Teoria współbieżności

Laboratorium 8

Natalia Bratek
31.12.2024

1. Podstawowe niepodzielnie zadania obliczeniowe

Mamy układ równań.
n – rozmiar macierzy

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

Można przekształcić do macierzy uzupełnionej, poprzez dołączenie wektora wyrazów wolnych do macierzy.

$$\begin{bmatrix} M_{1,1} & \cdots & M_{1,n} & | & M_{1,(n+1)} \\ \vdots & \ddots & \vdots & | & \vdots \\ M_{n,1} & \cdots & M_{n,n} & | & M_{n,(n+1)} \end{bmatrix}$$

$M_{n,(n+1)} = y_n$

Można wyróżnić trzy typy niepodzielnych zadań:

- $A_{k,i}$ - znalezienie mnożnika dla wiersza i, do odejmowania go od k-tego wiersza

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

- $B_{k,i,j}$ – pomnożenie j-tego elementu wiersza i przez mnożnik – do odejmowania od k-tego wiersza

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

- $C_{k,i,j}$ – odjęcie j-tego elementu wiersza i od wiersza k

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

2. Identyfikacja ciągu zadań obliczeniowych wykonywanych przez algorytm sekwencyjny

Mając macierz przekształconą do macierzy uzupełnionej można wykonywać na niej operacje.

1) Użycie pierwszego wiersza do „wyprodukowania” zer w pierwszej kolumnie.

Drugi wiersz = drugi wiersz - mnożnik * pierwszy wiersz

trzeci wiersz = trzeci wiersz - mnożnik * pierwszy wiersz

N wiersz = N wiersz - mnożnik * pierwszy wiersz

2) Analogicznie dla kolejnych kolumn

Algorytm sekwencyjny można zapisać jako:

$t_{k,i}$ – wyzerowanie odpowiedniego elementu z macierzy poprzez odjęcie od k-tego wiersza i-tego wiersza i pomnożenie go przez mnożnik

$$t_{k,i} = A_{k,i}, B_{k,i,i}, C_{k,i,i}, B_{k,i,i+1}, C_{k,i,i+1}, \dots, B_{k,i,n+1}, C_{k,i,N+1}$$

Następnie można zapisać:

$t_{2,1}, t_{3,1}, \dots, t_{n,1}, t_{3,2}, \dots, t_{n-1,n-2}, t_{n,n-2}, t_{n,n-1}$

$t_{2,1}, t_{3,1}, \dots, t_{n,1}$ - zeruje pierwszą kolumnę

3. Alfabet w sensie teorii śladów

$$\Sigma = \{A_{k,i}, B_{k,i,j}, C_{k,i,j} : \mathrm{i} \in [1, \mathrm{n}), \mathrm{k} \in (\mathrm{i}, \mathrm{n}], \mathrm{j} \in [\mathrm{i}, \mathrm{n}+1]\}$$

4. Relacje zależności i niezależności

Aby wyznaczyć relacje zależności szukam zmiennych zapisywanych w jednej operacji i jednocześnie są zapisywane w innej.
Można zauważyć, że mnożenie elementu jest zależne od mnożnika, ponieważ aby przemnożyć element przez mnożnik, najpierw trzeba obliczyć mnożnik.

$A_{k,i}$: $\boxed{m_{k,i}} = \frac{M_{k,i}}{M_{i,i}}$

$B_{k,i,j}$: $n_{k,i} = M_{i,j} * \boxed{m_{k,i}}$

Czyli $A_{k,i}$ i $B_{k,i,j}$ są zależne.

Odejmowanie jest także zależne od mnożenia dla zgodnych i, j, k.

$B_{k,i,j}$: $\boxed{n_{k,i}} = M_{i,j} * m_{k,i}$

$C_{k,i,j}$: $M_{k,j} = M_{k,j} - \boxed{n_{k,i}}$

$B_{k,i,j}\ i\ C_{k,i,j}$ są zależne.

Odejmowanie elementów wiersza jest uzależnione od wcześniejszego pomnożenia tych elementów przez odpowiedni mnożnik, wtw. gdy i dla mnożenia jest równe k dla odejmowania.

$B_{kb,ib,j}$: $n_{kb,ib} = \boxed{M_{ib,j}} * m_{kb,ib}$

$C_{kc,ic,j}$: $\boxed{M_{kc,j}} = M_{kc,j} - n_{kc,ic}$

Zanim $m_{ka,ia}$ zostanie wyznaczony, muszą być spełnione określone warunki: j-ty element wiersza ic powinien być odjęty od wiersza ka po przeprowadzeniu wszystkich operacji wpływających na wiersze kc oraz ka. Dla (ia = kc v ka = kc) ^ ia = j ^ ic = ia

$A_{ka,ia}$: $m_{ka,ia} = \boxed{\frac{M_{ka,ia}}{M_{ia,ia}}}$

$C_{kc,ic,j}$: $\boxed{M_{kc,j}} = M_{kc,j} - n_{kc,ic}$

Odejmowanie wierszy dla różnych i od wiersza k jest zależne.

$C_{k,i1,j}$: $\boxed{M_{k,j}} = M_{k,j} - n_{k,i1}$

$C_{k,i2,j}$: $M_{k,j} = \boxed{M_{k,j}} - n_{k,i2}$

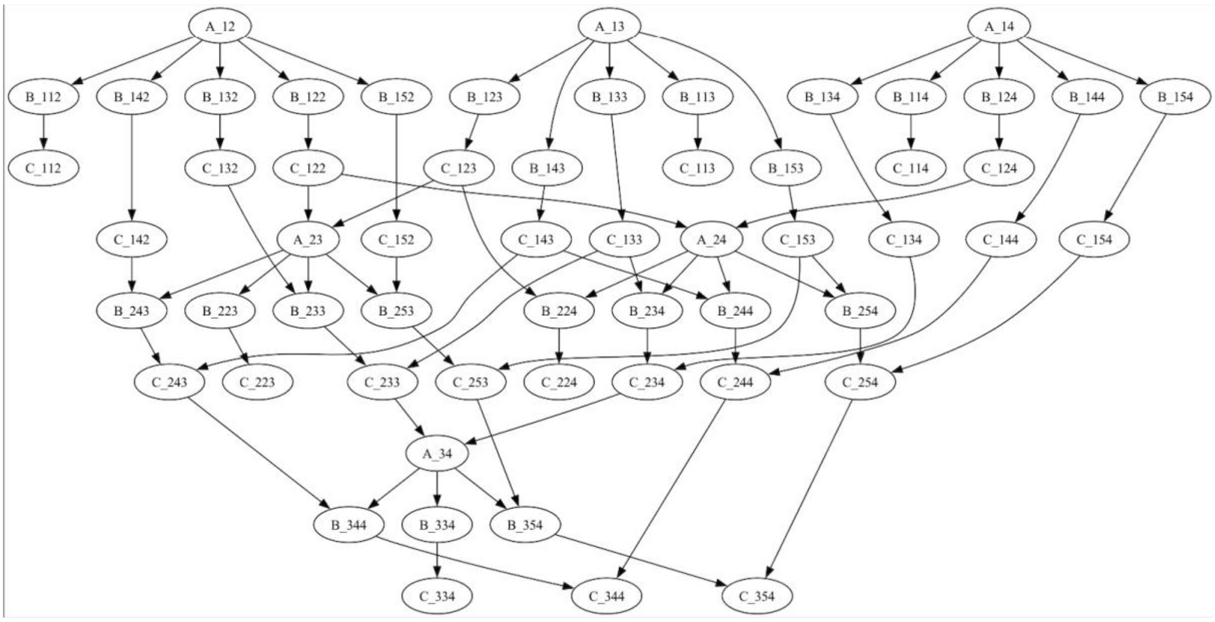
D = sym{$(A_{k,i}, B_{k,i,j}), (B_{k,i,j}, C_{k,i,j}), (C_{k,i1,j}, C_{k,i2,j}),$
$(C_{kc,ic,j}, B_{kb,ib,j}) : ib = kc,$
$(C_{kc,ic,j}, A_{ka,ia}) : (ia = kc \ \vee \ ka = kc) \wedge j = ia\} \cup I_\Sigma$

Relacja niezależności:

I = $\Sigma^2 - D$

5. Graf zależności Diekerta

Aby wyznaczyć graf Diekerta można wykorzystać relacje zależności oraz zbiór niepodzielnych zadań obliczeniowych.
W grafie Diekerta, wierzchołki V reprezentują niepodzielne zadania obliczeniowe Krawędzie grafu E reprezentują relacje zależności między zadaniami. W grafie Diekerta te zależności są bezprzechodnie, co oznacza, że każda krawędź bezpośrednio łączy zadania, które mają bezpośredni wpływ na siebie. Krawędzie grafu skierowane są zgodnie ze schematem działania algorytmu.



Rys. 1 Graf Diekerta dla n=4

6. Klasy Foata

Można zauważyć, że klasy Foata można podzielić na 3 rodzaje:

1) Operacje znalezienia mnożników (F_{A_m})
   - Znalezienie wszystkich niezbędnych mnożników do wyzerowania elementów w kolumnie m poniżej głównego elementu diagonalnego.

2) Operacje Przemnażania przez Znalezione Mnożniki (F_{B_m})
   - Przemnożenie elementów wiersza, który ma być odjęty, przez odpowiedni mnożnik, aby przygotować wiersz do odjęcia od innych wierszy w macierzy.

3) Operacje Odejmowania Wierszy (F_{C_m})
   - Odejmowanie przemnożonego wiersza od kolejnych wierszy w kolumnie m w celu wyzerowania elementów poniżej przekątnej.

$F_{A_m} = \{A_{k,m} : k \in (m, n]\}$
$F_{B_m} = \{B_{k,m,j} : k \in (m, n], \quad j \in (m, n+1]$
$F_{C_m} = \{C_{k,m,j} : k \in (m, n], \quad j \in (m, n+1]$

Dla m = {1, 2, 3, …. , n-1}

Postać normalna Foaty:
FNF = $[A_1][B_1][C_1][A_2][B_2][C_2]\ldots[A_{n-1}][B_{n-1}][C_{n-1}]$
Czyli:
$[F_{A_m}][F_{B_m}][F_{C_m}]$

7. Część implementacyjna

W ramach realizacji zadania zastosowałam język Python, korzystając z modułu concurrent.futures

7.1 Wyznaczanie alfabetu, relacji zależności, postaci normalnej Foaty

```python
class FoataNormalForm:
    def __init__(self, n):
        self.n = n
        self.alphabet, self.word = self.construct_alphabet()
        self.dependency_relations = self.construct_dependency_relation()
        self.graph = Graph(self.alphabet, self.dependency_relations)
        self.fnf = self.foata_normal_form()

    def task_a(self, i, k):
        return Task("A", i, None, k)

    def task_b_and_c(self, i, k, j):
        B = Task("B", i, j, k)
        C = Task("C", i, j, k)
        return B, C

    def construct_alphabet(self):
        word, alphabet = [], []
        for i in range(1, self.n + 1):
            for k in range(i + 1, self.n + 1):
                A = self.task_a(i, k)
                alphabet.append(A)
                word.append(A)
                for j in range(i, self.n + 2):
                    B, C = self.task_b_and_c(i, k, j)
                    alphabet.extend([B, C])
                    word.extend([B, C])
        return alphabet, word

    def construct_dependency_relation(self):
        dependency_rel = []
        for s in self.alphabet:
            for t in self.alphabet:
                if s.task_name == "A" and t.task_name == "C" and s.i - 1 == t.i and s.i == t.j and (s.i == t.k or s.k == t.k):
                    dependency_rel.append((s, t))
                elif s.task_name == "B" and t.task_name == "A" and s.i == t.i and s.k == t.k:
                    dependency_rel.append((s, t))
                elif s.task_name == "B" and t.task_name == "C" and s.i - 1 == t.i and s.j == t.j and s.k - 1 == t.k:
                    dependency_rel.append((s, t))
                elif s.task_name == "C" and t.task_name == "B" and s.i == t.i and s.j == t.j and s.k == t.k:
                    dependency_rel.append((s, t))
                elif s.task_name == "C" and t.task_name == "C" and s.i - 1 == t.i and s.j == t.j and s.k == t.k:
                    dependency_rel.append((s, t))
        return dependency_rel

    def initialize_classes(self, graph):
        all_targets = set()
        for edges in graph:
            all_targets.update(edges)
        num_vertices = len(graph)
        start_points = {i for i in range(num_vertices) if i not in all_targets}
        classes = [-1] * num_vertices
        for point in start_points:
            classes[point] = 0
        return classes

    def build_foata_normal_form(self, classes):
        graph = self.graph.graph_adj_list
        q = Queue()
        for vertex, cls in enumerate(classes):
            if cls == 0:
                q.put(vertex)
        while not q.empty():
            tmp = q.get()
            for neighbor in graph[tmp]:
                if classes[neighbor] == -1:
                    classes[neighbor] = classes[tmp] + 1
                    q.put(neighbor)
        max_class = max(classes)
        fnf_classes = [[] for _ in range(max_class + 1)]
        for vertex, cls in enumerate(classes):
            if cls != -1:
                fnf_classes[cls].append(self.alphabet[vertex])
        return fnf_classes

    def foata_normal_form(self):
        classes = self.initialize_classes(self.graph.graph_adj_list)
        return self.build_foata_normal_form(classes)
```

Rys.2 FNF

7.2 Wyznaczanie grafu Diekerta

```python
2 usages
class Graph:
    def __init__(self, alphabet, dependencies):
        self.alphabet = alphabet
        self.dependency_relations = dependencies
        self.graph = graphviz.Digraph()
        self.graph_adj_list = self.build_graph()
        self.reduced_graph = self.remove_edges()

    1 usage
    def build_graph(self):
        dependency_index = {}
        for dep in self.dependency_relations:
            if dep[1] not in dependency_index:
                dependency_index[dep[1]] = []
            dependency_index[dep[1]].append(dep[0])
        graph = []
        for letter in self.alphabet:
            if letter in dependency_index:
                index_set = {self.alphabet.index(dep) for dep in dependency_index[letter]}
            else:
                index_set = set()
            graph.append(index_set)
        return graph

    1 usage
    def remove_edges(self):
        total_vertices = len(self.graph_adj_list)
        for source in range(total_vertices):
            direct_neighbors = self.graph_adj_list[source].copy()
            for neighbor in direct_neighbors:
                self.graph_adj_list[source].remove(neighbor)
                if not self.bfs(source, neighbor):
                    self.graph_adj_list[source].add(neighbor)
        return self.graph_adj_list

    1 usage
    def bfs(self, start, end):
        visited = [False] * len(self.graph_adj_list)
        queue = [start]
        visited[start] = True
        while queue:
            current = queue.pop(0)
            if current == end:
                return True
            for adjacent in self.graph_adj_list[current]:
                if not visited[adjacent]:
                    visited[adjacent] = True
                    queue.append(adjacent)
        return False

    1 usage (1 dynamic)
    def draw_graph(self, save_path=None):
        [self.graph.node(str(i), elem.idx_task_name) for i, elem in enumerate(self.alphabet)]
        [self.graph.edge(str(i), str(j)) for i, neighbors in enumerate(self.reduced_graph) for j in neighbors]
        if save_path is None:
            save_path = 'output_graph'
        self.graph.render(save_path, format="png", view=True)
```

Rys.3 Graf Diekerta

7.3 Klasa Task do zarządzania zadaniami

```python
class Task:
    def __init__(self, task_name, i, j=None, k=None):
        self.task_name = task_name
        self.i = i
        self.j = j
        self.k = k
        self.idx_task_name = self.print_task_name()

    def print_task_name(self):
        if self.task_name == "A":
            return f"{self.task_name}_{self.i}{self.k}"
        elif self.task_name in ["B", "C"]:
            return f"{self.task_name}_{self.i}{self.j}{self.k}"

    def __repr__(self):
        return self.idx_task_name
```

Rys.4 Task

7.4 Implementacja Schedulera

```python
from concurrent.futures import ThreadPoolExecutor


class Scheduler:
    def __init__(self, max_workers=8):
        self.tasks = []
        self.max_workers = max_workers

    def add_task(self, task, *args):
        self.tasks.append((task, args))

    def run(self, auto_clear=True):
        results = []
        with ThreadPoolExecutor(max_workers=self.max_workers) as executor:
            futures = [executor.submit(task[0], *task[1]) for task in self.tasks]
            for future in futures:
                try:
                    result = future.result()
                    results.append(result)
                except Exception as e:
                    print(f"Error executing task: {e}")
                    results.append(None)
        if auto_clear:
            self.tasks = []
        return results
```

Rys.5 Scheduler

7.5 Implementacja eliminacji Gaussa

```python
class GaussianElimination:
    def __init__(self, matrix, n):
        self.matrix = matrix
        self.n = n
        self.m = np.empty((self.n, self.n))
        self.t = np.empty((self.n, self.n + 1, self.n))

    def task(fun):
        def wrapper(self, *args, **kwargs):
            print(f"Running task: {fun.__name__} with args {args}, {kwargs}")
            return fun
        return wrapper

    @task
    def task_A(self, i, k):
        self.m[i, k] = self.matrix[k, i] / self.matrix[i, i]

    @task
    def task_B(self, i, j, k):
        self.t[i, j, k] = self.matrix[i, j] * self.m[i, k]

    @task
    def task_C(self, i, j, k):
        self.matrix[k, j] -= self.t[i, j, k]

    def pivot(self, i):
        max_row = np.argmax(abs(self.matrix[i:, i])) + i
        if i != max_row:
            self.matrix[[i, max_row]] = self.matrix[[max_row, i]]
        if self.matrix[i, i] == 0:
            raise ValueError("Matrix is singular and cannot be solved")

    def resolve_backwards(self):
        for k in range(self.n - 1, 0, -1):
            if self.matrix[k, k] == 0:
                raise ValueError("Division by zero encountered in backward substitution")
            for i in range(k):
                self.reduce_row(k, i)
        return self.matrix

    def reduce_row(self, i, k):
        factor = self.matrix[k, i] / self.matrix[i, i]
        self.matrix[k, i:] -= factor * self.matrix[i, i:]
        return self.matrix

    def run(self):
        scheduler = Scheduler(self.n)
        for i in range(self.n):
            for k in range(i + 1, self.n):
                scheduler.add_task(self.reduce_row, i, k)
            scheduler.run()
        self.matrix /= self.matrix[np.arange(self.n), np.arange(self.n)][:, np.newaxis]
        self.matrix = self.resolve_backwards()
        return self.matrix
```

Rys. 6 Eliminacja Gaussa

7.6 Kod Uruchomieniowy

```python
1 usage
def load_matrix(filepath="in.txt"):
    try:
        with open(filepath, 'r') as file:
            data = file.readlines()
        if not data:
            raise ValueError("The file is empty.")
        size = int(data[0].strip())
        matrix = [list(map(float, line.split())) for line in data[1:size + 1]]
        additional_column = list(map(float, data[size + 1].strip().split()))
        for index, value in enumerate(additional_column):
            matrix[index].append(value)
    except FileNotFoundError:
        raise FileNotFoundError(f"File not found")
    except ValueError as e:
        raise ValueError(f"Error processing data: {e}")
    except Exception as e:
        raise Exception(f"Unexpected error: {e}")
    numpy_matrix = np.array(matrix)
    return numpy_matrix, size


1 usage
def write_matrix_to_file(matrix, size, file_path="result.txt"):
    with open(file_path, 'w') as output_file:
        output_file.write(f"{size}\n")
        for row in matrix:
            row_data = ' '.join(map(str, row[:-1])) + '\n'
            output_file.write(row_data)
        last_column_data = ' '.join(map(str, matrix[:, -1])) + '\n'
        output_file.write(last_column_data)


1 usage
def run_gaussian_elimination(filepath_source="in/in.txt", filepath_result="result/result.txt"):
    try:
        matrix, n = load_matrix(filepath_source)
        gauss = GaussianElimination(matrix, n)
        matrix = gauss.run()
        write_matrix_to_file(matrix, n, filepath_result)
        print(f"Results saved to file: {filepath_result}")
        return True
    except Exception as e:
        print(f"An error during testing: {e}")
        return False


1 usage
def main():
    n = 3
    fnf = FoataNormalForm(n)
    run_gaussian_elimination()
    wrapper = textwrap.TextWrapper(width=150)
    print("Alphabet and Dependency Relations")
    alphabet_output = "Alphabet = {" + ", ".join(sorted(i.idx_task_name for i in fnf.alphabet)) + "}"
    print(wrapper.fill(alphabet_output))
    dependency_output = "Dependency Relations = (" + ", ".join(f"({a.idx_task_name}, {b.idx_task_name})" for a, b in fnf.dependency_relations) + ")"
    print(wrapper.fill(dependency_output))
    print("Foata Normal Form (FNF)")
    fnf_output = "FNF = (" + ")(".join(", ".join(i.idx_task_name for i in sublist) for sublist in fnf.fnf) + ")"
    print(wrapper.fill(fnf_output))
    fnf.graph.draw_graph(save_path=f"graphs/graph_{fnf.n}")


if __name__ == "__main__":
    main()
```

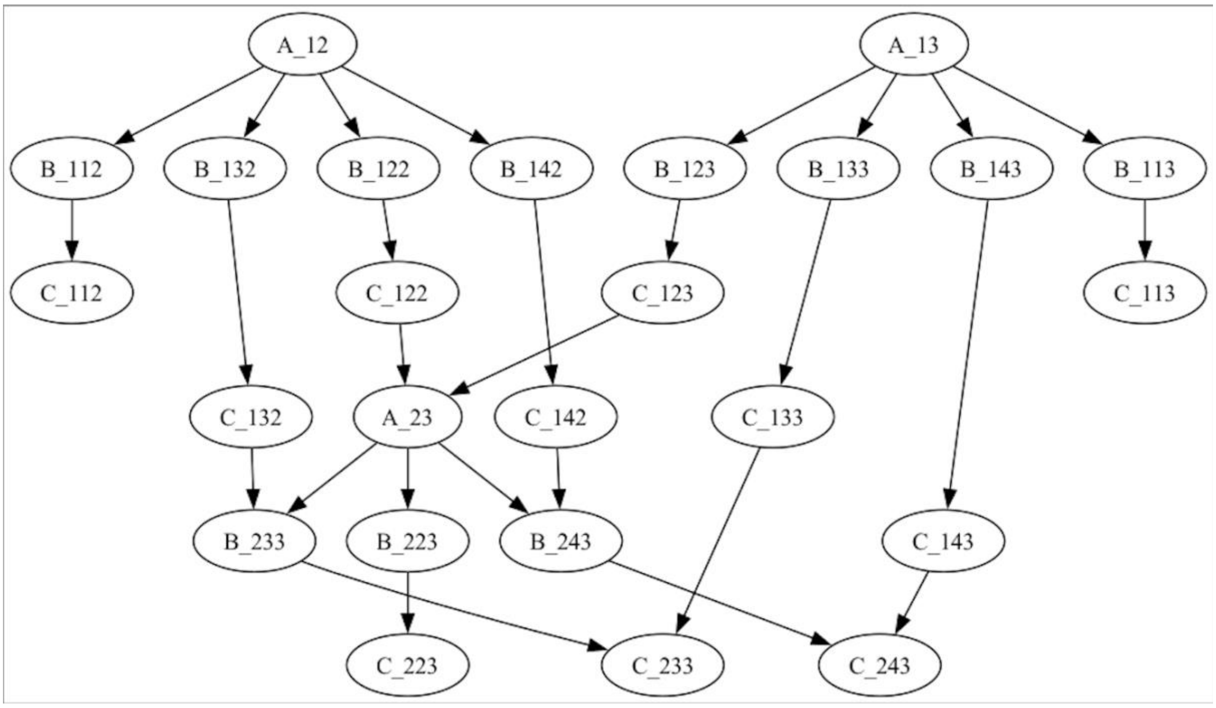
Rys. 7 Main

W folderze in znajdują się dane wejściowe. A po uruchomieniu, wyniki z eliminacji Gaussa zapisują się w folderze result jako result.txt
Po uruchomieniu wygenerują się też graf Diekerta oraz w terminalu wyświetli się:

```
/Users/nataliabratek/Desktop/zad3/.venv/bin/python /Users/nataliabratek/Desktop/zad3/main.py
Results saved to file: result/result.txt
Alphabet and Dependency Relations
Alphabet = {A_12, A_13, A_23, B_112, B_113, B_122, B_123, B_132, B_133, B_142, B_143, B_223, B_233, B_243, C_112, C_113, C_122, C_123, C_132, C_133,
C_142, C_143, C_223, C_233, C_243}
Dependency Relations = ((B_112, A_12), (C_112, B_112), (B_122, A_12), (C_122, B_122), (B_132, A_12), (C_132, B_132), (B_142, A_12), (C_142, B_142),
(B_113, A_13), (C_113, B_113), (B_123, A_13), (C_123, B_123), (B_133, A_13), (C_133, B_133), (B_143, A_13), (C_143, B_143), (A_23, C_122), (A_23,
C_123), (B_223, C_122), (B_223, A_23), (C_223, C_123), (C_223, B_223), (B_233, C_132), (B_233, A_23), (C_233, C_133), (C_233, B_233), (B_243, C_142),
(B_243, A_23), (C_243, C_143), (C_243, B_243))
Foata Normal Form (FNF)
FNF = (A_12, A_13)(B_112, B_122, B_132, B_142, B_113, B_123, B_133, B_143)(C_112, C_122, C_132, C_142, C_113, C_123, C_133, C_143)(A_23, B_233, C_233,
B_243, C_243)(B_223)(C_223)

Process finished with exit code 0
```

Rys. 8 Terminal



Rys.9 Graf Diekerta

8. Sprawdzarka

Aby zweryfikować poprawność implementacji eliminacji Gaussa, przeprowadziłam testy dla różnych rozmiarów macierzy.

Dla n = 3

```
/Users/nataliabratek/Library/Java/JavaVirtualMachines/corr
2
in.txt
result.txt
0.9999999999999994 0.9999999999999987 1.0000000000000007
1.0 1.0 1.0

Process finished with exit code 0
```

Rys.10 Checker dla n = 3

Dla n = 4

```
/Users/nataliabratek/Library/Java/JavaVirtualMachines/corretto-17.0.13/Cont
2
in4.txt
result4.txt
-77123.55496697404 4641.298731448853 39116.072988913715 29394.9823056558
-77123.55496697406 4641.298731448849 39116.072988913715 29394.9823056558

Process finished with exit code 0
```

Rys.11 Checker dla n = 4

n = 5

```
/Users/nataliabratek/Library/Java/JavaVirtualMachines/corretto-17.0.13/Contents/Home/bin/java -ja
2
in5.txt
result5.txt
-36.90925412509657 46.3891717791548 150.82530852532517 -187.8991875917358 0.6220711667243818
-36.9092541250949 46.389171779152726 150.8253085253183 -187.89918759172724 0.6220711667243818

Process finished with exit code 0
```

Rys.12 Checker dla n = 5

n = 10

```
/Users/nataliabratek/Library/Java/JavaVirtualMachines/corretto-17.0.13/Contents/Home/bin/java -javaagent:/Users/nataliabratek/Desktop/IntelliJ IDEA.app/Contents/li
2
in10.txt
result10.txt
0.460322336412967 0.4476808112784328 0.12472902951166166 0.7592894296742821 0.16109055688063728 0.2996712316065755 0.15190404634971288 -0.4659332943658147 -0.13220
0.46032233629375696 0.4476808109388959 0.1247290296423102 0.7592894295545325 0.16109055713605747 0.29967123153367936 0.1519040461848531 -0.46593329432033503 -0.132

Process finished with exit code 0
```

Rys.13 Checker dla n = 10

n = 20

```
/Users/nataliabratek/Library/Java/JavaVirtualMachines/corretto-17.0.13/Contents/Home/bin/java -javaagent:/Users/nataliabratek/Desktop/IntelliJ IDEA.app/Contents/li
2
in20.txt
result20.txt
-9.312571096033395 5.908379284787953 -4.390179543063113 7.416999458864289 1.430269195033032 2.9570176490731526 -0.8275129643763818 -0.38594783437484337 -0.4803000
-9.31257109600551 5.908379284669309 -4.390179543147461 7.416999459033197 1.4302691949847313 2.9570176491025975 -0.827512964431 -0.3859478342238456 -0.480300052166
```

Rys.14 Checker dla n = 20

Testy potwierdziły, że wyniki eliminacji Gaussa są zgodne z oczekiwaniami dla macierzy o różnych rozmiarach.