Zastosowanie teorii śladów do szeregowania wątków współbieżnej eliminacji Gaussa

Autor: Szymon Paja

Treść zadania

> Zadanie polega na wykonaniu następujących etapów (dla macierzy o rozmiarze N): (Proszę skoncentrować się na przykładzie z wykładu.)
> a) Proszę zlokalizować niepodzielne czynności wykonywane przez algorytm, nazwać je oraz zbudować alfabet w sensie teorii śladów
> b) Proszę skonstruować relację zależności dla alfabetu, opisującego algorytm eliminacji Gaussa.
> c) Proszę przedstawić algorytm eliminacji Gaussa w postaci ciągu symboli alfabetu.
> d) Proszę wygenerować graf zależności Diekerta.
> e) Proszę przekształcić ciąg symboli opisujący algorytm do postaci normalnej Foaty.
>
> Proszę zaprojektować i zaimplementować współbieżny algorytm eliminacji Gaussa. W szczególności proszę zwrócić uwagę na implementację jak najlepiej odwzorowującą graf zależności lub postać normalną Foaty.
> Program ma działać dla zadanych rozmiarów macierzy N.

Lokalizacja niepodzielnych czynności wykonywanych przez algorytm

W zadaniu będziemy rozwiązywać układ równań liniowych metodą Gaussa. Mamy zatem równanie $M \times x = y$, gdzie M jest macierzą kwadratową współczynników, x wektorem niewiadomych, a y wektorem wyników. Macierz M o rozmiarach $n \times n$ prezentuje się następująco

$$\begin{bmatrix} M_{1,1} & M_{1,2} & \cdots & M_{1,n} \\ M_{2,1} & M_{2,2} & \cdots & M_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ M_{n,1} & M_{n,2} & \cdots & M_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Zdefinujmy niepodzielne zadania obliczeniowe, służące do rozwiązania układu:

- $A_{i,k}$ - obliczenie mnożnika wiersza i w celu odjęcia go od wiersza k,
  $m_{k,i} = M_{k,i}/M_{i.i}$
- $B_{i,j,k}$ - pomnożenie j-tego elementu wiersza i przez wcześniej znaleziony mnożnik, w celu odjęcia od wiersza k,
  $n_{k,i} = M_{i,j} * m_{k,i}$
- $C_{i,j,k}$ - odjęcie j-tego elementu wiersza i od wiersza k,
  $M_{k,j} = M_{k,j} - n_{k,i}$.

Możemy w prosty sposób zasymulować rozwiązywanie takiego układu równań, wykorzystując powyżej zdefiniowane zadania obliczeniowe. Dla uproszczenia zastosujemy notację:

$$\left[\begin{array}{cccc|c} M_{1,1} & M_{1,2} & \cdots & M_{1,n} & y_1 \\ M_{2,1} & M_{2,2} & \cdots & M_{2,n} & y_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ M_{n,1} & M_{n,2} & \cdots & M_{n,n} & y_n \end{array}\right]$$

1. Wyzerowanie pierwszej kolumny używając pierwszego wiersza.

Znajdujemy mnożnik dla wiersza 1. w celu odjęcia go od wiersza drugiego - w tym celu skorzystamy z zadania obliczeniowego $A_{1,2}$. Następnie każdy j-ty element wiersza pierwszego przemnożymy przez wcześniej znaleziony mnożnik korzystając z operacji $B_{1,1,2}, \ldots, B_{1,n,2}$ i odejmiemy od j-tego elementu wiersza drugiego, korzystając z operacji $C_{1,1,2}, \ldots, C_{1,n,2}$. Zatem wykonanie wszystkich tych operacji możemy zapisać jako

$$A_{1,2}, B_{1,1,2}, C_{1,1,2}, \ldots, B_{1,n+1,2}, C_{1,n+1,2}$$

Postępując analogicznie dla każdego kolejnego wiersza otrzymujemy podobne ciągi operacji. Ostatecznie w celu wyzerowania pierwszej kolumny musimy wykonać następujące operacje

$$\begin{gathered} A_{1,2}, B_{1,1,2}, C_{1,1,2}, \ldots, B_{1,n+1,2}, C_{1,n+1,2}, \\ A_{1,3}, B_{1,1,3}, C_{1,1,3}, \ldots, B_{1,n+1,3}, C_{1,n+1,3}, \\ \vdots \\ A_{1,n}, B_{1,1,n}, C_{1,1,n}, \ldots, B_{1,n+1,n}, C_{1,n+1,n} \end{gathered}$$

Po ich wykonaniu nasza macierz prezentuje się jak poniżej

$$\left[\begin{array}{cccc|c} M_{1,1} & M_{1,2} & \cdots & M_{1,n} & y_1 \\ 0 & m_{2,2} & \cdots & m_{2,n} & Y_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & m_{n,2} & \cdots & m_{n,n} & Y_n \end{array}\right]$$

gdzie $m_{i,j}$ i Y_i to po prostu odpowiedniki $M_{i,j}$ i y_i po wykonanych operacjach.

2. Wyzerowanie pozostałych kolumn.

Postępujemy analogicznie do poprzedniego punktu. Aby wyzerować kolumnę drugą musimy wykonać takie operacje

$$\begin{gathered} A_{2,3}, B_{2,2,3}, C_{2,2,3}, \ldots, B_{2,n+1,3}, C_{2,n+1,3} \\ A_{2,4}, B_{2,2,4}, C_{2,2,4}, \ldots, B_{2,n+1,4}, C_{2,n+1,4} \\ \vdots \\ A_{2,n}, B_{2,2,n}, C_{2,2,n}, \ldots, B_{2,n+1,n}, C_{2,n+1,n} \end{gathered}$$

Postępując podobnie dla każdej kolejnej kolumny i wykonując takie operacje

$$\begin{gathered} A_{2,3}, B_{2,2,3}, C_{2,2,3}, \ldots, B_{2,n+1,3}, C_{2,n+1,3} \\ \vdots \\ A_{2,n}, B_{2,2,n}, C_{2,2,n}, \ldots, B_{2,n+1,n}, C_{2,n+1,n} \\ A_{3,4}, B_{3,3,4}, C_{3,3,4}, \ldots, B_{3,n+1,4}, C_{3,n+1,4} \\ \vdots \\ A_{3,n}, B_{3,3,n}, C_{3,3,n}, \ldots, B_{3,n+1,n}, C_{3,n+1,n} \\ \vdots \\ \vdots \\ A_{n-2,n-1}, B_{n-2,n-2,n-1}, C_{n-2,n-2,n-1}, \ldots, B_{n-2,n+1,n-1}, C_{n-2,n+1,n-1} \\ \vdots \\ A_{n-2,n}, B_{n-2,n-2,n}, C_{n-2,n-2,n}, \ldots, B_{n-2,n+1,n}, C_{n-2,n+1,n} \\ A_{n-1,n}, B_{n-1,n-1,n}, C_{n-1,n-1,n}, \ldots, B_{n-1,n+1,n}, C_{n-1,n+1,n} \end{gathered}$$

otrzymamy poniższą macierz

$$\left[\begin{array}{cccccc|c} M_{1,1} & M_{1,2} & M_{1,3} & \cdots & M_{1,n-1} & M_{1,n} & y_1 \\ 0 & m_{2,2} & m_{2,2} & \cdots & m_{2,n-1} & m_{2,n} & Y_2 \\ 0 & 0 & m_{3,3} & \cdots & m_{3,n-1} & m_{3,n} & Y_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & m_{n-1,n-1} & m_{n-1,n} & Y_{n-1} \\ 0 & 0 & 0 & \cdots & 0 & m_{n,n} & Y_n \end{array}\right]$$

gdzie $m_{i,j}$ i Y_i zdefiniowane tak jak wyżej.

Wyznaczenie alfabetu w sensie teorii śladów

Po wykonaniu tych dwóch kroków możemy określić nasz alfabet. W tym celu zdefiniujemy 3 zbiory odpowiednio dla zadań obliczeniowych A, B oraz C.

Zadania obliczeniowe A:

Zadania $A_{i,k}$ wykonujemy dla każdego wiersza i w taki sposób, że liczymy mnożnik w celu odjęcia od wiersza k dla $k \in i+1, \ldots, n$. Zatem nasz zbiór wygląda następująco

$$A = \{A_{i,k} | i \in \{1, \ldots, n\} \land k \in \{i+1, \ldots, n\}\}$$

Zadania obliczeniowe B:

Zadania $B_{i,j,k}$ wykonujemy dla każdego elementu wiersza i mnożąc go przez mnożnik w celu odjęcia od wiersza k. Zatem:

$$B = \{B_{i,j,k} | i \in \{1, \ldots, n\} \land k \in \{i+1, \ldots, n\} \land j \in \{1, \ldots, n+1\}\}$$

Zadania obliczeniowe C:

Zadania $C_{i,j,k}$ wykonujemy dla każdego elementu wiersza i odejmując go po wcześniejszym przemnożeniu od odpowiedniego elementu wiersza k. Zatem:

$$C = \{C_{i,j,k} | i \in \{1, \ldots, n\} \land k \in \{i+1, \ldots, n\} \land j \in \{1, \ldots, n+1\}\}$$

Ostatecznie alfabet możemy zapisać jako

$$\Sigma = A \cup B \cup C$$

Do wyznaczenia alfabetu tak naprawdę wystarczy nam tylko rozmiar n macierzy. Stąd funkcja wyznaczająca alfabet wygląda jak poniżej.

In [6]:
```
def create_alphabet(n):
    alphabet = []
    for i in range(1, n+1):
        for k in range(i+1, n+1):
            alphabet.append(f"A_{i},{k}")
            for j in range(i, n+2):
                alphabet.append(f'B_{i},{j},{k}')
                alphabet.append(f'C_{i},{j},{k}')
    return alphabet
```

Identyfikacja relacji zależności

Po określeniu wykonywanych operacji i zapisaniu alfabetu możemy wyznaczyć relacje zależności. Podobnie jak powyżej zrobimy to wyznaczając konkretne mniesze zbiory, a następnie zsumujemy je. Pierwszym zbiorem będzie po prostu relacja wszystkich zadań obliczeniowych A z odpowiednimi dla nich zadaniami B:

$$D_1 = \{(A_{i,k}, B_{i,j,k}) | A_{i,k} \in \Sigma \land B_{i,j,k} \in \Sigma\}$$

Drugim zbiorem będzie zbiór relacji wszystkich działań B z odpowiadającymi im działaniami C dla poszczególnych elementów:

$$D_2 = \{(B_{i,j,k}, C_{i,j,k}) | B_{i,j,k} \in \Sigma \land C_{i,j,k} \in \Sigma\}$$

Następny zbiór to relacja działań C z później wykonywanymi działaniami A wywoływanymi na wierszach, na których przed momentem wykonaliśmy operację C

$$D_3 = \{(C_{i_c,j_c,k_c}, A_{i_a,k_a}) | C_{i_c,j_c,k_c} \in \Sigma \land A_{i_a,k_a} \in \Sigma \land j_c = i_a = i_c + 1\}$$

Czwartym zbiorem jest relacja operacji C z zależnymi od nich później wywoływanych działań B

$$D_4 = \{(C_{i_c,j,k_c}, B_{i_b,j,k_b}) | C_{i_c,j,k_c} \in \Sigma \land B_{i_b,j,k_b} \in \Sigma \land i_b = i_c + 1 \land k_b = k_c + 1\}$$

Ostatnim zbiorem jest relacja dwóch operacji C wywoływanych na tym samym elemencie tego samego wiersza, ale wykorzystując inny wiersz do odejmowania

$$D_5 = \{(C_{i_1,j,k}, C_{i_2,j,k}) | C_{i_1,j,k} \in \Sigma \land C_{i_2,j,k} \in \Sigma \land i_1 < i_2\}$$

Ostatecznie nasz zbiór D będzie symetrią sumy tych zbiorów zsumowaną z macierzą identyczności I_Σ

$$D = sym\{\{D_1 \cup D_2 \cup D_3 \cup D_4 \cup D_5\}^+\} \cup I_\Sigma$$

Do wyznaczenia zbioru zależności służy poniższa funkcja, która przyjmuje rozmiar macierzy n oraz wygenerowany wcześniej alfabet.

In [7]:
```
def dependencies(n, alphabet):
    deps = []
    for i in range(1, n+1):
        for k in range(i+1, n+1):
            for j in range(i, n+2):
                # D_1
                deps.append((f'A_{i},{k}', f'B_{i},{j},{k}'))
                # D_2
                deps.append((f'B_{i},{j},{k}', f'C_{i},{j},{k}'))

    # D_3
    for i_a in range(2, n+1):
        for k_a in range(i_a+1, n+1):
            for k_c in range(i_a, n+1):
                deps.append((f'C_{i_a-1},{i_a},{k_c}', f'A_{i_a},{k_a}'))

    # D_4
    for i_c in range(1, n):
        for k_c in range(i_c+1, n+1):
            for j in range(i_c, n+2):
                if f'B_{i_c+1},{j},{k_c+1}' in alphabet:
                    deps.append((f'C_{i_c},{j},{k_c}', f'B_{i_c+1},{j},{k_c+1}'))

    # D_5
    for i1 in range(1, n):
        for i2 in range(i1+1, n+1):
            for k in range(i1+1, n+1):
                for j in range(1, n+2):
                    if f'C_{i1},{j},{k}' in alphabet and f'C_{i2},{j},{k}' in alphabet:
                        deps.append((f'C_{i1},{j},{k}', f'C_{i2},{j},{k}'))

    return deps
```

Obliczanie klas Foaty

Do wyznaczenia klas Foaty posłużą nam poniższe funkcje, które wykorzystane były już w sprawozdaniu z laboratorium nr 5, poza ostatnią z nich, która jest nowa. Prezentują się one tak:

- `FNF(edges, w)` - służy do wyznaczenia postaci normalnej Foaty,
- `f(G, fnf, vertex, step, visited)` - funkcja pomocnicza dla funkcji `FNF`,
- `determine_levels(fnf)` - służy do wyznaczenia poziomów postaci normalnej Foaty.

In [8]:

```
def FNF(edges, w):
    fnf = [[] for _ in range(len(w))]
    G = [[] for _ in range(len(w))]
    visited = [False for _ in range(len(w))]
    for edge in edges:
        G[edge[0]].append(edge[1])

    f(G, fnf, 0, 0, visited)
    for i in range(len(w)):
        if visited[i] is False:
            f(G, fnf, i, 0, visited)

    # Numbers to letters
    fnf = [sublist for sublist in fnf if sublist]
    for i in range(len(fnf)):
        for j in range(len(fnf[i])):
            fnf[i][j] = w[fnf[i][j]]

    return fnf, G

def f(G, fnf, vertex, step, visited):
    flag = False
    for i in range(len(fnf)):
        if vertex in fnf[i]:
            flag = True
    if flag is False:
        fnf[step].append(vertex)
    visited[vertex] = True
    for v in G[vertex]:
        visited[v] = True
        f(G, fnf, v, step+1, visited)

def determine_levels(fnf):
    levels = len(fnf)
    task_levels = {}
    for i in range(levels):
        for j in range(len(fnf[i])):
            task_levels[f'{fnf[i][j]}'] = i
    return task_levels
```

Wyprowadzenie grafu Diekerta

Graf Diekerta wyprowadzany jest korzystając z funkcji użytych w sprawozdaniu z laboratorium nr 5, jednak nieco zmodyfikowanych. Prezentują się one następująco:

- `create_graph(D, w)` - służy do stworzenia pierwszego grafu zależności, przyjmująca listę zależności i sekwencję wykonywania operacji,
- `delete_edges(G, number_graph)` - służy do usunięcia niepotrzebnych krawędzi, przyjmuje dwa grafy zwrócone przez powyższą funkcję,
- `show_graph(G, word, filename)` - służy do narysowania grafu Diekerta, przyjmuje graf, sekwencję operacji oraz nazwę pliku, do którego ma zapisać narysowany graf,
- `solve(n)` - funkcja główna wywołująca wszystkie powyższe.

Dodatkowo musimy zaimportować bibliotekę `graphviz` do wygenerowania grafu i funkcję `deepcopy` z pakietu `copy`.

In [20]:

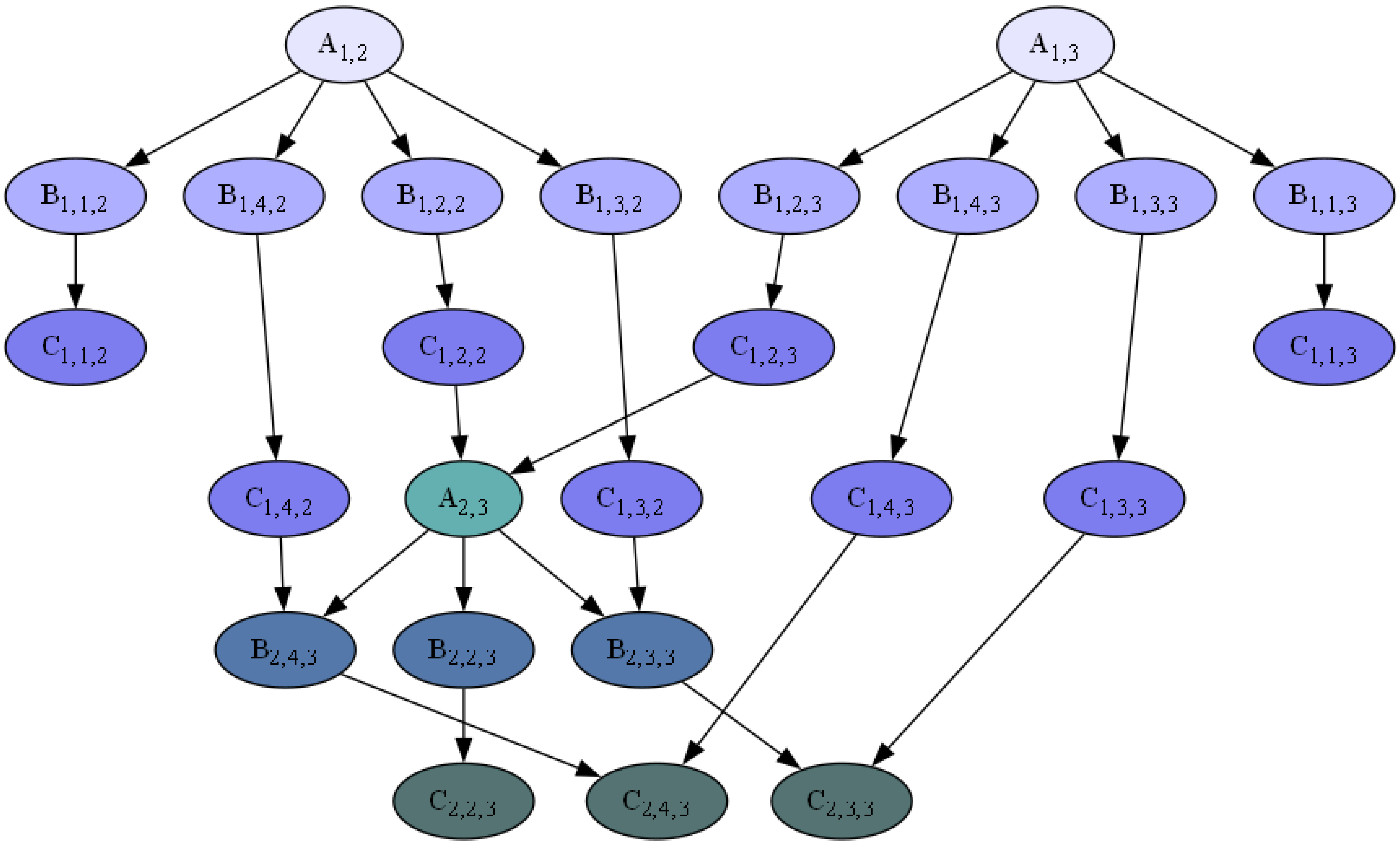
```
from copy import deepcopy
import graphviz

def create_graph(D, w):
    G = [[] for _ in range(len(w))]
    number_graph = [[] for _ in range(len(w))]
    for i in range(len(w)-1):
        letter = w[i]
        for j in range(i+1, len(w)):
            next_letter = w[j]
            if D.__contains__((letter, next_letter)) or D.__contains__((next_letter, letter)):
                G[i].append(next_letter)
                number_graph[i].append(j)
    return G, number_graph

def delete_edges(G, number_graph):
    edges = []
    for i in range(len(number_graph)):
        for j in range(len(number_graph[i])):
            edges.append((i, number_graph[i][j]))
```

```python
    edges_copy = deepcopy(edges)

    i = 0
    while i < len(edges_copy):
        j = 0
        while j < len(edges_copy):
            if edges_copy[i][1] == edges_copy[j][0]:  # if edges can be connected

                edges_copy.append((edges_copy[i][0], edges_copy[j][1]))

                # Removing duplicates
                counter = 0
                index = 0
                while index < len(edges_copy):
                    if edges_copy[index] == (edges_copy[i][0], edges_copy[j][1]):
                        counter += 1
                    index += 1
                if counter > 1 and ((edges_copy[i][0], edges_copy[j][1]) in edges):
                    edges.remove((edges_copy[i][0], edges_copy[j][1]))
            j += 1
        i += 1

    return edges

def show_graph(G, word, filename, task_levels):
    image = graphviz.Digraph(name=f'{filename}', format='png')
    image.attr(size="9,6")
    colors = ['#e6e6ff', '#b3b3ff', '#8080f2', '#68b3b3', '#577cad', '#557777']
    for v in range(len(G)):
        for u in G[v]:
            image.edge(str(v), str(u))
        label = word[v].replace("_", "<SUB>") + "</SUB>"
        image.node(str(v), label=f"<{label}>", style='filled', fillcolor=colors[task_levels[word[v]] % 6])

    image.render(view=True)

def solve(n):
    alph = create_alphabet(n)
    print("Alfabet: ", alph)
    D = dependencies(n, alph)
    print("Relacje zależności: ", D)
    G, number_graph = create_graph(D, alph)
    edges = delete_edges(G, number_graph)
    fnf, G = FNF(edges, alph)
    print("Postać normalna Foaty: ", fnf)
    task_levels = determine_levels(fnf)
    show_graph(G, alph, f'case{n}', task_levels)
```

Wywołajmy zatem funkcję `solve(n)` dla macierzy 3×3 w celu zobaczenia rezultatów:

In [22]:
```python
solve(3)
```

```
Alfabet:  ['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']
Relacje zależności:  [('A_1,2', 'B_1,1,2'), ('B_1,1,2', 'C_1,1,2'), ('A_1,2', 'B_1,2,2'), ('B_1,2,2', 'C_1,2,2'), ('A_1,2', 'B_1,3,2'), ('B_1,3,2', 'C_1,3,2'), ('A_1,2', 'B_1,4,2'), ('B_1,4,2', 'C_1,4,2'), ('A_1,3', 'B_1,1,3'), ('B_1,1,3', 'C_1,1,3'), ('A_1,3', 'B_1,2,3'), ('B_1,2,3', 'C_1,2,3'), ('A_1,3', 'B_1,3,3'), ('B_1,3,3', 'C_1,3,3'), ('A_1,3', 'B_1,4,3'), ('B_1,4,3', 'C_1,4,3'), ('A_2,3', 'B_2,2,3'), ('B_2,2,3', 'C_2,2,3'), ('A_2,3', 'B_2,3,3'), ('B_2,3,3', 'C_2,3,3'), ('A_2,3', 'B_2,4,3'), ('B_2,4,3', 'C_2,4,3'), ('C_1,2,2', 'A_2,3'), ('C_1,2,3', 'A_2,3'), ('C_1,2,2', 'B_2,2,3'), ('C_1,3,2', 'B_2,3,3'), ('C_1,4,2', 'B_2,4,3'), ('C_1,2,3', 'C_2,2,3'), ('C_1,3,3', 'C_2,3,3'), ('C_1,4,3', 'C_2,4,3')]
Postać normalna Foaty:  [['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']]
```

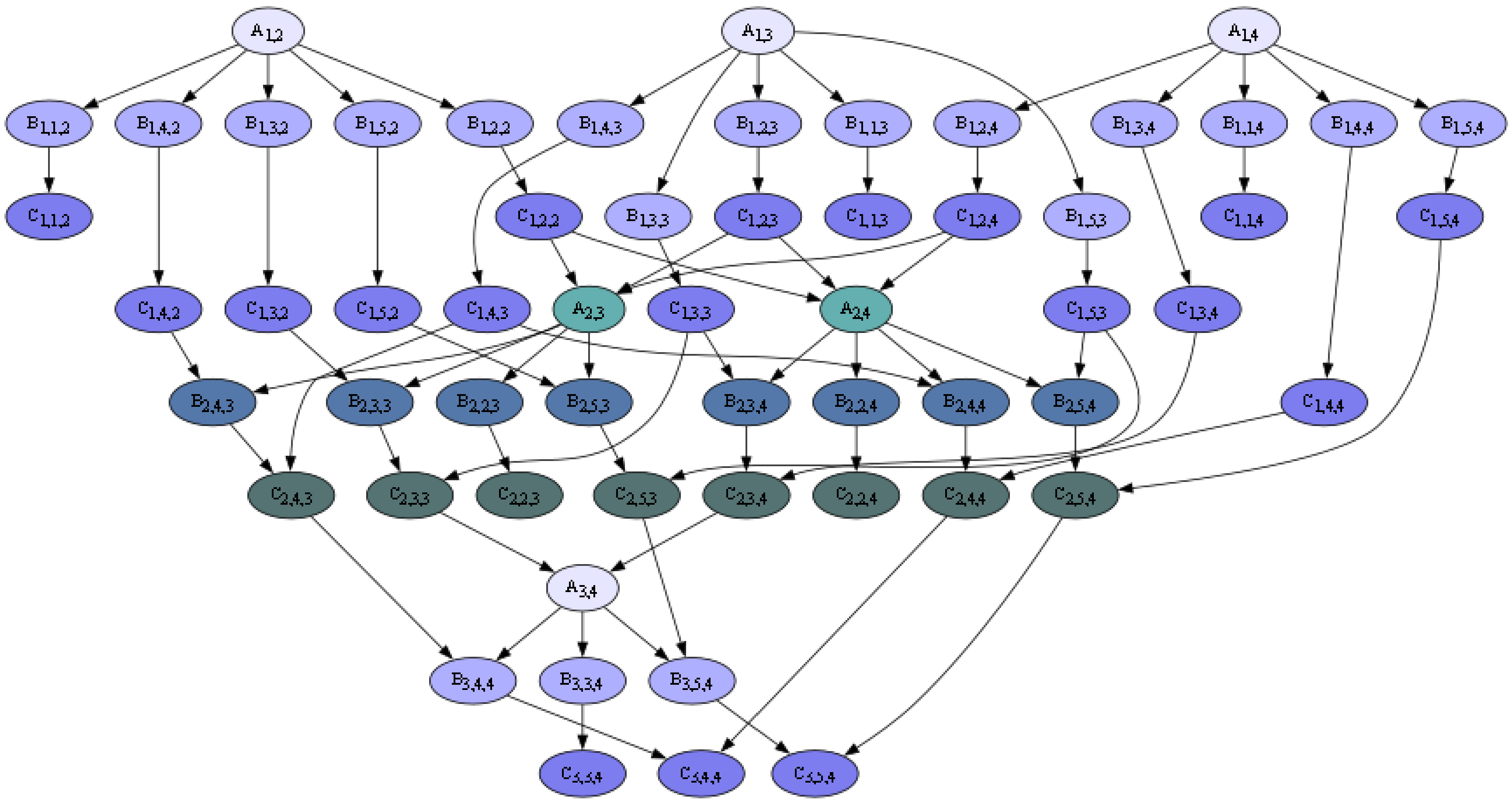


Wyniki wyglądają poprawnie, sprawdźmy więc dla większych macierzy.

$n = 4$:

In [24]:
```
solve(4)
```

```
Alfabet:  ['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']
Relacje zależności:  [('A_1,2', 'B_1,1,2'), ('B_1,1,2', 'C_1,1,2'), ('A_1,2', 'B_1,2,2'), ('B_1,2,2', 'C_1,2,2'), ('A_1,2', 'B_
1,3,2'), ('B_1,3,2', 'C_1,3,2'), ('A_1,2', 'B_1,4,2'), ('B_1,4,2', 'C_1,4,2'), ('A_1,2', 'B_1,5,2'), ('B_1,5,2', 'C_1,5,2'),
('A_1,3', 'B_1,1,3'), ('B_1,1,3', 'C_1,1,3'), ('A_1,3', 'B_1,2,3'), ('B_1,2,3', 'C_1,2,3'), ('A_1,3', 'B_1,3,3'), ('B_1,3,3',
'C_1,3,3'), ('A_1,3', 'B_1,4,3'), ('B_1,4,3', 'C_1,4,3'), ('A_1,3', 'B_1,5,3'), ('B_1,5,3', 'C_1,5,3'), ('A_1,4', 'B_1,1,4'),
('B_1,1,4', 'C_1,1,4'), ('A_1,4', 'B_1,2,4'), ('B_1,2,4', 'C_1,2,4'), ('A_1,4', 'B_1,3,4'), ('B_1,3,4', 'C_1,3,4'), ('A_1,4',
'B_1,4,4'), ('B_1,4,4', 'C_1,4,4'), ('A_1,4', 'B_1,5,4'), ('B_1,5,4', 'C_1,5,4'), ('A_2,3', 'B_2,2,3'), ('B_2,2,3', 'C_2,2,3'),
('A_2,3', 'B_2,3,3'), ('B_2,3,3', 'C_2,3,3'), ('A_2,3', 'B_2,4,3'), ('B_2,4,3', 'C_2,4,3'), ('A_2,3', 'B_2,5,3'), ('B_2,5,3',
'C_2,5,3'), ('A_2,4', 'B_2,2,4'), ('B_2,2,4', 'C_2,2,4'), ('A_2,4', 'B_2,3,4'), ('B_2,3,4', 'C_2,3,4'), ('A_2,4', 'B_2,4,4'),
('B_2,4,4', 'C_2,4,4'), ('A_2,4', 'B_2,5,4'), ('B_2,5,4', 'C_2,5,4'), ('A_3,4', 'B_3,3,4'), ('B_3,3,4', 'C_3,3,4'), ('A_3,4',
'B_3,4,4'), ('B_3,4,4', 'C_3,4,4'), ('A_3,4', 'B_3,5,4'), ('B_3,5,4', 'C_3,5,4'), ('C_1,2,2', 'A_2,3'), ('C_1,2,3', 'A_2,3'),
('C_1,2,4', 'A_2,3'), ('C_1,2,2', 'A_2,4'), ('C_1,2,3', 'A_2,4'), ('C_1,2,4', 'A_2,4'), ('C_2,3,3', 'A_3,4'), ('C_2,3,4', 'A_3,
4'), ('C_1,2,2', 'B_2,2,3'), ('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_1,2,3', 'B_2,2,4'),
('C_1,3,3', 'B_2,3,4'), ('C_1,4,3', 'B_2,4,4'), ('C_1,5,3', 'B_2,5,4'), ('C_2,3,3', 'B_3,3,4'), ('C_2,4,3', 'B_3,4,4'), ('C_2,
5,3', 'B_3,5,4'), ('C_1,2,3', 'C_2,2,3'), ('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_1,2,4',
'C_2,2,4'), ('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_1,3,4', 'C_3,3,4'), ('C_1,4,4', 'C_3,4,
4'), ('C_1,5,4', 'C_3,5,4'), ('C_2,3,4', 'C_3,3,4'), ('C_2,4,4', 'C_3,4,4'), ('C_2,5,4', 'C_3,5,4')]
Postać normalna Foaty:  [['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']]
```



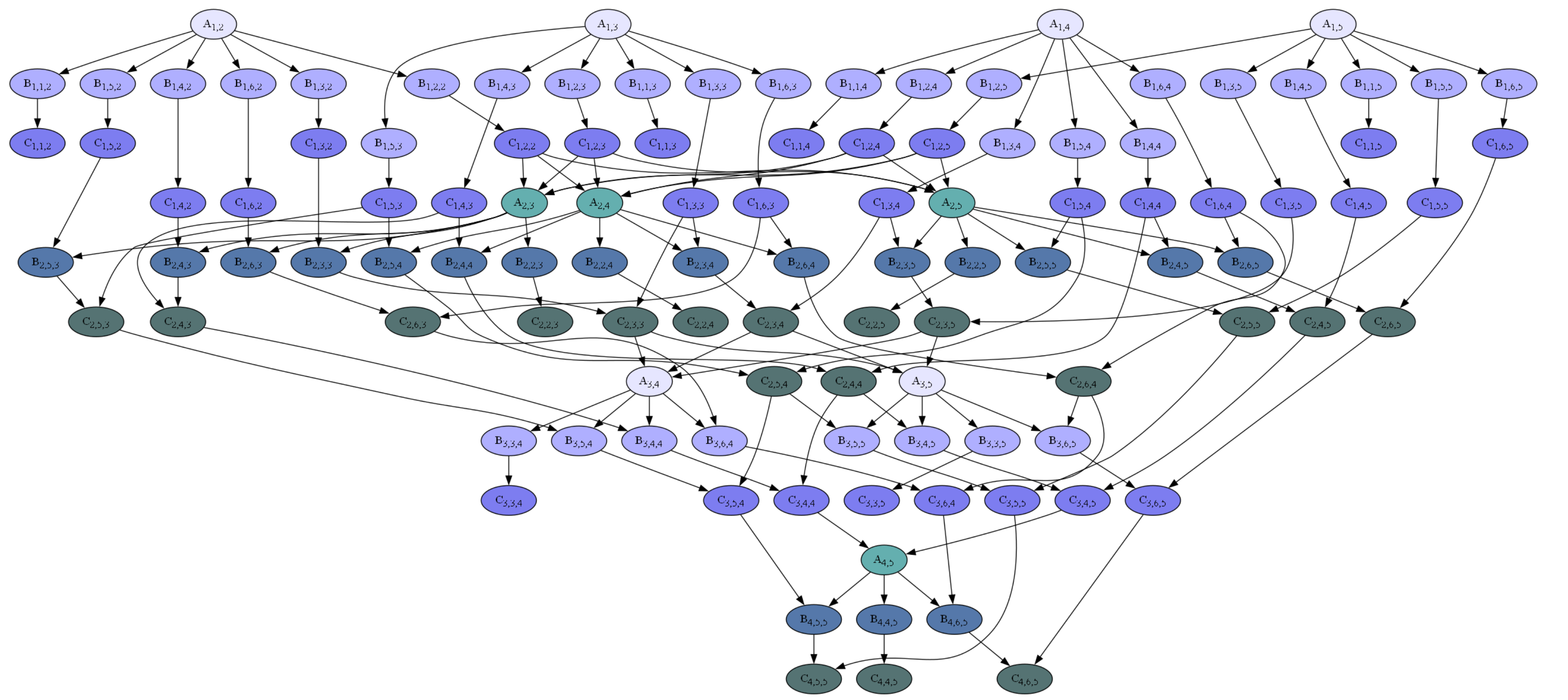
$n = 5$:

In [19]:
```
solve(5)
```

Alfabet: ['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', 'B_1,6,2', 'C_1,6,2', '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', 'B_1,6,3', 'C_1,6,3', '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', 'B_1,6,4', 'C_1,6,4', 'A_1,5', 'B_1,1,5', 'C_1,1,5', 'B_1,2,5', 'C_1,2,5', 'B_1,3,5', 'C_1,3,5', 'B_1,4,5', 'C_1,4,5', 'B_1,5,5', 'C_1,5,5', 'B_1,6,5', 'C_1,6,5', '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', 'B_2,6,3', 'C_2,6,3', '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', 'B_2,6,4', 'C_2,6,4', 'A_2,5', 'B_2,2,5', 'C_2,2,5', 'B_2,3,5', 'C_2,3,5', 'B_2,4,5', 'C_2,4,5', 'B_2,5,5', 'C_2,5,5', 'B_2,6,5', 'C_2,6,5', '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', 'B_3,6,4', 'C_3,6,4', 'A_3,5', 'B_3,3,5', 'C_3,3,5', 'B_3,4,5', 'C_3,4,5', 'B_3,5,5', 'C_3,5,5', 'B_3,6,5', 'C_3,6,5', 'A_4,5', 'B_4,4,5', 'C_4,4,5', 'B_4,5,5', 'C_4,5,5', 'B_4,6,5', 'C_4,6,5']

Relacje zależności: [('A_1,2', 'B_1,1,2'), ('B_1,1,2', 'C_1,1,2'), ('A_1,2', 'B_1,2,2'), ('B_1,2,2', 'C_1,2,2'), ('A_1,2', 'B_1,3,2'), ('B_1,3,2', 'C_1,3,2'), ('A_1,2', 'B_1,4,2'), ('B_1,4,2', 'C_1,4,2'), ('A_1,2', 'B_1,5,2'), ('B_1,5,2', 'C_1,5,2'), ('A_1,2', 'B_1,6,2'), ('B_1,6,2', 'C_1,6,2'), ('A_1,3', 'B_1,1,3'), ('B_1,1,3', 'C_1,1,3'), ('A_1,3', 'B_1,2,3'), ('B_1,2,3', 'C_1,2,3'), ('A_1,3', 'B_1,3,3'), ('B_1,3,3', 'C_1,3,3'), ('A_1,3', 'B_1,4,3'), ('B_1,4,3', 'C_1,4,3'), ('A_1,3', 'B_1,5,3'), ('B_1,5,3', 'C_1,5,3'), ('A_1,3', 'B_1,6,3'), ('B_1,6,3', 'C_1,6,3'), ('A_1,4', 'B_1,1,4'), ('B_1,1,4', 'C_1,1,4'), ('A_1,4', 'B_1,2,4'), ('B_1,2,4', 'C_1,2,4'), ('A_1,4', 'B_1,3,4'), ('B_1,3,4', 'C_1,3,4'), ('A_1,4', 'B_1,4,4'), ('B_1,4,4', 'C_1,4,4'), ('A_1,4', 'B_1,5,4'), ('B_1,5,4', 'C_1,5,4'), ('A_1,4', 'B_1,6,4'), ('B_1,6,4', 'C_1,6,4'), ('A_1,5', 'B_1,1,5'), ('B_1,1,5', 'C_1,1,5'), ('A_1,5', 'B_1,2,5'), ('B_1,2,5', 'C_1,2,5'), ('A_1,5', 'B_1,3,5'), ('B_1,3,5', 'C_1,3,5'), ('A_1,5', 'B_1,4,5'), ('B_1,4,5', 'C_1,4,5'), ('A_1,5', 'B_1,5,5'), ('B_1,5,5', 'C_1,5,5'), ('A_1,5', 'B_1,6,5'), ('B_1,6,5', 'C_1,6,5'), ('A_2,3', 'B_2,2,3'), ('B_2,2,3', 'C_2,2,3'), ('A_2,3', 'B_2,3,3'), ('B_2,3,3', 'C_2,3,3'), ('A_2,3', 'B_2,4,3'), ('B_2,4,3', 'C_2,4,3'), ('A_2,3', 'B_2,5,3'), ('B_2,5,3', 'C_2,5,3'), ('A_2,3', 'B_2,6,3'), ('B_2,6,3', 'C_2,6,3'), ('A_2,4', 'B_2,2,4'), ('B_2,2,4', 'C_2,2,4'), ('A_2,4', 'B_2,3,4'), ('B_2,3,4', 'C_2,3,4'), ('A_2,4', 'B_2,4,4'), ('B_2,4,4', 'C_2,4,4'), ('A_2,4', 'B_2,5,4'), ('B_2,5,4', 'C_2,5,4'), ('A_2,4', 'B_2,6,4'), ('B_2,6,4', 'C_2,6,4'), ('A_2,5', 'B_2,2,5'), ('B_2,2,5', 'C_2,2,5'), ('A_2,5', 'B_2,3,5'), ('B_2,3,5', 'C_2,3,5'), ('A_2,5', 'B_2,4,5'), ('B_2,4,5', 'C_2,4,5'), ('A_2,5', 'B_2,5,5'), ('B_2,5,5', 'C_2,5,5'), ('A_2,5', 'B_2,6,5'), ('B_2,6,5', 'C_2,6,5'), ('A_3,4', 'B_3,3,4'), ('B_3,3,4', 'C_3,3,4'), ('A_3,4', 'B_3,4,4'), ('B_3,4,4', 'C_3,4,4'), ('A_3,4', 'B_3,5,4'), ('B_3,5,4', 'C_3,5,4'), ('A_3,4', 'B_3,6,4'), ('B_3,6,4', 'C_3,6,4'), ('A_3,5', 'B_3,3,5'), ('B_3,3,5', 'C_3,3,5'), ('A_3,5', 'B_3,4,5'), ('B_3,4,5', 'C_3,4,5'), ('A_3,5', 'B_3,5,5'), ('B_3,5,5', 'C_3,5,5'), ('A_3,5', 'B_3,6,5'), ('B_3,6,5', 'C_3,6,5'), ('A_4,5', 'B_4,4,5'), ('B_4,4,5', 'C_4,4,5'), ('A_4,5', 'B_4,5,5'), ('B_4,5,5', 'C_4,5,5'), ('A_4,5', 'B_4,6,5'), ('B_4,6,5', 'C_4,6,5'), ('C_1,2,2', 'A_2,3'), ('C_1,2,3', 'A_2,3'), ('C_1,2,4', 'A_2,3'), ('C_1,2,5', 'A_2,3'), ('C_1,2,2', 'A_2,4'), ('C_1,2,3', 'A_2,4'), ('C_1,2,4', 'A_2,4'), ('C_1,2,5', 'A_2,4'), ('C_1,2,2', 'A_2,5'), ('C_1,2,3', 'A_2,5'), ('C_1,2,4', 'A_2,5'), ('C_1,2,5', 'A_2,5'), ('C_2,3,3', 'A_3,4'), ('C_2,3,4', 'A_3,4'), ('C_2,3,5', 'A_3,4'), ('C_2,3,3', 'A_3,5'), ('C_2,3,4', 'A_3,5'), ('C_2,3,5', 'A_3,5'), ('C_3,4,4', 'A_4,5'), ('C_3,4,5', 'A_4,5'), ('C_1,2,2', 'B_2,2,3'), ('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_1,6,2', 'B_2,6,3'), ('C_1,2,3', 'B_2,2,4'), ('C_1,3,3', 'B_2,3,4'), ('C_1,4,3', 'B_2,4,4'), ('C_1,5,3', 'B_2,5,4'), ('C_1,6,3', 'B_2,6,4'), ('C_1,2,4', 'B_2,2,5'), ('C_1,3,4', 'B_2,3,5'), ('C_1,4,4', 'B_2,4,5'), ('C_1,5,4', 'B_2,5,5'), ('C_1,6,4', 'B_2,6,5'), ('C_2,3,3', 'B_3,3,4'), ('C_2,4,3', 'B_3,4,4'), ('C_2,5,3', 'B_3,5,4'), ('C_2,6,3', 'B_3,6,4'), ('C_2,3,4', 'B_3,3,5'), ('C_2,4,4', 'B_3,4,5'), ('C_2,5,4', 'B_3,5,5'), ('C_2,6,4', 'B_3,6,5'), ('C_3,4,4', 'B_4,4,5'), ('C_3,5,4', 'B_4,5,5'), ('C_3,6,4', 'B_4,6,5'), ('C_1,2,3', 'C_2,2,3'), ('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_1,6,3', 'C_2,6,3'), ('C_1,2,4', 'C_2,2,4'), ('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_1,6,4', 'C_2,6,4'), ('C_1,2,5', 'C_2,2,5'), ('C_1,3,5', 'C_2,3,5'), ('C_1,4,5', 'C_2,4,5'), ('C_1,5,5', 'C_2,5,5'), ('C_1,6,5', 'C_2,6,5'), ('C_1,3,4', 'C_3,3,4'), ('C_1,4,4', 'C_3,4,4'), ('C_1,5,4', 'C_3,5,4'), ('C_1,6,4', 'C_3,6,4'), ('C_1,3,5', 'C_3,3,5'), ('C_1,4,5', 'C_3,4,5'), ('C_1,5,5', 'C_3,5,5'), ('C_1,6,5', 'C_3,6,5'), ('C_1,4,5', 'C_4,4,5'), ('C_1,5,5', 'C_4,5,5'), ('C_1,6,5', 'C_4,6,5'), ('C_2,3,4', 'C_3,3,4'), ('C_2,4,4', 'C_3,4,4'), ('C_2,5,4', 'C_3,5,4'), ('C_2,6,4', 'C_3,6,4'), ('C_2,3,5', 'C_3,3,5'), ('C_2,4,5', 'C_3,4,5'), ('C_2,5,5', 'C_3,5,5'), ('C_2,6,5', 'C_3,6,5'), ('C_2,4,5', 'C_4,4,5'), ('C_2,5,5', 'C_4,5,5'), ('C_2,6,5', 'C_4,6,5'), ('C_3,4,5', 'C_4,4,5'), ('C_3,5,5', 'C_4,5,5'), ('C_3,6,5', 'C_4,6,5')]

Postać normalna Foaty: [['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']]



Powtarzanie się kolorów na niższych poziomach nie oznacza, że operacje, których wierzchołki są bardzo daleko od siebie w sensie pośredniczących krawędzi, są w tej samej klasie Foaty - po prostu lista kolorów jest ograniczona i te kolory będą się powtarzać.

Algorytm eliminacji Gaussa

Oznaczmy:
$S_{i,k}$ - odjęcie odpowiednio przemnożonego wiersza i od wiersza k w celu wyzerowania danej kolumny.
$S_{i,k} = (A_{i,k}, B_{i,1,k}, C_{i,1,k}, \ldots, B_{i,n+1,k}, C_{i,n+1,k})$
Zatem wyzerowanie pierwszej kolumny macierzy $n \times n$ możemy zapisać jako ciąg

$$S_{1,2}, S_{1,3}, \ldots, S_{1,n}$$

Postępując analogicznie dla każdej kolejnej kolumny otrzymamy cały ciąg przedstawiający algorytm eliminacji Gaussa

$$g = (S_{1,2}, S_{1,3}, \ldots, S_{1,n}, S_{2,3}, S_{2,4}, \ldots, S_{2,n}, \ldots\ldots, S_{n-2,n-1}, S_{n-2,n}, S_{n-1,n})$$

Wykorzystując ten schemat przeprowadzimy eliminację Gaussa wielowątkowo za pomocą wcześniej wyprowadzonej postaci normalnej Foaty. W tym celu zmodyfikujemy nieco funkcję `solve(n)` tak, żeby teraz przyjmowała plik tekstowy z macierzą wejściową i nie generowała grafu Diekerta, ponieważ będzie on wyglądał tak samo jak dla przykładów powyżej. Ponadto dodamy trzy funkcje służące do operacji A, B i C na macierzy oraz trzy inne funkcje:

- `gaussian_elimination(M, fnf)` - przeprowadzająca wielowątkowo eliminację Gaussa,
- `worker(M, operation, A_tab, B_tab)` - wykonująca operacje macierzowe w danym wątku,
- `read_file(file)` - wczytująca macierz z pliku tekstowego.

Dodatkowo musimy zaimportować pakiet `threading` do obsługi wielowątkowości.

In [25]:
```python
import threading

def A(M, i, k):
    return M[k][i] / M[i][i]

def B(M, i, j, k, multiplier):
    return M[i][j] * multiplier

def C(M, i, j, k, to_substract):
    M[k][j] -= to_substract


def gaussian_elimination(M, fnf):
    levels = len(fnf)
    n = len(M)
    A_tab = [[0 for _ in range(n)] for i in range(n)]
    B_tab = [[[0 for _ in range(n)] for j in range(n+1)] for k in range(n)]
    for level in range(levels):
        threads_no = len(fnf[level])
        th = threading.Thread()
        threads = [th for _ in range(threads_no)]
        for t in range(threads_no):
            thread = threading.Thread(target=worker, args=(M, fnf[level][t], A_tab, B_tab))
            threads[t] = thread
            thread.start()
        for thread in threads:
            thread.join()
    return M

def worker(M, operation, A_tab, B_tab):
    if operation[0] == "A":
        i, k = int(operation[2])-1, int(operation[4])-1
        multiplier = A(M, i, k)
        A_tab[i][k] = multiplier
    elif operation[0] == "B":
        i, j, k = int(operation[2])-1, int(operation[4])-1, int(operation[6])-1
        to_subtract = B(M, i, j, k, A_tab[i][k])
        B_tab[i][j][k] = to_subtract
    elif operation[0] == "C":
        i, j, k = int(operation[2])-1, int(operation[4])-1, int(operation[6])-1
        C(M, i, j, k, B_tab[i][j][k])

def read_file(file):
    file1 = open(file, mode='r')

    n = int(file1.readline().strip())
    M = [[0 for _ in range(n)] for i in range(n)]   # Matrix
    y = [0 for _ in range(n)]   # y vector

    data = file1.readlines()
    counter = 0

    for line in data:
        if counter < n:
            linee = line.strip().split()
            for i in range(n):
                M[counter][i] = float(linee[i])
            counter += 1
        Y = line.strip().split()
        for i in range(n):
```

```python
            y[i] = float(Y[i])

    return n, M, y

def solve_2(file):
    n, M, y = read_file(file)
    alph = create_alphabet(n)
    D = dependencies(n, alph)
    G, number_graph = create_graph(D, alph)
    edges = delete_edges(G, number_graph)
    fnf, G = FNF(edges, alph)

    for i in range(n):
        M[i].append(y[i])

    result = gaussian_elimination(M, fnf)
    return result
```

Przetestujmy tą eliminację dla przykładu `in1.txt`.

In [26]:
```python
solve_2('in1.txt')
```

Out[26]:
```
[[2.0, 1.0, 3.0, 6.0], [0.0, 1.0, 2.0, 3.0], [0.0, 0.0, 3.0, 3.0]]
```

Wyniki wyglądają bardzo dobrze, gdyż zaszły takie zmiany jak poniżej:

$$\left[\begin{array}{ccc|c} 2.0 & 1.0 & 3.0 & 6.0 \\ 4.0 & 3.0 & 8.0 & 15.0 \\ 6.0 & 5.0 & 16.0 & 27.0 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 2.0 & 1.0 & 3.0 & 6.0 \\ 0.0 & 1.0 & 2.0 & 3.0 \\ 0.0 & 0.0 & 3.0 & 3.0 \end{array}\right]$$

In [30]:
```python
solve_2('in2.txt')
```

Out[30]:
```
[[12.0, 10.0, 3.0, 8.0, 73.0],
 [0.0, 26.666666666666668, 7.0, 7.333333333333334, 103.66666666666667],
 [0.0, 0.0, 2.4250000000000007, -15.65, -55.32499999999999],
 [0.0, 0.0, 0.0, -17.46391752577319, -69.85567010309276]]
```

Stworzymy jeszcze funkcję, która sprowadza macierz zwróconą przez `solve_2(file)` do macierzy jednostkowej.

In [51]:
```python
M = solve_2('in2.txt')

def matrix_to_diagonal(M):
    n =len(M)
    M[n-1][n] /= M[n-1][n-1]
    M[n-1][n-1] = 1.0
    for i in range(n-2, -1, -1):
        suma = M[i][n]
        for j in range(n-1, i, -1):
            suma -= M[i][j] * M[j][n]
            M[i][j] -= M[i][j]
        M[i][n] = (suma / M[i][i])
        M[i][i] = 1.0
    return M

matrix_to_diagonal(M)
```

Out[51]:
```
[[1.0, 0.0, 0.0, 0.0, 0.9999999999999994],
 [0.0, 1.0, 0.0, 0.0, 1.9999999999999991],
 [0.0, 0.0, 1.0, 0.0, 3.0000000000000044],
 [0.0, 0.0, 0.0, 1.0, 4.0]]
```

Wyniki wyglądają świetnie, poza tym, że z powodu zaokrągleń w trakcie wykonywania programu niektóre wartości niewiadomych są w niewielkim stopniu nie takie jak powinny. Sprawdźmy jeszcze dla innych przykładów.

In [54]:
```python
matrix_to_diagonal(solve_2('in4.txt'))
```

Out[54]:
```
[[1.0, 0.0, 0.0, 0.0, 18.0],
 [0.0, 1.0, 0.0, 0.0, 31.0],
 [0.0, 0.0, 1.0, 0.0, 26.0],
 [0.0, 0.0, 0.0, 1.0, 31.0]]
```

In [55]:
```python
matrix_to_diagonal(solve_2('in1.txt'))
```

Out[55]:
```
[[1.0, 0.0, 0.0, 1.0], [0.0, 1.0, 0.0, 1.0], [0.0, 0.0, 1.0, 1.0]]
```