

Министерство образования и науки РФ
Пермский национальный исследовательский политехнический университет
Электротехнический факультет
Кафедра информационных технологий и автоматизированных систем

Дискретная математика и математическая логика
Лабораторная работа № 2

Тема: «Разработка калькулятора свойств отношений»

Выполнил: студент группы
ИВТ-23-1Б
Долганова Диана Евгеньевна
Проверил: ст. пр.
Рустамханова Г. И.

Оглавление

Цель работы	1
Задачи работы	2
Этапы выполнения	3
1. Класс Relation	4
2. Чтение матрицы из файла	4
3. Проверка рефлексивности	5
4. Проверка антирефлексивности	6
5. Проверка симметричности	6
6. Проверка антисимметричности	7
7. Проверка асимметричности	7
8. Проверка транзитивности	8
9. Проверка связности	8
10. Главная функция	9
Тестирование программы	10
Заключение	12
Список используемой литературы	12

Цель работы

Разработать калькулятор свойств отношений.

Задачи работы

1. Реализовать ввод матрицы отношений из файла.
2. Вывести список свойств, которым обладает данное отношение (рефлексивность, антирефлексивность, симметричность, антисимметричность, ассимметричность, транзитивность, полнота).

Этапы выполнения

Программа была разработана на языке C++ в программе среде Microsoft Visual Studio 2022.

1. Класс Relation

Определение класса Relation, который будет представлять отношение. Метод checkProperties() выводит на экран свойства отношения, вызывая соответствующие методы для их проверки и выводя результат (рис. 1).

```
#include <iostream>
#include <fstream>
#include <vector>

using namespace std;

class Relation {
public:
    Relation(const string& filename) {
        readFromFile(filename);
    }

    void checkProperties() {
        cout << "Свойства отношения:\n";
        cout << "Рефлексивность: " << (isReflexive() ? "Да" : "Нет") << endl;
        cout << "Антирефлексивность: " << (isIrreflexive() ? "Да" : "Нет") << endl;
        cout << "Симметричность: " << (isSymmetric() ? "Да" : "Нет") << endl;
        cout << "Антисимметричность: " << (isAntisymmetric() ? "Да" : "Нет") << endl;
        cout << "Ассиметричность: " << (isAsymmetric() ? "Да" : "Нет") << endl;
        cout << "Транзитивность: " << (isTransitive() ? "Да" : "Нет") << endl;
        cout << "Связность: " << (isConnected() ? "Да" : "Нет") << endl;
    }
}
```

Рисунок 1 - Метод checkProperties()

2. Чтение матрицы из файла

Метод чтения матрицы из файла реализован следующим образом (рис. 2):

- 1) vector<vector<int>> matrix;; Двумерный вектор для хранения матрицы отношений.
- 2) void readFromFile(const string& filename): Метод для чтения матрицы из файла.

- 3) `ifstream file(filename);`: Открывает файл с заданным именем.
- 4) Если файл не удастся открыть, выводит сообщение об ошибке и завершает программу.
- 5) `while (!file.eof())`: Цикл для чтения значений из файла.
- 6) `file >> value`: Читает следующие целое число.
- 7) Если достигнут конец строки или файла, добавляет заполненную строку (`row`) в матрицу и очищает её для следующей строки.

```
private:
    vector<vector<int>> matrix;

    void readFromFile(const string& filename) { //Метод для чтения матрицы из файла
        ifstream file(filename);
        if (!file.is_open()) {
            cerr << "Ошибка подключения к файлу!" << endl;
            exit(1);
        }

        vector<int> row;
        while (!file.eof()) {
            int value;
            if (file >> value) {
                row.push_back(value);
            }
            if (file.peek() == '\n' || file.eof()) {
                if (!row.empty()) {
                    matrix.push_back(row);
                    row.clear();
                }
            }
        }

        file.close();
    }
}
```

Рисунок 2 - Метод `readFromFile`

3. Проверка рефлексивности

Отношение $[R, \Omega]$ называется рефлексивным, если каждый элемент множества находится в отношении R сам с собой. Функция `isReflexive()` Проверяет, что каждый элемент на главной диагонали равен 1 (т.е. `matrix[i][i]`). Если есть хотя бы один 0, возвращает `false` (рис. 3).

```

// Проверка рефлексивности
bool isReflexive() {
    int n = matrix.size();
    for (int i = 0; i < n; ++i) {
        if (matrix[i][i] == 0) {
            return false;
        }
    }
    return true;
}

```

Рисунок 3 - Функция isReflexive()

4. Проверка антирефлексивности

Отношение $[R, \Omega]$ называется антирефлексивным, если ни один элемент из множества не находится в отношении R сам с собой. Функция isIrreflexive() Проверяет, что каждый элемент на главной диагонали равен 0. Если есть хотя бы один 1, возвращает false (рис. 4).

```

// Проверка антирефлексивности
bool isIrreflexive() {
    int n = matrix.size();
    for (int i = 0; i < n; ++i) {
        if (matrix[i][i] == 1) {
            return false;
        }
    }
    return true;
}

```

Рисунок 4 - Функция isIrreflexive()

5. Проверка симметричности

Отношение $[R, \Omega]$ называется симметричным, если вместе с упорядоченной парой (x, y) отношение содержит и упорядоченную пару (y, x) . Функция isSymmetric() Проверяет, что для всех i и j выполняется: $matrix[i][j] == matrix[j][i]$. Если есть несоответствие, возвращает false (рис. 5)

```

// Проверка симметричности
bool isSymmetric() {
    int n = matrix.size();
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (matrix[i][j] != matrix[j][i]) {
                return false;
            }
        }
    }
    return true;
}

```

Рисунок 5 - Функция isSymmetric()

6. Проверка антисимметричности

Отношение $[R, \Omega]$ называется антисимметричным, если, если для всякой упорядоченной пары $(x, y) \in R$ упорядоченная пара $(y, x) \in R$, только в случае $x = y$. Функция isAntisymmetric() проверяет, что если $matrix[i][j] == 1$ и $matrix[j][i] == 1$, при этом i и j разные, то возвращает false (рис. 6).

```

// Проверка антисимметричности
bool isAntisymmetric() {
    int n = matrix.size();
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (matrix[i][j] == 1 && matrix[j][i] == 1 && i != j) {
                return false; // Если обе связи существуют для разных элементов
            }
        }
    }
    return true;
}

```

Рисунок 6 - isAntisymmetric()

7. Проверка асимметричности

Отношение $[R, \Omega]$ называется асимметричным, если оно антирефлексивно и для всякой упорядоченной пары $(x, y) \in R$ упорядоченная пара $(y, x) \notin R$. Функция isAsymmetric() Проверяет, что если $matrix[i][j] == 1$, то $matrix[j][i]$ должно быть равно 0. Если это не так, возвращает false (рис. 7).


```

// Проверка ассиметричности
bool isAsymmetric() {
    int n = matrix.size();
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            if (matrix[i][j] == 1 && matrix[j][i] == 1) {
                return false; // Если существует обратная связь
            }
        }
    }
    return true;
}

```

Рисунок 7 - Функция isAsymmetric()

8. Проверка транзитивности

Отношение $[R, \Omega]$ называется транзитивным, если для всяких упорядоченных пар $(x, y), (y, z) \in R$, в отношении R найдется упорядоченная пара $(x, z) \in R$. Функция isTransitive() Проверяет, что если $matrix[i][j] == 1$ и $matrix[j][k] == 1$, то $matrix[i][k]$ также должно быть 1. Если это не так, возвращает false (рис. 8).

```

// Проверка транзитивности
bool isTransitive() {
    int n = matrix.size();
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            for (int k = 0; k < n; ++k) {
                if (matrix[i][j] == 1 && matrix[j][k] == 1 && matrix[i][k] == 0) {
                    return false; // Транзитивность нарушена
                }
            }
        }
    }
    return true;
}

```

Рисунок 8 - Функция isTransitive()

9. Проверка связности

Отношение $[R, \Omega]$ называется полным (связным), если для любых двух элементов $(y, z) \in \Omega$ один из них находится в отношении с другим. Функция isConnected() Проверяет, что каждый элемент (строка) имеет хотя бы одну 1,

указывающую на связь с другим элементом, кроме себя. Если такой связи нет, возвращает false (рис. 9).

```
// Проверка связности
bool isConnected() {
    int n = matrix.size();
    for (int i = 0; i < n; ++i) {
        bool connected = false;
        for (int j = 0; j < n; ++j) {
            if (i != j && matrix[i][j] == 1) {
                connected = true;
                break;
            }
        }
        if (!connected) {
            return false;
        }
    }
    return true;
};
```

Рисунок 9 - Функция isConnected()

10. Главная функция


Функция int main() Создает объект relation класса Relation, считывая матрицу из файла matrix.txt. Вызывает метод checkProperties() для проверки и распечатки свойств отношения (рис. 10).

```
int main() {
    setlocale(LC_ALL, "RU");
    Relation relation("matrix.txt");
    relation.checkProperties();
    return 0;
}
```

Рисунок 10 - Главная функция


Тестирование программы

Для проверки программы на корректность запишем в файл matrix.txt матрицы отношений и запустим программу (рис. 11 - 16).

 matrix.txt – Блокнот


```
Файл  Правка  Формат  Вид  Справка
0 1 0
0 0 1
0 0 0
```

Рисунок 11 - Первая матрица отношений

 Консоль отладки Microsoft Visual Studio

```
Свойства отношения:
Рефлексивность: Нет
Антирефлексивность: Да
Симметричность: Нет
Антисимметричность: Да
Ассиметричность: Да
Транзитивность: Нет
Связность: Нет
```

Рисунок 12 - Свойства первого отношения

 *matrix.txt – Блокнот

```
Файл  Правка  Формат  Вид  Справка
1 1 0 1
0 1 1 1
0 0 1 1
1 0 1 1
```

Рисунок 13 - Вторая матрица отношений

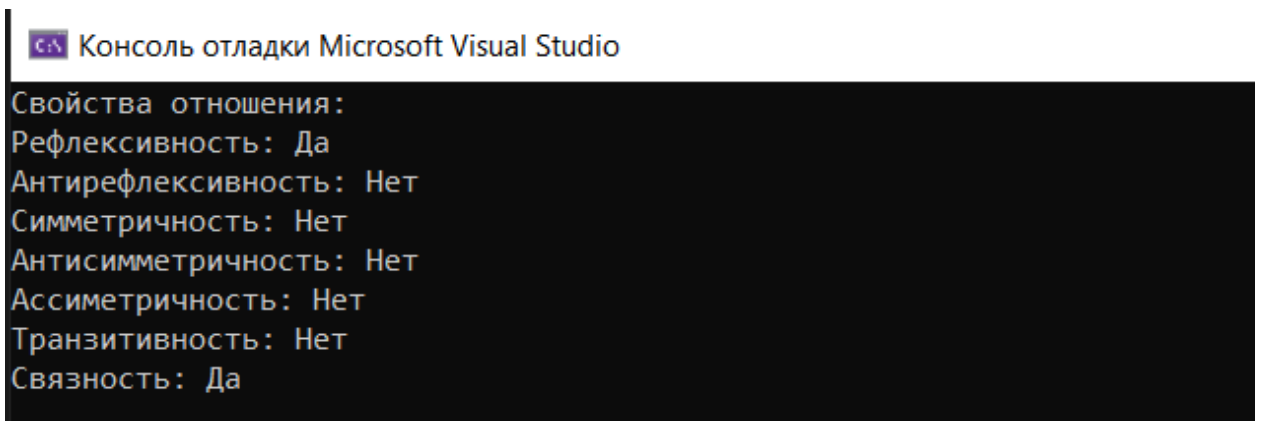


Рисунок 14 - Свойства второй матрицы отношений

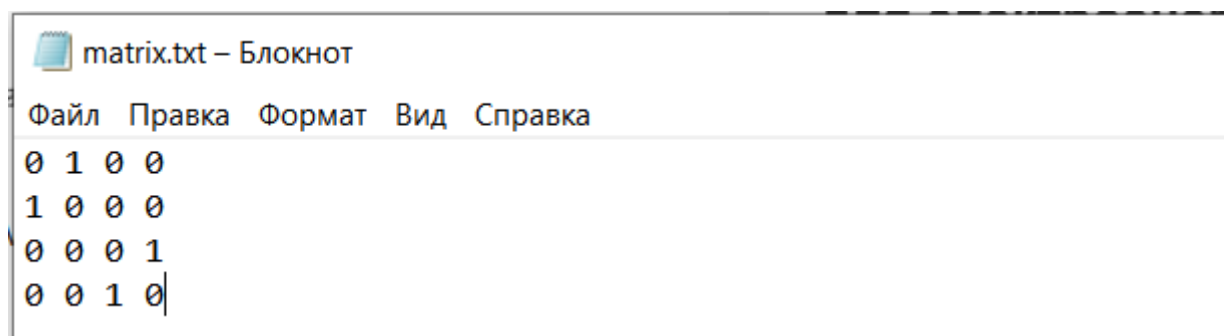


Рисунок 15 - Третья матрица отношений



Рисунок 16 - Свойства третьей матрицы отношений

Заключение

Был разработан калькулятор свойств отношений. Были изучены свойства отношений. Научились определять свойства отношения по матрице отношений.

Список используемой литературы

1. Отношения. Часть I / Хабр URL: <https://habr.com/ru/articles/515014/> (дата обращения: 22.10.2024).
2. Матрицы конечных бинарных отношений URL: <https://studfile.net/preview/7497108/page:10/> (дата обращения: 24.10.2024).