# Teoria Współbieżności

# Raport 3 – Dla problemu współbieżnej eliminacji Gaussa – Relacja Zależności i Niezależności, Postać Normalna Foaty oraz graf Diekerta

**Krzysztof Swędzioł**

**Macierz na której tłumaczę kod :**

$$\left[\begin{array}{ccc|c} 2 & 1 & 3 & 6 \\ 4 & 3 & 8 & 15 \\ 6 & 5 & 16 & 27 \end{array}\right]$$

**Macierz końcowa :**

$$\left[\begin{array}{ccc|c} 2 & 1 & 3 & 6 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 3 & 3 \end{array}\right]$$

**Opis kodu :**

1. **create_summary_of_matrix(Matrix) –** funkcja, która tworzy nam sumę wszystkich niepodzielnych operacji, jakie należy wykonać w celu przeprowadzenia eliminacji Gaussa. Zrobiłem to tak żeby output był formatem możliwie zbliżony do przykładu ze skryptu. Funkcja przy okazji spisywania niepodzielnych operacji także rozwiązuje macierz niewspółbieżnie, dzięki czemu będzie można porównać wynik z funkcją, która robi to współbieżnie

wynik w skrypcie :

$$\Sigma = \{A_{1,2}, B_{1,1,2}, C_{1,1,2}, B_{1,2,2}, C_{1,2,2}, B_{1,3,2}, C_{1,3,2}, B_{1,4,2}, C_{1,4,2},$$
$$A_{1,3}, B_{1,1,3}, C_{1,1,3}, B_{1,2,3}, C_{1,2,3}, B_{1,3,3}, C_{1,3,3}, B_{1,4,3}, C_{1,4,3},$$
$$A_{2,3}, B_{2,2,3}, C_{2,2,3}, B_{2,3,3}, C_{2,3,3}, B_{2,4,3}, C_{2,4,3}\}$$

Mój wynik :

```
['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']
```

Jak widać, wszystko się zgadza.

Kod funkcji :

```
def create_summary_of_matrix(Matrix):
    Matrix_modified = copy.deepcopy(Matrix)
    rows_amount = len(Matrix)
    columns_amount = len(Matrix[0])
    row_index = 1;
    column_index = 0;
    operations_summary = []

    for i in range(rows_amount - 1):
        Matrix_modified = sorted(Matrix_modified, key=lambda row: count_leading_zeros(row))
        for j in range(row_index, rows_amount):

            operation = []
            curr_multiplier = Matrix_modified[j][column_index] / Matrix_modified[i][column_index]
            operation.append(["A", [i, j]])
            for k in range(column_index, columns_amount):
                curr_multiplied = Matrix_modified[i][k] * curr_multiplier
                Matrix_modified[j][k] = Matrix_modified[j][k] - curr_multiplied
                operation.append(["B", [i, k, j]])
                operation.append(["C", [i, k, j]])
            operations_summary.append(operation)

        row_index += 1
        column_index += 1

    readible_summary = copy.deepcopy(operations_summary)
    readible_summary = modify_for_better_readibility(readible_summary)

    return operations_summary, readible_summary, Matrix, Matrix_modified
```

2. **modify_for_better_readibility(operations_summary) i convert_readible_summary_to_strings(readible_summary)**

**-** Są to funkcje pomocnicze. Pierwsza dodaje do wszystkiego w outpucie 1 żeby indeksowanie było od 1 tak jak w skrypcie a nie od 0. Druga funkcja konwertuje tablicę wejściową na czytelne stringi.

Kody funkcji :

```python
def modify_for_better_readibility(operations_summary):
    for i in range(len(operations_summary)):
        for j in range(len(operations_summary[i])):
            for k in range(len(operations_summary[i][j][1])):
                operations_summary[i][j][1][k] = operations_summary[i][j][1][k] + 1
    return operations_summary

1 usage
def convert_readible_summary_to_strings(readible_summary):
    summary_copy = []
    for block in readible_summary:
        new_block = []
        for op in block:
            letter = op[0]
            indices = op[1]
            if letter == "A":
                new_block.append(f"A{indices[0]},{indices[1]}")
            elif letter == "B":
                new_block.append(f"B{indices[0]},{indices[1]},{indices[2]}")
            elif letter == "C":
                new_block.append(f"C{indices[0]},{indices[1]},{indices[2]}")
        summary_copy.append(new_block)
    return summary_copy
```

3. **count_leading_zeros(row) –** funkcja, która przesuwa wiersze jeśli okazało się że przy robieniu schodków, gdzieś wygenerowało się 0 obok tego zera, które chcieliśmy osiągnąć. W poleceniu było napisane że można założyć że taka sytuacja nie wystąpi, ale przeczytałem to po tym jak już zaimplementowałem funkcję więc szkoda ją było wyrzucać.

Kod funkcji :

```python
def count_leading_zeros(row):
    count = 0
    for elem in row:
        if elem == 0:
            count += 1
        else:
            break
    return count
```

4. **create_alphabet(readible_summary) –** Funkcja, która tworzy alfabet na podstawie przekazanego readible_summary (czytelnej formy Summary)

Wynik :

```
Alfabet dla danego problemu :
{'C1,4,3', 'C1,1,2', 'B2,2,3', 'C2,3,3', 'B1,4,3', 'B1,2,3', 'B1,2,2', 'A1,2', 'B2,4,3', 'C2,2,3', 'B1,3,2', 'B1,4,2', 'C1,3,3', 'A1,3', 'B2,3,3', 'C1,1,3', 'C1,3,2', 'B1,1,3'
```

```
'C2,4,3', 'C1,2,3', 'C1,2,2', 'C1,4,2', 'B1,3,3', 'B1,1,2', 'A2,3'}
```

Wynik przekopiowany z outputu (dla czytelności) :

{'C1,4,3', 'C1,1,2', 'B2,2,3', 'C2,3,3', 'B1,4,3', 'B1,2,3', 'B1,2,2', 'A1,2', 'B2,4,3', 'C2,2,3', 'B1,3,2', 'B1,4,2', 'C1,3,3', 'A1,3', 'B2,3,3', 'C1,1,3', 'C1,3,2', 'B1,1,3', 'C2,4,3', 'C1,2,3', 'C1,2,2', 'C1,4,2', 'B1,3,3', 'B1,1,2', 'A2,3'}

5. **create_dependencies(summary, matrix_size) –** funkcja tworząca tablicę zależności między operacjami na podstawie summary.

Kod funkcji :

```python
def create_dependencies(summary, matrix_size):
    dependencies = []
    n = len(summary)
    for i in range(n):
        m = len(summary[i])
        for j in range(m):
            curr = summary[i][j]
            if curr[0] == "A":
                if i > matrix_size-2:
                    first_index = curr[1][0]
                    second_index = curr[1][1]
                    depend1 = ["C", [first_index - 1, first_index, second_index]]
                    depend2 = ["C", [first_index - 1, first_index, first_index]]
                    dependencies.append([depend1, curr])
                    dependencies.append([depend2, curr])

            if curr[0] == "B":
                first_index = curr[1][0]
                second_index = curr[1][1]
                third_index = curr[1][2]
                if i > matrix_size-2:
                    depend1 = ["A", [first_index, third_index]]
                    depend2 = ["C", [first_index - 1, second_index, first_index]]
                    dependencies.append([depend1, curr])
                    dependencies.append([depend2, curr])

                else:
                    depend = ["A", [first_index, third_index]]
                    dependencies.append([depend, curr])
```

```python
                else:
                    depend = ["A", [first_index, third_index]]
                    dependencies.append([depend, curr])

            if curr[0] == "C":
                first_index = curr[1][0]
                second_index = curr[1][1]
                third_index = curr[1][2]
                if i > matrix_size-2:
                    depend1 = ["B", [first_index, second_index, third_index]]
                    depend2 = ["C", [first_index - 1, second_index, third_index]]
                    dependencies.append([depend1, curr])
                    dependencies.append([depend2, curr])

                else:
                    depend = ["B", [first_index, second_index, third_index]]
                    dependencies.append([depend, curr])
    return dependencies
```

Wyniki ze skryptu :

$$
\begin{aligned}
D = \text{sym}\{\{&(A_{1,2}, B_{1,1,2}), (A_{1,2}, B_{1,2,2}), (A_{1,2}, B_{1,3,2}), (A_{1,2}, B_{1,4,2}),\\
&(B_{1,1,2}, C_{1,1,2}), (B_{1,2,2}, C_{1,2,2}), (B_{1,3,2}, C_{1,3,2}), (B_{1,4,2}, C_{1,4,2}),\\
&(A_{1,3}, B_{1,1,3}), (A_{1,3}, B_{1,2,3}), (A_{1,3}, B_{1,3,3}), (A_{1,3}, B_{1,4,3}),\\
&(B_{1,1,3}, C_{1,1,3}), (B_{1,2,3}, C_{1,2,3}), (B_{1,3,3}, C_{1,3,3}), (B_{1,4,3}, C_{1,4,3}),\\
&(A_{2,3}, B_{2,2,3}), (A_{2,3}, B_{2,3,3}), (A_{2,3}, B_{2,4,3}),\\
&(B_{2,2,3}, C_{2,2,3}), (B_{2,3,3}, C_{2,3,3}), (B_{2,4,3}, C_{2,4,3}),\\
&(C_{1,2,2}, A_{2,3}), (C_{1,2,3}, A_{2,3}), (C_{1,2,2}, B_{2,2,3}), (C_{1,2,3}, B_{2,2,3}),\\
&(C_{1,3,2}, B_{2,3,3}), (C_{1,3,3}, C_{2,3,3}), (C_{1,4,2}, B_{2,4,3}), (C_{1,4,3}, C_{2,4,3})\}^{+}\} \cup I_{\Sigma}
\end{aligned}
$$

Moje wyniki :

```
Czytelne zależności w formie takiej jak w przykładzie ze skryptu :
['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'], ['C1,2,3', 'A2,3'], ['C1,2,2', 'A2,3'], ['A2,3', 'B2,2,3'], ['C1,2,2', 'B2,2,3']
['B2,2,3', 'C2,2,3'], ['C1,2,3', 'C2,2,3'], ['A2,3', 'B2,3,3'], ['C1,3,2', 'B2,3,3'], ['B2,3,3', 'C2,3,3']
['C1,3,3', 'C2,3,3'], ['A2,3', 'B2,4,3'], ['C1,4,2', 'B2,4,3'], ['B2,4,3', 'C2,4,3'], ['C1,4,3', 'C2,4,3']
```

Nie są w takiej samej kolejności, natomiast wszystko co miało być jest, tylko w innych miejscach tablicy (nie miałem pomysłu jak to posortować tak aby output wyglądał jak w skrypcie) Natomiast wszystko się zgadza.

6. **create_independencies(summary, dependencies) –** funkcja tworząca niezależności w grafie. Za pomocą funkcji pomocniczych przedstawionych poniżej działa tak – Najpierw tworzy tablicę zależności poprzez dodanie wszystkiego, czego nie ma w dependencies – no ale przecież coś może być od czegoś zależne poprzez przechodniość np. a zależne z b a b zależne z c implikuje że c nie jest niezależne od a. Rozwiązałem to w taki sposób że tworzę pomocniczy graf z zależności i sprawdzam czy z pierwszego elementu independencies można dojść do drugiego elementu independencies. Jeśli tak to znaczy że nie są one prawdziwie niezależne i należy je usunąć z independencies.

Kod funkcji :

```python
def create_independencies(summary, dependencies):
    independencies = []
    unique_set = []
    for block in summary:
        for op in block:
            if op not in unique_set:
                unique_set.append(op)

    for i in range(len(unique_set)):
        curr = unique_set[i]
        for j in range(len(unique_set)):
            curr_neigh = unique_set[j]
            if j != i:
                curr_set = [curr, curr_neigh]
                curr_set_reversed = [curr_neigh, curr]
                if curr_set not in independencies:
                    if curr_set_reversed not in independencies:
                        if curr_set not in dependencies:
                            if curr_set_reversed not in dependencies:
                                independencies.append(curr_set)

    to_remove = []
    for independency in independencies:
        if not is_really_independent(independency, dependencies):
            to_remove.append(independency)

    for rem in to_remove:
        if rem in independencies:
            independencies.remove(rem)

    return independencies
```

7. **is_really_independent(independency, dependencies), can_reach(start, target, dependencies), build_graph_ind(dependencies), op_to_tuple(op)**

**–** Szereg funkcji do sprawdzania czy dana niezależność rzeczywiście się zgadza i powinna zostać w tablicy. Ich działanie opisałem w opisie poprzedniej funkcji – can_reach sprawdza czy możliwe jest dojście z jednego wierzchołka do drugiego w grafie zależności tworzonym przez build_graph_ind

Kod funkcji :

```
def is_really_independent(independency, dependencies):
    op1, op2 = independency[0], independency[1]
    if can_reach(op1, op2, dependencies) or can_reach(op2, op1, dependencies):
        return False
    return True

2 usages
def can_reach(start, target, dependencies):
    graph = build_graph_ind(dependencies)
    start_node = op_to_tuple(start)
    target_node = op_to_tuple(target)
    visited = set()
    stack = [start_node]

    while stack:
        current = stack.pop()
        if current == target_node:
            return True
        if current not in visited:
            visited.add(current)
            if current in graph:
                stack.extend(graph[current])
    return False
```

```
def build_graph_ind(dependencies):
    graph = {}
    for dep in dependencies:
        opA, opB = dep[0], dep[1]
        nodeA = op_to_tuple(opA)
        nodeB = op_to_tuple(opB)
        if nodeA not in graph:
            graph[nodeA] = []
        graph[nodeA].append(nodeB)
    return graph

8 usages
def op_to_tuple(op):
    return (op[0], tuple(op[1]))
```

7. **create_readible_dependencies(dependencies) –** jeszcze jedna funkcja zwiększająca czytelność wyniku zależności

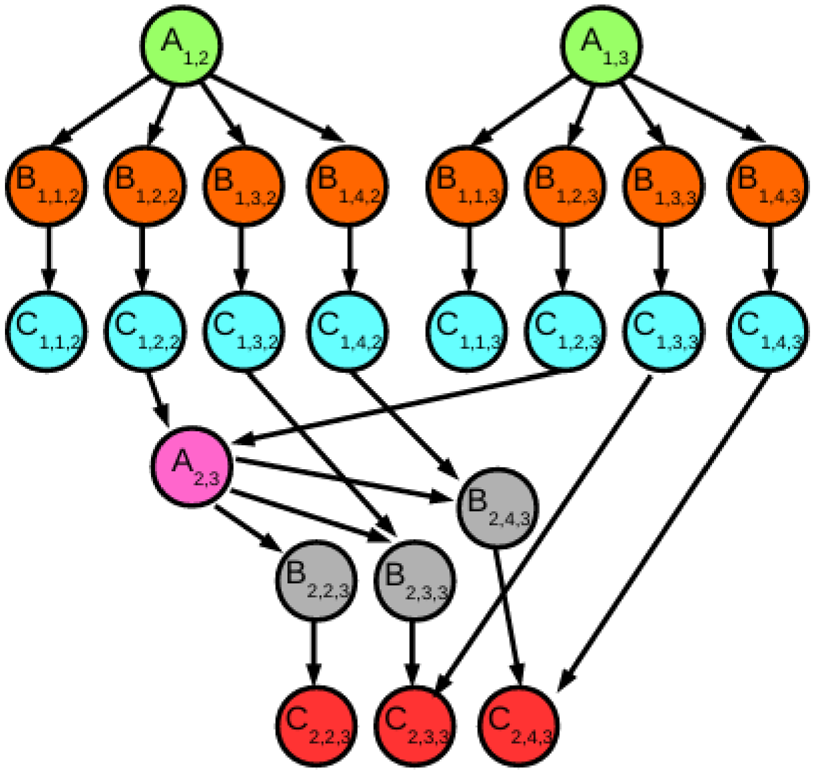
Kod funkcji :

```
def create_readible_dependencies(dependencies):
    readable_list = []
    for pair in dependencies:
        readable_pair = []
        for dep in pair:
            letter = dep[0]
            indices = dep[1]
            indices_str = ",".join(map(str, indices))
            readable_pair.append(f"{letter}{indices_str}")
        readable_list.append(readable_pair)
    return readable_list
```

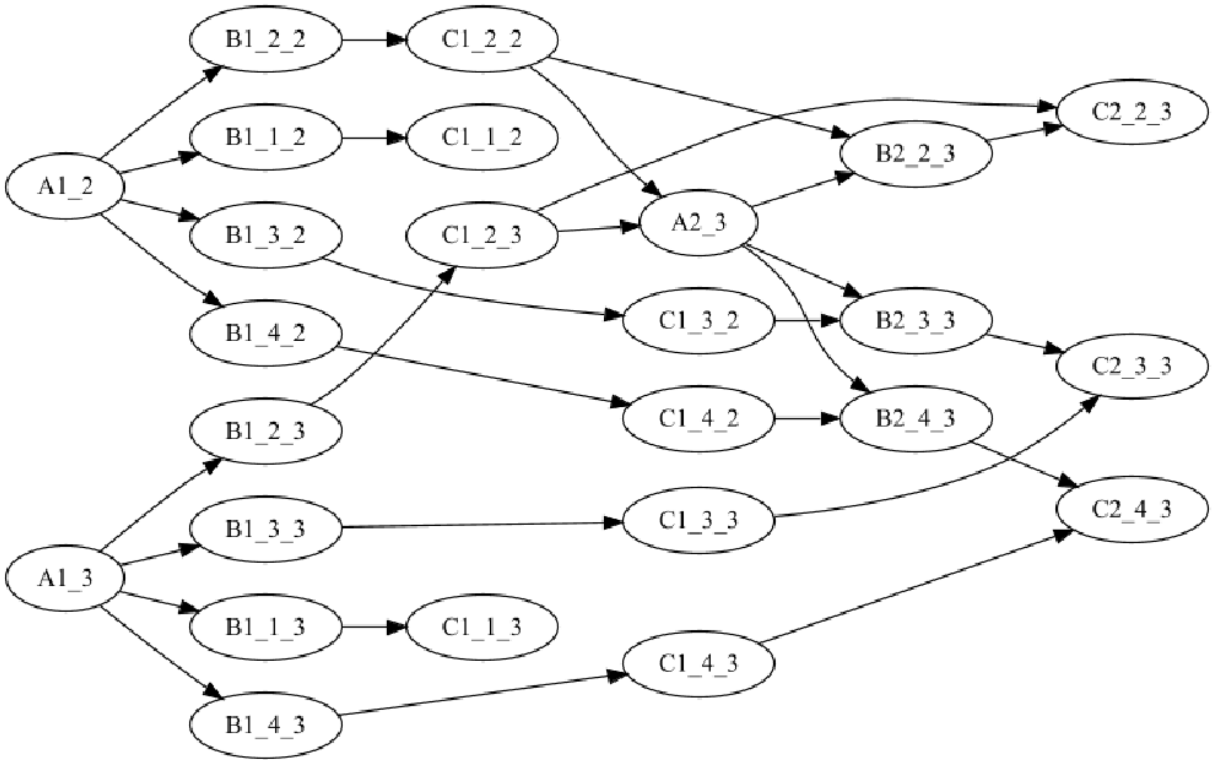
8. **create_diekert_graph(dependencies, filename="diekert_graph.dot") -**

Funkcja tworząca plik graficzny z grafem Diekerta.

Wynik ze skryptu :



Wynik działania Funkcji :




Jak widać tutaj również (z pominięciem skracania tras, co nie zmienia wyników) wszystko się zgadza.

Kod funkcji :

```python
def create_diekert_graph(dependencies, filename="diekert_graph.dot"):
    def op_to_str(op):
        letter = op[0]
        indices = op[1]
        if letter == "A":
            return f"A{indices[0]}_{indices[1]}"
        elif letter == "B":
            return f"B{indices[0]}_{indices[1]}_{indices[2]}"
        elif letter == "C":
            return f"C{indices[0]}_{indices[1]}_{indices[2]}"

    nodes = set()
    for dep in dependencies:
        opA, opB = dep[0], dep[1]
        nodes.add(op_to_str(opA))
        nodes.add(op_to_str(opB))

    with open(filename, "w") as f:
        f.write("digraph diekert {\n")
        f.write("    rankdir=LR;\n")
        for n in nodes:
            f.write(f"    \"{n}\";\n")
        for dep in dependencies:
            opA, opB = dep[0], dep[1]
            f.write(f"    \"{op_to_str(opA)}\" -> \"{op_to_str(opB)}\";\n")
        f.write("}\n")

    print(f"Graf Diekerta zapisano do pliku {filename}. Można go zwizualizować np. używając Graphviz.")
```

8. **foata_by_longest_path(dependencies) –** Przejście po grafie i wybór „Fal" o tej samej odległości od wierzchołków początkowych, czyli poszeregowanie w tablicy operacji – jeśli operacje są na tym samym poziomie to oznacza że można je wykonać współbieżnie.

Wynik ze skryptu :

Postać Normalna Foaty:

$$
\begin{aligned}
t = [\langle A \rangle]_{\equiv_I^+} &= [\{A_{1,2}, A_{1,3}\}]_{\equiv_I^+} \\
&\frown [\{B_{1,1,2}, B_{1,2,2}, B_{1,3,2}, B_{1,4,2}, B_{1,1,3}, B_{1,2,3}, B_{1,3,3}, B_{1,4,3}\}]_{\equiv_I^+} \\
&\frown [\{C_{1,1,2}, C_{1,2,2}, C_{1,3,2}, C_{1,4,2}, C_{1,1,3}, C_{1,2,3}, C_{1,3,3}, C_{1,4,3}\}]_{\equiv_I^+} \\
&\frown [\{A_{2,3}\}]_{\equiv_I^+} \frown [\{B_{2,2,3}, B_{2,3,3}, B_{2,4,3}\}]_{\equiv_I^+} \\
&\frown [\{C_{2,2,3}, C_{2,3,3}, C_{2,4,3}\}]_{\equiv_I^+} = \\
&[F_1]_{\equiv_I^+} \frown [F_2]_{\equiv_I^+} \frown [F_3]_{\equiv_I^+} \frown [F_4]_{\equiv_I^+} \frown [F_5]_{\equiv_I^+} \frown [F_6]_{\equiv_I^+}
\end{aligned}
$$

Mój wynik :

```
Postać normalna Foaty :
['A1_2', 'A1_3']
['B1_1_2', 'B1_1_3', 'B1_3_2', 'B1_4_3', 'B1_3_3', 'B1_2_2', 'B1_4_2', 'B1_2_3']
['C1_4_2', 'C1_2_3', 'C1_1_2', 'C1_1_3', 'C1_3_2', 'C1_4_3', 'C1_3_3', 'C1_2_2']
['A2_3']
['B2_2_3', 'B2_4_3', 'B2_3_3']
['C2_3_3', 'C2_2_3', 'C2_4_3']
```

Widać że tutaj również wszystko się zgadza.

Kod funkcji :

```python
def foata_by_longest_path(dependencies):
    graph, indeg = build_graph(dependencies)

    start_nodes = [n for n in indeg if indeg[n] == 0]

    level = {}
    for n in graph.keys():
        level[n] = -1
    for s in start_nodes:
        level[s] = 0

    q = deque(start_nodes)

    while q:
        u = q.popleft()
        for v in graph[u]:
            if level[v] < level[u] + 1:
                level[v] = level[u] + 1
            indeg[v] -= 1
            if indeg[v] == 0:
                q.append(v)

    max_level = max(level.values())
    foata_layers = [[] for _ in range(max_level + 1)]
    for node, lvl in level.items():
        foata_layers[lvl].append(op_to_str_from_tuple(node))

    return foata_layers
```

9. **op_to_str_from_tuple(t), build_graph(dependencies) –** Funkcje pomocnicze do tworzenia postaci normalnej Foaty. Pierwsza przekształca format na string a druga buduje graf żeby można było po nim przejść i powyznaczać odległości od startów.

Kod funkcji :

```python
def op_to_str_from_tuple(t):
    letter = t[0]
    indices = t[1]
    if letter == "A":
        return f"A{indices[0]}_{indices[1]}"
    elif letter == "B":
        return f"B{indices[0]}_{indices[1]}_{indices[2]}"
    elif letter == "C":
        return f"C{indices[0]}_{indices[1]}_{indices[2]}"
```

```python
def build_graph(dependencies):
    graph = {}
    indeg = {}
    nodes = set()

    for dep in dependencies:
        opA, opB = dep[0], dep[1]
        nodeA = op_to_tuple(opA)
        nodeB = op_to_tuple(opB)
        nodes.add(nodeA)
        nodes.add(nodeB)

    for n in nodes:
        graph[n] = []
        indeg[n] = 0

    for dep in dependencies:
        opA, opB = dep[0], dep[1]
        nodeA = op_to_tuple(opA)
        nodeB = op_to_tuple(opB)
        graph[nodeA].append(nodeB)
        indeg[nodeB] += 1

    return graph, indeg
```

10.**summary_printer(operations_summary), matrix_printer(Matrix), dependency_printer(dependencies), parse_op(op_str) –** Szereg funkcji do schludnego wypisywania wyników

Kod funkcji :

```python
def summary_printer(operations_summary):
    n = len(operations_summary)
    for i in range(n):
        print(operations_summary[i])

3 usages
def matrix_printer(Matrix):
    n = len(Matrix)
    for i in range(n):
        print(Matrix[i])

2 usages
def dependency_printer(dependencies):
    dep_list = list(dependencies)
    for i in range(0, len(dep_list), 5):
        line_deps = dep_list[i:i+5]
        print(", ".join(str(d) for d in line_deps))
```

```python
def parse_op(op_str):
    parts = op_str.split('_')
    letter = parts[0][0]
    if letter == 'A':
        i = int(parts[0][1:])
        j = int(parts[1])
        return ('A', [i,j])
    elif letter == 'B':
        i = int(parts[0][1:])
        k = int(parts[1])
        j = int(parts[2])
        return ('B', [i,k,j])
    elif letter == 'C':
        i = int(parts[0][1:])
        k = int(parts[1])
        j = int(parts[2])
        return ('C', [i,k,j])
    else:
        raise ValueError(f"Unknown operation: {op_str}")
```

**10.run_foata_layers(matrix, foata_layers), operation_A(matrix, i, j, multipliers), operation_B(matrix, i, k, j, multipliers, n_values), operation_C(matrix, i, k, j, n_values) –** Szereg funkcji do równoległego rozwiązywania macierzy na podstawie wcześniej wyliczonej postaci normalnej Foaty

Wynik ze skryptu :

$$\left[\begin{array}{ccc|c} 2 & 1 & 3 & 6 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 3 & 3 \end{array}\right]$$

Mój wynik z wykonania operacji współbieżnie :

```
Macierz po równoległej eliminacji Gaussa :
[2.0, 1.0, 3.0, 6.0]
[0.0, 1.0, 2.0, 3.0]
[0.0, 0.0, 3.0, 3.0]
```

Tutaj również wszystko się zgadza.

Kody funkcji :

```python
def operation_A(matrix, i, j, multipliers):
    i0 = i-1
    j0 = j-1
    pivot = matrix[i0][i0]
    if pivot == 0:
        mji = 0
    else:
        mji = matrix[j0][i0] / pivot
    multipliers[('A', i, j)] = mji

1 usage
def operation_B(matrix, i, k, j, multipliers, n_values):
    i0 = i-1
    j0 = j-1
    k0 = k-1
    mji = multipliers.get(('A', i, j))
    if mji is None:
        mji = 0
    nji = matrix[i0][k0]*mji
    n_values[('B', i, k, j)] = nji

1 usage
def operation_C(matrix, i, k, j, n_values):
    i0 = i-1
    j0 = j-1
    k0 = k-1
    nji = n_values.get(('B', i, k, j))
    if nji is None:
        nji = 0
    matrix[j0][k0] = matrix[j0][k0] - nji
```

```python
def run_foata_layers(matrix, foata_layers):
    multipliers = {}
    n_values = {}
    for layer in foata_layers:
        with ThreadPoolExecutor() as executor:
            futures = []
            for op_str in layer:
                letter, indices = parse_op(op_str)
                if letter == 'A':
                    i, j = indices
                    futures.append(executor.submit(operation_A, matrix, i, j, multipliers))
                elif letter == 'B':
                    i, k, j = indices
                    futures.append(executor.submit(operation_B, matrix, i, k, j, multipliers, n_values))
                elif letter == 'C':
                    i, k, j = indices
                    futures.append(executor.submit(operation_C, matrix, i, k, j, n_values))
            for f in futures:
                f.result()
```

10. **do_everything(given_matrix) –** Funkcja, która aktywuje wszystkie inne funkcje, wystarczy podać do niej macierz w formacie podanym poniżej i wszystko zrobi i wypisze

Format macierzy :

```python
test_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]]
do_everything(test_matrix)
```

Cały output :

Graf Diekerta zapisano do pliku diekert_graph.dot. Można go zwizualizować np. używając Graphviz.

Summary w wersji czytelnej (takiej jak w opisywanym w skrypcie przykładzie) Summary to suma wszystkich operacji po kolei, które muszą być wykonane w celu rozwiązania macierzy metodą eliminacji Gaussa :

['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']

-----------------------------------------------

Macierz wejściowa bez modyfikacji :

[2.0, 1.0, 3.0, 6.0]

[4.0, 3.0, 8.0, 15.0]

[6.0, 5.0, 16.0, 27.0]

-----------------------------------------------

Zmodyfikowana macierz. W trakcie pierwszej operacji - wyznaczania niepodzielnych operacji (Summary) możliwe jest jednoczesne rozwiązanie macierzy niewspółbieżnie. Robię to żeby potem móc porównać wyniki tej operacji z  rozwiązywaniem macierzy współbieżnie :

[2.0, 1.0, 3.0, 6.0]

[0.0, 1.0, 2.0, 3.0]

[0.0, 0.0, 3.0, 3.0]

-----------------------------------------------

Alfabet dla danego problemu :

{'C1,2,2', 'C1,2,3', 'A1,3', 'B2,3,3', 'A2,3', 'A1,2', 'B1,2,3', 'C1,3,3', 'C1,1,2', 'B1,2,2', 'C1,1,3', 'B1,1,2', 'C1,4,2', 'B1,4,3', 'C1,4,3', 'B1,3,2', 'B2,4,3', 'B2,2,3', 'C1,3,2', 'C2,4,3', 'C2,3,3', 'B1,4,2', 'C2,2,3', 'B1,1,3', 'B1,3,3'}

-----------------------------------------------

Czytelne zależności w formie takiej jak w przykładzie ze skryptu :

['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'], ['C1,2,3', 'A2,3'], ['C1,2,2', 'A2,3'], ['A2,3', 'B2,2,3'], ['C1,2,2', 'B2,2,3']

['B2,2,3', 'C2,2,3'], ['C1,2,3', 'C2,2,3'], ['A2,3', 'B2,3,3'], ['C1,3,2', 'B2,3,3'], ['B2,3,3', 'C2,3,3']

['C1,3,3', 'C2,3,3'], ['A2,3', 'B2,4,3'], ['C1,4,2', 'B2,4,3'], ['B2,4,3', 'C2,4,3'], ['C1,4,3', 'C2,4,3']

-----------------------------------------------------

Czytelne niezależności w formie takiej jak w przykładzie ze skryptu :

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

['A1,2', 'B1,3,3'], ['A1,2', 'C1,3,3'], ['A1,2', 'B1,4,3'], ['A1,2', 'C1,4,3'], ['B1,1,2', 'B1,2,2']

['B1,1,2', 'C1,2,2'], ['B1,1,2', 'B1,3,2'], ['B1,1,2', 'C1,3,2'], ['B1,1,2', 'B1,4,2'], ['B1,1,2', 'C1,4,2']

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

['B1,1,2', 'B1,3,3'], ['B1,1,2', 'C1,3,3'], ['B1,1,2', 'B1,4,3'], ['B1,1,2', 'C1,4,3'], ['B1,1,2', 'A2,3']

['B1,1,2', 'B2,2,3'], ['B1,1,2', 'C2,2,3'], ['B1,1,2', 'B2,3,3'], ['B1,1,2', 'C2,3,3'], ['B1,1,2', 'B2,4,3']

['B1,1,2', 'C2,4,3'], ['C1,1,2', 'B1,2,2'], ['C1,1,2', 'C1,2,2'], ['C1,1,2', 'B1,3,2'], ['C1,1,2', 'C1,3,2']

['C1,1,2', 'B1,4,2'], ['C1,1,2', 'C1,4,2'], ['C1,1,2', 'A1,3'], ['C1,1,2', 'B1,1,3'], ['C1,1,2', 'C1,1,3']

['C1,1,2', 'B1,2,3'], ['C1,1,2', 'C1,2,3'], ['C1,1,2', 'B1,3,3'], ['C1,1,2', 'C1,3,3'], ['C1,1,2', 'B1,4,3']

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

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

['B1,2,2', 'B1,4,2'], ['B1,2,2', 'C1,4,2'], ['B1,2,2', 'A1,3'], ['B1,2,2', 'B1,1,3'], ['B1,2,2', 'C1,1,3']

['B1,2,2', 'B1,2,3'], ['B1,2,2', 'C1,2,3'], ['B1,2,2', 'B1,3,3'], ['B1,2,2', 'C1,3,3'], ['B1,2,2', 'B1,4,3']

['B1,2,2', 'C1,4,3'], ['C1,2,2', 'B1,3,2'], ['C1,2,2', 'C1,3,2'], ['C1,2,2', 'B1,4,2'], ['C1,2,2', 'C1,4,2']

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

['C1,2,2', 'B1,3,3'], ['C1,2,2', 'C1,3,3'], ['C1,2,2', 'B1,4,3'], ['C1,2,2', 'C1,4,3'], ['B1,3,2', 'B1,4,2']

['B1,3,2', 'C1,4,2'], ['B1,3,2', 'A1,3'], ['B1,3,2', 'B1,1,3'], ['B1,3,2', 'C1,1,3'], ['B1,3,2', 'B1,2,3']

['B1,3,2', 'C1,2,3'], ['B1,3,2', 'B1,3,3'], ['B1,3,2', 'C1,3,3'], ['B1,3,2', 'B1,4,3'], ['B1,3,2', 'C1,4,3']

['B1,3,2', 'A2,3'], ['B1,3,2', 'B2,2,3'], ['B1,3,2', 'C2,2,3'], ['B1,3,2', 'B2,4,3'], ['B1,3,2', 'C2,4,3']

['C1,3,2', 'B1,4,2'], ['C1,3,2', 'C1,4,2'], ['C1,3,2', 'A1,3'], ['C1,3,2', 'B1,1,3'], ['C1,3,2', 'C1,1,3']

['C1,3,2', 'B1,2,3'], ['C1,3,2', 'C1,2,3'], ['C1,3,2', 'B1,3,3'], ['C1,3,2', 'C1,3,3'], ['C1,3,2', 'B1,4,3']

['C1,3,2', 'C1,4,3'], ['C1,3,2', 'A2,3'], ['C1,3,2', 'B2,2,3'], ['C1,3,2', 'C2,2,3'], ['C1,3,2', 'B2,4,3']

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

['B1,4,2', 'C1,2,3'], ['B1,4,2', 'B1,3,3'], ['B1,4,2', 'C1,3,3'], ['B1,4,2', 'B1,4,3'], ['B1,4,2', 'C1,4,3']

['B1,4,2', 'A2,3'], ['B1,4,2', 'B2,2,3'], ['B1,4,2', 'C2,2,3'], ['B1,4,2', 'B2,3,3'], ['B1,4,2', 'C2,3,3']

['C1,4,2', 'A1,3'], ['C1,4,2', 'B1,1,3'], ['C1,4,2', 'C1,1,3'], ['C1,4,2', 'B1,2,3'], ['C1,4,2', 'C1,2,3']

['C1,4,2', 'B1,3,3'], ['C1,4,2', 'C1,3,3'], ['C1,4,2', 'B1,4,3'], ['C1,4,2', 'C1,4,3'], ['C1,4,2', 'A2,3']

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

['B1,1,3', 'C1,2,3'], ['B1,1,3', 'B1,3,3'], ['B1,1,3', 'C1,3,3'], ['B1,1,3', 'B1,4,3'], ['B1,1,3', 'C1,4,3']

['B1,1,3', 'A2,3'], ['B1,1,3', 'B2,2,3'], ['B1,1,3', 'C2,2,3'], ['B1,1,3', 'B2,3,3'], ['B1,1,3', 'C2,3,3']

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

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

['C1,1,3', 'C2,2,3'], ['C1,1,3', 'B2,3,3'], ['C1,1,3', 'C2,3,3'], ['C1,1,3', 'B2,4,3'], ['C1,1,3', 'C2,4,3']

['B1,2,3', 'B1,3,3'], ['B1,2,3', 'C1,3,3'], ['B1,2,3', 'B1,4,3'], ['B1,2,3', 'C1,4,3'], ['C1,2,3', 'B1,3,3']

['C1,2,3', 'C1,3,3'], ['C1,2,3', 'B1,4,3'], ['C1,2,3', 'C1,4,3'], ['B1,3,3', 'B1,4,3'], ['B1,3,3', 'C1,4,3']

['B1,3,3', 'A2,3'], ['B1,3,3', 'B2,2,3'], ['B1,3,3', 'C2,2,3'], ['B1,3,3', 'B2,3,3'], ['B1,3,3', 'B2,4,3']

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

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

['B1,4,3', 'B2,2,3'], ['B1,4,3', 'C2,2,3'], ['B1,4,3', 'B2,3,3'], ['B1,4,3', 'C2,3,3'], ['B1,4,3', 'B2,4,3']

['C1,4,3', 'A2,3'], ['C1,4,3', 'B2,2,3'], ['C1,4,3', 'C2,2,3'], ['C1,4,3', 'B2,3,3'], ['C1,4,3', 'C2,3,3']

['C1,4,3', 'B2,4,3'], ['B2,2,3', 'B2,3,3'], ['B2,2,3', 'C2,3,3'], ['B2,2,3', 'B2,4,3'], ['B2,2,3', 'C2,4,3']

['C2,2,3', 'B2,3,3'], ['C2,2,3', 'C2,3,3'], ['C2,2,3', 'B2,4,3'], ['C2,2,3', 'C2,4,3'], ['B2,3,3', 'B2,4,3']

['B2,3,3', 'C2,4,3'], ['C2,3,3', 'B2,4,3'], ['C2,3,3', 'C2,4,3']

-----------------------------------------------------

Postać normalna Foaty :

['A1_2', 'A1_3']

['B1_2_2', 'B1_4_2', 'B1_2_3', 'B1_1_2', 'B1_1_3', 'B1_3_2', 'B1_4_3', 'B1_3_3']

['C1_2_2', 'C1_4_2', 'C1_2_3', 'C1_1_2', 'C1_1_3', 'C1_3_2', 'C1_4_3', 'C1_3_3']

['A2_3']

['B2_3_3', 'B2_2_3', 'B2_4_3']

['C2_3_3', 'C2_2_3', 'C2_4_3']

-----------------------------------------------------

Macierz po równoległej eliminacji Gaussa :

[2.0, 1.0, 3.0, 6.0]

[0.0, 1.0, 2.0, 3.0]

[0.0, 0.0, 3.0, 3.0]

Kod funkcji :

```python
def do_everything(given_matrix):
    n = len(given_matrix)
    summary, readible_summary, Matrix, Matrix_modified = create_summary_of_matrix(given_matrix)
    readible_string_summary = convert_readible_summary_to_strings(readible_summary)
    alphabet = create_alphabet(readible_summary)
    dependencies = create_dependencies(readible_summary, n)
    readible_dependencies = create_readible_dependencies(dependencies)
    independencies = create_independencies(readible_summary, dependencies)
    readible_independencies = create_readible_dependencies(independencies)
    create_diekert_graph(dependencies, filename="diekert_graph.dot")
    layers = foata_by_longest_path(dependencies)

    # print("Summary w postaci inżynierskiej (do użycia w dalszych etapach programu"
    #       "Summary to suma wszystkich operacji po kolei, które muszą być wykonane"
    #       "w celu rozwiązania macierzy metodą eliminacji Gaussa. :" )
    # summary_printer(summary)
    # print("-----------------------------------------------")
    print("Summary w wersji czytelnej (takiej jak w opisywanym w skrypcie przykładzie)"
          " Summary to suma wszystkich operacji po kolei, które muszą być wykonane"
           " w celu rozwiązania macierzy metodą eliminacji Gaussa : ")
    summary_printer(readible_string_summary)
    print("-------------------------------------------------")
    print("Macierz wejściowa bez modyfikacji : ")
    matrix_printer(Matrix)
    print("-------------------------------------------------")
    print("Zmodyfikowana macierz. W trakcie pierwszej operacji -"
          " wyznaczania niepodzielnych operacji (Summary) możliwe jest"
          " jednoczesne rozwiązanie macierzy niewspółbieżnie. Robię to"
          " żeby potem móc porównać wyniki tej operacji z "
          " rozwiązywaniem macierzy współbieżnie : ")
```

```python
matrix_printer(Matrix_modified)
print("--------------------------------------------------")
print("Alfabet dla danego problemu : ")
print(alphabet)
# print("--------------------------------------------------")
# print("Zależności w operacjach na danej macierzy w wersji do późniejszych modyfikacji w programie : ")
# dependency_printer(dependencies)
print("--------------------------------------------------")
print("Czytelne zależności w formie takiej jak w przykładzie ze skryptu : ")
dependency_printer(readible_dependencies)
# print("--------------------------------------------------")
# print("niezależności w operacjach na danej macierzy w wersji do późniejszych modyfikacji w programie :
# dependency_printer(independencies)
print("-----------------------------------------------------")
print("Czytelne niezależności w formie takiej jak w przykładzie ze skryptu : ")
dependency_printer(readible_independencies)
print("-----------------------------------------------------------")
print("Postać normalna Foaty :")
for l in layers:
    print(l)
parallel_matrix = copy.deepcopy(given_matrix)
run_foata_layers(parallel_matrix, layers)
print("-----------------------------------------------------------")
print("Macierz po równoległej eliminacji Gaussa :")
matrix_printer(parallel_matrix)
```

Cały kod :

```python
import copy
from concurrent.futures import ThreadPoolExecutor
from collections import deque
#A, [1,2] - znalezienie mnożnika 1 wiersza do odejmowania go od 2-giego wiersza
#B, [1,3,2] - pomnożenie 3-ciego elementu wiersza 1 przez mnożnik do odejmowania od 2-giego wiersza
#C, [1,3,2] - odjęcie 3-ciego elementu wiersza 1 od wiersza 2

def modify_for_better_readibility(operations_summary):
    for i in range(len(operations_summary)):
        for j in range(len(operations_summary[i])):
            for k in range(len(operations_summary[i][j][1])):
                operations_summary[i][j][1][k] = operations_summary[i][j][1][k] + 1
```

```
    return operations_summary

def convert_readible_summary_to_strings(readible_summary):
    summary_copy = []
    for block in readible_summary:
        new_block = []
        for op in block:
            letter = op[0]
            indices = op[1]
            if letter == "A":
                new_block.append(f"A{indices[0]},{indices[1]}")
            elif letter == "B":
                new_block.append(f"B{indices[0]},{indices[1]},{indices[2]}")
            elif letter == "C":
                new_block.append(f"C{indices[0]},{indices[1]},{indices[2]}")
        summary_copy.append(new_block)
    return summary_copy

def count_leading_zeros(row):
    count = 0
    for elem in row:
        if elem == 0:
            count += 1
        else:
            break
    return count

def create_summary_of_matrix(Matrix):
    Matrix_modified = copy.deepcopy(Matrix)
    rows_amount = len(Matrix)
    columns_amount = len(Matrix[0])
    row_index = 1;
    column_index = 0;
    operations_summary = []
```

```
    for i in range(rows_amount - 1):
        Matrix_modified = sorted(Matrix_modified, key=lambda row:
count_leading_zeros(row))
        for j in range(row_index, rows_amount):

            operation = []
            curr_multiplier = Matrix_modified[j][column_index] /
Matrix_modified[i][column_index]
            operation.append(["A", [i, j]])
            for k in range(column_index, columns_amount):
                curr_multiplied = Matrix_modified[i][k] * curr_multiplier
                Matrix_modified[j][k] = Matrix_modified[j][k] - curr_multiplied
                operation.append(["B", [i, k, j]])
                operation.append(["C", [i, k, j]])
            operations_summary.append(operation)

        row_index += 1
        column_index += 1

    readible_summary = copy.deepcopy(operations_summary)
    readible_summary = modify_for_better_readibility(readible_summary)

    return operations_summary, readible_summary, Matrix,
Matrix_modified

def create_alphabet(readible_summary):
    alphabet = set()
    for operation_block in readible_summary:
        for op in operation_block:
            letter = op[0]
            indices = op[1]
            if letter == "A":
                symbol = f"A{indices[0]},{indices[1]}"
            elif letter == "B":
                symbol = f"B{indices[0]},{indices[1]},{indices[2]}"
```

```
        elif letter == "C":
            symbol = f"C{indices[0]},{indices[1]},{indices[2]}"
        alphabet.add(symbol)
    return alphabet

def create_dependencies(summary, matrix_size):
    dependencies = []
    n = len(summary)
    for i in range(n):
        m = len(summary[i])
        for j in range(m):
            curr = summary[i][j]
            if curr[0] == "A":
                if i > matrix_size-2:
                    first_index = curr[1][0]
                    second_index = curr[1][1]
                    depend1 = ["C", [first_index - 1, first_index, second_index]]
                    depend2 = ["C", [first_index - 1, first_index, first_index]]
                    dependencies.append([depend1, curr])
                    dependencies.append([depend2, curr])

            if curr[0] == "B":
                first_index = curr[1][0]
                second_index = curr[1][1]
                third_index = curr[1][2]
                if i > matrix_size-2:
                    depend1 = ["A", [first_index, third_index]]
                    depend2 = ["C", [first_index - 1, second_index, first_index]]
                    dependencies.append([depend1, curr])
                    dependencies.append([depend2, curr])

                else:
                    depend = ["A", [first_index, third_index]]
                    dependencies.append([depend, curr])
```

```
        if curr[0] == "C":
            first_index = curr[1][0]
            second_index = curr[1][1]
            third_index = curr[1][2]
            if i > matrix_size-2:
                depend1 = ["B", [first_index, second_index, third_index]]
                depend2 = ["C", [first_index - 1, second_index, third_index]]
                dependencies.append([depend1, curr])
                dependencies.append([depend2, curr])

            else:
                depend = ["B", [first_index, second_index, third_index]]
                dependencies.append([depend, curr])
    return dependencies

def create_independencies(summary, dependencies):
    independencies = []
    unique_set = []
    for block in summary:
        for op in block:
            if op not in unique_set:
                unique_set.append(op)

    for i in range(len(unique_set)):
        curr = unique_set[i]
        for j in range(len(unique_set)):
            curr_neigh = unique_set[j]
            if j != i:
                curr_set = [curr, curr_neigh]
                curr_set_reversed = [curr_neigh, curr]
                if curr_set not in independencies:
                    if curr_set_reversed not in independencies:
                        if curr_set not in dependencies:
                            if curr_set_reversed not in dependencies:
                                independencies.append(curr_set)
```

```
    to_remove = []
    for independency in independencies:
        if not is_really_independent(independency, dependencies):
            to_remove.append(independency)

    for rem in to_remove:
        if rem in independencies:
            independencies.remove(rem)

    return independencies

def is_really_independent(independency, dependencies):
    op1, op2 = independency[0], independency[1]
    if can_reach(op1, op2, dependencies) or can_reach(op2, op1,
dependencies):
        return False
    return True

def can_reach(start, target, dependencies):
    graph = build_graph_ind(dependencies)
    start_node = op_to_tuple(start)
    target_node = op_to_tuple(target)
    visited = set()
    stack = [start_node]

    while stack:
        current = stack.pop()
        if current == target_node:
            return True
        if current not in visited:
            visited.add(current)
            if current in graph:
                stack.extend(graph[current])
    return False
```

```
def build_graph_ind(dependencies):
   graph = {}
   for dep in dependencies:
      opA, opB = dep[0], dep[1]
      nodeA = op_to_tuple(opA)
      nodeB = op_to_tuple(opB)
      if nodeA not in graph:
         graph[nodeA] = []
      graph[nodeA].append(nodeB)
   return graph

def op_to_tuple(op):
   return (op[0], tuple(op[1]))

def create_readible_dependencies(dependencies):
   readable_list = []
   for pair in dependencies:
      readable_pair = []
      for dep in pair:
         letter = dep[0]
         indices = dep[1]
         indices_str = ",".join(map(str, indices))
         readable_pair.append(f"{letter}{indices_str}")
      readable_list.append(readable_pair)
   return readable_list

def create_diekert_graph(dependencies, filename="diekert_graph.dot"):
   def op_to_str(op):
      letter = op[0]
      indices = op[1]
      if letter == "A":
         return f"A{indices[0]}_{indices[1]}"
      elif letter == "B":
         return f"B{indices[0]}_{indices[1]}_{indices[2]}"
```

```
    elif letter == "C":
      return f"C{indices[0]}_{indices[1]}_{indices[2]}"

  nodes = set()
  for dep in dependencies:
    opA, opB = dep[0], dep[1]
    nodes.add(op_to_str(opA))
    nodes.add(op_to_str(opB))

  with open(filename, "w") as f:
    f.write("digraph diekert {\n")
    f.write("    rankdir=LR;\n")
    for n in nodes:
      f.write(f"    \"{n}\";\n")
    for dep in dependencies:
      opA, opB = dep[0], dep[1]
      f.write(f"    \"{op_to_str(opA)}\" -> \"{op_to_str(opB)}\";\n")
    f.write("}\n")

  print(f"Graf Diekerta zapisano do pliku {filename}. Można go
zwizualizować np. używając Graphviz.")


def foata_by_longest_path(dependencies):
  graph, indeg = build_graph(dependencies)

  start_nodes = [n for n in indeg if indeg[n] == 0]

  level = {}
  for n in graph.keys():
    level[n] = -1
  for s in start_nodes:
    level[s] = 0

  q = deque(start_nodes)
```

```
    while q:
        u = q.popleft()
        for v in graph[u]:
            if level[v] < level[u] + 1:
                level[v] = level[u] + 1
            indeg[v] -= 1
            if indeg[v] == 0:
                q.append(v)

    max_level = max(level.values())
    foata_layers = [[] for _ in range(max_level + 1)]
    for node, lvl in level.items():
        foata_layers[lvl].append(op_to_str_from_tuple(node))

    return foata_layers



def op_to_str_from_tuple(t):
    letter = t[0]
    indices = t[1]
    if letter == "A":
        return f"A{indices[0]}_{indices[1]}"
    elif letter == "B":
        return f"B{indices[0]}_{indices[1]}_{indices[2]}"
    elif letter == "C":
        return f"C{indices[0]}_{indices[1]}_{indices[2]}"

def build_graph(dependencies):
    graph = {}
    indeg = {}
    nodes = set()

    for dep in dependencies:
        opA, opB = dep[0], dep[1]
```

```
        nodeA = op_to_tuple(opA)
        nodeB = op_to_tuple(opB)
        nodes.add(nodeA)
        nodes.add(nodeB)

    for n in nodes:
        graph[n] = []
        indeg[n] = 0

    for dep in dependencies:
        opA, opB = dep[0], dep[1]
        nodeA = op_to_tuple(opA)
        nodeB = op_to_tuple(opB)
        graph[nodeA].append(nodeB)
        indeg[nodeB] += 1

    return graph, indeg


def summary_printer(operations_summary):
    n = len(operations_summary)
    for i in range(n):
        print(operations_summary[i])

def matrix_printer(Matrix):
    n = len(Matrix)
    for i in range(n):
        print(Matrix[i])

def dependency_printer(dependencies):
    dep_list = list(dependencies)
    for i in range(0, len(dep_list), 5):
        line_deps = dep_list[i:i+5]
        print(", ".join(str(d) for d in line_deps))
```

```
def parse_op(op_str):
    parts = op_str.split('_')
    letter = parts[0][0]
    if letter == 'A':
        i = int(parts[0][1:])
        j = int(parts[1])
        return ('A', [i,j])
    elif letter == 'B':
        i = int(parts[0][1:])
        k = int(parts[1])
        j = int(parts[2])
        return ('B', [i,k,j])
    elif letter == 'C':
        i = int(parts[0][1:])
        k = int(parts[1])
        j = int(parts[2])
        return ('C', [i,k,j])
    else:
        raise ValueError(f"Unknown operation: {op_str}")

def operation_A(matrix, i, j, multipliers):
    i0 = i-1
    j0 = j-1
    pivot = matrix[i0][i0]
    if pivot == 0:
        mji = 0
    else:
        mji = matrix[j0][i0] / pivot
    multipliers[('A', i, j)] = mji

def operation_B(matrix, i, k, j, multipliers, n_values):
    i0 = i-1
    j0 = j-1
    k0 = k-1
    mji = multipliers.get(('A', i, j))
```

```
    if mji is None:
        mji = 0
    nji = matrix[i0][k0]*mji
    n_values[('B', i, k, j)] = nji

def operation_C(matrix, i, k, j, n_values):
    i0 = i-1
    j0 = j-1
    k0 = k-1
    nji = n_values.get(('B', i, k, j))
    if nji is None:
        nji = 0
    matrix[j0][k0] = matrix[j0][k0] - nji

def run_foata_layers(matrix, foata_layers):
    multipliers = {}
    n_values = {}
    for layer in foata_layers:
        with ThreadPoolExecutor() as executor:
            futures = []
            for op_str in layer:
                letter, indices = parse_op(op_str)
                if letter == 'A':
                    i, j = indices
                    futures.append(executor.submit(operation_A, matrix, i, j,
multipliers))
                elif letter == 'B':
                    i, k, j = indices
                    futures.append(executor.submit(operation_B, matrix, i, k, j,
multipliers, n_values))
                elif letter == 'C':
                    i, k, j = indices
                    futures.append(executor.submit(operation_C, matrix, i, k, j,
n_values))
            for f in futures:
```

```
            f.result()


def do_everything(given_matrix):
   n = len(given_matrix)
   summary, readible_summary, Matrix, Matrix_modified =
create_summary_of_matrix(given_matrix)
   readible_string_summary =
convert_readible_summary_to_strings(readible_summary)
   alphabet = create_alphabet(readible_summary)
   dependencies = create_dependencies(readible_summary, n)
   readible_dependencies =
create_readible_dependencies(dependencies)
   independencies = create_independencies(readible_summary,
dependencies)
   readible_independencies =
create_readible_dependencies(independencies)
   create_diekert_graph(dependencies, filename="diekert_graph.dot")
   layers = foata_by_longest_path(dependencies)

   # print("Summary w postaci inżynierskiej (do użycia w dalszych
etapach programu"
   #      "Summary to suma wszystkich operacji po kolei, które muszą być
wykonane"
   #      "w celu rozwiązania macierzy metodą eliminacji Gaussa. :" )
   # summary_printer(summary)
   # print("-----------------------------------------------")
   print("Summary w wersji czytelnej (takiej jak w opisywanym w skrypcie
przykładzie)"
        " Summary to suma wszystkich operacji po kolei, które muszą być
wykonane"
         " w celu rozwiązania macierzy metodą eliminacji Gaussa : ")
   summary_printer(readible_string_summary)
   print("-----------------------------------------------")
   print("Macierz wejściowa bez modyfikacji : ")
```

```
    matrix_printer(Matrix)
    print("------------------------------------------------")
    print("Zmodyfikowana macierz. W trakcie pierwszej operacji -"
          " wyznaczania niepodzielnych operacji (Summary) możliwe jest"
          " jednoczesne rozwiązanie macierzy niewspółbieżnie. Robię to"
          " żeby potem móc porównać wyniki tej operacji z "
          " rozwiązywaniem macierzy współbieżnie : ")
    matrix_printer(Matrix_modified)
    print("------------------------------------------------")
    print("Alfabet dla danego problemu : ")
    print(alphabet)
    # print("------------------------------------------------")
    # print("Zależności w operacjach na danej macierzy w wersji do późniejszych modyfikacji w programie : ")
    # dependency_printer(dependencies)
    print("------------------------------------------------")
    print("Czytelne zależności w formie takiej jak w przykładzie ze skryptu : ")
    dependency_printer(readible_dependencies)
    # print("------------------------------------------------")
    # print("niezależności w operacjach na danej macierzy w wersji do późniejszych modyfikacji w programie : ")
    # dependency_printer(independencies)
    print("--------------------------------------------------")
    print("Czytelne niezależności w formie takiej jak w przykładzie ze skryptu : ")
    dependency_printer(readible_independencies)
    print("----------------------------------------------------")
    print("Postać normalna Foaty :")
    for l in layers:
        print(l)
    parallel_matrix = copy.deepcopy(given_matrix)
    run_foata_layers(parallel_matrix, layers)
    print("----------------------------------------------------")
    print("Macierz po równoległej eliminacji Gaussa :")
```

```
matrix_printer(parallel_matrix)



test_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]]
do_everything(test_matrix)
```

## Wyniki :

W skrypcie :

Mamy zadany następujący układ równań

$$\begin{bmatrix} 2 & 1 & 3 \\ 4 & 3 & 8 \\ 6 & 5 & 16 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 15 \\ 27 \end{bmatrix} \tag{2}$$

Dla uproszczenia zapisu będziemy stosować poniższą notację.

$$\left[\begin{array}{ccc|c} 2 & 1 & 3 & 6 \\ 4 & 3 & 8 & 15 \\ 6 & 5 & 16 & 27 \end{array}\right] \tag{3}$$

Używamy pierwszego wiersza do „wyprodukowania" zer w pierwszej kolumnie.

Drugi wiersz = drugi wiersz - 2 * pierwszy wiersz

$$A_{1,2}, B_{1,1,2}, C_{1,1,2}, B_{1,2,2}, C_{1,2,2}, B_{1,3,2}, C_{1,3,2}, B_{1,4,2}, C_{1,4,2} \tag{4}$$

$$\left[\begin{array}{ccc|c} 2 & 1 & 3 & 6 \\ 0 & 1 & 2 & 3 \\ 6 & 5 & 16 & 27 \end{array}\right] \tag{5}$$

Trzeci wiersz = trzeci wiersz - 3 * pierwszy wiersz

$$A_{1,3}, B_{1,1,3}, C_{1,1,3}, B_{1,2,3}, C_{1,2,3}, B_{1,3,3}, C_{1,3,3}, B_{1,4,3}, C_{1,4,3} \tag{6}$$

$$\left[\begin{array}{ccc|c} 2 & 1 & 3 & 6 \\ 0 & 1 & 2 & 3 \\ 0 & 2 & 7 & 9 \end{array}\right] \tag{7}$$

Używamy drugiego wiersza do „wyprodukowania" zer w drugiej kolumnie

Trzeci wiersz = trzeci wiersz - 2 * drugi wiersz

$$A_{2,3}, B_{2,1,3}, C_{2,1,3}, B_{2,2,3}, C_{2,2,3}, B_{2,3,3}, C_{2,3,3}, B_{2,4,3}, C_{2,4,3} \tag{8}$$

$$\left[\begin{array}{ccc|c} 2 & 1 & 3 & 6 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 3 & 3 \end{array}\right] \tag{9}$$

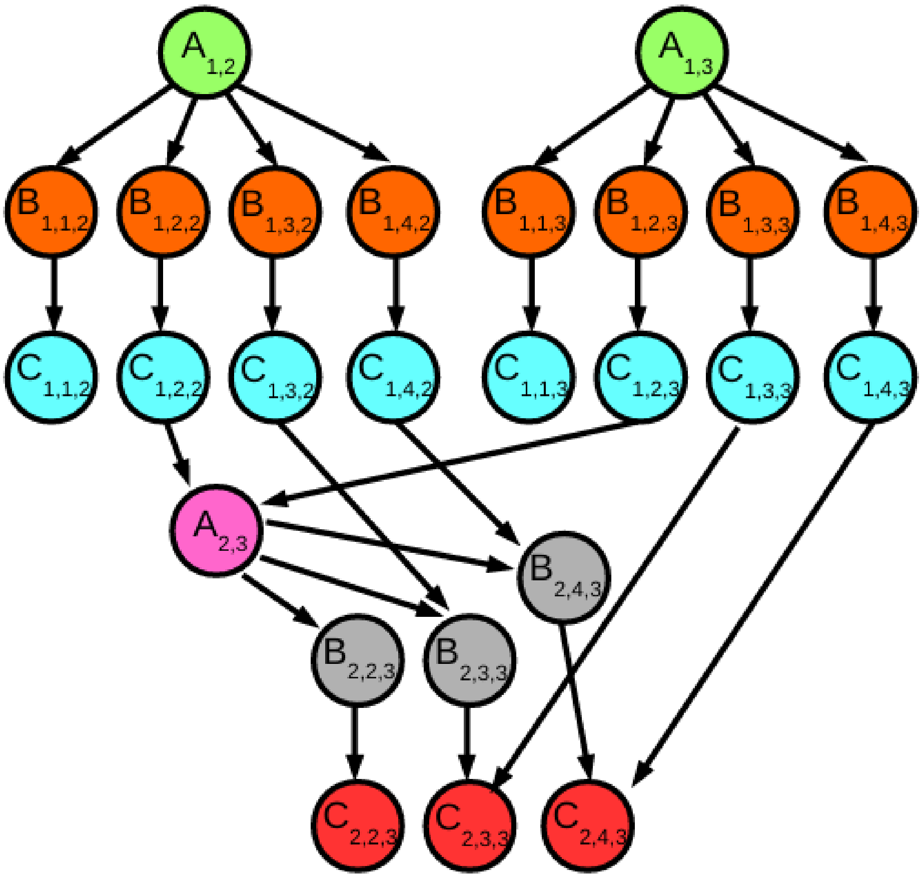
$$\begin{aligned}\Sigma = \{&A_{1,2}, B_{1,1,2}, C_{1,1,2}, B_{1,2,2}, C_{1,2,2}, B_{1,3,2}, C_{1,3,2}, B_{1,4,2}, C_{1,4,2}, \\ &A_{1,3}, B_{1,1,3}, C_{1,1,3}, B_{1,2,3}, C_{1,2,3}, B_{1,3,3}, C_{1,3,3}, B_{1,4,3}, C_{1,4,3}, \\ &A_{2,3}, B_{2,2,3}, C_{2,2,3}, B_{2,3,3}, C_{2,3,3}, B_{2,4,3}, C_{2,4,3}\}\end{aligned} \tag{10}$$

$$
\begin{aligned}
D = \mathrm{sym}\{\{&(A_{1,2}, B_{1,1,2}), (A_{1,2}, B_{1,2,2}), (A_{1,2}, B_{1,3,2}), (A_{1,2}, B_{1,4,2}),\\
&(B_{1,1,2}, C_{1,1,2}), (B_{1,2,2}, C_{1,2,2}), (B_{1,3,2}, C_{1,3,2}), (B_{1,4,2}, C_{1,4,2}),\\
&(A_{1,3}, B_{1,1,3}), (A_{1,3}, B_{1,2,3}), (A_{1,3}, B_{1,3,3}), (A_{1,3}, B_{1,4,3}),\\
&(B_{1,1,3}, C_{1,1,3}), (B_{1,2,3}, C_{1,2,3}), (B_{1,3,3}, C_{1,3,3}), (B_{1,4,3}, C_{1,4,3}),\\
&(A_{2,3}, B_{2,2,3}), (A_{2,3}, B_{2,3,3}), (A_{2,3}, B_{2,4,3}),\\
&(B_{2,2,3}, C_{2,2,3}), (B_{2,3,3}, C_{2,3,3}), (B_{2,4,3}, C_{2,4,3}),\\
&(C_{1,2,2}, A_{2,3}), (C_{1,2,3}, A_{2,3}), (C_{1,2,2}, B_{2,2,3}), (C_{1,2,3}, B_{2,2,3}),\\
&(C_{1,3,2}, B_{2,3,3}), (C_{1,3,3}, C_{2,3,3}), (C_{1,4,2}, B_{2,4,3}), (C_{1,4,3}, C_{2,4,3})\}^{+}\} \cup I_{\Sigma}
\end{aligned}
\tag{11}
$$

$$
\begin{aligned}
t = [w]_{\equiv_I^+} = [\langle &A_{1,2}, B_{1,1,2}, C_{1,1,2}, B_{1,2,2}, C_{1,2,2}, B_{1,3,2}, C_{1,3,2}, B_{1,4,2}, C_{1,4,2},\\
&A_{1,3}, B_{1,1,3}, C_{1,1,3}, B_{1,2,3}, C_{1,2,3}, B_{1,3,3}, C_{1,3,3}, B_{1,4,3}, C_{1,4,3},\\
&A_{2,3}, B_{2,2,3}, C_{2,2,3}, B_{2,3,3}, C_{2,3,3}, B_{2,4,3}, C_{2,4,3}\rangle]_{\equiv_I^+}
\end{aligned}
\tag{12}
$$

Postać Normalna Foaty:

$$
\begin{aligned}
t = [\langle A\rangle]_{\equiv_I^+} &= [\{A_{1,2}, A_{1,3}\}]_{\equiv_I^+}\\
&\frown[\{B_{1,1,2}, B_{1,2,2}, B_{1,3,2}, B_{1,4,2}, B_{1,1,3}, B_{1,2,3}, B_{1,3,3}, B_{1,4,3}\}]_{\equiv_I^+}\\
&\frown[\{C_{1,1,2}, C_{1,2,2}, C_{1,3,2}, C_{1,4,2}, C_{1,1,3}, C_{1,2,3}, C_{1,3,3}, C_{1,4,3}\}]_{\equiv_I^+}\\
&\frown[\{A_{2,3}\}]_{\equiv_I^+}\frown[\{B_{2,2,3}, B_{2,3,3}, B_{2,4,3}\}]_{\equiv_I^+}\\
&\frown[\{C_{2,2,3}, C_{2,3,3}, C_{2,4,3}\}]_{\equiv_I^+} =\\
&[F_1]_{\equiv_I^+}\frown[F_2]_{\equiv_I^+}\frown[F_3]_{\equiv_I^+}\frown[F_4]_{\equiv_I^+}\frown[F_5]_{\equiv_I^+}\frown[F_6]_{\equiv_I^+}
\end{aligned}
\tag{13}
$$




Rysunek 1: Graf Diekerta wraz z kolorowaniem

Moje wyniki :

Graf Diekerta zapisano do pliku diekert_graph.dot. Można go zwizualizować np. używając Graphviz.

Summary w wersji czytelnej (takiej jak w opisywanym w skrypcie przykładzie) Summary to suma wszystkich operacji po kolei, które muszą być wykonane w celu rozwiązania macierzy metodą eliminacji Gaussa :

['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']

-----------------------------------------------

Macierz wejściowa bez modyfikacji :

[2.0, 1.0, 3.0, 6.0]

[4.0, 3.0, 8.0, 15.0]

[6.0, 5.0, 16.0, 27.0]

-----------------------------------------------

Zmodyfikowana macierz. W trakcie pierwszej operacji - wyznaczania niepodzielnych operacji (Summary) możliwe jest jednoczesne rozwiązanie macierzy niewspółbieżnie. Robię to żeby potem móc porównać wyniki tej operacji z rozwiązywaniem macierzy współbieżnie :

[2.0, 1.0, 3.0, 6.0]

[0.0, 1.0, 2.0, 3.0]

[0.0, 0.0, 3.0, 3.0]

-----------------------------------------------

Alfabet dla danego problemu :

{'C1,4,2', 'C1,2,3', 'C1,2,2', 'C1,1,3', 'A2,3', 'C1,3,3', 'B2,3,3', 'C2,4,3', 'C1,3,2', 'A1,2', 'B1,3,3', 'B1,4,2', 'B1,2,2', 'B1,3,2', 'C1,4,3', 'C2,2,3', 'B1,1,3', 'B1,1,2', 'B1,4,3', 'C1,1,2', 'C2,3,3', 'B2,4,3', 'B1,2,3', 'A1,3', 'B2,2,3'}

-----------------------------------------------

Czytelne zależności w formie takiej jak w przykładzie ze skryptu :

['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'], ['C1,2,3', 'A2,3'], ['C1,2,2', 'A2,3'], ['A2,3', 'B2,2,3'], ['C1,2,2', 'B2,2,3']

['B2,2,3', 'C2,2,3'], ['C1,2,3', 'C2,2,3'], ['A2,3', 'B2,3,3'], ['C1,3,2', 'B2,3,3'], ['B2,3,3', 'C2,3,3']

['C1,3,3', 'C2,3,3'], ['A2,3', 'B2,4,3'], ['C1,4,2', 'B2,4,3'], ['B2,4,3', 'C2,4,3'], ['C1,4,3', 'C2,4,3']

-------------------------------------------------------

Postać normalna Foaty :

['A1_2', 'A1_3']

['B1_1_2', 'B1_1_3', 'B1_3_2', 'B1_4_3', 'B1_3_3', 'B1_2_2', 'B1_4_2', 'B1_2_3']

['C1_1_3', 'C1_3_2', 'C1_4_3', 'C1_3_3', 'C1_2_2', 'C1_4_2', 'C1_2_3', 'C1_1_2']

['A2_3']

['B2_2_3', 'B2_4_3', 'B2_3_3']

['C2_4_3', 'C2_3_3', 'C2_2_3']

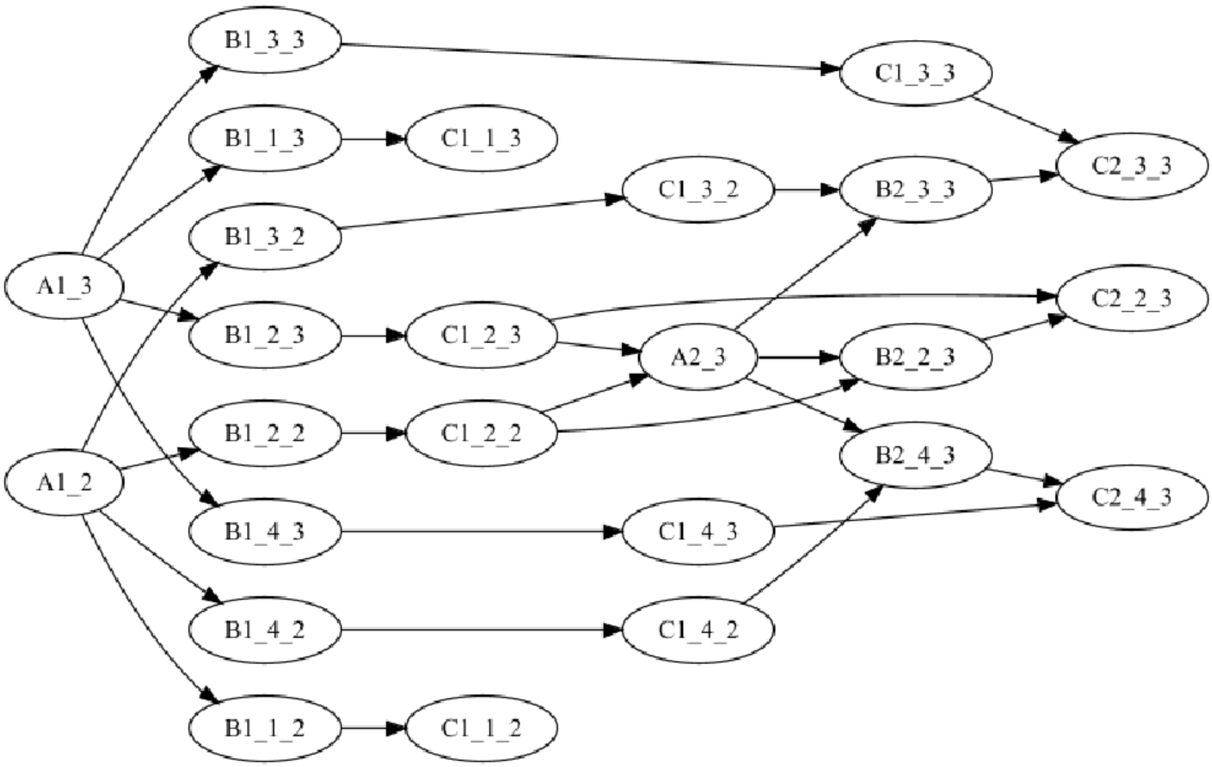
-------------------------------------------------------

Macierz po równoległej eliminacji Gaussa :

[2.0, 1.0, 3.0, 6.0]

[0.0, 1.0, 2.0, 3.0]

[0.0, 0.0, 3.0, 3.0]




Dla podanego inputu :

```
3
2.0 1.0 3.0
4.0 3.0 8.0
6.0 5.0 16.0
6.0 15.0 27.0
```

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

Output mojego programu :

Graf Diekerta zapisano do pliku diekert_graph.dot. Można go zwizualizować np. używając Graphviz.

Summary w wersji czytelnej (takiej jak w opisywanym w skrypcie przykładzie) Summary to suma wszystkich operacji po kolei, które muszą być wykonane w celu rozwiązania macierzy metodą eliminacji Gaussa :

['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']

['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']

['A2,3', 'B2,2,3', 'C2,2,3', 'B2,3,3', 'C2,3,3', 'B2,4,3', 'C2,4,3']

['A2,4', 'B2,2,4', 'C2,2,4', 'B2,3,4', 'C2,3,4', 'B2,4,4', 'C2,4,4']

['A3,4', 'B3,3,4', 'C3,3,4', 'B3,4,4', 'C3,4,4']

------------------------------------------------

Macierz wejściowa bez modyfikacji :

[2.0, 1.0, 3.0, 3.0]

[4.0, 3.0, 8.0, 8.0]

[6.0, 5.0, 16.0, 16.0]

[6.0, 15.0, 27.0, 27.0]

------------------------------------------------

Zmodyfikowana macierz. W trakcie pierwszej operacji - wyznaczania niepodzielnych operacji (Summary) możliwe jest jednoczesne rozwiązanie macierzy niewspółbieżnie. Robię to żeby potem móc porównać wyniki tej operacji z rozwiązywaniem macierzy współbieżnie :

[2.0, 1.0, 3.0, 3.0]

[0.0, 1.0, 2.0, 2.0]

[0.0, 0.0, 3.0, 3.0]

[0.0, 0.0, 0.0, 0.0]

------------------------------------------------

Alfabet dla danego problemu :

{'C1,3,2', 'B1,1,3', 'C1,2,2', 'B1,2,4', 'C1,3,4', 'C2,4,4', 'B1,4,3', 'C3,4,4', 'C1,1,4', 'B3,3,4', 'B1,3,3', 'B1,1,2', 'B1,4,2', 'B1,2,2', 'A2,3', 'C1,2,3', 'B2,3,3', 'C1,4,4', 'C2,4,3', 'B1,2,3', 'B1,4,4', 'C1,4,3', 'C1,2,4', 'A1,3', 'B2,4,4', 'B2,4,3', 'C1,3,3', 'A2,4', 'A1,2', 'B2,2,3', 'B2,3,4', 'C1,4,2', 'A1,4', 'C2,3,4', 'A3,4', 'C1,1,2', 'C2,3,3', 'B3,4,4', 'C1,1,3', 'B1,3,4', 'C2,2,3', 'B1,3,2', 'B2,2,4', 'C3,3,4', 'B1,1,4', 'C2,2,4'}

-----------------------------------------------

Czytelne zależności w formie takiej jak w przykładzie ze skryptu :

['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'], ['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'], ['C1,2,3', 'A2,3']

['C1,2,2', 'A2,3'], ['A2,3', 'B2,2,3'], ['C1,2,2', 'B2,2,3'], ['B2,2,3', 'C2,2,3'], ['C1,2,3', 'C2,2,3']

['A2,3', 'B2,3,3'], ['C1,3,2', 'B2,3,3'], ['B2,3,3', 'C2,3,3'], ['C1,3,3', 'C2,3,3'], ['A2,3', 'B2,4,3']

['C1,4,2', 'B2,4,3'], ['B2,4,3', 'C2,4,3'], ['C1,4,3', 'C2,4,3'], ['C1,2,4', 'A2,4'], ['C1,2,2', 'A2,4']

['A2,4', 'B2,2,4'], ['C1,2,2', 'B2,2,4'], ['B2,2,4', 'C2,2,4'], ['C1,2,4', 'C2,2,4'], ['A2,4', 'B2,3,4']

['C1,3,2', 'B2,3,4'], ['B2,3,4', 'C2,3,4'], ['C1,3,4', 'C2,3,4'], ['A2,4', 'B2,4,4'], ['C1,4,2', 'B2,4,4']

['B2,4,4', 'C2,4,4'], ['C1,4,4', 'C2,4,4'], ['C2,3,4', 'A3,4'], ['C2,3,3', 'A3,4'], ['A3,4', 'B3,3,4']

['C2,3,3', 'B3,3,4'], ['B3,3,4', 'C3,3,4'], ['C2,3,4', 'C3,3,4'], ['A3,4', 'B3,4,4'], ['C2,4,3', 'B3,4,4']

['B3,4,4', 'C3,4,4'], ['C2,4,4', 'C3,4,4']

---

Postać normalna Foaty :

['A1_2', 'A1_3', 'A1_4']

['B1_3_4', 'B1_4_2', 'B1_2_3', 'B1_2_4', 'B1_1_2', 'B1_1_3', 'B1_3_2', 'B1_1_4', 'B1_4_3', 'B1_3_3', 'B1_4_4', 'B1_2_2']

['C1_1_3', 'C1_3_2', 'C1_1_4', 'C1_4_3', 'C1_3_3', 'C1_4_4', 'C1_2_2', 'C1_3_4', 'C1_4_2', 'C1_2_3', 'C1_2_4', 'C1_1_2']

['A2_3', 'A2_4']

['B2_3_3', 'B2_3_4', 'B2_2_3', 'B2_4_3', 'B2_2_4', 'B2_4_4']

['C2_2_3', 'C2_4_3', 'C2_2_4', 'C2_4_4', 'C2_3_3', 'C2_3_4']

['A3_4']

['B3_4_4', 'B3_3_4']

['C3_4_4', 'C3_3_4']

-----------------------------------------------------

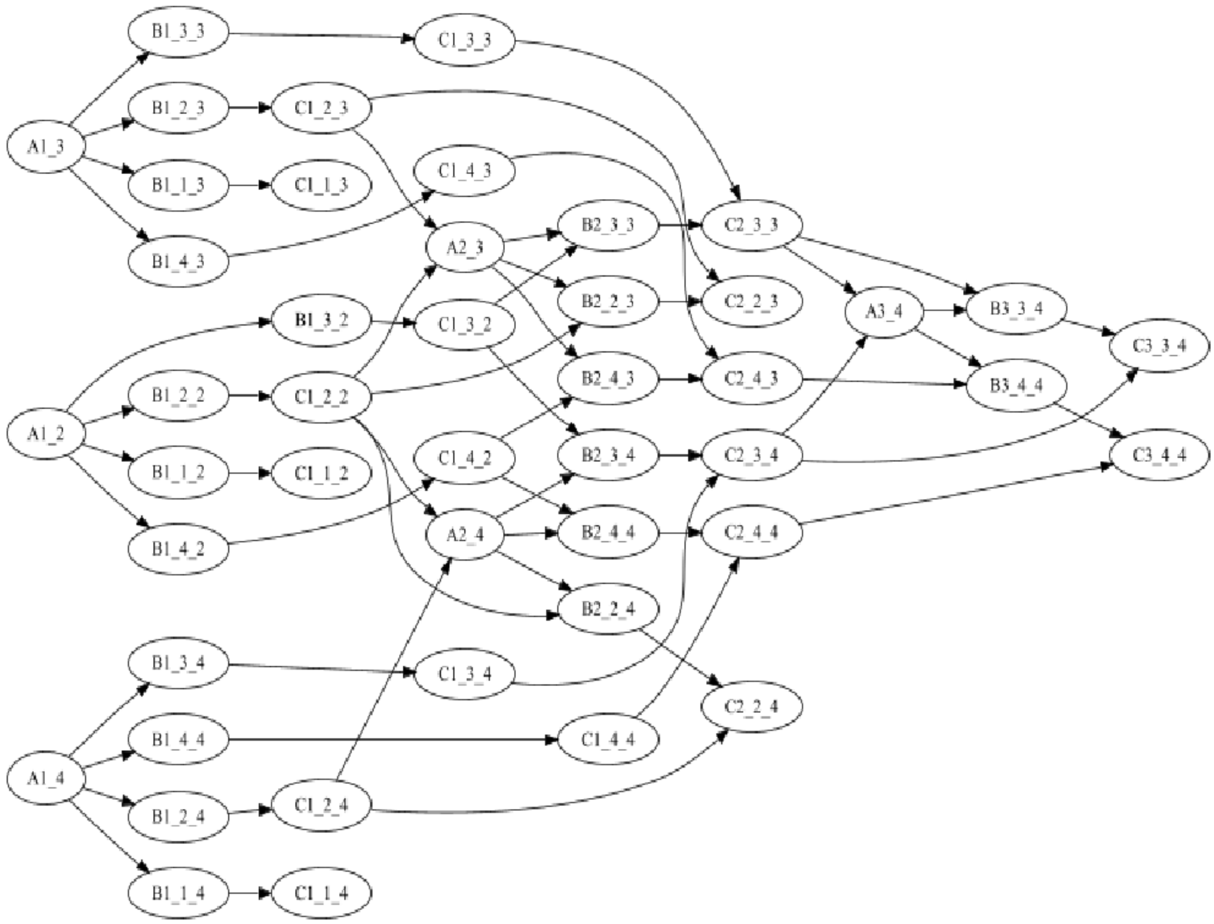
Macierz po równoległej eliminacji Gaussa :

[2.0, 1.0, 3.0, 3.0]

[0.0, 1.0, 2.0, 2.0]

[0.0, 0.0, 3.0, 3.0]

[0.0, 0.0, 0.0, 0.0]




Jak widać wszystko się zgadza i program rozwiązuje zadanie poprawnie.