1. Jak uruchomić program?

Z powodu użycia bibliotek `matplotlib` i `networkx` na początku należy stworzyć środowisko wirtualne i zainstalować wymagane paczki z dołączonego pliku `requirements.txt`.

```
cd matrices/
python3 -m venv .
source bin/activate
pip3 install -r requirements.txt
```

Program spodziewa się otrzymać ścieżkę do pliku z danymi wejściowymi oraz ścieżkę do pliku gdzie ma zapisać rozwiązanie w postaci akceptowanej przez sprawdzarkę. Przygotowałem kilka przykładowych plików z danymi wejściowymi, znajdują się one w folderze `inputs`.

```
python3 main.py inputs/in1.txt outputs/out1.txt
python3 main.py inputs/in2.txt outputs/out2.txt
```

2. Opis problemu

Zadanie polega na stworzeniu algorytmu, który dla macierzy o dowolnym rozmiarze n zrealizuje następujące kroki:

1. Zlokalizowanie niepodzielnych czynności wykonywanych podczas eliminacji Gaussa i nazwanie ich.
2. Zbudowanie alfabetu w sensie teorii śladów.
3. Skonstruowanie relacji zależności dla wyznaczonego alfabetu.
4. Przedstawienie algorytmu eliminacji Gaussa jako ciągu symboli alfabetu.
5. Wygenerowanie grafu zależności Diekerta.
6. Przekształcenie ciągu symboli do postaci normalnej Foaty.
7. Rozwiązanie problemu eliminacji Gaussa współbieżnie, za pomocą postaci normalnej Foaty.

3. Algorytm

3.1. Niepodzielne czynności

Dla ogólnego przypadku przyjmujemy oznaczenia niepodzielnych czynności:

- $A_{i,k}$ to znalezienie mnożnika, który wykorzystamy do odjęcia i-tego wiersza od k-tego wiersza. Niech $m_{k,i}$ będzie oznaczać szukany mnożnik, a $M_{i,j}$ oznacza wyraz z i-tego wiersza i j-tej kolumny macierzy uzupełnionej o wektor wyrazów wolnych. Wówczas zachodzi:

$$m_{k,i} = \frac{M_{k,i}}{M_{i,i}}$$

- $B_{i,j,k}$ to pomnożenie j-tej komórki i-tego wiersza przez obliczony wcześniej mnożnik w celu odjęcia i-tego wiersza od k-tego wiersza. Wartość po pomnożeniu oznaczamy jako $n_{k,i,j}$ i spełnia ona:

$$n_{k,i,j} = M_{i,j} * m_{k,i}$$

- $C_{i,j,k}$ to odjęcie przemnożonego j-tego elementu i-tego wiersza od j-tego elementu k-tego wiersza, zachodzi wówczas:

$$M_{k,j} = M_{k,j} - n_{k,i,j}$$

3.2. Alfabet

Działamy na macierzy uzupełnionej o dodatkowy wektor wyrazów wolnych, zatem mamy n wierszy i $n+1$ kolumn. Dodatkowo łatwo zauważyć, że używając i-tego wiersza będziemy działać tylko na wiersze $i+1, i+2, ..., n$, a w tych wierszach tylko na kolumny $i, i+1, ..., n+1$. Zatem alfabet będzie w postaci:

$$\begin{gathered}\Sigma = \{A_{1,2}, B_{1,1,2}, C_{1,1,2}, ..., B_{1,n+1,2}, C_{1,n+1,2},\\ A_{1,3}, B_{1,1,3}, C_{1,1,3}, ..., B_{1,n+1,3}, C_{1,n+1,3}\\ ...\\ A_{1,n}, B_{1,1,n}, C_{1,1,n}, ..., B_{1,n+1,n}, C_{1,n+1,n}\\ A_{2,3}, B_{2,2,3}, C_{2,2,3}, ..., B_{2,n+1,3}, C_{2,n+1,3}\\ ...\\ A_{n-1,n}, B_{n-1,n-1,n}, C_{n-1,n-1,n}, B_{n-1,n,n}, C_{n-1,n,n}, B_{n-1,n+1,n}, C_{n-1,n+1,n}\}\end{gathered}$$

3.3. Relacja zależności

Oczywistym jest, że zależnościami są $(A_{i,k}, B_{i,j,k})$ oraz $(B_{i,j,k}, C_{i,j,k})$. Oprócz tego, przyjmując pewne $l, l > k$, musimy uwzględnić również że:

- od $C_{i,i,k}$ nie będzie zależeć żadna operacja, bo to zadanie zeruje komórkę, a wyzerowanej komórki nie potrzebujemy już do niczego.
- od $C_{i,i+1,k}$ oraz $C_{i,i+1,l}$ zależy operacja $A_{k,l}$, bo zmieniają one elementy używane w liczeniu mnożnika.
- dla $j \in \{i+2, i+3, ..., n+1\}$ od każdej operacji $C_{i,j,k}$ zależy $B_{k,j,l}$, bo zmienia ona element używany w zadaniu B.
- dla $j \in \{i+2, i+3, ..., n+1\}$ od każdej operacji $C_{i,j,l}$ zależy $C_{k,j,l}$, bo zmienia element używany w następnym zadaniu C.

Po uwzględnieniu wszystkich powyższych relacji otrzymujemy

$$\begin{gathered}L = \{(A_{1,2}, B_{1,1,2}), ..., (A_{1,2}, B_{1,n+1,2}),\\ (B_{1,1,2}, C_{1,1,2}), ..., (B_{1,n+1,2}, C_{1,n+1,2}),\\ (A_{1,3}, B_{1,1,3}), ..., (A_{1,3}, B_{1,n+1,3}),\\ (B_{1,1,3}, C_{1,1,3}), ..., (B_{1,n+1,3}, C_{1,n+1,3}),\\ ...,\\ (A_{1,n}, B_{1,1,n}), ..., (A_{1,n}, B_{1,n+1,n}),\\ (B_{1,1,n}, C_{1,1,n}), ..., (B_{1,n+1,n}, C_{1,n+1,n}),\\ (A_{2,3}, B_{2,2,3}), ..., (A_{2,3}, B_{2,n+1,3}),\\ (B_{2,2,3}, C_{2,2,3}), ..., (B_{2,n+1,3}, C_{2,n+1,3}),\\ ...,\\ (A_{n-1,n}, B_{n-1,n-1,n}), (A_{n-1,n}, B_{n-1,n,n}), (A_{n-1,n}, B_{n-1,n+1,n}),\\ (B_{n-1,n-1,n}, C_{n-1,n-1,n}), (B_{n-1,n,n}, C_{n-1,n,n}), (B_{n-1,n+1,n}, C_{n-1,n+1,n}),\\ (C_{1,2,2}, A_{2,3}), (C_{1,2,3}, A_{2,3}), (C_{1,3,2}, B_{2,3,3}), (C_{1,3,3}, C_{2,3,3}), ..., (C_{1,n+1,2}, B_{2,n+1,3}), (C_{1,n+1,3}, C_{2,n+1,3}),\\ ...,\\ (C_{n-2,n-1,n-1}, A_{n-1,n}), (C_{n-2,n-1,n}, A_{n-1,n}), (C_{n-2,n,n-1}, B_{n-1,n,n})\\ (C_{n-2,n,n}, C_{n-1,n,n}), (C_{n-2,n+1,n-1}, B_{n-1,n+1,n}), (C_{n-2,n+1,n}, C_{n-1,n+1,n})\}\end{gathered}$$

$$D = \text{sym}\{L^+\} \cup I_\Sigma$$

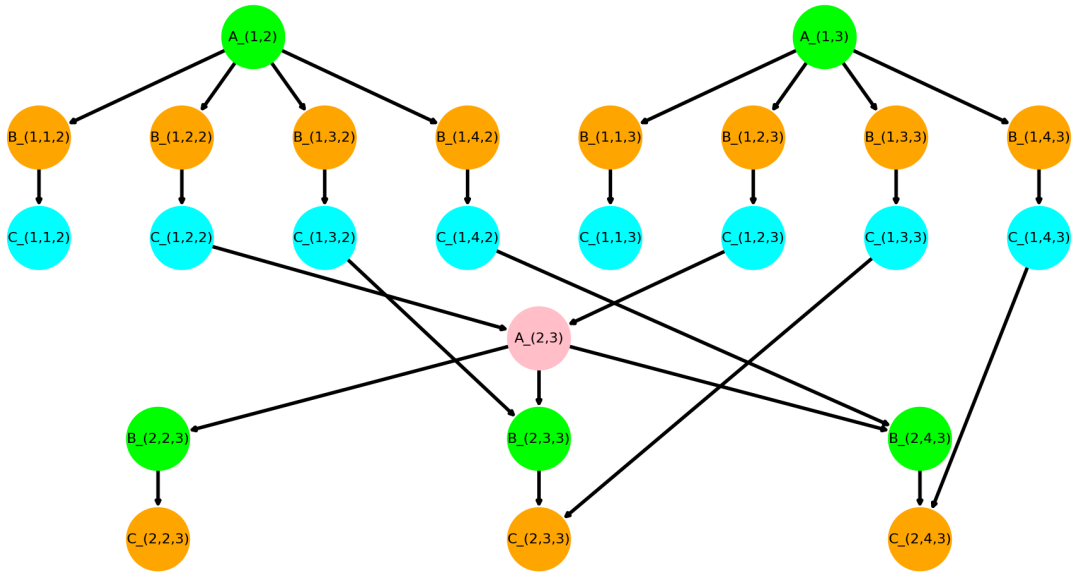
3.4. Algorytm jako ciąg symboli

Przykładowy ślad wykonania algorytmu będzie miał postać:

$$\begin{gathered} w = A_{1,2}B_{1,1,2}C_{1,1,2}...B_{1,n+1,2}C_{1,n+1,2} \\ A_{1,3}B_{1,1,3}C_{1,1,3}...B_{1,n+1,3}C_{1,n+1,3} \\ ... \\ A_{1,n}B_{1,1,n}C_{1,1,n}...B_{1,n+1,n}C_{1,n+1,n} \\ A_{2,3}B_{2,2,3}C_{2,2,3}...B_{2,n+1,3}C_{2,n+1,3} \\ ... \\ A_{n-1,n}B_{n-1,n-1,n}C_{n-1,n-1,n}B_{n-1,n,n}C_{n-1,n,n}B_{n-1,n+1,n}C_{n-1,n+1,n} \end{gathered}$$

3.5. Graf zależności Diekerta

Graf zależności wykonywany jest na podstawie relacji zależności. Do przedstawienia go użyto biblioteki `matplotlib`. Przykładowy graf, dla $n = 3$, prezentuje się następująco:



Rysunek 1: Graf zależności Diekerta dla $n = 3$

3.6. Postać normalna Foaty

Przy okazji poprzedniego wykresu można zauważyć, że klasy układają się warstwami. Dla pewnego wiersza $i \in \{1, 2, ..., n\}$ będziemy mieli kolejno warstwy zadań $A_{i,k}, k \in \{2, ..., n\}$, następnie operacje $B_{i,j,k}, j \in \{1, 2, ..., n+1\}, k \in \{2, ..., n\}$ i na koniec analogicznie warstwę zadań C. W miarę jak algorytm eliminacji Gaussa przechodzi przez kolejne wiersze macierzy, na wykresie będziemy obserwować kolejne warstwy zadań A, B, C.

3.7. Implementacja

Implementację algorytmu współbieżnego wykonano w języku Python i podzielono na 2 pliki: `main.py`, zawierający jedynie funkcję główną, oraz `utils.py`, zawierający funkcje pomocnicze:

- `process_file(file_path: str)` - funkcja przetwarzająca dane z pliku na macierze i listy wykorzystywane później
- `print_operation(operation: dict)` - funkcja prezentujące operację danego typu na danej komórce macierzy w postaci napisu

- `get_alphabet(n: int)` - funkcja generująca operacje, które algorytm będzie musiał wykonać, na podstawie rozmiaru macierzy
- `get_dependent_transaction(n: int)` - funkcja generująca zbiór relacji zależności na podstawie rozmiaru macierzy
- `get_fnf(n: int)` - funkcja wyznaczająca postać normalną Foaty na podstawie rozmiaru macierzy
- `draw_graph(dependent: list[tuple[dict, dict]], fnf: list[list[dict]])` - funkcja rysująca graf zależności Diekerta
- `parallel_gauss(matrix: list[list[float]], fnf: list[list[dict]])` - funkcja wykonująca współbieżnie algorytm eliminacji Gaussa
- `backward_substitution(eM: list[list[float]], n: int)` - funkcja sprowadzająca macierz do postaci jednostkowej
- `A(M: list[list[float]], i: int, k: int)` - funkcja wykonująca zadanie $A_{i,k}$
- `B(M: list[list[float]], i: int, j: int, k: int)` - funkcja wykonująca zadanie $B_{i,j,k}$
- `C(M: list[list[float]], i: int, j: int, k: int)` - funkcja wykonująca zadanie $C_{i,j,k}$

4. Wyniki

W folderze z implementacją algorytmu znajdują się foldery `inputs` z danymi dla algorytmu (plik `in1.txt` zawiera przykład z UPeLa) oraz `outputs` gdzie algorytm zapisuje wyniki działania w postaci oczekiwanej przez sprawdzarkę.

4.1. Dla `inputs/in1.txt`

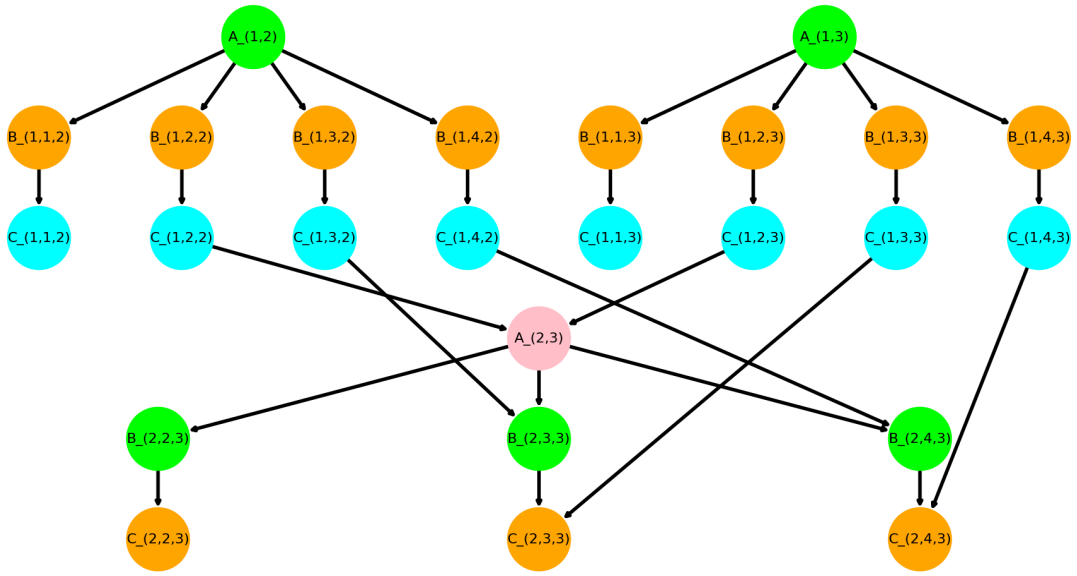
4.1.1. Wywołanie

```
python3 main.py inputs/in1.txt outputs/out1.txt
```

4.1.2. Rezultat wywołania

```
Alfabet
Sigma = {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), 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),
         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)}

Relacja zależności
D = sym{{(A_(1,2),B_(1,1,2)), (A_(1,2),B_(1,2,2)), (A_(1,2),B_(1,3,2)),
(A_(1,2),B_(1,4,2)), (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)), (A_(1,3),B_(1,1,3)),
(A_(1,3),B_(1,2,3)), (A_(1,3),B_(1,3,3)), (A_(1,3),B_(1,4,3)),
(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)), (A_(2,3),B_(2,2,3)), (A_(2,3),B_(2,3,3)),
(A_(2,3),B_(2,4,3)), (B_(2,2,3),C_(2,2,3)), (B_(2,3,3),C_(2,3,3)),
(B_(2,4,3),C_(2,4,3)), (C_(1,2,2),A_(2,3)), (C_(1,2,3),A_(2,3)),
(C_(1,3,2),B_(2,3,3)), (C_(1,4,2),B_(2,4,3)), (C_(1,3,3),C_(2,3,3)),
(C_(1,4,3),C_(2,4,3))}^+} u I_Σ

Ślad wykonania algorytmu
w = 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)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)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)

Postać normalna Foaty
FNF([w]) = [A_(1,2)A_(1,3)]
[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)]
[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)]
[B_(2,2,3)B_(2,3,3)B_(2,4,3)][C_(2,2,3)C_(2,3,3)C_(2,4,3)]

Macierz po rozwiązaniu układu
[1.0, 0.0, 0.0, 1.0]
[0.0, 1.0, 0.0, 1.0]
[0.0, 0.0, 1.0, 1.0]

Transponowany wektor rozwiązań
[1.0, 1.0, 1.0]
```

4.1.3. Uzyskany graf Diekerta



Rysunek 2: Graf Diekerta dla `inputs/in1.txt`, $n = 3$

4.2. Dla `inputs/in2.txt`

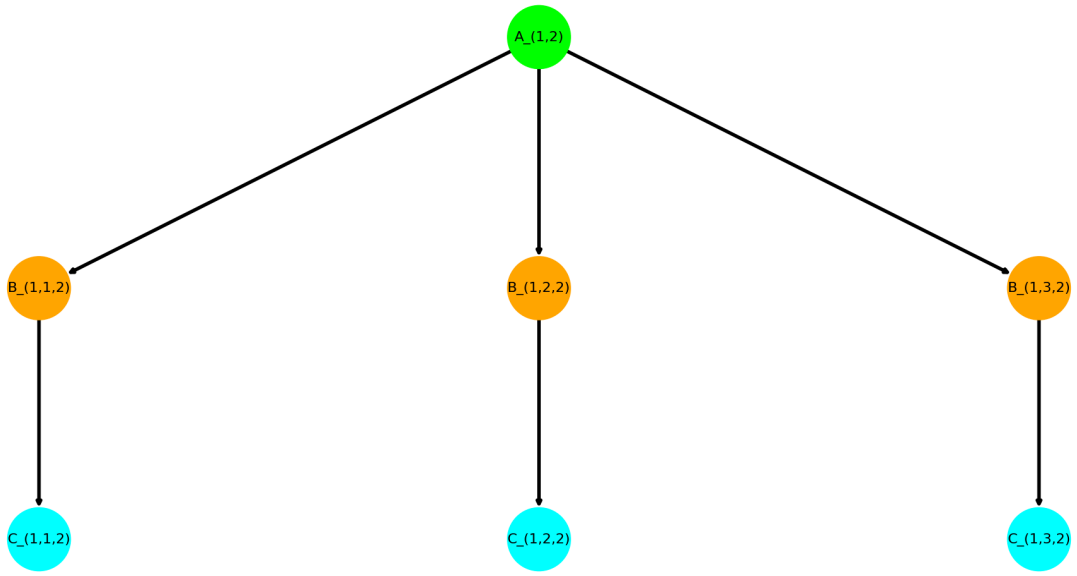
4.2.1. Wywołanie

```
python3 main.py inputs/in2.txt outputs/out2.txt
```

4.2.2. Rezultat wywołania

```
Alfabet
Sigma = {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)}

Relacja zależności
D = sym{{(A_(1,2),B_(1,1,2)), (A_(1,2),B_(1,2,2)), (A_(1,2),B_(1,3,2)),
(B_(1,1,2),C_(1,1,2)), (B_(1,2,2),C_(1,2,2)), (B_(1,3,2),C_(1,3,2))}^+} u I_Σ

Ślad wykonania algorytmu
w = 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)

Postać normalna Foaty
FNF([w]) = [A_(1,2)][B_(1,1,2)B_(1,2,2)B_(1,3,2)]
           [C_(1,1,2)C_(1,2,2)C_(1,3,2)]

Macierz po rozwiązaniu układu
[1.0, 0.0, 0.3414634146341462]
[0.0, 1.0, 0.4634146341463415]

Transponowany wektor rozwiązań
[0.3414634146341462, 0.4634146341463415]
```

4.2.3. Uzyskany graf Diekerta



Rysunek 3: Graf Diekerta dla `inputs/in2.txt`, $n = 2$

4.3. Dla `inputs/in3.txt`

4.3.1. Wywołanie

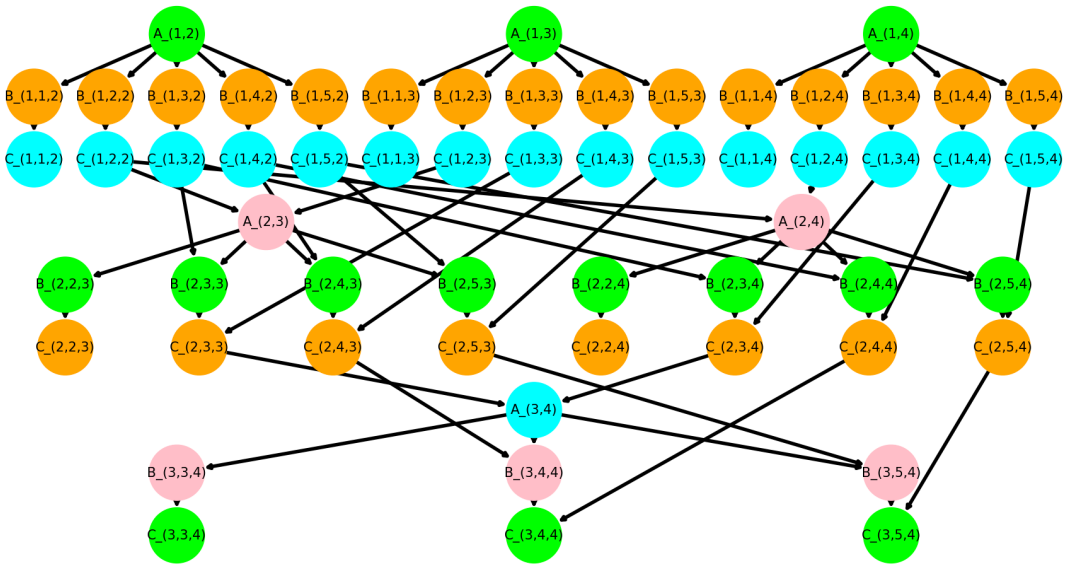
```
python3 main.py inputs/in3.txt outputs/out3.txt
```

4.3.2. Rezultat wywołania

```
Alfabet
Sigma = {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), B_(1,5,2), C_(1,5,2), 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),
B_(1,5,3), C_(1,5,3), A_(1,4), B_(1,1,4), C_(1,1,4), B_(1,2,4), C_(1,2,4),
B_(1,3,4), C_(1,3,4), B_(1,4,4), C_(1,4,4), B_(1,5,4), C_(1,5,4), 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), B_(2,5,3),
C_(2,5,3), A_(2,4), B_(2,2,4), C_(2,2,4), B_(2,3,4), C_(2,3,4), B_(2,4,4),
C_(2,4,4), B_(2,5,4), C_(2,5,4), A_(3,4), B_(3,3,4), C_(3,3,4), B_(3,4,4),
C_(3,4,4), B_(3,5,4), C_(3,5,4)}

Relacja zależności
D = sym{{(A_(1,2),B_(1,1,2)), (A_(1,2),B_(1,2,2)), (A_(1,2),B_(1,3,2)),
(A_(1,2),B_(1,4,2)), (A_(1,2),B_(1,5,2)), (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)),
(B_(1,5,2),C_(1,5,2)), (A_(1,3),B_(1,1,3)), (A_(1,3),B_(1,2,3)),
(A_(1,3),B_(1,3,3)), (A_(1,3),B_(1,4,3)), (A_(1,3),B_(1,5,3)),
(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)), (B_(1,5,3),C_(1,5,3)), (A_(1,4),B_(1,1,4)),
(A_(1,4),B_(1,2,4)), (A_(1,4),B_(1,3,4)), (A_(1,4),B_(1,4,4)),
(A_(1,4),B_(1,5,4)), (B_(1,1,4),C_(1,1,4)), (B_(1,2,4),C_(1,2,4)),
(B_(1,3,4),C_(1,3,4)), (B_(1,4,4),C_(1,4,4)), (B_(1,5,4),C_(1,5,4)),
(A_(2,3),B_(2,2,3)), (A_(2,3),B_(2,3,3)), (A_(2,3),B_(2,4,3)),
(A_(2,3),B_(2,5,3)), (B_(2,2,3),C_(2,2,3)), (B_(2,3,3),C_(2,3,3)),
(B_(2,4,3),C_(2,4,3)), (B_(2,5,3),C_(2,5,3)), (A_(2,4),B_(2,2,4)),
(A_(2,4),B_(2,3,4)), (A_(2,4),B_(2,4,4)), (A_(2,4),B_(2,5,4)),
(B_(2,2,4),C_(2,2,4)), (B_(2,3,4),C_(2,3,4)), (B_(2,4,4),C_(2,4,4)),
(B_(2,5,4),C_(2,5,4)), (A_(3,4),B_(3,3,4)), (A_(3,4),B_(3,4,4)),
(A_(3,4),B_(3,5,4)), (B_(3,3,4),C_(3,3,4)), (B_(3,4,4),C_(3,4,4)),
(B_(3,5,4),C_(3,5,4)), (C_(1,2,2),A_(2,3)), (C_(1,2,3),A_(2,3)),
(C_(1,2,2),A_(2,4)), (C_(1,2,4),A_(2,4)), (C_(1,3,2),B_(2,3,3)),
(C_(1,4,2),B_(2,4,3)), (C_(1,5,2),B_(2,5,3)), (C_(1,3,2),B_(2,3,4)),
(C_(1,4,2),B_(2,4,4)), (C_(1,5,2),B_(2,5,4)), (C_(1,3,3),C_(2,3,3)),
(C_(1,4,3),C_(2,4,3)), (C_(1,5,3),C_(2,5,3)), (C_(1,3,4),C_(2,3,4)),
(C_(1,4,4),C_(2,4,4)), (C_(1,5,4),C_(2,5,4)), (C_(2,3,3),A_(3,4)),
(C_(2,3,4),A_(3,4)), (C_(2,4,3),B_(3,4,4)), (C_(2,5,3),B_(3,5,4)),
(C_(2,4,4),C_(3,4,4)), (C_(2,5,4),C_(3,5,4))}^+} u I_Σ

Ślad wykonania algorytmu
w =
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)
B_(1,5,2)C_(1,5,2)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)B_(1,5,3)C_(1,5,3)A_(1,4)B_(1,1,4)C_(1,1,4)B_(1,2,4)C_(1,2,4)
B_(1,3,4)C_(1,3,4)B_(1,4,4)C_(1,4,4)B_(1,5,4)C_(1,5,4)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)B_(2,5,3)C_(2,5,3)A_(2,4)B_(2,2,4)C_(2,2,4)
B_(2,3,4)C_(2,3,4)B_(2,4,4)C_(2,4,4)B_(2,5,4)C_(2,5,4)A_(3,4)B_(3,3,4)C_(3,3,4)
B_(3,4,4)C_(3,4,4)B_(3,5,4)C_(3,5,4)
```

```
Postać normalna Foaty
FNF([w]) = [A_(1,2)A_(1,3)A_(1,4)]
[B_(1,1,2)B_(1,2,2)B_(1,3,2)B_(1,4,2)B_(1,5,2)B_(1,1,3)B_(1,2,3)B_(1,3,3)
B_(1,4,3)B_(1,5,3)B_(1,1,4)B_(1,2,4)B_(1,3,4)B_(1,4,4)B_(1,5,4)]
[C_(1,1,2)C_(1,2,2)C_(1,3,2)C_(1,4,2)C_(1,5,2)C_(1,1,3)C_(1,2,3)C_(1,3,3)
C_(1,4,3)C_(1,5,3)C_(1,1,4)C_(1,2,4)C_(1,3,4)C_(1,4,4)C_(1,5,4)]
[A_(2,3)A_(2,4)]
[B_(2,2,3)B_(2,3,3)B_(2,4,3)B_(2,5,3)B_(2,2,4)B_(2,3,4)B_(2,4,4)B_(2,5,4)]
[C_(2,2,3)C_(2,3,3)C_(2,4,3)C_(2,5,3)C_(2,2,4)C_(2,3,4)C_(2,4,4)C_(2,5,4)]
[A_(3,4)][B_(3,3,4)B_(3,4,4)B_(3,5,4)]
[C_(3,3,4)C_(3,4,4)C_(3,5,4)]

Macierz po rozwiązaniu układu
[1.0, 0.0, 0.0, 0.0, 16.81403269754774]
[0.0, 1.0, 0.0, 0.0, 0.4275658492279746]
[0.0, 7.105427357601002e-15, 1.0, 0.0, 0.26135331516802984]
[0.0, 0.0, 0.0, 1.0, -17.942325158946478]

Transponowany wektor rozwiązań
[16.81403269754774, 0.4275658492279746, 0.26135331516802984, -17.942325158946478]
```

4.3.3. Uzyskany graf Diekerta



Rysunek 4: Graf Diekerta dla `inputs/in3.txt`, $n = 4$

4.4. Dla `inputs/in4.txt`

4.4.1. Wywołanie

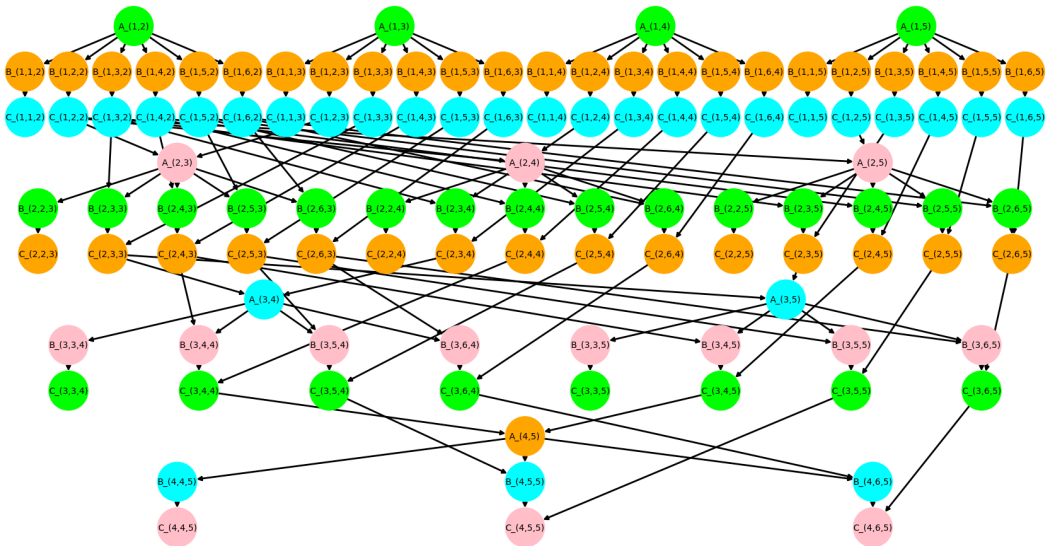
```
python3 main.py inputs/in4.txt outputs/out4.txt
```

4.4.2. Rezultat wywołania

Ze względu na obszerność logów, pokazuję tylko końcówkę

```
Postać normalna Foaty
FNF([w]) = [A_(1,2)A_(1,3)A_(1,4)A_(1,5)]
[B_(1,1,2)B_(1,2,2)B_(1,3,2)B_(1,4,2)B_(1,5,2)B_(1,6,2)B_(1,1,3)
B_(1,2,3)B_(1,3,3)B_(1,4,3)B_(1,5,3)B_(1,6,3)B_(1,1,4)B_(1,2,4)
B_(1,3,4)B_(1,4,4)B_(1,5,4)B_(1,6,4)B_(1,1,5)B_(1,2,5)B_(1,3,5)
B_(1,4,5)B_(1,5,5)B_(1,6,5)]
[C_(1,1,2)C_(1,2,2)C_(1,3,2)C_(1,4,2)C_(1,5,2)C_(1,6,2)C_(1,1,3)
C_(1,2,3)C_(1,3,3)C_(1,4,3)C_(1,5,3)C_(1,6,3)C_(1,1,4)C_(1,2,4)
C_(1,3,4)C_(1,4,4)C_(1,5,4)C_(1,6,4)C_(1,1,5)C_(1,2,5)C_(1,3,5)
C_(1,4,5)C_(1,5,5)C_(1,6,5)][A_(2,3)A_(2,4)A_(2,5)]
[B_(2,2,3)B_(2,3,3)B_(2,4,3)B_(2,5,3)B_(2,6,3)B_(2,2,4)B_(2,3,4)
B_(2,4,4)B_(2,5,4)B_(2,6,4)B_(2,2,5)B_(2,3,5)B_(2,4,5)B_(2,5,5)B_(2,6,5)]
[C_(2,2,3)C_(2,3,3)C_(2,4,3)C_(2,5,3)C_(2,6,3)C_(2,2,4)C_(2,3,4)C_(2,4,4)
C_(2,5,4)C_(2,6,4)C_(2,2,5)C_(2,3,5)C_(2,4,5)C_(2,5,5)C_(2,6,5)]
[A_(3,4)A_(3,5)]
[B_(3,3,4)B_(3,4,4)B_(3,5,4)B_(3,6,4)B_(3,3,5)B_(3,4,5)B_(3,5,5)B_(3,6,5)]
[C_(3,3,4)C_(3,4,4)C_(3,5,4)C_(3,6,4)C_(3,3,5)C_(3,4,5)C_(3,5,5)C_(3,6,5)]
[A_(4,5)]
[B_(4,4,5)B_(4,5,5)B_(4,6,5)][C_(4,4,5)C_(4,5,5)C_(4,6,5)]

Macierz po rozwiązaniu układu
[1.0, 0.0, 0.0, 0.0, 0.0, 1.2239062723187453]
[0.0, 1.0, 0.0, 0.0, 0.0, 0.369371822709914]
[0.0, -3.552713678800501e-15, 1.0, 0.0, 0.0, -0.4206112934004898]
[0.0, 1.7763568394002505e-15, 0.0, 1.0, 0.0, -1.1955427764595583]
[0.0, 0.0, 0.0, 0.0, 1.0, 1.42371575052126]

Transponowany wektor rozwiązań
[1.2239062723187453, 0.369371822709914, -0.4206112934004898, -1.1955427764595583,
1.42371575052126]
```

4.4.3. Uzyskany graf Diekerta



Rysunek 5: Graf Diekerta dla `inputs/in4.txt`, $n = 5$

4.5. Dla `inputs/in5.txt`

4.5.1. Wywołanie

```
python3 main.py inputs/in5.txt outputs/out5.txt
```

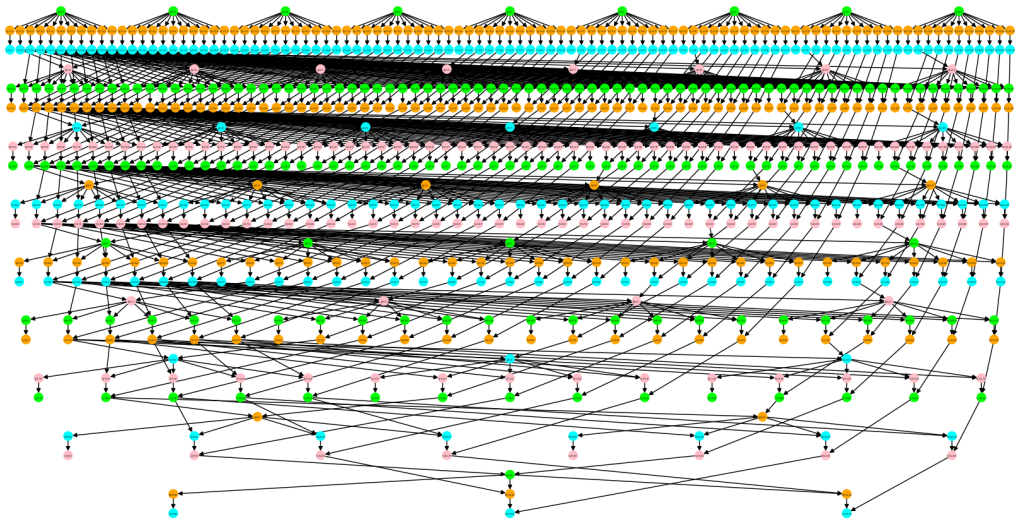
4.5.2. Rezultat wywołania

Ze względu na obszerność logów, pokazuję tylko końcówkę

```
Macierz po rozwiązaniu układu
[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.8482857874915397]
[0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.1245205517598427]
[0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.19701820378976617]
[0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.07485705506078451]
[0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.37938587752593367]
[0.0, 0.0, 0.0, 4.440892098500626e-16, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0,
-0.42025285645908567]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, -0.9763314686895471]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, -0.47913700307416773]
[0.0, 0.0, 0.0, 0.0, 0.0, 8.881784197001252e-16, 0.0, 0.0, 1.0, 0.0,
0.7238925762218943]
[0.0, 1.7763568394002505e-15, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0,
0.46832076324109606]

Transponowany wektor rozwiązań
[-0.8482857874915397, 2.1245205517598427, -0.19701820378976617,
-0.07485705506078451, 0.37938587752593367, -0.42025285645908567,
-0.9763314686895471, -0.47913700307416773, 0.7238925762218943,
0.46832076324109606]
```

4.5.3. Uzyskany graf Diekerta

Jako że w tym przykładzie $n = 10$ to graf jest już na tyle duży, że nie można go czytelnie przedstawić w całości.



Rysunek 6: Graf Diekerta dla `inputs/in5.txt`, $n = 10$

5. Błędy obliczeniowe

W trakcie wykonywania programu kumulują się błędy obliczeniowe wynikające z reprezentacji liczb zmiennoprzecinkowych. Są one dla sprawdzonych danych rzędu 10^{-15} więc mamy duży zapas względem błędu akceptowanego przez sprawdzarkę.