i in 1 to N:
(b)   for k in i+1 to N:
(c)     m(i, k) = $M_{k,i}$ / $M_{i,i}$
(d)     for j in i to N+1:
(e)       d(i, j, k) = $M_{i,j}$ * m(i, k)
(f)       $M_{k,j}$ -= d(i, j, k)

Widzimy, że wykonuje się trzy rodzaje niezależnych zadań. Nazwę je i opiszę poniżej.

**$A_{i,k}$** (linijka c) – wyznaczenie mnożnika dla odejmowania od k-tego wiersza i-tego wiersza.
**$B_{i,j,k}$** (linijka e) – wymnożenie j-tego elementu z wiersza i z mnożnikiem z zadania $A_{i,k}$.
**$C_{i,j,k}$** (linijka f) – odjęcie obliczonej wartości w zadaniu $B_{i,j,k}$ od $M_{k,j}$.

Wyznaczyliśmy trzy klasy zadań A, B i C. Po opisie od razu zauważamy, że zadanie B korzysta z wyniku zadania A, natomiast zadanie C z wyniku zadania B.

## 4. Alfabet w sensie teorii śladów

Analizując pseudokod z punktu 3 możemy zapisać alfabet naszego zadania.

$$\Sigma = \left\{A_{i,k}, B_{i,j,k}, C_{i,j,k} : i \in \{1, \dots, N-1\};\ j \in \{i, \dots, N+1\}; k \in \{i+1, \dots, N\}\right\}$$

Ograniczenia dla indeksów wynikają z następujących faktów:

- i –> i wskazuje na indeks lewego górnego rogu podmacierzy w trakcie wykonywania eliminacji Gaussa, i nie osiąga N bo dla jedno elementowej macierzy nie wykonujemy eliminacji.
- j –> j wskazuje na numer kolumny w podmacierzy w której wykonujemy eliminację, zaczynamy od i bo tam zaczyna się podmacierz i sięgamy do końca macierzy, a więc do N+1.
- k –> k to numer wiersza od którego odejmujemy i-ty wiersz, wiersze te znajdują się pod i-tym więc dlatego indeks przebiega od i+1, przechodzimy do końca więc do N.

## 5. Słowo w sensie teorii śladów

Postać słowa zapiszę za pomocą pseudokodu, który je generuje.

```
(a) ω = empty list
(b) for i in 1 to N:
(c)   for k in i+1 to N:
(d)     ω.append(A_{i,k})
(e)     for j in i to N+1:
(f)       ω.append(B_{i,j,k})
(g)       ω.append(C_{i,j,k})
(h) return ω
```

Możemy zauważyć pewną intuicję jak to jest generowane. Najpierw pętla po i wyznacza nam podmacierz na której działamy. Następnie dla każdej takiej podmacierzy przechodzimy po jej wierszach od i+1 do końca macierzy. Dla każdego wiersza obliczamy mnożnik i odejmujemy wiersz i przemnożony przez ten mnożnik. Kluczowa jest obserwacja podziału zadania na podmacierze, ponieważ będę z niej korzystał podczas wyznaczania relacji zależności.

## 6. Relacje zależności i niezależności

Zajmę się określeniem relacji zależności poprzez wyznaczenie kilku zbiorów, które w sumie dadzą nam relację zależności. Zacznę od najprostszych relacji które wynikają wprost z określenie trzech klas zadań. W punkcie 3 zauważyłem, że zadanie $B_{i,j,k}$

korzysta z liczby obliczonej w zadaniu $A_{i,k}$, a zadanie $C_{i,j,k}$ korzysta z liczby obliczonej w zadaniu $B_{i,j,k}$. Z tej obserwacji wynikają następujące relacje.

$$D_1 = \{(A_{i,k}, B_{i,j,k}) : i \in \{1, \dots, N-1\};\ j \in \{i, \dots, N+1\}; k \in \{i+1, \dots, N\}\}$$
$$D_2 = \{(B_{i,j,k}, C_{i,j,k}) : i \in \{1, \dots, N-1\};\ j \in \{i, \dots, N+1\}; k \in \{i+1, \dots, N\}\}$$

$D_1$ mówi, że aby przemnożyć przez mnożnik najpierw potrzebujemy obliczyć mnożnik. $D_2$ pokazuje, że aby odjąć odpowiednią wartość, najpierw trzeba ją obliczyć.

Aby wyznaczyć kolejne relacje zależności zastanówmy się nad tym problemem podobnie jak w zadaniu z licznikami. Określę dla każdego zadania jakie elementy tablicy potrzebują i jakie zmieniają. Najpierw przypomnę równania występujące w eliminacji Gaussa.

- $A_{i,k}$ –> $m(i, k) = M_{k,i} / M_{i,i}$
- $B_{i,j,k}$ –> $d(i, j, k) = M_{i,j} * m(i, k)$
- $C_{i,j,k}$ –> $M_{k,j} = M_{k,j} - d(i, j, k)$

Z równań tych wprost wynikają zależności ze zbiorów $D_1$ i $D_2$, biorą się one z obliczanych wartości m i d. Z racji że każda wartość m i d z konkretnymi indeksami jest obliczana tylko raz i zależności z nią związane są już wypisane, więc nie będziemy się już nimi zajmować. Na podstawie równań uzupełnię tabelkę.

| Zadanie | Elementy odczytywane | Elementy zmieniane |
|---|---|---|
| $A_{i,k}$ | $M_{k,i}$, $M_{i,i}$ | - |
| $B_{i,j,k}$ | $M_{i,j}$ | - |
| $C_{i,j,k}$ | $M_{k,j}$ | $M_{k,j}$ |

A więc widzimy, że w każdej parze zadań zależnych jedno z nich będzie należało do klasy C, ponieważ tylko w tej klasie zmieniana jest wartość w macierzy. Relacja zachodzi pomiędzy zadaniem z klasy C a zadaniem z klasy X jeśli zadanie X odczytuje element zmieniony przez zadanie z klasy C. Elementem zmienianym jest zawsze $M_{k,j}$. Zastanówmy się więc nad trzema możliwościami.
Relacja pomiędzy C i A.
Zadanie $A_{i,k}$ odczytuje pole $M_{k,i}$, a więc mamy następujące pary:

$$D_3 = \{(C_{i,j,k}, A_{j,k}) : i \in \{1, \dots, N-1\};\ j \in \{i, \dots, N+1\}; k \in \{i+1, \dots, N\}\}$$

Zadanie $A_{i,k}$ odczytuje pole $M_{i,i}$, a więc mamy następujące pary:

$$D_4 = \left\{(C_{i,k,k}, A_{k,t}) : i \in \{1, \dots, N-1\};\ k \in \{i+1, \dots, N-1\};\ t \in \{k+1, \dots, N\}\right\}$$

Relacja pomiędzy C i B.
Zadanie $B_{i,j,k}$ odczytuje pole $M_{i,j}$, a więc mamy następujące pary:

$$D_5 = \left\{ \begin{matrix} (C_{i,j,k}, B_{k,j,t}) : i \in \{1, \dots, N-1\}; k \in \{i+1, \dots, N-1\}; \\ j \in \{k, \dots, N+1\};\ t \in \{k+1, \dots, N\} \end{matrix} \right\}$$

Relacja pomiędzy C i C.

Zadanie $C_{i,j,k}$ odczytuje pole $M_{k,i}$, a więc mamy następujące pary (indeks t większy równy i, bo pozostałe załatwia nam symetria):

$$D_6 = \left\{ \begin{matrix} (C_{i,j,k}, C_{t,j,k}) : i \in \{1, \dots, N-1\};\ t \in \{i, \dots, N-1\}; \\ j \in \{\mathrm{t}, \dots, N+1\}; k \in \{\mathrm{t}+1, \dots, N\} \end{matrix} \right\}$$

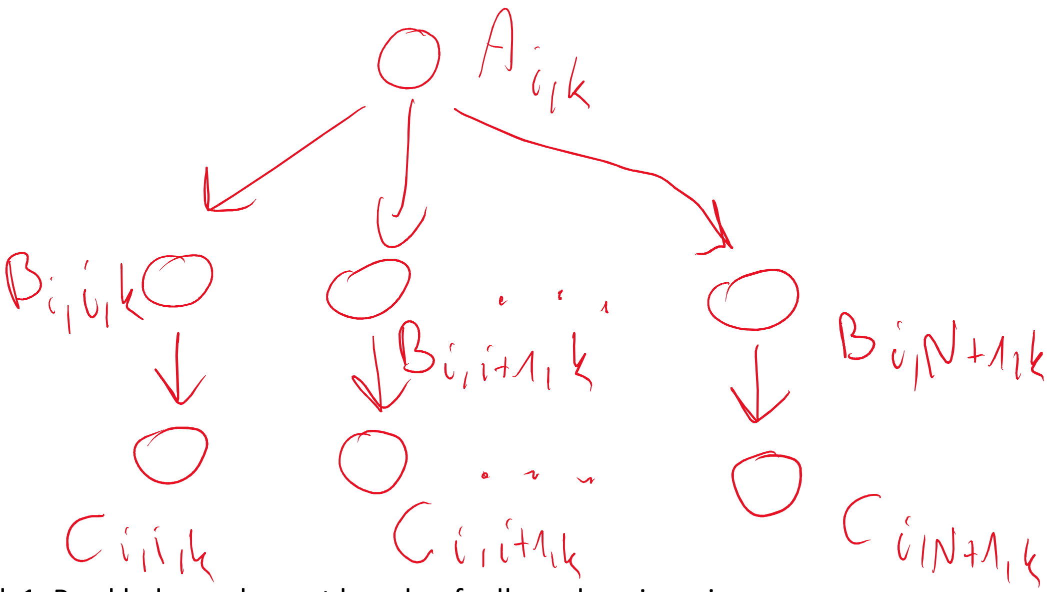
Rozważyliśmy wszystkie możliwości a więc ostatecznie otrzymujemy.

$$D = sym((D_1 \cup D_2 \cup D_3 \cup D_4 \cup D_5 \cup D_6)^+) \ \cup \ id_\Sigma$$
$$I = \Sigma^2 \setminus D$$

## 7. Graf zależności

Aby poprawnie zredukować krawędzie do postaci grafu Dickerta potrzebujemy zauważyć kilka istotnych faktów. Moje obserwacje będą oparte na podmacierzach naszego problemu, a pojęcie to wprowadziłem wyżej. Z oczywistych względów krawędzie powstałe poprzez przechodniość nie interesują nas w minimalnym grafie.

Najpierw będę chciał pokazać, że zadania wykonywane na danej podmacierzy tworzą pewien podgraf, dla którego jeśli zastosujemy krawędzie wynikające ze zbioru $D_1$ i $D_2$ to krawędzi tych nie da się zredukować w grafie zależności.




Rysunek 1: Przykładowy element k podgrafu dla podmacierzy i.

Podgraf ten składa się z elementów takich jak w rysunku 1 dla kolejnych k. Jeden taki element dla ustalonego k możemy zapisać formalnie.

$$E_{i,k} = \{(A_{i,k}, B_{i,j,k}), (B_{i,j,k}, C_{i,j,k}) : j \in \{i, \dots, N+1\}\}$$

Natomiast cały podgraf to suma po tych elementach.

$$P_i = \bigcup_{k=i+1}^{N} E_{i,k}$$

Zauważmy, że w słowie ω graf ten jest prezentowany jako spójna część, więc wszystkie ścieżki rozpoczynające się w naszym podgrafie po wyjściu z niego już do niego nie wracają. Więc widzimy, że na pewno nie możemy tego podgrafu zredukować. Pytanie więc nasuwa się, czy nie pominęliśmy jakiejś krawędzi w tym podgrafie. Patrząc na kolejność tych zadań w słowie, rozpatrzymy kolejne możliwości łączenia krawędzi z różnych klas:

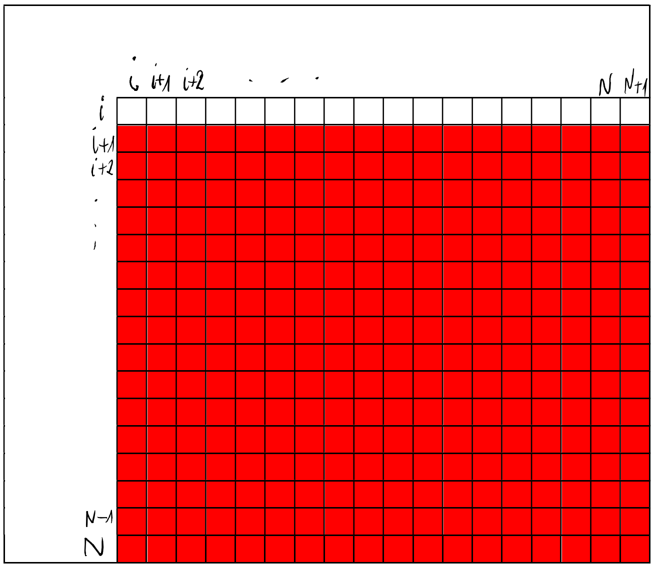
- AxA – zadania A w danej podmacierzy są miedzy sobą niezależne (odczytują wspólnie jedno pole, które się nie zmienia).
- BxB – niezależne między sobą, odczytują pola, które się nie zmieniają w trakcie działania dla danej podmacierzy.
- CxC – każde z zadań C działa na innym polu macierzy. Dodatkowo patrząc na zbiór $D_6$ widzimy że występujące trójki indeksów nie dają pary należącej do tego zbioru.
- AxB – wszystkie krawędzie wynikające z tej relacji są.
- AxC – relacja zachodzi pomiędzy $A_{i,k}$ a $C_{i,i,k}$; krawędź ta jest redukowalna bo mamy ścieżkę poprzez $B_{i,i,k}$.
- BxC - patrząc na $D_5$ widzimy, że relacja pomiędzy B i C zachodzi tylko gdy mają tą samą drugą współrzędną i różne współrzędne pierwsze a taka sytuacja nie zachodzi dla jednej podmacierzy.

Wynika nam z tego że nasz zredukowany graf będzie zbudowany z N-1 podgrafów dla każdej podmacierzy połączonym pomiędzy sobą krawędziami. W kolejnych rozważaniach będziemy przyjmować, że każdy podgraf dla każdej podmacierzy zawiera tylko tego typu krawędzie, bo wiemy że pozostałe znikną w redukcji a dążymy do zredukowanej postaci.

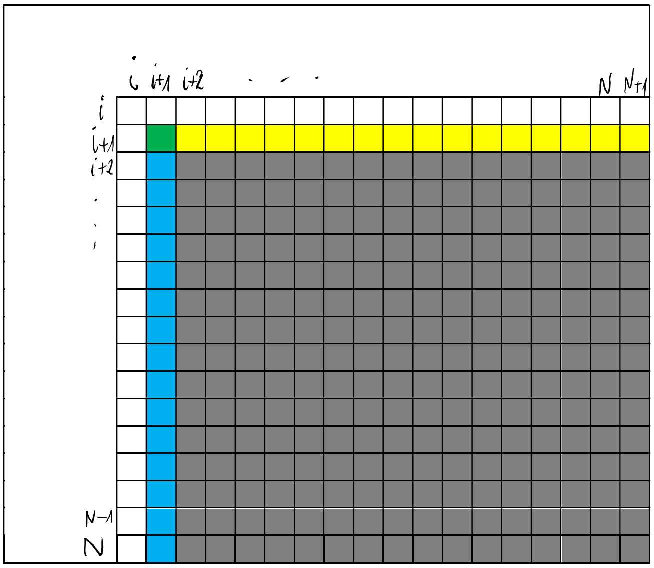
W kolejnym etapie pokaże jakie krawędzie istnieją w pełnym grafie pomiędzy dwiema sąsiednimi macierzami. Weźmy podmacierz i oraz i+1. Po analizie słowa ω wiemy, że istnieją tylko krawędzie z podgrafu dla podmacierzy i do podgrafu i+1 i nie istnieje ścieżka rozpoczynająca się w podgrafie i+1, która posiadała by wierzchołki z podgrafu i. Aby pokazać jakie krawędzie łączą te podgrafy przedstawię kilka rysunków macierzy.

Oznaczenia kolorów:

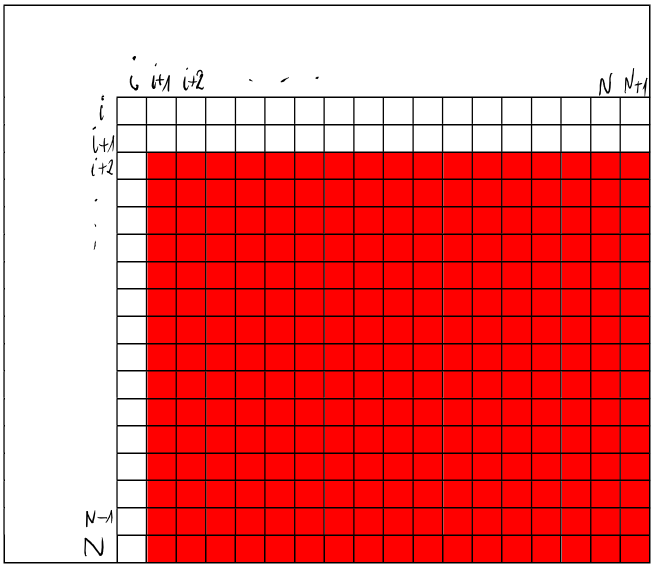
- czerwony – pola modyfikowane przez zadanie C
- szary + niebieski – pola odczytywane przez zadanie C
- niebieski – pola odczytywane przez zadanie A
- zielony – pole odczytywane przez każde zadanie A
- żółty + zielony – pola odczytywane przez zadanie B



Rysunek 2: Pola modyfikowane przez podmacierz i.



Rysunek 3: Pola odczytywane przez podmacierz i+1.



Rysunek 4: Pola modyfikowane przez podmacierz i+1.

Rysunek 2 ukazuje nam pola, które zmienia podmacierz i, wiemy, że każde z tych pól oznacza zadanie C wykonane na tym polu. A więc ok każdego takiego zadania prowadzi krawędź do każdego z zadań, które odczytują to pole w podmacierzach większych od i. W szczególności możemy odczytać z rysunku 3 do których zadań w podmacierzy i+1 prowadzi. Zauważmy, że jeśli dane pole jest odczytywane przez dwa różne zadania to jesteśmy w stanie zredukować każdą z krawędzi oprócz do zadań z najwyższej z hierarchii. Hierarchia zadań to oczywiście A, B, C. Dzieje się tak ponieważ jeśli dane pole odczytuje zadanie A i B to istnieje krawędź z A do B w podgrafie i+1, analogicznie dla pozostałych. Natomiast ta krawędź, która pozostanie jest nieredukowalna bo z zadania C z podgrafu i prowadzą tylko krawędzie do niższych podmacierzy, więc jeśli prowadzi do zadania niższego w hierarchii to na pewno nie ma krawędzi do zadania wyżej, a jeśli do dalszych podmacierzy to też nie da się wrócić ścieżką do tego zadania. Więc pokazaliśmy, że każda tak utworzona krawędź występuje w zredukowanym grafie zależności. Teraz opiszę te krawędzie.
Zadanie C które zmienia element o indeksie (i+1, i+1) trafia do każdego zadania typu A. Zadania C które zmieniają elementy z i+1 kolumny trafiają w odpowiednie zadania typu A. Zmieniające i+1 wiersz trafiają w zadania typu B. Natomiast zadania zawarte w podmacierzy i+2 trafiają na zadania typu C. Zapisując to formalnie dla zadań C pochodzących z i podmacierzy otrzymujemy następujący zbiór krawędzi.

$$\begin{aligned} C_i = &\left\{\left(C_{i,i+1,i+1}, A_{i+1,k}\right) : k \in \{i+2, \dots, N\}\right\} \cup \left\{\left(C_{i,i+1,k}, A_{i+1,k}\right) : k \in \{i+2, \dots, N\}\right\} \\ &\cup \left\{\left(C_{i,j,i+1}, B_{i+1,j,k}\right) : j \in \{i+2, \dots, N+1\};\ k \in \{i+2, \dots, N\}\right\} \\ &\cup \left\{\left(C_{i,j,k}, C_{i+1,j,k}\right) : j \in \{i+2, \dots, N+1\};\ k \in \{i+2, \dots, N\}\right\} \end{aligned}$$

Na koniec pokażę, że mamy już opisane wszystkie potrzebne krawędzie do postaci zredukowanej. Mamy opisane krawędzie w każdym podgrafie. Wiemy, że nie istnieją krawędzie z podmacierzy i do podmacierzy o mniejszym indeksie. Opisaliśmy krawędzie z podmacierzy i do podmacierzy i+1. Pozostaje pokazać, że każdą krawędź z podmacierzy i do podmacierzy i+k, gdzie k większe od 1 da się zredukować. Zauważmy, że jeśli wierzchołek jest w podmacierzy i+k to oznacza że pole które zmienia lub odczytuje ma indeksy co najmniej i+k. Z tego faktu wynika, że w każdej podmacierzy o indeksie mniejszym niż i+k istnieje zadanie C, które to pole modyfikuje (porównanie rysunków 2 i 3). A więc w szczególności istnieje takie zadanie C(i) w podmacierzy i, C(i+1) w podmacierzy i+1 itd. aż do zadania C(i+k-1). Natomiast zadanie z podmacierzy i+k nazwijmy X. Teraz z poprzednich rozważań wiemy, że istnieje krawędź pomiędzy C(i) i C(i+1), C(i+1) i C(i+2), …, C(i+k-1) i X, a więc istnieje ścieżka od C(i) do X taka która nie zawiera krawędzi (C(i), X), więc krawędź ta jest redukowalna.

Podsumowując graf Dickerta zawiera krawędzie dla każdego podgrafu takie jak w zbiorach P oraz krawędzie z podmacierzy i do podmacierzy i+1 dla każdego sensownego i. Są to wszystkie krawędzie grafu Dickerta. Wierzchołki grafu Dickerta to elementy słowa, które pokrywają się z alfabetem.

$$G = \left\{ \begin{matrix} V = \Sigma \\ E = \bigcup_{i=1}^{N-1} P_i \cup \bigcup_{i=1}^{N-2} C_i \end{matrix} \right\}$$

## 8. Postać Normalna Foaty

Z wcześniejszych obserwacji, zauważamy, że nie możemy wykonywać operacji z warstwy i dopóki nie wyliczą się prawie wszystkie(oprócz lewej kolumny, która i tak nie bierze udziału w dalszych obliczeniach) zadania z warstwy i-1. Natomiast patrząc na podgraf dla danej podmacierzy możemy go podzielić na trzy klasy Foaty: wszystkie zadania typu A, wszystkie zadania typu B, wszystkie zadania typu C. Widzimy, że nie ma możliwości wykonania szybciej tych zadań (nie istnieje zadanie w klasie drugiej, które możemy wykonać w klasie pierwszej i analogicznie dla klasy trzeciej). A więc z tego wynika że klasy Foaty będą wykonywać podmacierz po podmacierzy dzieląc każdą z nich na trzy klasy Foaty. Formalnie będzie to wyglądać następująco.

$$F_i = [A_{i,k}][B_{i,j,k}][C_{i,j,k}] \;\; dla \;\; j \in \{i, \dots, N+1\}; k \in \{i+1, \dots, N\}$$

$$FNF = F_1 F_2 \dots F_N$$

Można łatwo pokazać dowód indukcyjny, że dla każdego elementu w danej klasie Foaty nie da się go przenieść do wyższej klasy Foaty. Pokazuje nam to, że jest to prawidłowa postać normalna Foaty.

## 9. Implementacja schedulera

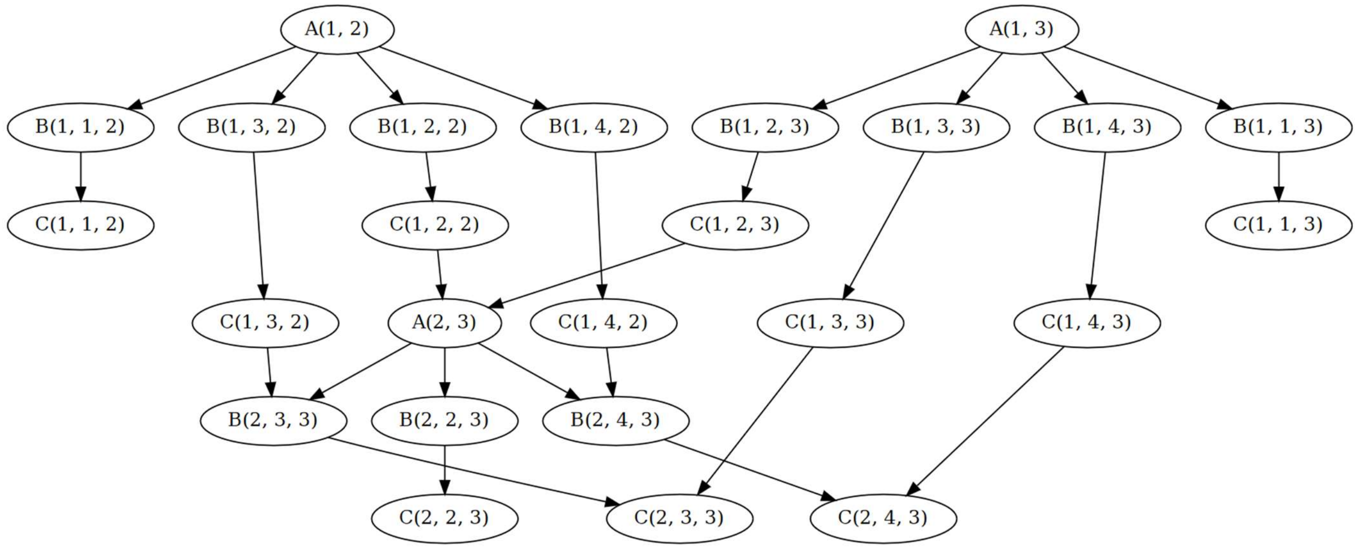
Implementację wykonałem w języku Python. Składa się ona z kilku części, które poniżej omówię. Aby uruchomić program należy w folderze z plikiem gaussian_elimination.py posiadać pliki in.txt i out.txt, które odpowiednio zawierają macierz wejściową oraz wyniki eliminacji.

Program zawiera klasę Scheduler. Jest to prosta klasa która zawiera listę zadań do wykonania. Jako parametr możemy podać ilość wątków na której będzie działać Scheduler. Klasa posiada metodę add_task, która służy do dodawania zadań do listy, metodę clear do czyszczenia listy zadań oraz metodę run, która wykonuje zadania. Metoda run korzysta z ThreadPoolExecutor, który pozwala symulować współbieżne wykonywanie zadań. Dodatkowo dodałem w niej opcjonalne permutowanie zadań na potrzeby ćwiczenia, aby pokazać że permutacja zadań w danej klasie Foaty nie zmienia rezultatu.

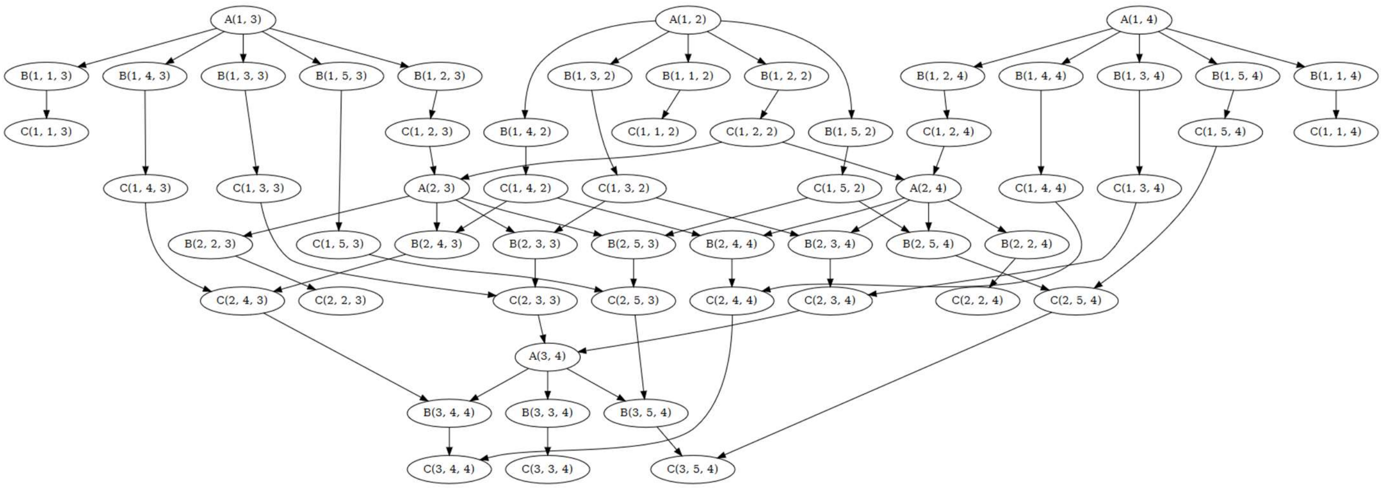
Klasa GaussianElimination jako argumenty przyjmuje macierz oraz jej rozmiar. Ma zaimplementowane zadania A, B i C. Metoda run przechodzi po kolejnych klasach Foaty wyznaczonych wyżej i dla każdej z nich uruchamia scheduler. Na końcu wynik jest normalizowany i następuje podstawianie wsteczne, aby dostosować rezultat do macierzy wygenerowanej w programie Java.

Dodatkowo w programie znajdziemy funkcję do odczytywania macierzy z pliku oraz funkcję zapisującą do pliku.

Napisałem również funkcję, która z pomocą biblioteki graphviz rysuje graf Dickerta. Dodaje ona najpierw do grafu wierzchołki, a później krawędzie zgodnie z wcześniejszym wyprowadzeniem. Dla małych N uzyskujemy sensowne wizualizacje, które pokrywają się z materiałem przedstawionym na ćwiczeniu. Funkcję tą umieściłem w pliku draw_graph.py. Należy ją wywołać z argumentem N.




Rysunek 5: Graf Dickerta dla N = 3.



Rysunek 6: Graf Dickerta dla N = 4.

## 10. Wnioski

- Otrzymywane wyniki są zgodne z wynikami generowanymi przez program z ćwiczeń, więc implementacja przebiegła prawidłowo.
- Zadanie pozwoliło szerzej spojrzeć na eliminację Gaussa.
- Dla dużych macierzy FNF pozwala na przyspieszeniu działania wykorzystując równoległe działanie.
- Gdybyśmy jako pojedyncze zadanie dali jedno drzewo takie jak na rysunku 1 można by zaimplementować szybki program, który nie stracił by poprzez rozdzielenie zadań, a wykonując je równolegle można by zyskać przyspieszenie. Zadania te dla każdej podmacierzy można by wykonywać równolegle.