МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Направление: 02.03.01 Математика и компьютерные науки

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

по дисциплине Дискретная математика

# Исследование булевых функций


Студент,
группы 5130201/40003 ________________ Адиатуллин Т.Р

Доцент
________________ Востров А.В


«_____»________________ 20__ г.

Санкт-Петербург, 2025

# Содержание

# Введение

В данной работе задана булева функция (порядок определен по возрастанию элементов). Лабораторная работа состоит из двух частей:

1. **Расчетная часть**. Построить таблицу истинности, семантическое дерево. Найти наиболее компактную бинарную диаграмму решений (вручную, шаги построения БДР должны быть в отчете), записать по ней формулу в виде ДНФ и построить синтаксическое дерево по ней. Порядок следования переменных выбрать самостоятельно и аргументировать выбор. Вывести первые производные по каждой переменной. По таблице истинности написать формулу четвертой производной. Найти минимальную ДНФ форму и построить по ней логическую схему (с помощью вентилей 'и', 'или', 'не').

2. **Программная часть**. По заданной таблице истинности программно построить СДНФ и СКНФ. Реализовать программно хранение полученной в первом пункте наиболее компактной бинарной диаграммы решений и вычисление по ней значения (по пользовательскому вводу значения переменных). Для заданной функции построить программно полином Жегалкина (любым способом) и вычислить по нему значение булевой функции согласно пользовательскому вводу.

Исходная функция в десятичной системе счисления:

$$f = 52375$$

# 1 Математическое описание

## 1.1 Булевы функции

Функции $f : E_2^n \to E_2$, где $E_2 \overset{\mathrm{Def}}{=} \{0, 1\}$, называются функциями алгебры логики, или булевыми функциями от $n$ переменных, по имени Дж. Буля.

Множество булевых функций от $n$ переменных обозначим

$$P_n, \quad P_n \overset{\mathrm{Def}}{=} \{f : E_2^n \to E_2\}.$$

Булеву функцию от $n$ переменных можно задать таблицей истинности.

| $x_1$ | $x_2$ | $\cdots$ | $x_n$ | $f(x_1, \ldots, x_n)$ |
|---|---|---|---|---|
| 0 | 0 | $\cdots$ | 0 | $f(0, \ldots, 0)$ |
| 0 | 0 | $\cdots$ | 1 | $f(0, \ldots, 1)$ |
| $\vdots$ | $\vdots$ | $\cdots$ | $\vdots$ | $\vdots$ |
| 1 | 1 | $\cdots$ | 0 | $f(1, \ldots, 0)$ |
| 1 | 1 | $\cdots$ | 1 | $f(1, \ldots, 1)$ |

Если число переменных равно $n$, то таблица истинности булевой функции $f(x_1, \ldots, x_n)$ содержит $2^n$ строк, соответствующих всем возможным наборам значений переменных.

Исходная функция в десятичной системе счисления

$$f = 52375_{10}$$

Была переведена в двоичную систему:

$$f(x_1, x_2, x_3, x_4) = (1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1)$$

Построим таблицу истинности по этой функции:

| $x_1$ | $x_2$ | $x_3$ | $x_4$ | $f(x_1, x_2, x_3, x_4)$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Таблица 1: Таблица истенности для булевой функции $f(x_1, x_2, x_3, x_4) = (1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1)$

## 1.2 Деревья решений

Таблицу истинности булевой функции $n$ переменных можно представить в виде полного бинарного дерева высоты $n + 1$. Ярусы дерева соответствуют переменным, дуги дерева соответствуют значениям переменных, скажем, левая дуга - 0 (пунктир), а правая - 1 (сплошная). Листья дерева на последнем ярусе хранят значение функции на кортеже, соответствующем пути из корня в этот лист. Такое дерево называется деревом решений (или семантическим деревом).

Семантическое дерево для исходной булевой функции представленно на рисунке 1

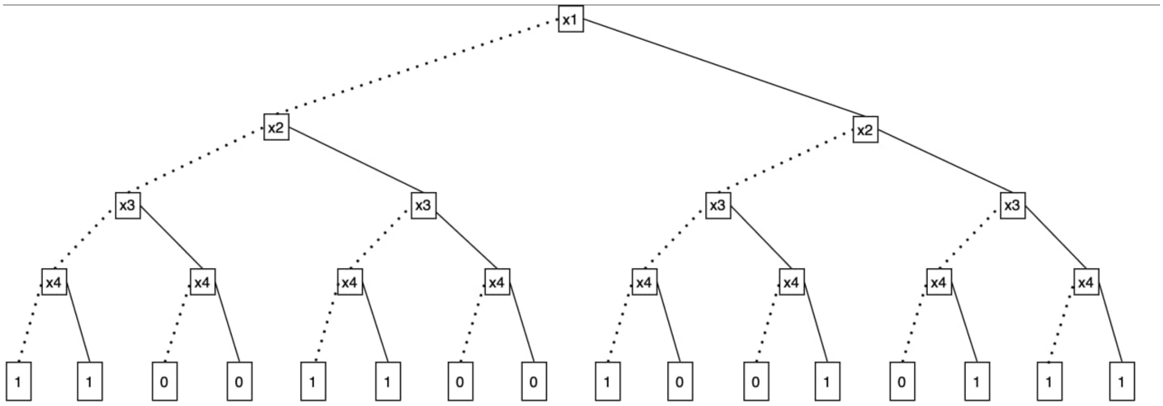



Рис. 1: Семантическое дерево булевой функции

Если ветки дерева ведут к одному и тому же листу, его можно упростить(см. рис 2)

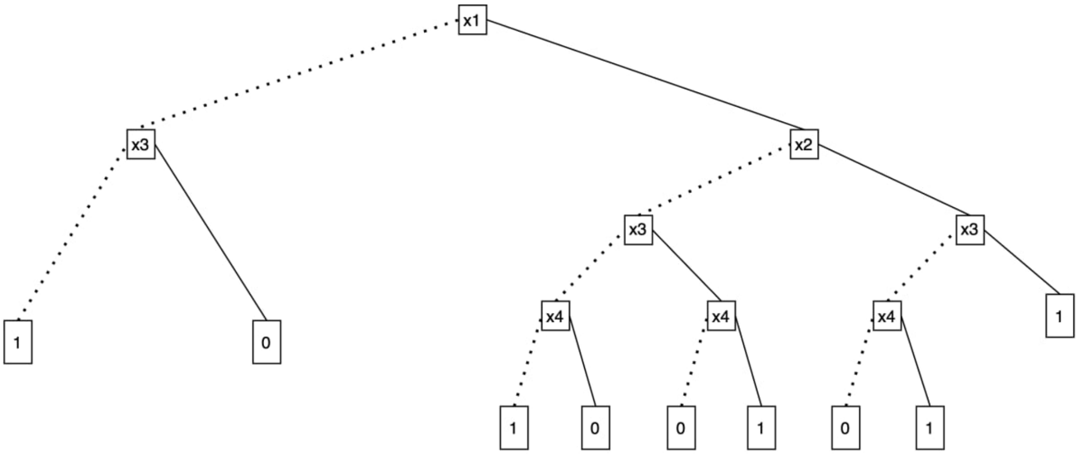



Рис. 2: Сокращенное семантическое дерево булевой функции

Дерево решений можно еще упростить, если отказаться от древовидности связей, то есть сделать так чтобы несколько ребер входили в листы $\{0, 1\}$. Следовательно мы можем получить бинарную диаграмму решений (БДР). Прежде чем строить БДР определим последовательность элементов для построения дерева. Для этого была использован алгоритм просеивания Руделла (Rudell's sifting). Идея алгоритма заключается в том, что порядок переменных в БДР влияет на его размер. Алгоритм просеивания последовательно выбирает одну переменную и просеивает её через текущий порядок перемещает вверх и вниз по списку переменных, измеряя размер БДР при каждом положении. После оценки всех возможных позиций переменной фиксируется то её положение, при котором размер диаграммы минимален, и затем алгоритм переходит к обработке следующей переменной.

В данной работе алгоритм просеивания использовался для выбора порядка переменных перед построением диаграммы. После перебора возможных перестановок был определён порядок $x_1, x_2, x_4, x_3$, при котором количество узлов в диаграмме оказалось минимальным. Этот порядок и был использован при формировании конечного BDD.(см. рис 3)

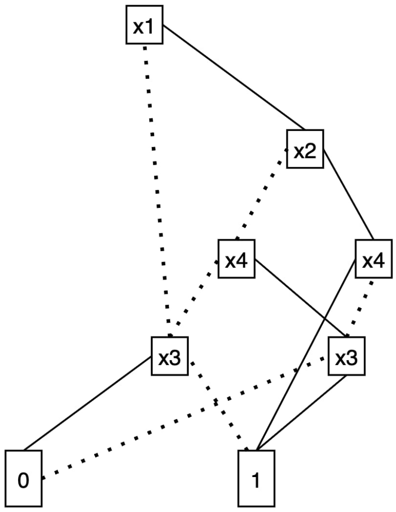



Рис. 3: Оптимальное Бинарное Диаграмма Решений

## 1.3 Нормальные формы

Дизъюнктивная нормальная форма, ДНФ — нормальная форма, в которой булева функция имеет вид дизъюнкции нескольких простых конъюнктов.

Реализация булевой функции $f(x_1, \dots, x_n)$ в виде формулы

$$\bigvee_{\{(\sigma_1,\dots,\sigma_n)|f(\sigma_1,\dots,\sigma_n)=1\}} (x_1^{\sigma_1} \wedge \dots \wedge x_n^{\sigma_n})$$

называется совершенной дизъюнктивной нормальной формой (СДНФ).

Конъюнктивная нормальная форма, КНФ — нормальная форма, в которой булева функция имеет вид конъюнкции нескольких простых дизъюнктов.

Реализация булевой функции $f(x_1, \dots, x_n)$ в виде формулы

$$\bigwedge_{(\sigma_1,\dots,\sigma_n)|f(\sigma_1,\dots,\sigma_n)=1} (x_1^{\sigma_1} \vee \dots \vee x_n^{\sigma_n})$$

называется совершенной конъюнктивной нормальной формой (СКНФ).

По БДР было записана формула для функции $f(x_1, x_2, x_3, x_4)$ в виде ДНФ.

$$\text{ДНФ} f = \overline{x}_1\,\overline{x}_3 \vee x_1\,\overline{x}_2\,\overline{x}_3\,\overline{x}_4 \vee x_1\,x_2\,x_3 \vee x_1\,\overline{x}_2\,x_3\,x_4 \vee x_1\,x_2\,\overline{x}_3\,x_4$$

По полученной ДНФ было построенно синтаксическое дерево (см. рис.4)

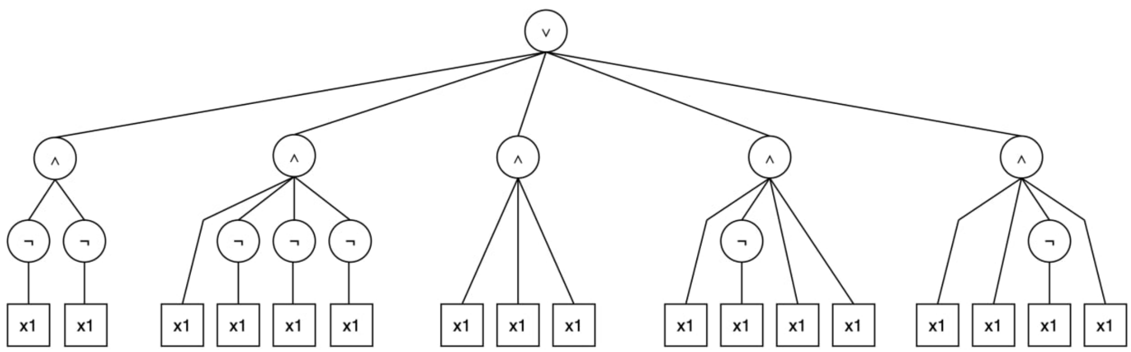



Рис. 4: Синтаксическое дерево

## 1.4 Производная булевой функции

Как и любая булева функция, производная булевой функции принимает значения 0 или 1. В случае, если булева функция при изменении одного из её аргументов не меняет своего значения, булева производная по этому аргументу равна 0. В противном случае производная равна 1, независимо от того, как именно с (0→1 или 1→0) меняется функция при изменении аргумента 0→1.

В формальном виде определение булевой производной записывается следующим образом:

$$\frac{df}{dx_i} = f(x_1, \ldots, 0, \ldots, x_n) \oplus f(x1, \ldots, 1, \ldots, x_n)$$

По каждой переменной функции были найдены её первые производные:

**Производная по $x_1$:**

$$\frac{\partial f}{\partial x_1} = (\overline{x}_2\overline{x}_3\overline{x}_4 \vee x_2x_3 \vee \overline{x}_2x_3x_4 \vee x_2\overline{x}_3x_4) \oplus \overline{x}_3$$

$$\frac{\partial f}{\partial x_1} = \overline{x}_3 \oplus (\overline{x}_2\overline{x}_3\overline{x}_4 \vee x_2x_3 \vee \overline{x}_2x_3x_4 \vee x_2\overline{x}_3x_4)$$

$$\frac{\partial f}{\partial x_1} = (\overline{x}_3 \wedge \overline{\overline{x}_2\overline{x}_3\overline{x}_4 \vee x_2x_3 \vee \overline{x}_2x_3x_4 \vee x_2\overline{x}_3x_4}) \vee (x_3 \wedge (\overline{x}_2\overline{x}_3\overline{x}_4 \vee x_2x_3 \vee \overline{x}_2x_3x_4 \vee x_2\overline{x}_3x_4))$$

$$\frac{\partial f}{\partial x_1} = (\overline{x}_3\overline{x}_2\overline{x}_4 \vee \overline{x}_3x_2\overline{x}_4 \vee \overline{x}_3\overline{x}_2x_4) \vee (x_3x_2 \vee x_3\overline{x}_2x_4)$$

$$\frac{\partial f}{\partial x_1} = \overline{x}_3x_2\overline{x}_4 \vee \overline{x}_3\overline{x}_2x_4 \vee x_3x_2 \vee x_3x_4$$

$$\frac{\partial f}{\partial x_1} = \overline{x_2}x_4 \vee x_2\overline{x_4} \vee x_3x_4$$

**Производная по $x_2$:**

$$\frac{\partial f}{\partial x_2} = (\overline{x}_1\overline{x}_3 \vee x_1\overline{x}_3\overline{x}_4 \vee x_1x_3x_4) \oplus (\overline{x}_1\overline{x}_3 \vee x_1x_3 \vee x_1\overline{x}_3x_4)$$

$$\frac{\partial f}{\partial x_2} = \overline{(\overline{x}_1\overline{x}_3)} \wedge [(x_1\overline{x}_3\overline{x}_4 \vee x_1x_3x_4) \oplus (x_1x_3 \vee x_1\overline{x}_3x_4)]$$

$$\frac{\partial f}{\partial x_2} = (x_1 \vee x_3) \wedge x_1 \wedge [(\overline{x}_3\overline{x}_4 \vee x_3x_4) \oplus (x_3 \vee \overline{x}_3x_4)]$$

$$\frac{\partial f}{\partial x_2} = x_1 \wedge [(\overline{x}_3\overline{x}_4 \vee x_3x_4) \oplus (x_3 \vee x_4)]$$

$$\frac{\partial f}{\partial x_2} = x_1 \wedge (\overline{x}_3 \vee \overline{x}_4)$$

$$\frac{\partial f}{\partial x_2} = x_1\overline{x}_3 \vee x_1\overline{x}_4$$

**Производная по $x_3$:**

$$\frac{\partial f}{\partial x_3} = (\overline{x}_1 \vee x_1\overline{x}_2\overline{x}_4 \vee x_1x_2x_4) \oplus (x_1x_2 \vee x_1\overline{x}_2x_4)$$

$$\frac{\partial f}{\partial x_3} = (\overline{x}_1 \vee \overline{x}_2\overline{x}_4 \vee x_2x_4) \oplus (x_1x_2 \vee x_1x_4)$$

$$\frac{\partial f}{\partial x_3} = (\overline{x}_1 \vee \overline{x}_2\overline{x}_4) \vee (x_1x_2\overline{x}_4 \vee x_1\overline{x}_2x_4)$$

$$\frac{\partial f}{\partial x_3} = \overline{x}_1 \vee \overline{x}_2\overline{x}_4 \vee x_1x_2\overline{x}_4 \vee x_1\overline{x}_2x_4$$

$$\frac{\partial f}{\partial x_3} = \overline{x}_1 \vee (\overline{x}_2\overline{x}_4 \vee x_2\overline{x}_4) \vee \overline{x}_2x_4$$

$$\frac{\partial f}{\partial x_3} = \overline{x}_1 \vee \overline{x}_4 \vee \overline{x}_2x_4$$

$$\frac{\partial f}{\partial x_3} = \overline{x}_1 \vee \overline{x}_4 \vee \overline{x}_2$$

**Производная по $x_4$:**

$$\frac{\partial f}{\partial x_4} = (\overline{x}_1\overline{x}_3 \vee x_1\overline{x}_2\overline{x}_3 \vee x_1x_2x_3) \oplus (\overline{x}_1\overline{x}_3 \vee x_1x_2x_3 \vee x_1\overline{x}_2x_3 \vee x_1x_2\overline{x}_3)$$

$$\frac{\partial f}{\partial x_4} = \overline{(\overline{x}_1\overline{x}_3 \vee x_1x_2x_3)} \wedge (x_1\overline{x}_2\overline{x}_3 \oplus (x_1\overline{x}_2x_3 \vee x_1x_2\overline{x}_3))$$

$$\frac{\partial f}{\partial x_4} = ((x_1 \vee x_3) \wedge (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)) \wedge x_1 \wedge (\overline{x}_2\overline{x}_3 \oplus (\overline{x}_2x_3 \vee x_2\overline{x}_3))$$

$$\frac{\partial f}{\partial x_4} = (x_1\overline{x}_2 \vee x_1\overline{x}_3 \vee \overline{x}_1x_3 \vee \overline{x}_2x_3) \wedge x_1 \wedge (\overline{x}_2 \vee \overline{x}_3)$$

$$\frac{\partial f}{\partial x_4} = (x_1\overline{x}_2 \vee x_1\overline{x}_3) \wedge (\overline{x}_2 \vee \overline{x}_3)$$

$$\frac{\partial f}{\partial x_4} = x_1\overline{x}_2 \vee x_1\overline{x}_3$$

Также была найдена формула четверой производной с помощью таблицы истинности:

$$\frac{d^4 f}{d(x1, x2, x3, x4)} =$$

$$= \overline{x}_1\,\overline{x}_2\,x_3\,\overline{x}_4 \lor x_1\,x_2\,\overline{x}_3\,x_4 \lor \overline{x}_1\,x_2\,\overline{x}_3\,x_4 \lor x_1\,\overline{x}_2\,x_3\,\overline{x}_4 \lor \overline{x}_1\,x_2\,x_3\,x_4 \lor x_1\,\overline{x}_2\,\overline{x}_3\,\overline{x}_4$$

## 1.5 Логическая схема

Для построения логической схемы была найдеда минимальная ДНФ

ДНФ f $= \overline{x}_1\,\overline{x}_3 \lor x_1\,\overline{x}_2\,\overline{x}_3\,\overline{x}_4 \lor x_1\,x_2\,x_3 \lor x_1\,\overline{x}_2\,x_3\,x_4 \lor x_1\,x_2\,\overline{x}_3\,x_4 =$

$$= \overline{x}_1\,\overline{x}_3 \lor \overline{x}_2\,\overline{x}_3\,\overline{x}_4 \lor x_1\,x_3\,x_4 \lor x_1\,x_2\,x_3 \lor x_1\,x_2\,x_4$$

Логическая схема была построена в программе Multisim (см. рис.5)

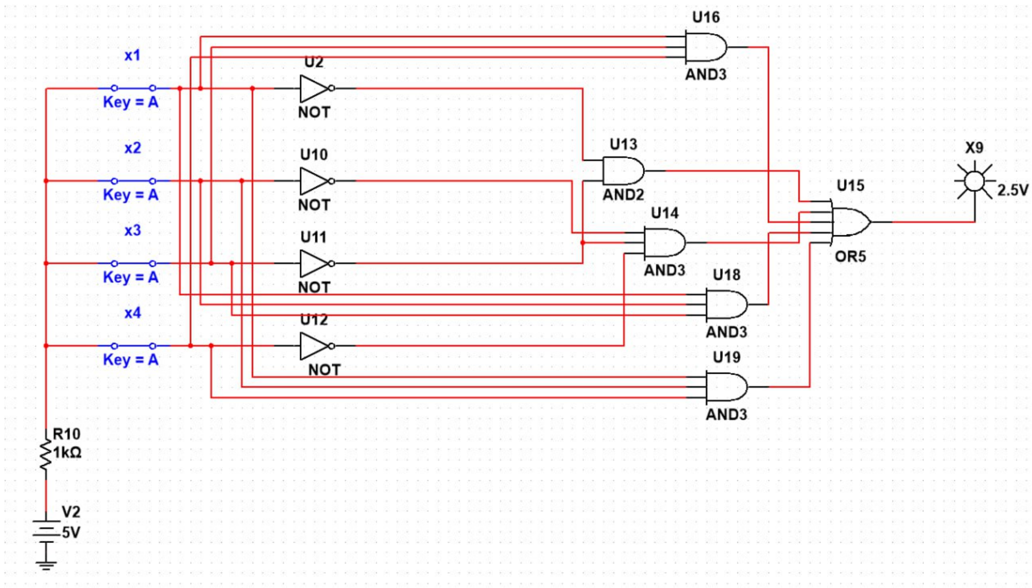



Рис. 5: Логическая схема ДНФ

# 2 Особенности реализации

Программная часть проекта реализована на языке C++ с применением стандартной библиотеки шаблонов (STL). Код программы структурирован по отдельным файлам, каждый из которых отвечает за определённый участок функционала, что обеспечивает модульность системы и упрощает её дальнейшее сопровождение и развитие.

## 2.1 Структура проекта

Проект имеет следующую структуру:

- `config.h` содержит используемые переменные и заданный вектор булевой функции
- `zheg.h` отвечает за построение полинома Жигалкина
- `bdd.h`: отвечает за хранения БДР в виде графа
- `help.h`: содержит функции построения СДНФ и СКНФ
- `ui.h`: отвечают за пользовательский интерфейс и взаимодействие с пользователем.
- `main.cpp`: точка входа в программу, инициализация объектов и запуск пользовательского интерфейса.
- `CMakeLists.txt`: автоматизация процесса сборки проекта.

## 2.2 Структуры данных

Для представления и анализа двоичных функций в проекте используются два класса: **ZhegalkinPolynomial** и **BDDGraph**. Они реализуют ключевые методы работы с булевыми функциями и предоставляют интерфейс для вычислений.

Класс **ZhegalkinPolynomial** формирует полином Жегалкина по заданной таблице истинности. Он хранит значения функции, треугольник коэффициентов и генерируемые термы.

Класс **BDDGraph** реализует бинарную диаграмму решений. Он содержит узлы графа, связи между ними и механизм обхода диаграммы, обеспечивая вычисление значения функции

## 2.3 Класс ZhegalkinPolynomial

### 2.3.1 Внутренняя структура класса ZhegalkinPolynomial

Класс содержит следующие поля:

- `int numsVars`: Количество переменных булевой функции
- `vector<int> truthTable`: Поле для хранения таблицы истинности
- `vector< vector<int> > triangle`: Двумерный массив для построения треугольника Жегалкина, в котором вычисляются коэффициенты полинома.
- `vector<string> terms`: Массив строковых представлений термов полинома Жегалкина.

### 2.3.2 Интерфейс класса ZhegalkinPolynomial

```
class ZhegalkinPolynomial {
private:
    int numVars;
    vector<int> truthTable;
    vector<vector<int>> triangle;
    vector<string> terms;

public:
    ZhegalkinPolynomial(int n) : numVars(n) {
        int size = pow(2, n);
        truthTable.resize(size);
        triangle.resize(size);
        for (int i = 0; i < size; i++) {
            triangle[i].resize(size - i);
        }
    }

    void setTruthTableFromVector(const vector<int>& F);

    void buildTriangle();
    void printTriangle();

    string generateTerm(int index);
    string buildPolynomial();

    void printResult();

    int evaluatePolynomial(const vector<int>& vars);
};
```

Листинг 1: Объявление класса ZhegalkinPolynomial

## 2.4 Реализация функций класса ZhegalkinPolynomial

### 2.4.1 Функция setTruthTableFromVector(const vector<int> F)

**Назначение:** Загружает таблицу истинности булевой функции из переданного вектора значений и выводит её в консоль в табличном виде.

**Вход:**

- `const vector<int> F` - вектор значений функции, где каждый элемент соответствует выходу булевой функции для определённого набора входных переменных.

**Выход:**

- `vector<int> truthTable` - таблица истинности в классе ZhegalkinPolynomial

**Алгоритм:** Функция проверяет, совпадает ли размер переданного вектора F с числом всех комбинаций входных переменных. В случае корректного размера вектор сохраняется в поле truthTable, после чего формируется и выводится таблица истинности, где каждая строка соответствует набору входных переменных и значению функции.

```cpp
void setTruthTableFromVector(const vector<int>& F) {
        int size = pow(2, numVars);

        if (F.size() != size) {
            cerr << "Ошибка! Размер вектора F (" << F.size()
                 << ") не соответствует количеству переменных (ожидается "
                 << size << ")" << endl;
            return;
        }

        truthTable = F;

        cout << "\n=== Таблица истинности загружена ===" << endl;
        for (int i = 0; i < numVars; i++) {
            cout << "x" << (i + 1) << " ";
        }
        cout << "| f" << endl;
        cout << string(numVars * 3 + 3, '-') << endl;

        // Вывод значений
        for (int i = 0; i < size; i++) {
            for (int j = numVars - 1; j >= 0; j--) {
                cout << ((i >> j) & 1) << "  ";
            }
            cout << "| " << truthTable[i] << endl;
        }
        cout << endl;
    }
```

Листинг 2: Функция setTruthTableFromVector(const vector<int> F)

### 2.4.2 Функция buildTriangle()

**Назначения:** Построение треугольника Жегалкина для вычисления коэффициентов полинома Жегалкина на основе таблицы истинности.

**Вход:**

- `int numVars` - количество переменных булевой функции

**Выход:**

- `vector< vector<int> > triangle` - двумерный массив для построения треугольника Жегалкина, в котором пошагово вычисляются коэффициенты полинома.

```
void buildTriangle() {
    int size = pow(2, numVars);

    for (int i = 0; i < size; i++) {
        triangle[0][i] = truthTable[i];
    }

    for (int row = 1; row < size; row++) {
        for (int col = 0; col < size - row; col++) {
            triangle[row][col] = triangle[row - 1][col] ^
               triangle[row - 1][col + 1];
        }
    }
}
```

Листинг 3: Функция buildTriangle()

### 2.4.3 Функция buildPolynomial()

**Назначение:** Построение полинома Жегалкина в аналитической форме на основе ранее сформированного треугольника Жегалкина.
**Вход:**

- `int numVars` — количество переменных булевой функции.
- `vector<vector<int» triangle` — треугольник Жегалкина с рассчитанными коэффициентами.

**Выход:**

- `string polynomial` — строковое представление полинома Жегалкина.

**Алгоритм:** Функция просматривает левый столбец треугольника Жегалкина, где хранятся коэффициенты полинома. Для каждого коэффициента, равного 1, формируется соответствующий терм с помощью метода generateTerm(), который добавляется в итоговый полином и в список terms. Если полином пуст (нет ненулевых коэффициентов), возвращается строка "0".

```
string buildPolynomial() {
    int size = pow(2, numVars);
    string polynomial = "";
    bool first = true;

    minterms.clear();

    // левая колонка треугольника содержит коэффициенты
    for (int i = 0; i < size; i++) {
        if (triangle[i][0] == 1) {
            string term = generateTerm(i);
            terms.push_back(term);

            if (!first) {
                polynomial += "      "; // символ XOR
            }
            polynomial += term;
            first = false;
        }
    }

    if (polynomial.empty()) {
        polynomial = "0";
    }

    return polynomial;
}
```

Листинг 4: Функция buildPolynomial()

## 2.5 Класс BDDGraph

### 2.5.1 Внутренняя структура класса BDDGraph

Класс содержит следующие поля:

- `map<int, BDDNode> nodes`: Поле в для хранения узлов в графе
- `int rootId`: Номер корневого узла графа

### 2.5.2 Описание структуры узла графа BDDNode:

- `int id` — уникальный идентификатор узла.
- `string var` — имя логической переменной в узле.
- `int zero` — идентификатор дочернего узла, по которому продолжается вычисление при значении переменной 0 (пунктирная линия).
- `int one` — идентификатор дочернего узла, по которому продолжается вычисление при значении переменной 1 (сплошная линия).
- `bool isSheet` — признак того, является ли узел листом.

### 2.5.3 Интерфейс класса BDDGraph

```
struct BDDNode {
    int id;
    string var;
    int zero;
    int one;
    bool isSheet;

    BDDNode() : id(-1), var(""), zero(-1), one(-1),
       isSheet(false) {}

    BDDNode(int _id, string _var, int _zero, int _one, bool
       _term)
        : id(_id), var(_var), zero(_zero), one(_one),
           isSheet(_term) {}
};

class BDDGraph {
private:
    map<int, BDDNode> nodes;
    int rootId;

public:
    BDDGraph() : rootId(-1) {}

```

```cpp
    void addNode(int id, string var, int zero, int one, bool
       isSheet) {
        nodes[id] = BDDNode(id, var, zero, one, isSheet);
    }
    void setRoot(int id);

    void buildFromDiagram();

    int evaluate(map<string, int>& values, vector<string>& path);

private:
    int evaluateNode(int nodeId, map<string, int>& values,
       vector<string>& path);

public:
    void printGraph();
};
```

Листинг 5: Объявление класса ZhegalkinPolynomial

## 2.6 Реализация ключевых функций класса BDDGraph

### 2.6.1 Функция addNode()

**Назначение:** Добавление узла в BDD с указанием идентификатора, переменной и связей с дочерними узлами.

**Вход:**

- `int id` — уникальный идентификатор узла.
- `string var` — имя логической переменной в узле.
- `int zero` — идентификатор дочернего узла, по которому продолжается вычисление при значении переменной 0 (пунктирная линия).
- `int one` — идентификатор дочернего узла, по которому продолжается вычисление при значении переменной 1 (сплошная линия).
- `bool isSheet` — признак того, является ли узел листом (терминалом).

**Выход:**

- Узел добавляется в контейнер `nodes` класса `BDDGraph`.

**Алгоритм:** Функция создаёт узел и сохраняет его в словаре `nodes`, используя `id` в качестве ключа. Это обеспечивает быстрый доступ к узлам и позволяет формировать структуру графа

```
void addNode(int id, string var, int zero, int one, bool
   isSheet) {
nodes[id] = BDDNode(id, var, zero, one, isSheet);
}
```

Листинг 6: Функция addNode()

### 2.6.2 Функция buildFromDiagram()

**Назначение:** Построение структуры BDD-графа.
**Вход:**

- Структура BDDGraph.

**Выход:**

- Модифицированная структура BDDGraph

**Алгоритм:** Функция создаёт листья графа, представляющие значения 0 и 1, с помощью метода `addNode()`. Затем формируется граф исходя из BDD.

```
    void buildFromDiagram() {
        // листья
        addNode(0, "0", -1, -1, true);   // 0
        addNode(1, "1", -1, -1, true);   // 1

    // внутренние ноды
        addNode(2, "x3", 1, 0, false);
        addNode(3, "x3", 0, 1, false);
        addNode(4, "x4", 2, 3, false);
        addNode(5, "x4", 3, 1, false);
        addNode(6, "x2", 4, 5, false);
        addNode(7, "x1", 2, 6, false);

    // корневая нода
        setRoot(7);
    }
```

Листинг 7: Функция buildFromDiagram()

## 2.7 Реализация функций построения нормальных форм

### 2.7.1 Функция buildSDNF()

**Назначение:** Построение СДНФ на основе таблицы истинности.
**Вход:**

- `const vector<string> x` — список переменных булевой функции.
- `const vector<int> F` — вектор значений функции.

**Выход:**

- `string result` — строковое представление СДНФ.

**Алгоритм:** Функция перебирает все комбинации значений переменных. Для каждой комбинации, при которой функция равна 1, формируется конъюнкция переменных, где переменные с нулевым значением инвертируются через ¬. Все такие конъюнкции объединяются через логическое ИЛИ , формируя итоговую СДНФ.

```
string buildSDNF(const vector<string>& x, const vector<int>& F) {
    bool f = false; // первый операнд дизъюнкции
    string result;
    int n = x.size();
    int total = 1 << n; // 2^n

    for (int i = 0; i < total; i++) {
    // СДНФ строится по единицам функции
        if (F[i] == 1) {
            if (f) {
                result += "     ";
            } else {
                f = true; // первое выражение
            }
            bool g = false; // первый операнд конъюнкции
            result += "(";

            for (int j = 0; j < n; j++) {
                if (g) {
                    result += "     ";
                } else {
                    g = true; // первый операнд
                }

                int v = (i >> (n - j - 1)) & 1; // j-й разряд кортежа i

                if (v == 0) {
                    result += "  ";
                }

                result += x[j];
            }

            result += ")";
        }
    }

    return result;
}
```

Листинг 8: Функция buildFromDiagram()

### 2.7.2 Функция buildSKNF()

**Назначение:** Построение СКНФ на основе таблицы истинности.
**Вход:**

- `const vector<string> x` — список переменных булевой функции.
- `const vector<int> F` — вектор значений функции.

**Выход:**

- `string` — строковое представление СКНФ.

**Алгоритм:** Функция перебирает все комбинации значений входных переменных. Для каждой комбинации, при которой функция равна 0, формируется дизъюнкция переменных, где переменные с единичным значением инвертируются через ¬. Все такие дизъюнкции объединяются через логическое И, формируя СКНФ.

```
string buildSKNF(const vector<string>& x, const vector<int>& F) {
    bool f = false; // левый операнд конъюнкции
    string result;
    int n = x.size();
    int total = 1 << n; // 2^n

    for (int i = 0; i < total; i++) {
    // СКНФ строится по нулям функции
        if (F[i] == 0) {
            if (f) {
                result += "      ";
            } else {
                f = true; // первое выражение
            }

            bool g = false; // первый операнд дизъюнкции
            result += "(";

            for (int j = 0; j < n; j++) {
                if (g) {
                    result += "      ";
                } else {
                    g = true;
                }

                int v = (i >> (n - j - 1)) & 1; // j-й разряд кортежа i

                if (v == 1) {
                    result += "  "; // здесь инверсия наоборот
                }

                result += x[j];
```

```
                }

                result += ")";
            }
        }

        return result;
}
```

Листинг 9: Функция buildSKNF(const vector<string> x, const vector<int> F)

# 3 Результаты работы программы

Программа использует консольный интерфейс для работы с булевыми функциями, выполняя все операции через меню и корректно обрабатывая как правильный, так и неверный ввод.

## 3.1 Главное меню программы

При запуске программы пользователю предоставляется главное меню с набором доступных операций (см. рис. 6). Меню включает следующие пункты:

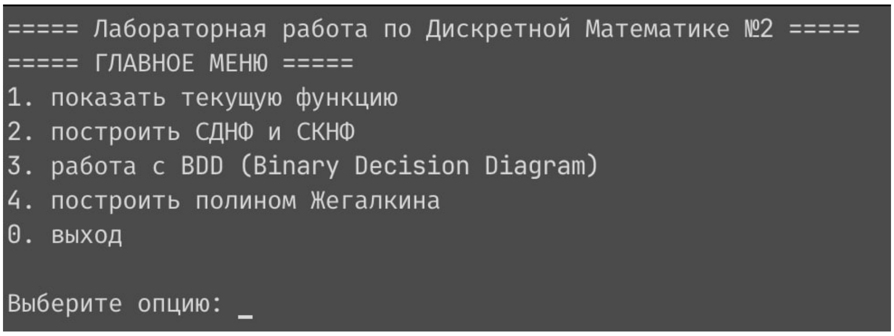



Рис. 6: Главное меню программы

.

## 3.2 Отображение текущей конфигурации булевой функции

При выборе первого пункта меню программа выводит полную информацию о текущей конфигурации булевой функции (см. рис. 7).

```
===== ТЕКУЩАЯ КОНФИГУРАЦИЯ БУЛЕВОЙ ФУНКЦИИ =====
x1 x2 x3 x4 | f
---------------
0  0  0  0  | 1
0  0  0  1  | 1
0  0  1  0  | 0
0  0  1  1  | 0
0  1  0  0  | 1
0  1  0  1  | 1
0  1  1  0  | 0
0  1  1  1  | 0
1  0  0  0  | 1
1  0  0  1  | 0
1  0  1  0  | 0
1  0  1  1  | 1
1  1  0  0  | 0
1  1  0  1  | 1
1  1  1  0  | 1
1  1  1  1  | 1

Переменные: x1, x2, x3, x4
Вектор функции: [1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1]

нажмите любую кнопку для продолжения..._
```

Рис. 7: Текущая конфигурация булевой функции

## 3.3 Построение совершенных нормальных форм

Программа реализует автоматическое построение СДНФ и СКНФ на основе таблицы истинности(см. рис. 8, 9).

```
СДНФ:
(¬x1 ∧ ¬x2 ∧ ¬x3 ∧ ¬x4) ∨ (¬x1 ∧ ¬x2 ∧ ¬x3 ∧ x4) ∨ (¬x1 ∧ x2 ∧ ¬x3 ∧ ¬x4) ∨ (¬x1
∧ x2 ∧ ¬x3 ∧ x4) ∨ (x1 ∧ ¬x2 ∧ ¬x3 ∧ ¬x4) ∨ (x1 ∧ ¬x2 ∧ x3 ∧ x4) ∨ (x1 ∧ x2 ∧ ¬x3
 ∧ x4) ∨ (x1 ∧ x2 ∧ x3 ∧ ¬x4) ∨ (x1 ∧ x2 ∧ x3 ∧ x4)
```

Рис. 8: Совершенная дизъюнктивная нормальная форма

```
СКНФ:
(x1 ∨ x2 ∨ ¬x3 ∨ x4) ∧ (x1 ∨ x2 ∨ ¬x3 ∨ ¬x4) ∧ (x1 ∨ ¬x2 ∨ ¬x3 ∨ x4) ∧ (x1 ∨ ¬x2
∨ ¬x3 ∨ ¬x4) ∧ (¬x1 ∨ x2 ∨ x3 ∨ ¬x4) ∧ (¬x1 ∨ x2 ∨ ¬x3 ∨ x4) ∧ (¬x1 ∨ ¬x2 ∨ x3 ∨
x4)
```

Рис. 9: Совершенная конъюнктивная нормальная форма

## 3.4 Работа с бинарной диаграммой решений

На рис. 10 показана структура построенного BDD графа. Программа выводит информацию о каждом узле графа:

```
=== Структура BDD графа ===
Корневой узел: 7

Узел 0: 0 (лист)
Узел 1: 1 (лист)
Узел 2: x3 -> zero: 1, one: 0
Узел 3: x3 -> zero: 0, one: 1
Узел 4: x4 -> zero: 2, one: 3
Узел 5: x4 -> zero: 3, one: 1
Узел 6: x2 -> zero: 4, one: 5
Узел 7: x1 -> zero: 2, one: 6
===========================
```

Рис. 10: Структура бинарной диаграммы решений

### 3.4.1 Вычисление функции по BDD

На рис. 11 продемонстрирован процесс вычисления значения функции для конкретного набора входных переменных:

```
=== Вычисление по BDD ===
Введите значения переменных (0 или 1):
x1 = 1
x2 = 1
x3 = 0
x4 = 1

=== Результат вычисления ===
Входные значения: x1=1, x2=1, x3=0, x4=1

Путь по графу:
  1. Узел 7 (x1 = 1) -> (сплошная)
  2. Узел 6 (x2 = 1) -> (сплошная)
  3. Узел 5 (x4 = 1) -> (сплошная)
  4. Достигнут лист: 1

Итоговый результат: 1
```

Рис. 11: Вычисление значения функции по BDD

## 3.5 Построение полинома

Программа реализует построение полинома Жегалкина методом треугольника Паскаля (см. рис. 12).

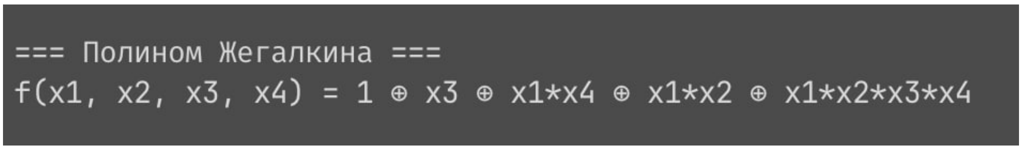
```
=== Треугольник (метод Паскаля) ===
1   1   0   0   1   1   0   0   1   0   0   1   0   1   1   1
  0   1   0   1   0   1   0   1   1   0   1   1   1   0   0
    1   1   1   1   1   1   1   0   1   1   0   0   1   0
      0   0   0   0   0   0   1   1   0   1   0   1   1
        0   0   0   0   0   1   0   1   1   1   1   0
          0   0   0   0   1   1   1   0   0   0   1
            0   0   0   1   0   0   1   0   0   1
              0   0   1   1   0   1   1   0   1
                0   1   0   1   1   0   1   1
                  1   1   1   0   1   1   0
                    0   0   1   1   0   1
                      0   1   0   1   1
                        1   1   1   0
                          0   0   1
                            0   1
                              1

Левая колонка треугольника - коэффициенты полинома Жегалкина!
```

Рис. 12: Треугольник для построения полинома Жегалкина

### 3.5.1 Результирующий полином

На основе коэффициентов из левого столбца треугольника формируется итоговый полином (см. рис. 13).

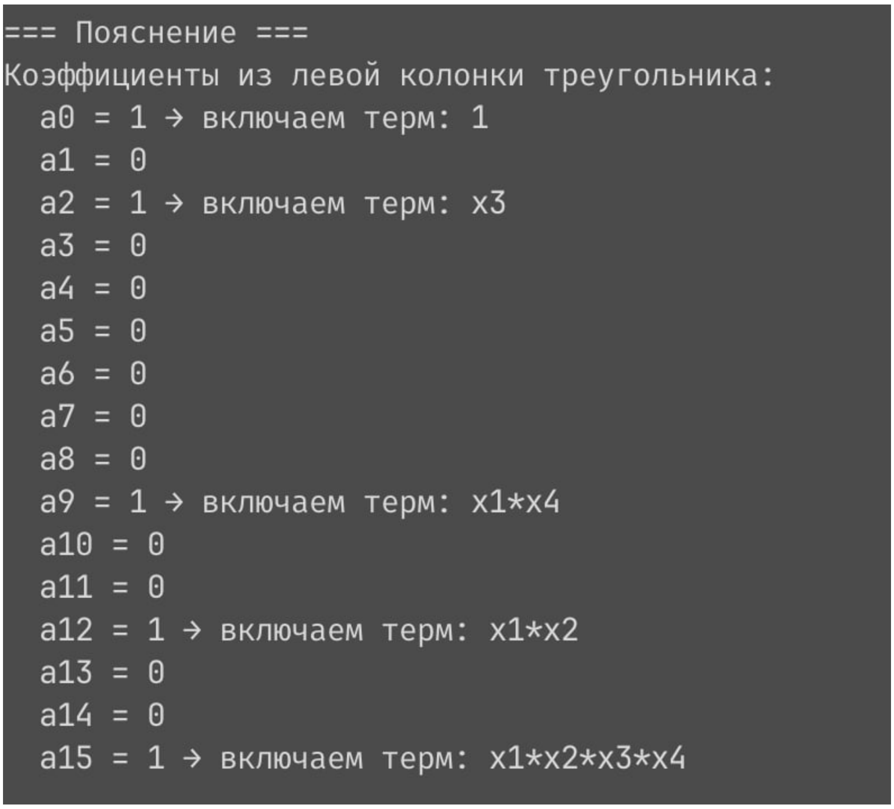


```
=== Полином Жегалкина ===
f(x1, x2, x3, x4) = 1 ⊕ x3 ⊕ x1*x4 ⊕ x1*x2 ⊕ x1*x2*x3*x4
```

Рис. 13: Полином Жегалкина

### 3.5.2 Описание коэффициентов

Для лучшего понимания процесса построения программа выводит подробную информацию о коэффициентах (см. рис. 14). Для каждого коэффициента указывается его значение и соответствующий ему терм.



```
=== Пояснение ===
Коэффициенты из левой колонки треугольника:
  a0 = 1 → включаем терм: 1
  a1 = 0
  a2 = 1 → включаем терм: x3
  a3 = 0
  a4 = 0
  a5 = 0
  a6 = 0
  a7 = 0
  a8 = 0
  a9 = 1 → включаем терм: x1*x4
  a10 = 0
  a11 = 0
  a12 = 1 → включаем терм: x1*x2
  a13 = 0
  a14 = 0
  a15 = 1 → включаем терм: x1*x2*x3*x4
```

Рис. 14: Детализация коэффициентов полинома Жегалкина

## 3.6 Обработка некорректного ввода

Программа реализует обработку некорректного пользовательского ввода (см. рис. 15, 16)

```
===== Лабораторная работа по Дискретной Математике №2 =====
===== ГЛАВНОЕ МЕНЮ =====
1. показать текущую функцию
2. построить СДНФ и СКНФ
3. работа с BDD (Binary Decision Diagram)
4. построить полином Жегалкина
0. выход

Выберите опцию: 234

Ошибка! Неверный выбор. Попробуйте снова.

нажмите любую кнопку для продолжения..._
```

Рис. 15: Некорректный ввод в главном меню

```
=== Вычисление по BDD ===
Введите значения переменных (0 или 1):
x1 = 25
Ошибка! Введите 0 или 1.
```

Рис. 16: Некорректный ввод значения переменных

# Заключение

В ходе лабораторной работы была реализована обработка булевой функции как вручную, так и программно. В расчётной части построены таблица истинности, семантическое дерево и наиболее компактная БДР, по которой получена формула ДНФ и построено синтаксическое дерево. Были вычислены первые производные по переменным и написана формула четвертой производной. Минимальная ДНФ использована для построения логической схемы на элементах И, ИЛИ, НЕ. Также была реализована программа для работы с булевыми функциями.

## Основные результаты

В программной части автоматизировано построение СДНФ и СКНФ по заданной таблице истинности, реализовано хранение и вычисление значений по БДР, а также построение полинома Жегалкина.

Реализованные алгоритмы показали корректность работы для различных входных данных и позволяют работать с булевыми функции.

## Достоинства реализации

Достоинством сделанной программы является то, что код структурирован по отдельным файлам, каждый из которых отвечает за определённый участок функционала, что обеспечивает модульность системы. Дополнительным преимуществом является то, что работа с графами реализована наглядно и просто: у каждого узла есть свой номер, имя переменной и переходы в зависимости от значения. Это делает структуру графа понятной. Узлы связаны между собой так, что можно легко проследить путь от корня до конечного результата, а если нужно, можно расширять или изменять граф

## Недостатки реализации

Пользователь не может самостоятельно менять порядок переменных в BDD, что ограничивает гибкость работы с диаграммой. Также проограмма способна работать с пятью и более переменными, однако из-за того, что граф задаётся вручную непосредственно в коде, нет возможности просто добавить новые переменные и автоматически получить корректный BDD. Каждый раз требуется вручную изменять структуру графа, что усложняет масштабирование и не позволяет гибко расширять систему.

## Возможности масштабирования и улучшения

Все операции выполняются через консольный интерфейс, что ограничивает наглядность работы с бинарными диаграммами решений, нормальными формами и полиномом Жегалкина. Визуализация сложных графов была бы удобнее через GUI при помощи фреймворка Qt. Также можно добавить обработку единичной и нулевой функции, чтобы программа корректно формировала СДНФ, СКНФ и BDD даже в случаях, когда функция всегда равна 1 или всегда равна 0.

# Список литературы


[1] Сайт кафедры с учебными материалами по курсу *«Дискретная математика»*. Ссылка: https://tema.spbstu.ru/dismath/ (Дата обращения: 30.11.2025).

[2] Новиков Ф.А. *Дискретная математика*. Учебник. Ссылка на PDF: https://stugum.wordpress.com/wp-content/uploads/2014/03/novikov.pdf (Дата обращения: 30.11.2025).

[3] Итмо Вики. *ДНФ*. Ссылка: https://neerc.ifmo.ru/wiki/index.php?title=ДНФ (Дата обращения: 30.11.2025).

[4] Итмо Вики. *КНФ*. Ссылка: https://neerc.ifmo.ru/wiki/index.php?title=КНФ (Дата обращения: 30.11.2025).

[5] Википедия. *Производная булевой функции*. Ссылка: ru.wikipedia.org/wiki/Производная_булевой_функции (Дата обращения: 30.11.2025).

[6] Digital Library *Dynamic variable ordering for ordered binary decision diagrams*. Ссылка: https://dl.acm.org/doi/10.5555/259794.259802