Zadanie domowe III

Współbieżna eliminacja Gaussa

Autor: Adrian Żerebiec

Kompilacja programu

Program został napisany w języku Python. Wykorzystano do tego następujące biblioteki: cocurrent.futures, grapviz. Aby je zainstalować należy skorzystać z polecenia:

pip install ‘nazwa_biblioteki’

Aby uruchomić program wystarczy użyć komendy będąc w katalogu głównym projektu:

py main.py

W celu ułatwienia instalacji jak i kompilacji polecam korzystać z programu PyCharm. Aby stworzyć wykresy można wykorzystać program dot z biblioteki Graphviz. Można ją pobrać na stronie: https://graphviz.gitlab.io/download/ . Ścieżkę do rozpakowanego archiwum trzeba dodać do zmiennej środowiskowej PATH np. C:\Graphviz\bin. Po zresetowaniu terminala wszystko powinno działać.

Struktura projektu

Głównym plikiem projektu jest main. W osobnym katalogu ‘Functions’ znajdują się natomiast wszystkie pozostałe pliki z implementacjami poszczególnych funkcji takich jak: znajdowanie FNF, tworzenie grafów, obliczanie zależności itp.

Struktura:

- Functions/
    - dependencies.py – obliczanie zależności
    -foata.py – obliczanie FNF
    - gauss.py – metoda Gaussa
    - graph.py – tworzenie i zapisywanie grafu
    - sigma.py – tworzenie zbioru wszystkich czynności
    - utils.py – funkcje pomocnicze
- graphs/

main.py – główna część programu

Wyjaśnienie oznaczeń w programie

W programie wykorzystałem kilka oznaczeń:

- Factors – są to nasze czynniki Ai,j, dzięki czemu możemy wyzerować elementy macierzy

- Sigma – zbiór wszystkich czynności wykonywanych w macierzy

- Dependencies – zbiór zależności w macierzy

- FNF – postać normalna Foaty dla macierzy

Symbole A, B,C są przedstawione w formie: Ai,j Bi,j,k, Ci,j,k

Wynik działania programu

Początkowo program pyta, czy chcemy skorzystać z przykładów czy może jednak samemu stworzyć sobie macierz:

Do you want to enter your own matrix or examples? (own/examples):

W zależności od wyboru albo sami dodamy swoją macierz, podając na początku wymiar a później elementy i wektor, lub jeśli wybierzemy, aby skorzystać z przykładów to rozpoczął się obliczenia.

Przykładowe macierze

Najpierw sprawdźmy, jak działają przykłady, zatem przygotowane wcześniej macierze.

Macierz pierwsza

Pierwsza testowana macierz to ta z przykładu zawartego w poleceniu, czyli:

```
[2.0, 1.0, 3.0, 6.0]
[4.0, 3.0, 8.0, 15.0]
[6.0, 5.0, 16.0, 27.0]
```

gdzie ostatnia kolumna to nasz wektor.

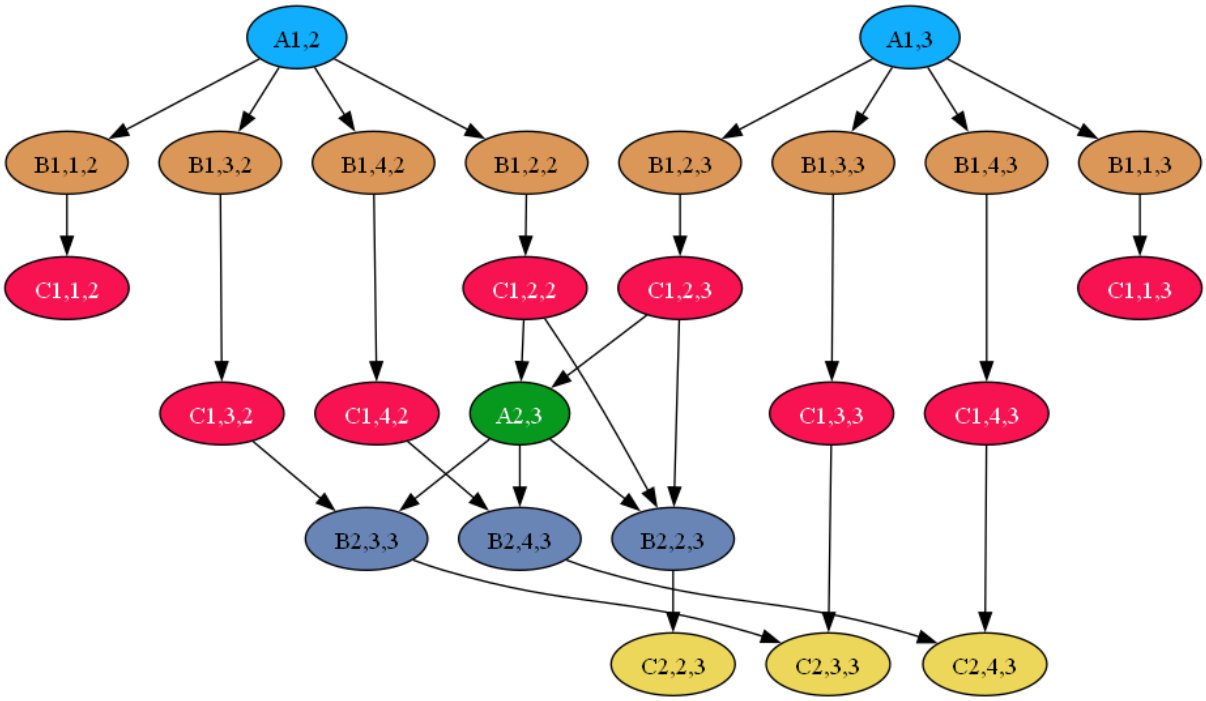
Wynik działania programu:

```
Original matrix:
2.0 1.0 3.0 | 6.0
4.0 3.0 8.0 | 15.0
6.0 5.0 16.0 | 27.0

Factors:
A1,2 -> 2.0
A1,3 -> 3.0
A2,3 -> 2.0

Matrix after parallel Gauss elimination:
2.0 1.0 3.0 | 6.0
```

```
0.0 1.0 2.0 | 3.0

0.0 0.0 3.0 | 3.0

Sigma:

['A1,2', 'B1,1,2', 'C1,1,2', 'B1,2,2', 'C1,2,2', 'B1,3,2', 'C1,3,2', 'B1,4,2', 'C1,4,2', 'A1,3', 'B1,1,3', 'C1,1,3',
'B1,2,3', 'C1,2,3', 'B1,3,3', 'C1,3,3', 'B1,4,3', 'C1,4,3', 'A2,3', 'B2,2,3', 'C2,2,3', 'B2,3,3', 'C2,3,3', 'B2,4,3',
'C2,4,3']


Dependencies:

[('A1,2', 'B1,1,2'), ('B1,1,2', 'C1,1,2'), ('A1,2', 'B1,2,2'), ('B1,2,2', 'C1,2,2'), ('A1,2', 'B1,3,2'), ('B1,3,2',
'C1,3,2'), ('A1,2', 'B1,4,2'), ('B1,4,2', 'C1,4,2'), ('A1,3', 'B1,1,3'), ('B1,1,3', 'C1,1,3'), ('A1,3', 'B1,2,3'),
('B1,2,3', 'C1,2,3'), ('A1,3', 'B1,3,3'), ('B1,3,3', 'C1,3,3'), ('A1,3', 'B1,4,3'), ('B1,4,3', 'C1,4,3'), ('A2,3',
'B2,2,3'), ('B2,2,3', 'C2,2,3'), ('A2,3', 'B2,3,3'), ('B2,3,3', 'C2,3,3'), ('A2,3', 'B2,4,3'), ('B2,4,3', 'C2,4,3'),
('C1,2,2', 'A2,3'), ('C1,2,3', 'A2,3'), ('C1,4,3', 'C2,4,3'), ('C1,4,2', 'B2,4,3'), ('C1,3,3', 'C2,3,3'), ('C1,3,2',
'B2,3,3'), ('C1,2,3', 'B2,2,3'), ('C1,2,2', 'B2,2,3')]


FNF:

[['A1,2', 'A1,3'], ['B1,1,2', 'B1,2,2', 'B1,3,2', 'B1,4,2', 'B1,1,3', 'B1,2,3', 'B1,3,3', 'B1,4,3'], ['C1,1,2',
'C1,2,2', 'C1,3,2', 'C1,4,2', 'C1,1,3', 'C1,2,3', 'C1,3,3', 'C1,4,3'], ['A2,3'], ['B2,2,3', 'B2,3,3', 'B2,4,3'],
['C2,2,3', 'C2,3,3', 'C2,4,3']]

Graph saved to graphs/directed_graph_example1.png
```

Dodatkowo wykres zapisuje się w pliku directed_graph_example1.png w katalogu graphs. Dzięki temu możemy mieć dostęp do grafu nawet po wyłączeniu programu. Kolory są ustawione na losowe, jednak, aby nieco ograniczyć losowość na stałe ustawiony jest seed.

Graf dla przykładu 1:



Jak widzimy jest on analogiczny jak ten pokazany w przykładzie zawartym w Zadaniu Domowym.

Macierz druga

Jako drugi przykład wykorzystujemy większą macierz, jednak, aby graf nie był zbyt duży dajemy kilka zer w niej.

[2.0, 4.0, 5.0, 6.0, 7.0, 1.0]
[0.0, 5.0, 6.0, 7.0, 8.0, 2.0]
[5.0, 6.0, 0.0, 8.0, 0.0, 3.0]
[0.0, 7.0, 8.0, 9.0, 10.0, 4.0]
[7.0, 8.0, 9.0, 10.0, 11.0, 5.0]

Wynik działania dla niej to:

```
Original matrix:
2.0 4.0 5.0 6.0 7.0 | 1.0
0.0 5.0 6.0 7.0 8.0 | 2.0
5.0 6.0 0.0 8.0 0.0 | 3.0
0.0 7.0 8.0 9.0 10.0 | 4.0
7.0 8.0 9.0 10.0 11.0 | 5.0

Factors:
A1,3 -> 2.5
```

A1,5 -> 3.5

A2,3 -> -0.8

A2,4 -> 1.4

A2,5 -> -1.2

A3,4 -> 0.05

A3,5 -> 0.17

A4,5 -> 3.25

Matrix after parallel Gauss elimination:

2.0 4.0 5.0 6.0 7.0 | 1.0

0.0 5.0 6.0 7.0 8.0 | 2.0

0.0 0.0 -7.7 -1.4 -11.1 | 2.1

0.0 0.0 0.0 -0.73 -0.62 | 1.09

0.0 0.0 0.0 0.0 0.0 | -0.0

Sigma:

['A1,3', 'B1,1,3', 'C1,1,3', 'B1,2,3', 'C1,2,3', 'B1,3,3', 'C1,3,3', 'B1,4,3', 'C1,4,3', 'B1,5,3', 'C1,5,3', 'B1,6,3', 'C1,6,3', 'A1,5', 'B1,1,5', 'C1,1,5', 'B1,2,5', 'C1,2,5', 'B1,3,5', 'C1,3,5', 'B1,4,5', 'C1,4,5', 'B1,5,5', 'C1,5,5', 'B1,6,5', 'C1,6,5', 'A2,3', 'B2,2,3', 'C2,2,3', 'B2,3,3', 'C2,3,3', 'B2,4,3', 'C2,4,3', 'B2,5,3', 'C2,5,3', 'B2,6,3', 'C2,6,3', 'A2,4', 'B2,2,4', 'C2,2,4', 'B2,3,4', 'C2,3,4', 'B2,4,4', 'C2,4,4', 'B2,5,4', 'C2,5,4', 'B2,6,4', 'C2,6,4', 'A2,5', 'B2,2,5', 'C2,2,5', 'B2,3,5', 'C2,3,5', 'B2,4,5', 'C2,4,5', 'B2,5,5', 'C2,5,5', 'B2,6,5', 'C2,6,5', 'A3,4', 'B3,3,4', 'C3,3,4', 'B3,4,4', 'C3,4,4', 'B3,5,4', 'C3,5,4', 'B3,6,4', 'C3,6,4', 'A3,5', 'B3,3,5', 'C3,3,5', 'B3,4,5', 'C3,4,5', 'B3,5,5', 'C3,5,5', 'B3,6,5', 'C3,6,5', 'A4,5', 'B4,4,5', 'C4,4,5', 'B4,5,5', 'C4,5,5', 'B4,6,5', 'C4,6,5']

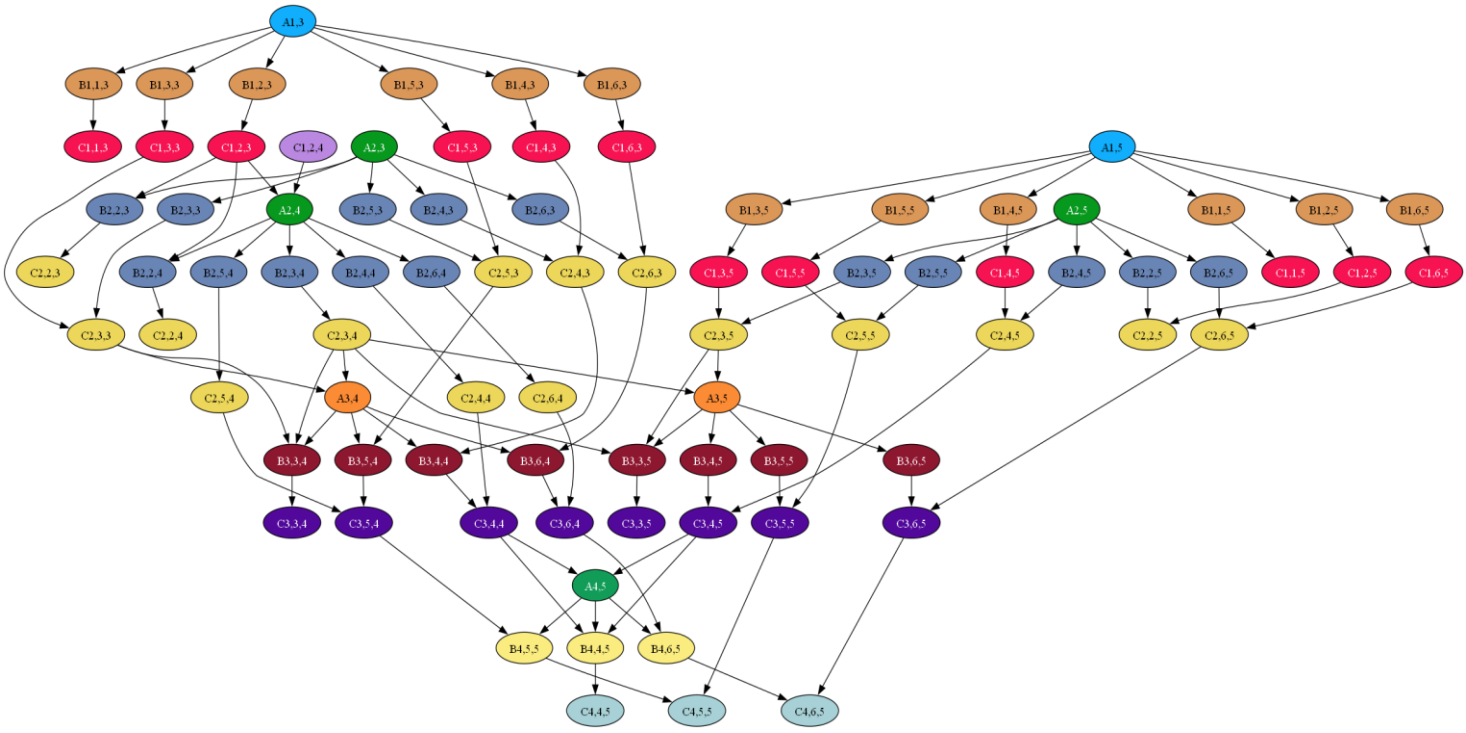
Dependencies:

[('A1,3', 'B1,1,3'), ('B1,1,3', 'C1,1,3'), ('A1,3', 'B1,2,3'), ('B1,2,3', 'C1,2,3'), ('A1,3', 'B1,3,3'), ('B1,3,3', 'C1,3,3'), ('A1,3', 'B1,4,3'), ('B1,4,3', 'C1,4,3'), ('A1,3', 'B1,5,3'), ('B1,5,3', 'C1,5,3'), ('A1,3', 'B1,6,3'), ('B1,6,3', 'C1,6,3'), ('A1,5', 'B1,1,5'), ('B1,1,5', 'C1,1,5'), ('A1,5', 'B1,2,5'), ('B1,2,5', 'C1,2,5'), ('A1,5', 'B1,3,5'), ('B1,3,5', 'C1,3,5'), ('A1,5', 'B1,4,5'), ('B1,4,5', 'C1,4,5'), ('A1,5', 'B1,5,5'), ('B1,5,5', 'C1,5,5'), ('A1,5', 'B1,6,5'), ('B1,6,5', 'C1,6,5'), ('A2,3', 'B2,2,3'), ('B2,2,3', 'C2,2,3'), ('A2,3', 'B2,3,3'), ('B2,3,3', 'C2,3,3'), ('A2,3', 'B2,4,3'), ('B2,4,3', 'C2,4,3'), ('A2,3', 'B2,5,3'), ('B2,5,3', 'C2,5,3'), ('A2,3', 'B2,6,3'), ('B2,6,3', 'C2,6,3'), ('A2,4', 'B2,2,4'), ('B2,2,4', 'C2,2,4'), ('A2,4', 'B2,3,4'), ('B2,3,4', 'C2,3,4'), ('A2,4', 'B2,4,4'), ('B2,4,4', 'C2,4,4'), ('A2,4', 'B2,5,4'), ('B2,5,4', 'C2,5,4'), ('A2,4', 'B2,6,4'), ('B2,6,4', 'C2,6,4'), ('C1,2,3', 'A2,4'), ('C1,2,4', 'A2,4'), ('A2,5', 'B2,2,5'), ('B2,2,5', 'C2,2,5'), ('A2,5', 'B2,3,5'), ('B2,3,5', 'C2,3,5'), ('A2,5', 'B2,4,5'), ('B2,4,5', 'C2,4,5'), ('A2,5', 'B2,5,5'), ('B2,5,5', 'C2,5,5'), ('A2,5', 'B2,6,5'), ('B2,6,5', 'C2,6,5'), ('A3,4', 'B3,3,4'), ('B3,3,4', 'C3,3,4'), ('A3,4', 'B3,4,4'), ('B3,4,4', 'C3,4,4'), ('A3,4', 'B3,5,4'), ('B3,5,4', 'C3,5,4'), ('A3,4', 'B3,6,4'), ('B3,6,4', 'C3,6,4'), ('C2,3,3', 'A3,4'), ('C2,3,4', 'A3,4'), ('A3,5', 'B3,3,5'), ('B3,3,5', 'C3,3,5'), ('A3,5', 'B3,4,5'), ('B3,4,5', 'C3,4,5'), ('A3,5', 'B3,5,5'), ('B3,5,5', 'C3,5,5'), ('A3,5', 'B3,6,5'), ('B3,6,5', 'C3,6,5'), ('C2,3,4', 'A3,5'), ('C2,3,5', 'A3,5'), ('A4,5', 'B4,4,5'),

```
('B4,4,5', 'C4,4,5'), ('A4,5', 'B4,5,5'), ('B4,5,5', 'C4,5,5'), ('A4,5', 'B4,6,5'), ('B4,6,5', 'C4,6,5'), ('C3,4,4',
'A4,5'), ('C3,4,5', 'A4,5'), ('C3,6,5', 'C4,6,5'), ('C3,6,4', 'B4,6,5'), ('C3,5,5', 'C4,5,5'), ('C3,5,4', 'B4,5,5'),
('C3,4,5', 'B4,4,5'), ('C3,4,4', 'B4,4,5'), ('C2,6,5', 'C3,6,5'), ('C2,6,4', 'C3,6,4'), ('C2,6,3', 'B3,6,4'),
('C2,5,5', 'C3,5,5'), ('C2,5,4', 'C3,5,4'), ('C2,5,3', 'B3,5,4'), ('C2,4,5', 'C3,4,5'), ('C2,4,4', 'C3,4,4'),
('C2,4,3', 'B3,4,4'), ('C2,3,5', 'B3,3,5'), ('C2,3,4', 'B3,3,5'), ('C2,3,4', 'B3,3,4'), ('C2,3,3', 'B3,3,4'),
('C1,6,5', 'C2,6,5'), ('C1,6,3', 'C2,6,3'), ('C1,5,5', 'C2,5,5'), ('C1,5,3', 'C2,5,3'), ('C1,4,5', 'C2,4,5'),
('C1,4,3', 'C2,4,3'), ('C1,3,5', 'C2,3,5'), ('C1,3,3', 'C2,3,3'), ('C1,2,5', 'C2,2,5'), ('C1,2,3', 'B2,2,4'),
('C1,2,3', 'B2,2,3')]

FNF:

[['A1,3', 'A1,5'], ['B1,1,3', 'B1,2,3', 'B1,3,3', 'B1,4,3', 'B1,5,3', 'B1,6,3', 'B1,1,5', 'B1,2,5', 'B1,3,5',
'B1,4,5', 'B1,5,5', 'B1,6,5'], ['C1,1,3', 'C1,2,3', 'C1,3,3', 'C1,4,3', 'C1,5,3', 'C1,6,3', 'C1,1,5', 'C1,2,5',
'C1,3,5', 'C1,4,5', 'C1,5,5', 'C1,6,5'], ['A2,3', 'A2,4', 'A2,5'], ['B2,2,3', 'B2,3,3', 'B2,4,3', 'B2,5,3', 'B2,6,3',
'B2,2,4', 'B2,3,4', 'B2,4,4', 'B2,5,4', 'B2,6,4', 'B2,2,5', 'B2,3,5', 'B2,4,5', 'B2,5,5', 'B2,6,5'], ['C2,2,3',
'C2,3,3', 'C2,4,3', 'C2,5,3', 'C2,6,3', 'C2,2,4', 'C2,3,4', 'C2,4,4', 'C2,5,4', 'C2,6,4', 'C2,2,5', 'C2,3,5',
'C2,4,5', 'C2,5,5', 'C2,6,5'], ['A3,4', 'A3,5'], ['B3,3,4', 'B3,4,4', 'B3,5,4', 'B3,6,4', 'B3,3,5', 'B3,4,5',
'B3,5,5', 'B3,6,5'], ['C3,3,4', 'C3,4,4', 'C3,5,4', 'C3,6,4', 'C3,3,5', 'C3,4,5', 'C3,5,5', 'C3,6,5'], ['A4,5'],
['B4,4,5', 'B4,5,5', 'B4,6,5'], ['C4,4,5', 'C4,5,5', 'C4,6,5']]

Graph saved to graphs/directed_graph_example2.png
```

Z racji, iż była ona sporo większa niż poprzednia to także graf jest bardziej skomplikowany.

Graf dla przykładu 2:



Widzimy, że graf jest znacznie bardziej rozbudowany. Nie może to dziwić, iż mamy do wykonania znacznie więcej operacji niż poprzednio.

Macierz trzecia

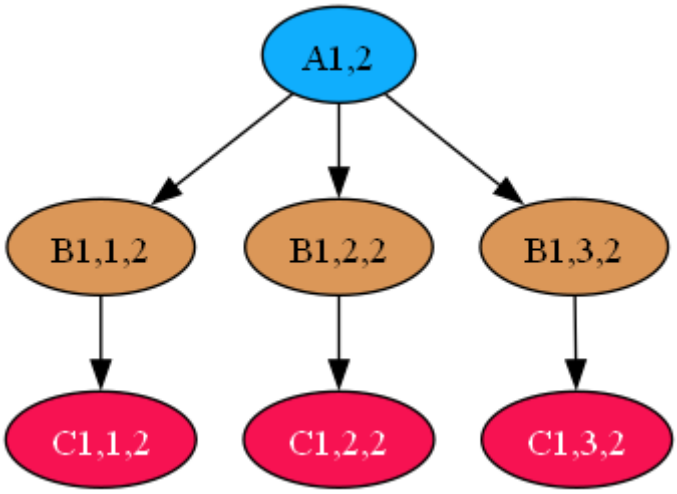
W trzecim przykładzie wykorzystujemy zdecydowanie najmniejszą macierz. Możemy zatem spodziewać się znacznie mniejszego grafu.

```
[90.0, 14.0, 22.0]
[14.0, 50.0, 4.0]
```

Wynik działania dla niej to:

```
Original matrix:

90.0 14.0 | 22.0

14.0 50.0 | 4.0

Factors:

A1,2 -> 0.16

Matrix after parallel Gauss elimination:

90.0 14.0 | 22.0

0.0 47.82 | 0.58

Sigma:

['A1,2', 'B1,1,2', 'C1,1,2', 'B1,2,2', 'C1,2,2', 'B1,3,2', 'C1,3,2']


Dependencies:

[('A1,2', 'B1,1,2'), ('B1,1,2', 'C1,1,2'), ('A1,2', 'B1,2,2'), ('B1,2,2', 'C1,2,2'), ('A1,2', 'B1,3,2'), ('B1,3,2', 'C1,3,2')]


FNF:

[['A1,2'], ['B1,1,2', 'B1,2,2', 'B1,3,2'], ['C1,1,2', 'C1,2,2', 'C1,3,2']]

Graph saved to graphs/directed_graph_example3.png
```

Graf dla przykładu 3:



Zatem wygląda on tak jak powinien, gdyż musimy wykonać operacje dla całego drugiego rzędu.

Macierz stworzona przez użytkownika

Program pozwala nam także dodać własną macierz. Zatem stwórzmy macierz typu:

```
[2.0, 12.0, 7.0, 35.0, 21.0]
[8.0, 2.0, 0.0, 35.0, 45.0]
[0.0, 24.0, 11.0, 17.0, 25.0]
[3.0, 90.0, 23.0, 62.0, 5.0]
```

Aby stworzyć macierz musimy dodawać rzędami wartości, a następnie dodać transponowany wektor, czyli [21.0, 45.0, 25.0, 5.0]. W ten sposób program stworzy nam powyższą macierz.

Wynik działania programu dla własnej macierzy:

```
Do you want to enter your own matrix or examples? (own/examples): own
Size of matrix: 4
Enter row 1 (space-separated values): 2 12 7 35
Enter row 2 (space-separated values): 8 2 0 35
Enter row 3 (space-separated values): 0 24 11 17
Enter row 4 (space-separated values): 3 90 23 62
Enter vector (space-separated values): 21 45 25 5
Original matrix:
2.0 12.0 7.0 35.0 | 21.0
8.0 2.0 0.0 35.0 | 45.0
0.0 24.0 11.0 17.0 | 25.0
3.0 90.0 23.0 62.0 | 5.0
```

Factors:

A1,2 -> 4.0

A1,4 -> 1.5

A2,3 -> -0.52

A2,4 -> -1.57

A3,4 -> 8.68

Matrix after parallel Gauss elimination:

2.0 12.0 7.0 35.0 | 21.0

0.0 -46.0 -28.0 -105.0 | -39.0

0.0 0.0 -3.61 -37.78 | 4.65

0.0 0.0 -0.0 173.13 | -127.93

Sigma:

['A1,2', 'B1,1,2', 'C1,1,2', 'B1,2,2', 'C1,2,2', 'B1,3,2', 'C1,3,2', 'B1,4,2', 'C1,4,2', 'B1,5,2', 'C1,5,2', 'A1,4', 'B1,1,4', 'C1,1,4', 'B1,2,4', 'C1,2,4', 'B1,3,4', 'C1,3,4', 'B1,4,4', 'C1,4,4', 'B1,5,4', 'C1,5,4', 'A2,3', 'B2,2,3', 'C2,2,3', 'B2,3,3', 'C2,3,3', 'B2,4,3', 'C2,4,3', 'B2,5,3', 'C2,5,3', 'A2,4', 'B2,2,4', 'C2,2,4', 'B2,3,4', 'C2,3,4', 'B2,4,4', 'C2,4,4', 'B2,5,4', 'C2,5,4', 'A3,4', 'B3,3,4', 'C3,3,4', 'B3,4,4', 'C3,4,4', 'B3,5,4', 'C3,5,4']

Dependencies:

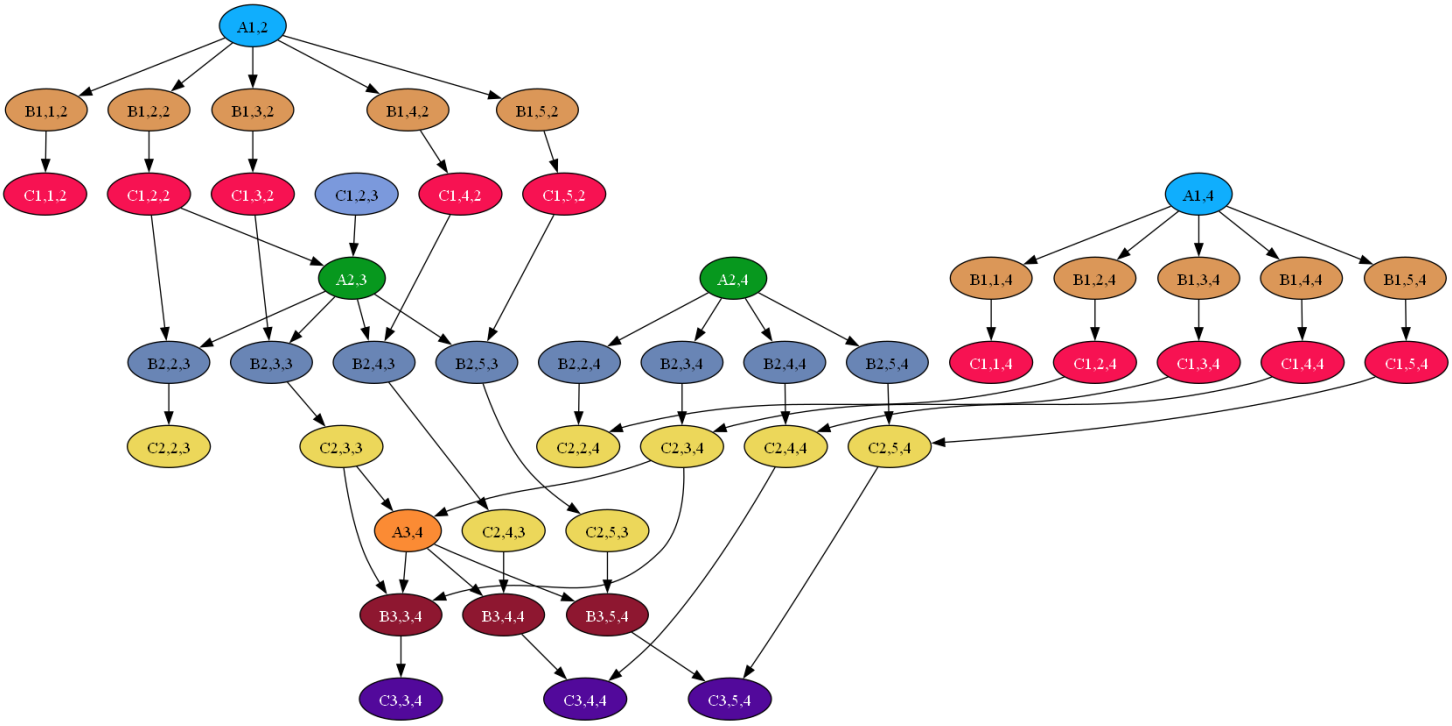
[('A1,2', 'B1,1,2'), ('B1,1,2', 'C1,1,2'), ('A1,2', 'B1,2,2'), ('B1,2,2', 'C1,2,2'), ('A1,2', 'B1,3,2'), ('B1,3,2', 'C1,3,2'), ('A1,2', 'B1,4,2'), ('B1,4,2', 'C1,4,2'), ('A1,2', 'B1,5,2'), ('B1,5,2', 'C1,5,2'), ('A1,4', 'B1,1,4'), ('B1,1,4', 'C1,1,4'), ('A1,4', 'B1,2,4'), ('B1,2,4', 'C1,2,4'), ('A1,4', 'B1,3,4'), ('B1,3,4', 'C1,3,4'), ('A1,4', 'B1,4,4'), ('B1,4,4', 'C1,4,4'), ('A1,4', 'B1,5,4'), ('B1,5,4', 'C1,5,4'), ('A2,3', 'B2,2,3'), ('B2,2,3', 'C2,2,3'), ('A2,3', 'B2,3,3'), ('B2,3,3', 'C2,3,3'), ('A2,3', 'B2,4,3'), ('B2,4,3', 'C2,4,3'), ('A2,3', 'B2,5,3'), ('B2,5,3', 'C2,5,3'), ('C1,2,2', 'A2,3'), ('C1,2,3', 'A2,3'), ('A2,4', 'B2,2,4'), ('B2,2,4', 'C2,2,4'), ('A2,4', 'B2,3,4'), ('B2,3,4', 'C2,3,4'), ('A2,4', 'B2,4,4'), ('B2,4,4', 'C2,4,4'), ('A2,4', 'B2,5,4'), ('B2,5,4', 'C2,5,4'), ('A3,4', 'B3,3,4'), ('B3,3,4', 'C3,3,4'), ('A3,4', 'B3,4,4'), ('B3,4,4', 'C3,4,4'), ('A3,4', 'B3,5,4'), ('B3,5,4', 'C3,5,4'), ('C2,3,3', 'A3,4'), ('C2,3,4', 'A3,4'), ('C2,5,4', 'C3,5,4'), ('C2,5,3', 'B3,5,4'), ('C2,4,4', 'C3,4,4'), ('C2,4,3', 'B3,4,4'), ('C2,3,4', 'B3,3,4'), ('C2,3,3', 'B3,3,4'), ('C1,5,4', 'C2,5,4'), ('C1,5,2', 'B2,5,3'), ('C1,4,4', 'C2,4,4'), ('C1,4,2', 'B2,4,3'), ('C1,3,4', 'C2,3,4'), ('C1,3,2', 'B2,3,3'), ('C1,2,4', 'C2,2,4'), ('C1,2,2', 'B2,2,3')]

FNF:

[['A1,2', 'A1,4'], ['B1,1,2', 'B1,2,2', 'B1,3,2', 'B1,4,2', 'B1,5,2', 'B1,1,4', 'B1,2,4', 'B1,3,4', 'B1,4,4', 'B1,5,4'], ['C1,1,2', 'C1,2,2', 'C1,3,2', 'C1,4,2', 'C1,5,2', 'C1,1,4', 'C1,2,4', 'C1,3,4', 'C1,4,4', 'C1,5,4'], ['A2,3', 'A2,4'], ['B2,2,3', 'B2,3,3', 'B2,4,3', 'B2,5,3', 'B2,2,4', 'B2,3,4', 'B2,4,4', 'B2,5,4'], ['C2,2,3',

```
'C2,3,3', 'C2,4,3', 'C2,5,3', 'C2,2,4', 'C2,3,4', 'C2,4,4', 'C2,5,4'], ['A3,4'], ['B3,3,4', 'B3,4,4', 'B3,5,4'],
['C3,3,4', 'C3,4,4', 'C3,5,4']]

Graph saved to graphs/directed_graph_own.png
```

Graf dla własnego przykładu:



Kod zadania domowego

main.py

```python
"""
Glowna czesc programu, ktora pozwala na wybor przykladowych macierzy lub wprowadzenie
wlasnej. Nastepnie wywoluje funkcje z Functions
"""
from Functions.dependencies import construct_D
from Functions.foata import construct_FNF
from Functions.gauss import parallel_gauss_elimination
from Functions.graph import draw_directed_graph
from Functions.sigma import construct_sigma
from Functions.utils import print_matrix, print_sigma, print_factors

mat = [[2.0, 1.0, 3.0, 6.0],
       [4.0, 3.0, 8.0, 15.0],
       [6.0, 5.0, 16.0, 27.0]]

mat1 = [[2.0, 4.0, 5.0, 6.0, 7.0, 1.0],
        [0.0, 5.0, 6.0, 7.0, 8.0, 2.0],
        [5.0, 6.0, 0.0, 8.0, 0.0, 3.0],
```

```
        [0.0, 7.0, 8.0, 9.0, 10.0, 4.0],
        [7.0, 8.0, 9.0, 10.0, 11.0, 5.0]]

mat2 = [[90.0, 14.0, 22.0],
        [14.0, 50.0, 4.0]]

matrixes = [mat, mat1, mat2]

if __name__ == "__main__":
    while True:
        n = input("Do you want to enter your own matrix or examples? (own/examples): ")
        if n == "own" or n == "examples":
            break
        else:
            print("Wrong input")
    if n == "own":
        n = int(input("Size of matrix: "))
        mat = []
        for i in range(n):
            row = list(map(float, input(f"Enter row {i + 1} (space-separated values):
").split()))
            while len(row) != n:
                print("Wrong input")
                row = list(map(float, input(f"Enter row {i + 1} (space-separated values):
").split()))
            mat.append(row)

        vector = list(map(float, input(f"Enter vector (space-separated values):
").split()))
        mat = [row + [vector[i]] for i, row in enumerate(mat)]
        print("Original matrix:")
        print_matrix(mat)
        factors = []
        parallel_gauss_elimination(mat, factors)
        print("\nFactors:")
        print_factors(factors)
        print("\nMatrix after parallel Gauss elimination:")
        print_matrix(mat)
        print("\nSigma:")
        sigma = construct_sigma(mat, factors)
        print_sigma(sigma)
        print("\nDependencies:")
        D = construct_D(construct_sigma(mat, factors))
        print(D)
        FNF = construct_FNF(sigma, D)
        print("\nFNF:")
        print(FNF)
        print("Graph saved to graphs/directed_graph_own.png")
        print("\n\n")
```

```
        draw_directed_graph(D, "own", FNF)
    if n == "examples":
        i = 0
        for mat in matrixes:
            i += 1
            print("Original matrix:")
            print_matrix(mat)
            factors = []
            parallel_gauss_elimination(mat, factors)
            print("\nFactors:")
            print_factors(factors)
            print("\nMatrix after parallel Gauss elimination:")
            print_matrix(mat)
            print("\nSigma:")
            sigma = construct_sigma(mat, factors)
            print_sigma(sigma)
            print("\nDependencies:")
            D = construct_D(construct_sigma(mat, factors))
            print(D)
            FNF = construct_FNF(sigma, D)
            print("\nFNF:")
            print(FNF)
            draw_directed_graph(D, "example" + str(i), FNF)
            print("Graph saved to graphs/directed_graph_example" + str(i) + ".png")
            print("\n\n")
```

Funtions/dependencies.py

```
"""
Funkcja tworzaca zaleznosci D. W tym celu pomagam sobie znaczkami "*", gdyz licze sobie ile
poszczegole operacje maja elementow
oraz pozycje gwiazdek. Nastepnie odpowiednimi operacjami dodajemy sobie zaleznosci. Warto
zauwazyc, iz momentami trudno dodac niektore
elementy i trzeba nieco kombinowac, stad chociazby flagi czy sortowania. Dodatkowo dzieki
nim udalo sie zachowac calkiem dobra kolejnosc
elementow w D.
"""

from Functions.utils import extract_two_numbers, extract_three_numbers


def construct_D(sigma):
    D = []
    n = len(sigma)
    stars = [-1]
    k = 0
    for i in range(n):
        if sigma[i] != "*":
```

```
            k += 1
        else:
            stars.append(k)
            k += 1

    for i in range(1, len(stars)):
        A = sigma[stars[i - 1] + 1]
        for j in range(stars[i - 1] + 1, stars[i] + 1):
            if "B" in sigma[j]:
                D.append((A, sigma[j]))
                D.append((sigma[j], sigma[j + 1]))

        x, y = extract_two_numbers(A)
        for j in range(1, i):
            x1, y1 = extract_two_numbers(sigma[stars[j - 1] + 1])
            if x - 1 == x1 and y - 1 == y1:
                k = (f"C{x - 1},{x},{y - 1}", A)
                k1 = (f"C{x - 1},{x},{y}", A)
                if k not in D and k1 not in D:
                    D.append(k)
                    D.append(k1)
    sigma = sorted(sigma, reverse=True)
    for i in range(n):
        if sigma[i] == "*" or "A" in sigma[i] or "B" in sigma[i]:
            continue
        x, y, z = extract_three_numbers(sigma[i])
        for j in range(0, n):
            flag = True
            if sigma[j] == "*" or "A" in sigma[j]:
                continue

            for k in D:
                if k[0] == sigma[i] and "A" in k[1] and "C" in k[0] and "C" in sigma[j]:
                    flag = False
                    break

            if not flag:
                continue

            x1, y1, z1 = extract_three_numbers(sigma[j])

            if "C" in sigma[j] and "C" in sigma[i]:
                if x + 1 == x1 and y == y1 and z == z1:
                    k = (sigma[i], sigma[j])
                    if k not in D:
                        D.append(k)

            if (sigma[i], f"C{x + 1},{y},{z}") in D:
                continue
```

```
            if "C" in sigma[i] and "B" in sigma[j]:
                if (x + 1 == x1 and y == y1 and z + 1 == z1) or (x + 1 == x1 and y == y1
and z == z1):
                    k = (sigma[i], sigma[j])
                    if k not in D:
                        D.append(k)
    return D
```

Functions/foata.py

```
"""
Funkcja ta tworzy nam postac normana Foaty. Z przykladu mozna wyciagnac wnioski, ze idziemy
w kolejnosci dzialan oraz z racji, iz
np. A1,2 jest niezalezne od A1,3 to sa one w tej samej klasie. Nastepnie musia byc elementy
z B, pozniej z C. Po tych operacjach
znowu wykonujemy A tak jak w przykladzie, zatem idac tym schematem mozemy stworzyc cale FNF
"""

from Functions.utils import extract_two_numbers, extract_three_numbers


def construct_FNF(sigma, D):
    FNF = []
    D.sort()
    A = []
    B = []
    C = []
    for i in range(len(sigma)):
        if "A" in sigma[i]:
            A.append(sigma[i])
        if "B" in sigma[i]:
            B.append(sigma[i])
        if "C" in sigma[i]:
            C.append(sigma[i])

    while A or B or C:
        x_max = float("inf")
        for i in range(len(A)):
            x, _ = extract_two_numbers(A[i])
            if x < x_max:
                x_max = x

        A1 = []
        i = 0
        while i < len(A):
            x, _ = extract_two_numbers(A[i])
```

```
            if x == x_max:
                A1.append(A[i])
                A.pop(i)
            else:
                i += 1

        FNF.append(A1)
        B1 = []
        i = 0

        while i < len(B):
            x, _, _ = extract_three_numbers(B[i])
            if x == x_max:
                B1.append(B[i])
                B.pop(i)
            else:
                i += 1

        C1 = []
        i = 0
        while i < len(C):
            x, _, _ = extract_three_numbers(C[i])
            if x == x_max:
                C1.append(C[i])
                C.pop(i)
            else:
                i += 1

        FNF.append(B1)
        FNF.append(C1)

    return FNF
```

Functions/gauss.py

```
"""
Eliminacje Gaussa wykonujemy standardowo. Jesli mamy 0 w sprawdzanym elemencie to nie ma
sensu tam nic robic dlatego pomijamy.
Do obliczen wykorzystujemy concurrent.futures. Odpalamy sobie dla kazdego rzedu osobny
watek aby przyspieszyc dzialanie.
"""

import concurrent.futures
def gauss_elimination(mat, row, factors):
    pivot = mat[row][row]
    for i in range(row + 1, len(mat)):
        if mat[i][row] == 0:
            continue
```

```
        factor = mat[i][row] / pivot
        factors.append(((row + 1, i + 1), factor))
        for j in range(row, len(mat[row])):
            mat[i][j] -= factor * mat[row][j]


def parallel_gauss_elimination(mat, factors):
    with concurrent.futures.ThreadPoolExecutor() as executor:
        futures = [executor.submit(gauss_elimination, mat, row, factors) for row in
range(len(mat))]
        concurrent.futures.wait(futures)
```

Functions/graph.py

```
"""
Do tworzenia grafow wykorzystujemy biblioteke graphviz, wykorzystywana juz poprzednio w
zadaniu domowym II. na podstawie stworzonych
zaleznosci dodajemy latwo odpowiednie wierzcholki oraz krawedzi i tworzymy graf. Pozostale
funkcja zlozone sa do tworzenia kolorow
"""
import random

import graphviz

def generate_contrast_color(hex_color):
    r = int(hex_color[1:3], 16)
    g = int(hex_color[3:5], 16)
    b = int(hex_color[5:7], 16)

    brightness = (r * 299 + g * 587 + b * 114) / 1000
    text_color = '#000000' if brightness > 128 else '#FFFFFF'

    return text_color

def blend_colors(color1, color2):
    r1, g1, b1 = int(color1[1:3], 16), int(color1[3:5], 16), int(color1[5:7], 16)
    r2, g2, b2 = int(color2[1:3], 16), int(color2[3:5], 16), int(color2[5:7], 16)

    blended_color = "#{:02x}{:02x}{:02x}".format((r1 + r2) // 2, (g1 + g2) // 2, (b1 + b2)
// 2)
    return blended_color

def draw_directed_graph(dependencies, name, FNF):
    dot = graphviz.Digraph(comment='Directed Graph')
    random.seed(10)

    color_dict = {}
```

```
    for i, fnf_list in enumerate(FNF):
        base_color = "#{:06x}".format(random.randint(0, 0xFFFFFF))
        for node in fnf_list:
            if node not in color_dict:
                color_dict[node] = (base_color, generate_contrast_color(base_color))

    for dependency in dependencies:
        fill_color_node_0, text_color_node_0 = color_dict.get(dependency[0],
("#{:06x}".format(random.randint(0, 0xFFFFFF)), '#000000'))
        fill_color_node_1, text_color_node_1 = color_dict.get(dependency[1],
("#{:06x}".format(random.randint(0, 0xFFFFFF)), '#000000'))

        dot.node(dependency[0], style='filled', fillcolor=fill_color_node_0,
fontcolor=text_color_node_0)
        dot.node(dependency[1], style='filled', fillcolor=fill_color_node_1,
fontcolor=text_color_node_1)
        dot.edge(dependency[0], dependency[1])

    dot.render(f'graphs/directed_graph_{name}', format='png', cleanup=True)
```

Functions/sigma.py

```
"""
Funkcja tworzy zbior operacji, ktore bedziemy wykonywac, czyli jak w przykladzie A1,2 A1,3
A2,3 B1,2,2 itp.
Z racji iz w przykladzie mozna bylo zauwazyc schemat ich powstawania go wykorzystalem,
zatem najpierw majac juz factors czyli nasze czynniki
latwo nam bylo stworzyc dzialania A np. A1,2. Nastepnie zauwazylem ze B i C wystepuja
parami w sensie maja takie same indeksy, dlatego
dodajemy je na raz. Indeksy tez wydaja sie w miare latwe do zrobienia. Na podstawie
przykladu stwierdzilem, ze pierwszy i trzeci sa takie same jak
indesky A. Srodkowy natomiast jest to liczba z przedzialu [pierwszy indeks A, dlugosc
wiersza macierzy). Znaczek "*" jest pomocniczy i na konsoli jest on pomijany.
"""

def construct_sigma(mat, factors):
    sigma = []
    n = len(mat[0])
    for i in range(len(factors)):
        sigma.append(f"A{factors[i][0][0]},{factors[i][0][1]}")  # A1,2
        for j in range(factors[i][0][0], n + 1):
            sigma.append(f"B{factors[i][0][0]},{j},{factors[i][0][1]}")  # B1,1,3
            sigma.append(f"C{factors[i][0][0]},{j},{factors[i][0][1]}")  # C1,1,3
        sigma.append("*")
    return sigma
```

Funtions/utils.py

```
"""
Funkcje pomocnicze sluzace glownie do wypisywania na konsoli ladniej wynikow oraz danych.
Dodatkowo funkcje extract_x_numbers(s)
sa do wydobycia z napisow potrzebnych liczb
"""


def extract_two_numbers(s):
    parts = ''.join(filter(lambda x: x.isdigit() or x == ',', s)).split(',')
    numbers = [int(part) for part in parts if part.isdigit()]
    return tuple(numbers)


def extract_three_numbers(s):
    parts = ''.join(filter(lambda x: x.isdigit() or x == ',', s)).split(',')
    numbers = [int(part) for part in parts if part.isdigit()]
    return numbers


def print_sigma(sigma):
    sigma1 = []
    for i in range(len(sigma)):
        if sigma[i] != "*":
            sigma1.append(sigma[i])
    print(sigma1)


def print_matrix(mat):
    for i in range(len(mat)):
        for j in range(len(mat[i])):
            mat[i][j] = round(mat[i][j], 2)
            if j == len(mat[i]) - 1:
                print("|", mat[i][j], end=" ")
            else:
                print(mat[i][j], end=" ")
        print()


def print_factors(factors):
    for i in range(len(factors)):
        print("A" + str(factors[i][0][0]) + "," + str(factors[i][0][1]), "->",
round(factors[i][1], 2))
```