МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Институт компьютерных наук
Кафедра автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №3

По предмету: “СИСТЕМЫ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА”

Студент АС-21-1 _____ Станиславчук С.М.
(подпись, дата) Коновалов К.А.

Руководитель
Профессор _____ Сараев П.В.
(подпись, дата)

Липецк, 2024 г.

Задание кафедры.

1. Реализуйте программное обеспечение для обработки нечетких систем логического вывода с одной и несколькими входными переменными. База правил и нечеткие множества должны храниться в отдельном текстовом файле. В программном обеспечении должна быть предусмотрена возможность использования различных импликаций, max-min композиции, различных операторов агрегации.

2. Придумайте содержательную задачу (или составьте на основе предметной области из лаб.раб. № 1,2), содержащую 1 входную и 1 выходную переменные. Постройте нечеткую систему логического вывода, состоящую не менее чем из 7 правил. Найдите выход системы на основе трех механизмов логического вывода (2 композиционных механизма, 1 механизм с использованием уровней истинности предпосылок правил). При необходимости используйте дефазификацию на основе метода центра тяжести.

3. Модифицируйте задачу с тем, чтобы она содержала несколько входных (не менее 3) и 1 выходную переменные. Постройте нечеткую систему логического вывода, состоящую не менее чем из 7 правил. Найдите выход системы на основе использования уровней истинности предпосылок правил. При необходимости используйте дефазификацию на основе метода центра тяжести.

4. Сравните результаты, полученные с использованием разных типов импликаций.

Конфиг файл 1 (БЗ):

Множество определения золото_в_минуту
300 450 600 750 900
Нечеткое множество нулевое
1 0.1 0.02 0 0
Нечеткое множество мало
0.2 0.9 0.1 0.05 0
Нечеткое множество немного
0.1 0.6 0.5 0.3 0.1
Нечеткое множество средне
0.05 0.2 1 0.1 0.02
Нечеткое множество много
0.02 0.1 0.3 0.8 0.1
Нечеткое множество огромное
0 0.05 0.1 0.6 0.6
Нечеткое множество гигантское
0 0.02 0.05 0.2 1

Множество определения навык_игры
0 2000 4000 6000 8000 10000
Нечеткое множество новичок
0 0 0.1 0.9 0.1 0
Нечеткое множество любитель
0.4 0.9 0.1 0 0 0
Нечеткое множество опытный
0 0.1 0.9 0.1 0 0
Нечеткое множество бывалый
0.9 0.2 0.1 0 0 0
Нечеткое множество эксперт
0 0 0.1 0.25 0.9 0.1
Нечеткое множество профессионал
0 0 0 0 0.2 0.9

Если золото_в_минуту нулевое то навык_игры новичок
Если золото_в_минуту мало то навык_игры новичок
Если золото_в_минуту немного то навык_игры любитель
Если золото_в_минуту средне то навык_игры опытный
Если золото_в_минуту много то навык_игры бывалый
Если золото_в_минуту огромное то навык_игры эксперт
Если золото_в_минуту гигантское то навык_игры профессионал

Пусть золото_в_минуту
0.2 0.3 0.6 0.75 1

Конфиг файл 2 (БЗ):

Множество определения обслуживание
0 3.3 6.6 10
Нечеткое множество грубое
1 0.8 0.3 0
Нечеткое множество среднее
0 0.5 0.9 0.3
Нечеткое множество идеальное
0 0 0.4 0.9

Множество определения еда
0 3.3 6.6 10
Нечеткое множество ужасное
1 0.7 0.3 0
Нечеткое множество среднее
0 0.6 0.9 0.4
Нечеткое множество превосходное
0 0 0.2 0.8

Множество определения ожидание
0 20 40 60
Нечеткое множество короткое
0.9 0.5 0 0
Нечеткое множество среднее
0.2 0.9 0.5 0
Нечеткое множество долгое
0 0.1 0.8 1

Множество определения оценка
0 3.3 6.6 10
Нечеткое множество очень плохо
1 0.7 0.3 0

Нечеткое множество удовлетворительно
0 0.6 0.9 0.3
Нечеткое множество идеально
0 0 0.4 0.9

Если обслуживание грубое и еда ужасное и ожидание долгое то оценка очень плохо
Если обслуживание грубое и еда среднее и ожидание среднее то оценка очень плохо
Если обслуживание среднее и еда среднее и ожидание среднее то оценка удовлетворительно
Если обслуживание среднее и еда превосходное и ожидание короткое то оценка удовлетворительно
Если обслуживание идеальное и еда превосходное и ожидание короткое то оценка идеально
Если обслуживание идеальное и еда среднее и ожидание короткое то оценка идеально
Если обслуживание идеальное и еда превосходное и ожидание среднее то оценка идеально

Пусть обслуживание
0.3 0.8 0.6 0.1
Пусть еда
0.4 0.7 0.9 0.2
Пусть ожидание
0.2 0.8 0.5 0.1

Программная реализация
Программа 1.

```
import os
import sys

import pandas as pd

# Import script with plots drawings
sys.path.append(os.path.dirname(os.path.abspath(__file__)))
import plt.plotting


def process_file(filename):
    # Матрицы для хранения данных
    A = {}
    B = {}
    rules = []
    given = []

    with open(filename, "r", encoding="utf-8") as file:
        lines = file.readlines()

    current_set_name = None
    current_set_values = []
    current_set_length = 0
    A_set = False
    a_name = ""
    b_name = ""

    i = 0
    while i < len(lines):
        line = lines[i].strip()

        if not line:
            i += 1
            continue

        elif line.startswith("Множество определения"):
            # Завершаем предыдущее множество, если оно есть
            if current_set_name:
                if A_set:
                    A[current_set_name] = current_set_values + [0] * (
                        current_set_length - len(current_set_values)
                    )
                else:
                    B[current_set_name] = current_set_values + [0] * (
                        current_set_length - len(current_set_values)
                    )

            A_set = not A_set

            # Начинаем обработку нового множества
            parts = line.split()
            current_set_name = parts[-1]
            if A_set:
                a_name = current_set_name
            else:
                b_name = current_set_name
            current_set_values = []
            current_set_length = 0

        elif line.startswith("0") or line.startswith("-") or line[0].isdigit():
            # Считываем числовые значения множества
            values = list(map(float, line.split()))
            if current_set_length == 0:
                current_set_length = len(values)
            current_set_values.extend(values)

        elif line.startswith("Нечеткое множество"):
            # Считываем название нечеткого множества
            parts = line.split()
            fuzzy_set_name = parts[-1]
            i += 1  # Переходим к следующей строке, содержащей значения
```

```
            if i < len(lines):
                fuzzy_set_values = list(map(float, lines[i].strip().split()))
                fuzzy_set_values.extend(
                    [0] * (current_set_length - len(fuzzy_set_values))
                )
                # Добавляем в матрицу множеств
                if A_set:
                    A[fuzzy_set_name] = fuzzy_set_values
                else:
                    B[fuzzy_set_name] = fuzzy_set_values

        elif line.startswith("Если"):
            # Обработка правила
            parts = line.split()
            antecedent = parts[2]  # Название нечеткого множества условия
            consequent = parts[-1]  # Название нечеткого множества результата
            rules.append((antecedent, consequent))

        elif line.startswith("Пусть"):
            # Обработка начального состояния
            i += 1
            line = lines[i].strip()
            values = list(map(float, line.split()))
            given.extend(values)

        i += 1

    # Обрабатываем последнее множество
    if current_set_name:
        if A_set:
            A[current_set_name] = current_set_values
        else:
            B[current_set_name] = current_set_values

    # Преобразуем матрицу множеств в DataFrame
    A = pd.DataFrame(A)
    B = pd.DataFrame(B)

    # Преобразуем правила в DataFrame
    rules = pd.DataFrame(rules, columns=["Условие", "Следствие"])

    return A, B, rules, given, a_name, b_name


def get_correspondences_Mamdani(A, B):
    print("===ИМПЛИКАЦИЯ МЕТОДОМ МАМДАНИ===")
    correspondences = []
    for r in range(len(rules["Условие"])):
        correspondence = []
        for i in range(len(A)):
            row = []
            for j in range(len(B)):
                row.append(min(A[rules["Условие"][r]][i], B[rules["Следствие"][r]][j]))
            correspondence.append(row)
        correspondences.append(correspondence)

    print("")
    for i, corr in enumerate(correspondences, start=1):
        print(f"Матрица зависимостей {i}")
        for row in corr:
            print([float(value) for value in row])
        print("")
    return correspondences


def get_correspondences_Larsen(A, B):
    print("===ИМПЛИКАЦИЯ МЕТОДОМ ЛАРСЕНА===")
    correspondences = []
    for r in range(len(rules["Условие"])):
        correspondence = []
        for i in range(len(A)):
            row = []
            for j in range(len(B)):
                row.append(
                    round(A[rules["Условие"][r]][i] * B[rules["Следствие"][r]][j], 2)
                )
            correspondence.append(row)
        correspondences.append(correspondence)
```

```
    print("")
    for i, corr in enumerate(correspondences, start=1):
        print(f"Матрица зависимостей {i}")
        for row in corr:
            print([float(value) for value in row])
        print("")
    return correspondences


def outputs_aggregation(correspondences, given):
    print("===ВЫЧИСЛЕНИЕ МЕТОДОМ АГРЕГАЦИИ ВЫХОДОВ===\n")
    outputs = []

    for num, correspondence in enumerate(correspondences, start=1):
        output = []
        for i in range(len(correspondence[0])):
            column = [min(given[j], correspondence[j][i]) for j in range(len(given))]
            output.append(max(column))

        print(f"Выход для правила {num}")
        print([float(value) for value in output])
        outputs.append(output)

    print("\nАггрегация выходов")
    aggregation = [max(output[i] for output in outputs) for i in range(len(outputs[0]))]
    print([float(value) for value in aggregation])
    return aggregation


def rules_aggregation(correspondences, given):
    print(f"===ВЫЧИСЛЕНИЕ МЕТОДОМ АГРЕГАЦИИ ПРАВИЛ===\n")
    aggregation = correspondences[0]
    for correspondence in correspondences:
        for i in range(len(A)):
            for j in range(len(B)):
                aggregation[i][j] = max(aggregation[i][j], correspondence[i][j])

    print(f"Агрегация правил")
    for i in range(len(aggregation)):
        print([float(value) for value in aggregation[i]])

    output = []
    for i in range(len(aggregation[0])):
        column = []
        for j in range(len(given)):
            column.append(min(given[j], aggregation[j][i]))
        output.append(max(column))

    print("")
    print(f"Значение выхода")
    print([float(value) for value in output])
    print("")
    return output


def defuzzification(output, values_a, a_name, given, values_b, b_name):
    sum_1 = 0
    sum_2 = 0
    for i in range(len(given)):
        sum_1 += given[i] * values_a[i]
        sum_2 += given[i]
    mid = sum_1 / sum_2
    print(f"ДЕФАЗАФИКАЦИЯ")
    print(f"Четкое значение входа {a_name}: {round(mid)}")
    sum_1 = 0
    sum_2 = 0
    for i in range(len(output)):
        sum_1 += output[i] * values_b[i]
        sum_2 += output[i]
    mid = sum_1 / sum_2
    print(f"Четкое значение выхода {b_name}: {round(mid)}")
    print("\n")


filename = "config_combined.txt"
A, B, rules, given, a_name, b_name = process_file(filename)
```

```
print("\nA:")
print(A)
print("\nB:")
print(B)
print("\nМатрица правил:")
print(rules)
print(f"\nПусть {a_name}:")
print(given)
print("")

input("Нажмите любую клавишу, чтобы посмотреть результат метода Мамдани...")

correspondences_M = get_correspondences_Mamdani(A, B)

input("Нажмите любую клавишу, чтобы посмотреть результат агрегации...")

output = outputs_aggregation(correspondences_M, given)
defuzzification(output, A[a_name], a_name, given, B[b_name], b_name)


output = rules_aggregation(correspondences_M, given)
defuzzification(output, A[a_name], a_name, given, B[b_name], b_name)

input("Нажмите любую клавишу, чтобы посмотреть результат метода Ларсена...")

correspondences_L = get_correspondences_Larsen(A, B)
output = outputs_aggregation(correspondences_L, given)
defuzzification(output, A[a_name], a_name, given, B[b_name], b_name)

input("Нажмите любую клавишу, чтобы посмотреть результат агрегации...")

output = rules_aggregation(correspondences_L, given)


input("Нажмите любую клавишу, чтобы посмотреть график...")
plt.plotting.plot_comparison_Q(correspondences_M, correspondences_L, A, B)
```

Программа 2.

```
import pandas as pd


def process_file(filename):
    # Матрицы для хранения данных
    A1 = {}
    A2 = {}
    A3 = {}
    B = {}
    rules = []
    given = []
    given1 = []

    with open(filename, "r", encoding="utf-8") as file:
        lines = file.readlines()

    current_set_name = None
    current_set_values = []
    current_set_length = 0
    set = 0
    given_set = 0
    a1_name = ""
    a2_name = ""
    a3_name = ""
    b_name = ""

    i = 0
    while i < len(lines):
        line = lines[i].strip()

        if not line:
            i += 1
            continue

        elif line.startswith("Множество определения"):
            # Завершаем предыдущее множество, если оно есть
            if current_set_name:
                if set == 1:
```

```
            A1[current_set_name] = current_set_values + [0] * (
                current_set_length - len(current_set_values)
            )
        elif set == 2:
            A2[current_set_name] = current_set_values + [0] * (
                current_set_length - len(current_set_values)
            )
        elif set == 3:
            A3[current_set_name] = current_set_values + [0] * (
                current_set_length - len(current_set_values)
            )
        elif set == 4:
            B[current_set_name] = current_set_values + [0] * (
                current_set_length - len(current_set_values)
            )

    set += 1

    # Начинаем обработку нового множества
    parts = line.split()
    current_set_name = parts[-1]
    if set == 1:
        a1_name = current_set_name
    elif set == 2:
        a2_name = current_set_name
    elif set == 3:
        a3_name = current_set_name
    elif set == 4:
        b_name = current_set_name
    current_set_values = []
    current_set_length = 0

elif line.startswith("0") or line.startswith("-") or line[0].isdigit():
    # Считываем числовые значения множества
    values = list(map(float, line.split()))
    if current_set_length == 0:
        current_set_length = len(values)
    current_set_values.extend(values)

elif line.startswith("Нечеткое множество"):
    # Считываем название нечеткого множества
    parts = line.split()
    fuzzy_set_name = parts[-1]
    i += 1  # Переходим к следующей строке, содержащей значения
    if i < len(lines):
        fuzzy_set_values = list(map(float, lines[i].strip().split()))
        fuzzy_set_values.extend(
            [0] * (current_set_length - len(fuzzy_set_values))
        )
        # Добавляем в матрицу множеств
        if set == 1:
            A1[fuzzy_set_name] = fuzzy_set_values
        elif set == 2:
            A2[fuzzy_set_name] = fuzzy_set_values
        elif set == 3:
            A3[fuzzy_set_name] = fuzzy_set_values
        elif set == 4:
            B[fuzzy_set_name] = fuzzy_set_values

elif line.startswith("Если"):
    # Обработка правила
    parts = line.split()
    antecedent1 = parts[2]
    antecedent2 = parts[5]
    antecedent3 = parts[8]
    consequent = parts[-1]
    rules.append((antecedent1, antecedent2, antecedent3, consequent))

elif line.startswith("Пусть"):
    # Обработка начального состояния
    i += 1
    line = lines[i].strip()
    values = list(map(float, line.split()))
    given1.extend(values)
    given.append(given1)
    given1 = []

i += 1
```

```
    # Обрабатываем последнее множество
    if current_set_name:
        if set == 1:
            A1[current_set_name] = current_set_values + [0] * (
                current_set_length - len(current_set_values)
            )
        elif set == 2:
            A2[current_set_name] = current_set_values + [0] * (
                current_set_length - len(current_set_values)
            )
        elif set == 3:
            A3[current_set_name] = current_set_values + [0] * (
                current_set_length - len(current_set_values)
            )
        elif set == 4:
            B[current_set_name] = current_set_values + [0] * (
                current_set_length - len(current_set_values)
            )

    # Преобразуем матрицу множеств в DataFrame
    A1 = pd.DataFrame(A1)
    A2 = pd.DataFrame(A2)
    A3 = pd.DataFrame(A3)
    B = pd.DataFrame(B)

    A = []
    A.append([])
    A.append(A1)
    A.append(A2)
    A.append(A3)

    name = []
    name.append(b_name)
    name.append(a1_name)
    name.append(a2_name)
    name.append(a3_name)

    # Преобразуем правила в DataFrame
    rules = pd.DataFrame(rules, columns=[a1_name, a2_name, a3_name, b_name])

    return A, B, rules, given, name


def get_outputs(B, rules, name, levels_of_truth):
    outputs = []
    for r in range(len(rules[name[0]])):
        output = []
        for i in range(len(B[rules[name[0]][r]])):
            output.append(min(B[rules[name[0]][r]][i], levels_of_truth[r]))

        print(f"Выход для правила {r+1}")
        print([float(value) for value in output])
        outputs.append(output)

    print("")
    return outputs


def outputs_aggregation(outputs):
    aggregation = []
    for i in range(len(outputs[0])):
        max_output = outputs[0][i]
        for output in outputs:
            max_output = max(max_output, output[i])
        aggregation.append(max_output)

    print("\nАггрегация выходов")
    aggregation = [max(output[i] for output in outputs) for i in range(len(outputs[0]))]
    print([float(value) for value in aggregation])
    print("")
    return aggregation


def defuzzification(A, B, name, given, aggregation):
    for j in range(3):
        sum_1 = 0
        sum_2 = 0
```

```
        for i in range(len(given)):
            sum_1 += given[j][i] * A[j + 1][name[j + 1]][i]
            sum_2 += given[j][i]
        mid = sum_1 / sum_2
        print(f"Четкое значение входа {name[j+1]}: {round(mid)}")

    sum_1 = 0
    sum_2 = 0
    for i in range(len(aggregation)):
        sum_1 += aggregation[i] * B[name[0]][i]
        sum_2 += aggregation[i]
    mid = sum_1 / sum_2
    print(f"Четкое значение выхода {name[0]}: {round(mid)}")
    print("\n")


def maxmin(a, b):
    row = []
    for i in range(len(a)):
        row.append(min(a[i], b[i]))
    output = max(row)
    return output


def levels_of_truth_of_premises(A, B, rules, given, name):
    levels_of_truth = []
    for i in range(len(rules[name[0]])):
        a1 = A[1][rules[name[1]][i]]
        a2 = A[2][rules[name[2]][i]]
        a3 = A[3][rules[name[3]][i]]

        level_a1 = maxmin(a1, given[0])
        level_a2 = maxmin(a2, given[1])
        level_a3 = maxmin(a3, given[2])

        level = min(level_a1, level_a2, level_a3)
        levels_of_truth.append(level)

    print("Уровни истинности предпосылок правил:")
    print([float(value) for value in levels_of_truth])
    print("")
    return levels_of_truth


filename = "config_many.txt"
A, B, rules, given, name = process_file(filename)

print("\nМатрица множеств A1:")
print(A[1])
print("\nМатрица множеств A2:")
print(A[2])
print("\nМатрица множеств A3:")
print(A[3])
print("\nМатрица множеств B:")
print(B)
print("\nМатрица правил:")
print(rules)
print("\nПусть:")
print(given)
print("")

levels_of_truth = levels_of_truth_of_premises(A, B, rules, given, name)
outputs = get_outputs(B, rules, name, levels_of_truth)
aggregation = outputs_aggregation(outputs)
defuzzification(A, B, name, given, aggregation)
```

```
stanik@archlinux /home/stanik/programmer/++Programmer/4_1/SII_SARAEV/lab/lr3
 > ./bin/python ./src/core_combined.py

A:
   нулевое  мало  немного  средне  много  огромное  гигантское  золото_в_минуту
0     1.00  0.20      0.1    0.05   0.02      0.00        0.00            300.0
1     0.10  0.90      0.6    0.20   0.10      0.05        0.02            450.0
2     0.02  0.10      0.5    1.00   0.30      0.10        0.05            600.0
3     0.00  0.05      0.3    0.10   0.80      0.60        0.20            750.0
4     0.00  0.00      0.1    0.02   0.10      0.60        1.00            900.0

B:
   новичок  любитель  опытный  бывалый  эксперт  профессионал  навык_игры
0      0.0       0.4      0.0      0.9     0.00           0.0         0.0
1      0.0       0.9      0.1      0.2     0.00           0.0      2000.0
2      0.1       0.1      0.9      0.1     0.10           0.0      4000.0
3      0.9       0.0      0.1      0.0     0.25           0.0      6000.0
4      0.1       0.0      0.0      0.0     0.90           0.2      8000.0
5      0.0       0.0      0.0      0.0     0.10           0.9     10000.0

Матрица правил:
      Условие     Следствие
0     нулевое       новичок
1        мало       новичок
2     немного      любитель
3      средне       опытный
4        много       бывалый
5    огромное       эксперт
6  гигантское  профессионал

Пусть золото_в_минуту:
[0.2, 0.3, 0.6, 0.75, 1.0]

Нажмите любую клавишу, чтобы посмотреть результат метода Мамдани...
===ИМПЛИКАЦИЯ МЕТОДОМ МАМДАНИ===

Матрица зависимостей 1
[0.0, 0.0, 0.1, 0.9, 0.1, 0.0]
[0.0, 0.0, 0.1, 0.1, 0.1, 0.0]
[0.0, 0.0, 0.02, 0.02, 0.02, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

Матрица зависимостей 2
```

Рисунок 1 — Пример выполнения программы 1

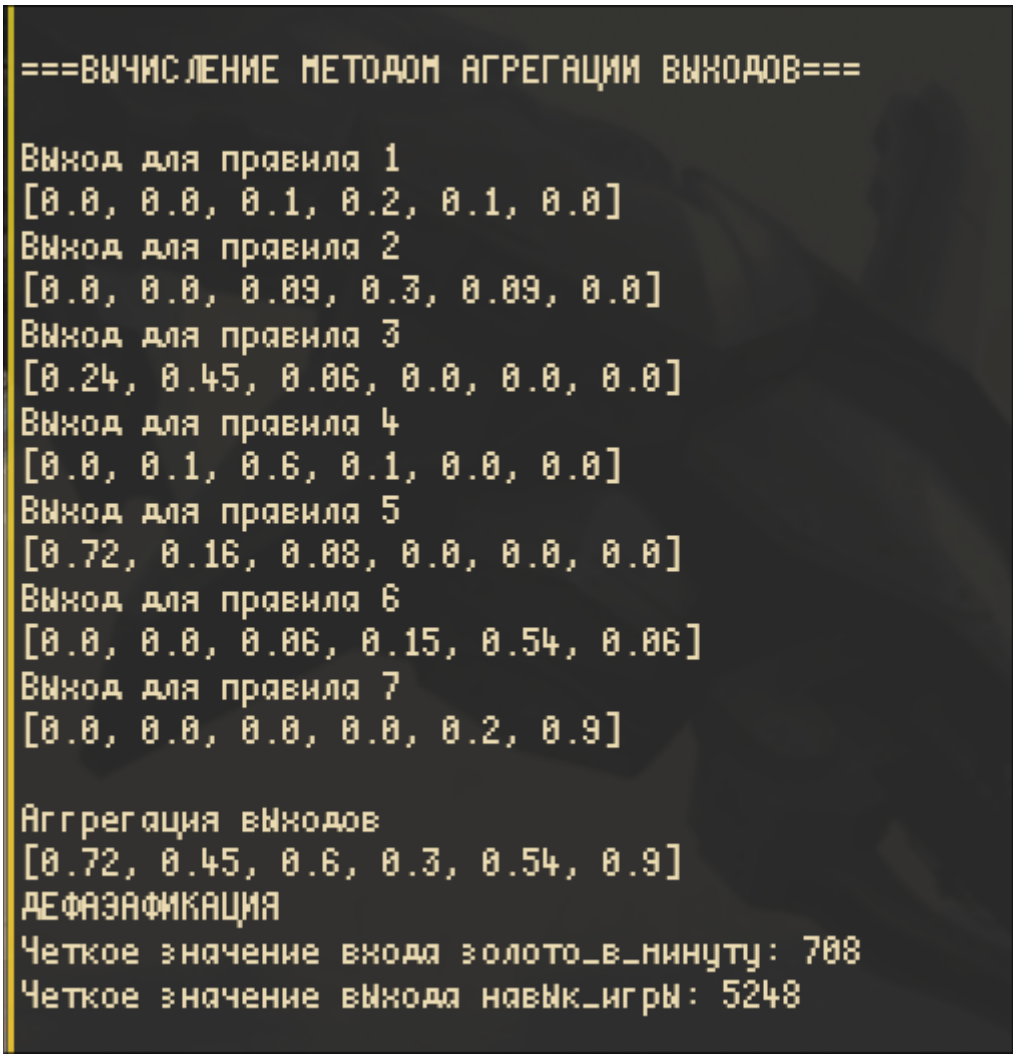

Рисунок 2 — Пример выполнения программы 1

Рисунок 2 — результат программы 1 в виде графов

```
Матрица правил:
  обслуживание          еда  ожидание             оценка
0       грубое       ужасное    долгое              плохо
1       грубое       среднее   среднее              плохо
2      среднее       среднее   среднее  удовлетворительно
3      среднее  превосходное  короткое  удовлетворительно
4    идеальное  превосходное  короткое           идеально
5    идеальное       среднее  короткое           идеально
6    идеальное  превосходное   среднее           идеально

Пусть:
[[0.3, 0.8, 0.6, 0.1], [0.4, 0.7, 0.9, 0.2], [0.2, 0.8, 0.5, 0.1]]

Уровни истинности предпосЫлок правил:
[0.5, 0.8, 0.6, 0.2, 0.2, 0.4, 0.2]

ВЫход для правила 1
[0.5, 0.5, 0.3, 0.0]
ВЫход для правила 2
[0.8, 0.7, 0.3, 0.0]
ВЫход для правила 3
[0.0, 0.6, 0.6, 0.3]
ВЫход для правила 4
[0.0, 0.2, 0.2, 0.2]
ВЫход для правила 5
[0.0, 0.0, 0.2, 0.2]
ВЫход для правила 6
[0.0, 0.0, 0.4, 0.4]
ВЫход для правила 7
[0.0, 0.0, 0.2, 0.2]


Аггрегация вЫходов
[0.8, 0.7, 0.6, 0.4]

Четкое значение входа обслуживание: 4
Четкое значение входа еда: 4
Четкое значение входа ожидание: 24
Четкое значение вЫхода оценка: 4
```

Вывод:

В ходе выполнения лабораторной работы реализовали 3 метода — три механизма логического вывода: Мамдани, Ларсена, матод истинности предпосылок.