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

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 Математика и компьютерные науки

Отчет о выполнении лабораторной работы №2
«Реализация булевой функции.»
Дисциплина «Дискретная математика»
Вариант 33

Выполнил студент группы
№5130201/30001 _____ Мелещенко С.И.

Проверил _____ Востров А. В.

Санкт-Петербург, 2024

Содержание

Введение

Данный отчет содержит в себе описание выполнения лабораторной работы №2 «Работа с булевой функцией. Построение таблицы истинности, дерева решений и бинарной диаграммы решений, синтаксического дерева для полинома Жегалкина. Построение СДНФ и СКНФ. Программное вычисление значений булевой функции по СДНФ, полиному Жегалкина и БДР.»

Была задана булева функция 4-х переменных 48045.

Было необходимо:

1) Построить таблицу истинности и по ней дерево решении и бинарную диаграмму решении, а также синтаксическое дерево для полинома Жегалкина (вручную). Бинарная диаграмма решений должна быть наиболее компактной. Реализовать программно хранение полученной бинарной диаграммы решений и вычисление по ней значения (по пользовательскому вводу значения переменных).

2) По таблице истинности программно построить СДНФ и СКНФ. Вычислить по СДНФ значение булевой функции (по пользовательскому вводу). Сверить полученные значения (в п. 1 и в п.2) с исходной таблицей истинности (автоматически).

3) Для заданной функции построить программно полином Жегалкина. Вывести его на экран и вычислить значение булевой функции согласно пользовательскому вводу.

Программа была реализована на языке программирования C++ в среде программирования Microsoft Visual Studio.

1 Математическое описание

1.1 Булева функция

Функции

$$f\colon E_2^n \to E_2, \text{где } E_2 \stackrel{\text{Def}}{=} \{0,1\}$$

называются функциями алгебры логики, или булевыми функциями от n переменных. Элементы множества E_2 называются истинностными значениями. Множество булевых функций от n переменных обозначим P_n.

$$P_n \stackrel{\text{Def}}{=} \{f \mid f\colon E_2^n \to E_2\}.$$

Булева функция (функция алгебры логики), функция, аргументы которой, равно как и сама функция, принимают значения из двухэлементного множества (обычно из множества 0,1). Булевы функции являются объектами дискретной математики, особенно часто они используются в математической логике, математической кибернетике и в технике

1.2 Таблица истинности

Булеву функцию от n переменных можно задать таблицей истинности:

x_1	$\dots$	x_{n-1}	x_n	$f(x_1,\dots,x_n)$
0	$\dots$	0	0	$f(0,\dots,0,0)$
0	$\dots$	0	1	$f(0,\dots,0,1)$
0	$\dots$	1	0	$f(0,\dots,1,0)$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
1	$\dots$	1	1	$f(1,\dots,1,1)$

Если число переменных равно n, то в таблице истинности имеется 2^n строк, соответствующих всех различным комбинациям значений переменных. Следовательно, существует 2^{2^n} различных столбцов, каждый из которых определяет булеву функцию от n переменных.

По условию было дано, что f = 48045.
$48045_{10} = 1011101110101101_2$
$2^n = 16 => n = 4$, т. е. функция 4-х переменных.
Составим таблицу истинности (см. [Рис. 1]).

	a	b	c	d	f
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	1	0	1
4	0	0	1	1	1
5	0	1	0	0	1
6	0	1	0	1	0
7	0	1	1	0	1
8	0	1	1	1	1
9	1	0	0	0	1
10	1	0	0	1	0
11	1	0	1	0	1
12	1	0	1	1	0
13	1	1	0	0	1
14	1	1	0	1	1
15	1	1	1	0	0
16	1	1	1	1	1

Рис. 1: Таблица истинности

1.3 Совершенная дизъюнктивная нормальная форма

Простой конъюнкцией или конъюнктом называется конъюнкция одной или нескольких переменных или их отрицаний, причём каждая переменная встречается не более одного раза.

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

Совершенная дизъюнктивная нормальная форма, СДНФ — ДНФ, удовлетворяющая условиям:

1) в ней нет одинаковых простых конъюнкций;

2) каждая простая конъюнкция полная.

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

Составим для данный функции по таблице совершенную дизъюнктивную нормальную форму.

СДНФ f = $\bar{a}\bar{b}\bar{c}\bar{d} \vee \bar{a}\bar{b}cd \vee \bar{a}b\bar{c}d \vee \bar{a}bc\bar{d} \vee \bar{a}bcd \vee a\bar{b}\bar{c}\bar{d} \vee a\bar{b}cd \vee ab\bar{c}\bar{d} \vee ab\bar{c}d \vee abcd$

1.4 Совершенная конъюнктивная нормальная форма

Простой дизъюнкцией или дизъюнктом называется дизъюнкция одной или нескольких переменных или их отрицаний, причём каждая переменная встречается не более одного раза.

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

Совершенная конъюнктивная нормальная форма, СКНФ — это такая КНФ, которая удовлетворяет условиям:

1) в ней нет одинаковых простых дизъюнкций;

2) каждая простая дизъюнкция полная.

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

Составим для данный функции по таблице совершенную конъюнктивную нормальную форму.

СКНФ f $= \neg(\bar{a}\bar{b}\bar{c}d \vee \bar{a}b\bar{c}d \vee a\bar{b}\bar{c}d \vee a\bar{b}cd \vee abc\bar{d}) = (a \vee b \vee c \vee \bar{d})(a \vee \bar{b} \vee c \vee \bar{d})(\bar{a} \vee b \vee c \vee \bar{d})(\bar{a} \vee b\bar{c} \vee \bar{d})(\bar{a} \vee \bar{b} \vee \bar{c} \vee d)$

1.5 Полином Жегалкина

Полином Жегалкина — полином с коэффициентами вида 0 и 1 , где в качестве произведения берётся конъюнкция, а в качестве сложения исключающее или. Полином был предложен в 1927 году И. И. Жегалкиным в качестве средства для представления функций булевой логики. Полином Жегалкина имеет следующий вид:

$$P = a \oplus \bigoplus_{1 \le i_1 < \ldots < i_k \le n;\ k \in \overline{1,n}} a_{i_1},\ldots,i_k \vee x_{i_1} \vee \ldots \vee x_{i_k}, a, a_{i_1,\ldots,i_k} \in 0,1$$

По теореме Поста, чтобы система булевых функций была полной, надо, чтобы в ней существовали:

1) хотя бы одна функция, не сохраняющая 0 ;

2) хотя бы одна функция, не сохраняющая 1 ;

3) хотя бы одна нелинейная функция;

4) хотя бы одна немонотонная функция;

5) хотя бы одна несамодвойственная функция.

Исходя из этого, система функций $\langle \wedge, \oplus, 1 \rangle$ является полной:

x_0	x_1	...	x_n	1	$\wedge$	$\oplus$
0	0	...	0	1	0	0
1	0	...	0	1	0	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	...	1	1	1	0
Сохраняет 0				0	1	1
Сохраняет 1				1	1	0
Самодвойственная				0	0	0
Монотонная				1	1	0
Линейная				1	0	1

На основе этой системы и строятся полиномы Жегалкина.

Построим поленом Жегалкина по заданной функции методом треугольника (см. [Рис. 2]).

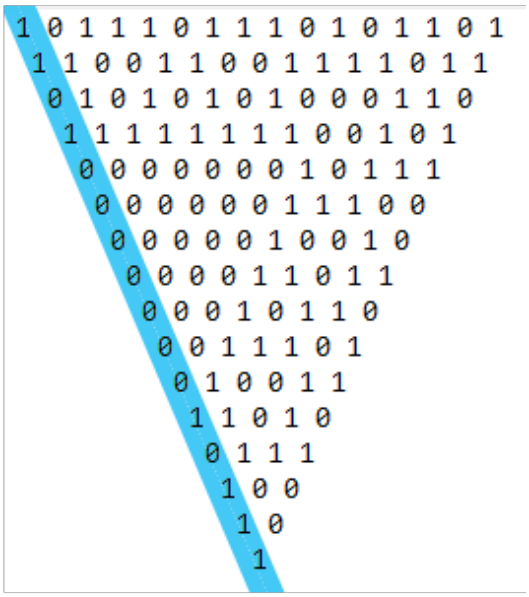

Рис. 2: Метод треугольника

Таким образом, f(a,b,c,d) $= 1 \oplus d \oplus cd \oplus acd \oplus abd \oplus abc \oplus abcd$

1.6 Семантическое дерево решений

Наиболее эффективными с точки зрения экономии памяти и времени оказываются представления, которые не имеют прямой связи с "естественными"представлениями функции в виде графика или формулы, но специально ориентированы на выполнение операций.

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. Листья на последнем ярусе хранят значение функции на кортеже, соответствующем пути из корня в этот лист. Такое дерево называется деревом решений (или семантическим деревом).

Составим дерево для нашей функции (см. [Рис. 3]).

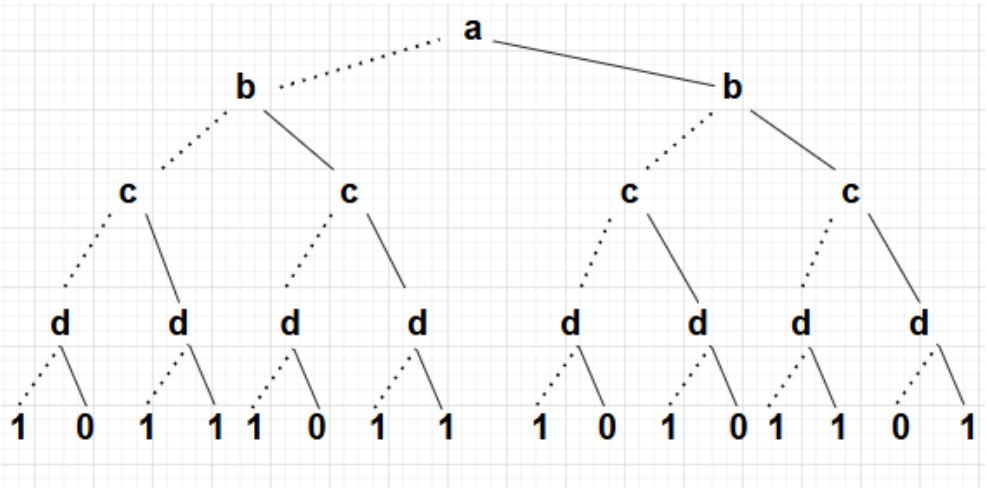

Рис. 3: Семантическое дерево

Дерево можно сократить (см. [Рис. 4], [Рис. 5]).

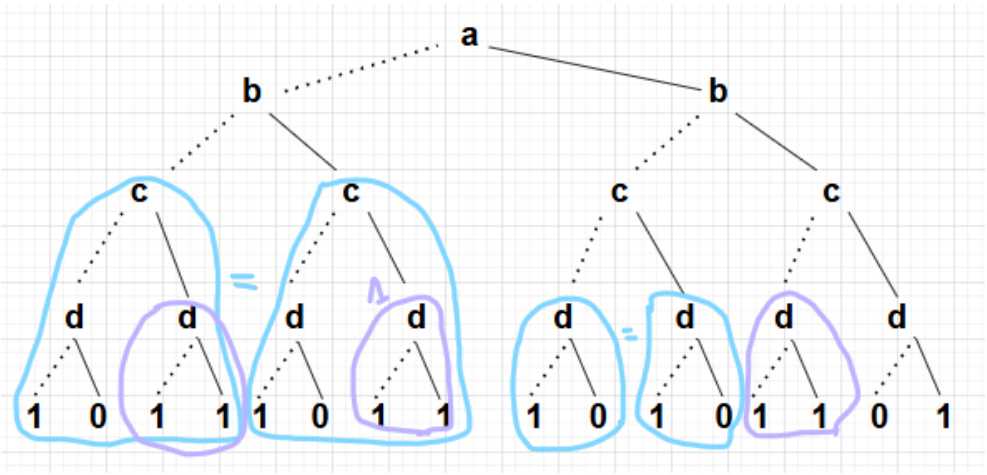

Рис. 4: Сокращение дерева

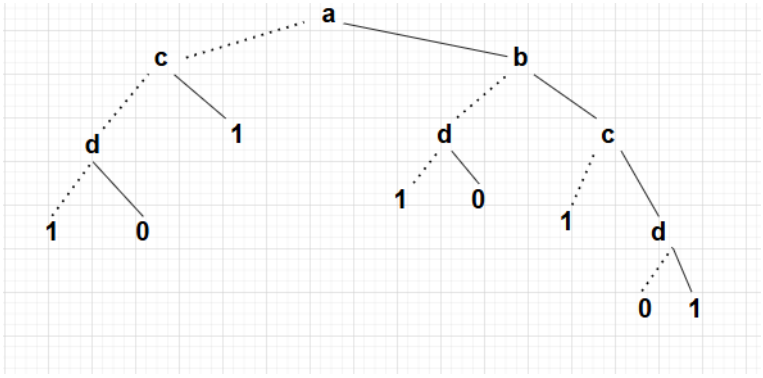

Рис. 5: Сокращенный вариант дерева

1.7 Бинарная диаграмма решений

Дерево решений можно сделать еще компактнее, если отказаться от древовидности связей, то есть допускать несколько дуг, входящих в узел. В таком случае мы получаем бинарную диаграмму решений. Бинарная диаграмма решений получается из бинарного дерева решений тремя последовательными преобразованиями:

1) отождествляются листовые узлы, содержащие 0 и содержащие 1;

2) в диаграмме выделяются изоморфные поддиаграммы и заменяются единственным их экземпляром;

3) исключаются узлы, обе исходящие дуги которых ведут в один узел.

Ниже приведена БДР для заданной функции (см. [Рис. 6]).

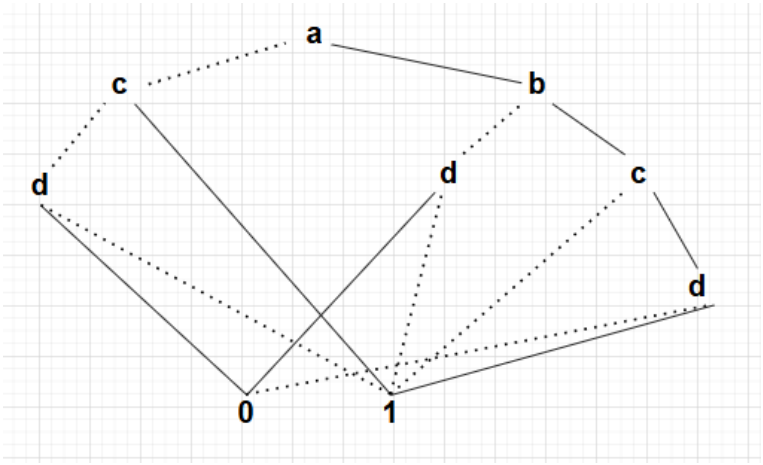

Рис. 6: Бинарная диаграмма решений

1.8 Синтаксическое дерево решений

Синтаксическое дерево (дерево операций) - это структура, представляющая собой результат работы синтаксического анализатора. Она отражает синтаксис конструкций входного языка и явно содержит в себе полную взаимосвязь операций.

В синтаксическом дереве внутренние узлы (вершины) соответствуют операциям, а листья представляют собой операнды. Как правило, листья синтаксического дерева связаны с записями в таблице идентификаторов. Структура синтаксического дерева отражает синтаксис языка программирования, на котором написана исходная программа.

Ниже представлено синтаксическое дерево построенное по полиному Жегалкина (см. [Рис. 7]).

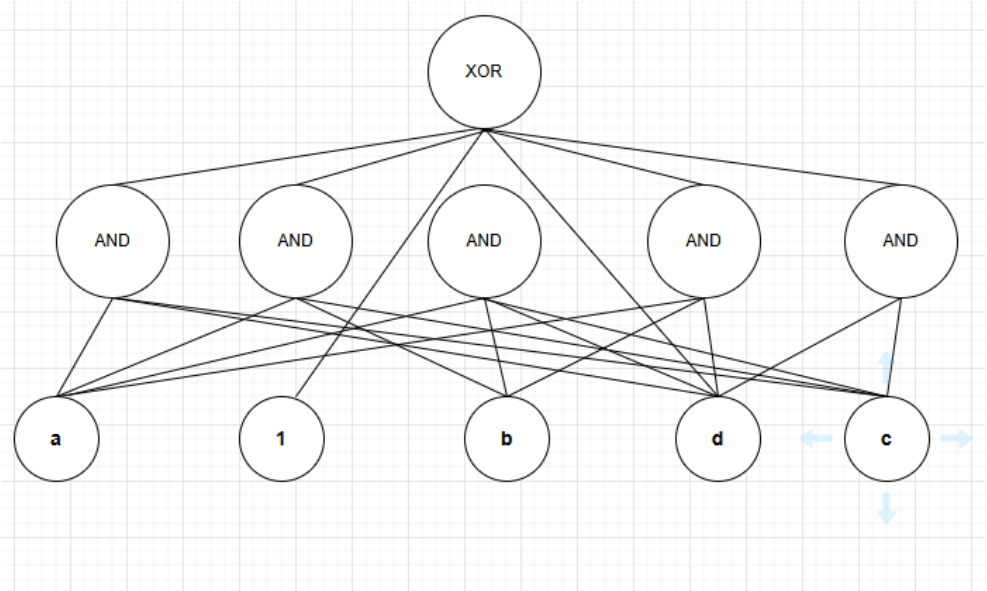

Рис. 7: Синтаксическое дерево решений

2 Особенности реализации

2.1 Библиотечные классы

iostream — ввод и вывод данных через стандартные потоки (std::cin, std::cout, std::cerr).

iomanip — форматирование вывода (ширина, точность, выравнивание).

map — ассоциативный контейнер для хранения пар "ключ-значение"с упорядоченными ключами.

vector — динамический массив с поддержкой автоматического изменения размера.

variant — контейнер для хранения значения одного из нескольких типов.

string — работа с текстовыми строками, поддержка операций над строками.

limits — информация о границах числовых типов (min, max, точность).

2.2 Класс Bin

2.2.1 Поля класса

map<vector<int>, int> table — хранит таблицу истинности, где ключ — набор значений переменных (vector<int>), а значение (int) — результат функции для этого набора.

vector<int> f — хранит значения функции для всех строк таблицы истинности.

string sdnf — хранит строковое представление совершенной дизъюнктивной нормальной формы (СДНФ).

string polynomial — хранит строковое представление полинома Жегалкина.

string sknf — хранит строковое представление совершенной конъюнктивной нормальной формы (СКНФ).

2.2.2 Конструктор

GreyCode() — вызывает метод для генерации кодов Грея, которые использованы для создания таблицы истинности.

fillf() — этот метод заполняет таблицу истинности.

SDNF() — вызывает метод для вычисления совершенной дизъюнктивной нормальной формы (СДНФ).

SKNF() — вызывает метод для вычисления совершенной конъюнктивной нормальной формы (СКНФ).

PZhegalkina() — вызывает метод для вычисления полинома Жегалкина.

Листинг 1: Bin::Bin()

```
GreyCode();
fillf();
SDNF();
SKNF();
PZhegalkina();
```

2.2.3 Методы бинарного кода Грея

Вход: таблица истинности table.

Выход: заполненная таблица истинности table.

GreyCode() — Генерирует коды Грея для 4-битных чисел (от 0000 до 1111). Каждый код Грея отличается от предыдущего только одним битом. Для каждого кода создается запись в таблице, где ключом является вектор, представляющий код Грея, а значением — 0 (это начальная кратность).

Листинг 2: vector<vector<int» Bin::GreyCode()

```
vector<int> B(4, 0);
    table[B] = 0;
    for (int i = 1; i < 16; ++i) {
        int elnum = Q(i);
        B[elnum - 1] = 1 - B[elnum - 1];
        table[B] = 0;
    }
    return {};
```

Вход: int i бит числа.

Выход: q позиция первого ненулевого бита.

Q(int i) — Рассчитывает порядковый номер бита, который должен измениться при генерации следующего кода Грея для числа i. Это значение указывает, какой бит следует инвертировать в процессе генерации.

Листинг 3: int Bin::Q(int i)

```
int q = 1;
    while (i % 2 == 0) {
        i /= 2;
        q++;
    }
    return q;
```

2.2.4 Метод fillf()

Вход: число 48045 - значения функции.

Выход: vector<int> f двоичное представление числа 48045.

Метод fillf() преобразует 48045 в двоичное представление.

Листинг 4: vector<int> Bin::fillf()

```
    int number = 48045;
    for (int i = 15; i >= 0; --i) {
        f.push_back((number & (1 << i)) ? 1 : 0);
    }
    return f;
```

2.2.5 Метод SDNF()

Вход: таблица истинности table, значения функции f.

Выход: sdnf по таблице истинности.

Метод SDNF() создает строку, представляющую СДНФ на основе значений, хранящихся в table и f. Если соответствующее значение функции f[index] равно 1, то создается слагаемое СДНФ, представляющее логическую дизъюнкцию для этих переменных.

Для каждой переменной (от a до d) проверяется, равен ли соответствующий бит 1 или 0. Если бит равен 1, то добавляется сама переменная (например, a). Если бит равен 0, добавляется отрицание переменной (например, !a). Между переменными используется логическое "И".

После формирования части добавляется символ OR, который соединяет слагаемые.

Также последний OR удаляется.

Листинг 5: void Bin::SDNF()

```
int index = 0;
    for (const auto& pair : table) {
        if (f[index] == 1) {
            sdnf += "(";
            for (size_t i = 0; i < pair.first.size(); ++i) {
                char variable = 'a' + i;
                if (pair.first[i] == 1) {
                    sdnf += variable + string(" ");
                }
                else {
                    sdnf += "!" + string(1, variable) + " ";
                }
                if (i < pair.first.size() - 1) {
                    sdnf += "AND ";
                }
            }
            sdnf += ") OR ";
```

```
        }
        index++;
    }
        sdnf = sdnf.substr(0, sdnf.size() - 4);
```

2.2.6 Метод SKNF()

Вход: таблица истинности table, значения функции f.

Выход: sknf по таблице истинности.

Метод SKNF() генерирует строку, представляющую СКНФ. Если соответствующее значение функции f[index] равно 0, то создается слагаемое для СКНФ, представляющее логическую конъюнкцию для этих переменных.

Для каждой переменной проверяется, равен ли соответствующий бит 0 или 1. Если бит равен 0, то добавляется сама переменная (например, a). Если бит равен 1, добавляется отрицание переменной (например, !a). Между переменными используется логическое "ИЛИ".

После формирования части добавляется логическое "И соединяющее множетели.

После формирования всей строки СКНФ, в конце удаляется лишний символ AND.

Листинг 6: void Bin::SKNF()

```
    int index = 0;
    for (const auto& pair : table) {
        if (f[index] == 0) {
            sknf += "(";
            for (size_t i = 0; i < pair.first.size(); ++i) {
                char variable = 'a' + i;
                if (pair.first[i] == 0) {
                    sknf += variable + string(" ");
                }
                else {
                    sknf += "!" + string(1, variable) + " ";
                }
                if (i < pair.first.size() - 1) {
                    sknf += "OR ";
                }
            }
            sknf += ") AND ";
        }
        index++;
    }
        sknf = sknf.substr(0, sknf.size() - 5);
```

2.2.7 Метод PZhegalkina()

Вход: значения функции f.

Выход: polynomial полином Жегалкина.

Вектор coeff копирует значения функции f. Внешний цикл проходит по каждому индексу вектора f. Внутри каждого шага создаётся новый вектор temp, в котором значения элементов coeff получаются через операцию XOR между соседними элементами. Таким образом, для каждой пары соседних значений выполняется операция XOR.

Вектор results содержит результат вычислений для каждого шага в цикле XOR. Если результат на текущем шаге равен 1, то для данного индекса добавляется соответствующий член в полином.

Если индекс равен 0, добавляется свободный член 1. Если индекс больше 0, проверяется, какие биты установлены в числе i (от старших к младшим). Это позволяет добавить соответствующие переменные в полином.

Добавляются операторы + между членами полинома.

Листинг 7: void Bin::PZhegalkina()

```
 vector<int> coeff = f;
vector<int> results;

    for (size_t i = 0; i < f.size(); ++i) {
        results.push_back(coeff[0]);
        vector<int> temp;
        for (size_t j = 0; j < coeff.size() - 1; ++j) {
            temp.push_back(coeff[j] ^ coeff[j + 1]);
        }
        coeff = temp;
    }

    for (size_t i = 0; i < results.size(); ++i) {
        if (results[i] == 1) {
            if (!polynomial.empty()) polynomial += " + ";
            if (i == 0) {
                polynomial += "1";
            }
            else {
                for (int j = 3; j >= 0; --j) {
                    if ((i >> j) & 1) {
                        polynomial += string(1, 'a' + (3 - j));
                    }
                }
            }
        }
```

```
}
```

2.2.8 Метод Operations(int op)

Вход: int op пользовательский ввод номера операции.

Выход: номер функции, к которой обращаемся.

Метод либо выводит нужную пользователю информацию, либо вызывает нужный метод благодаря конструкции switch.

Листинг 8: void Bin::Operations(int op)

```
switch (op)
{
case 0:
    break;
case 1:
    printTable();
    break;
case 2:
    cout << sdnf << endl;
    break;
case 3:
    cout << sknf << endl;
    break;
case 4:
    cout << polynomial << endl;
    break;
case 5:
    evaluateSDNF();
    break;
case 6:
   evaluatePolynomial();
    break;
case 7:
    evaluateBDRFromInput();
    break;
}
```

2.2.9 printTable()

Вход: f вектор значений функции, table таблица истинности.

Выход: вывод таблицы истинности.

Этот метод перебирает все элементы в таблице и выводит для каждой комбинации входных переменных соответствующее значение функции (из вектора f).

Листинг 9: void Bin::printTable()

```
 int index = 0;
     cout << "a b c d    f«< endl;for (const auto pair : table) for (const auto elem : pair.first) cout « elem « ;cout «  «< f[index];cout « endl;index++;cout « endl;
```

2.2.10 Метод checkBinaryInput(string& input)

Вход: string& input строка для пользовательского ввода.

Выход: true при корректном вводе.

Метод checkBinaryInput проверяет корректность пользовательского ввода, который должен быть строкой из ровно 4 битов (только символы 0 и 1). Если ввод некорректен, программа просит повторить попытку.

Листинг 10: bool Bin::checkBinaryInput(string input)

```
while (true) {
        cout << "Введите 4 бита (например, 1010): ";
        cin >> input;
        if (input.length() != 4) {
            cout << "Ошибка! Введите ровно 4 бита." << endl;
            continue;
        }
        bool isValid = true;
        for (char c : input) {
            if (c != '0' && c != '1') {
                isValid = false;
                break;
            }
        }
        if (isValid) {
            return true;
        }
        else {
            cout << "Ошибка! Ввод должен содержать только 0 и
1." << endl;
        }
    }
```

2.2.11 check(string vars)

Вход: string vars - пользовательский ввод.

Выход: строка с ответом по таблице.

Функция Bin::check проверяет значение булевой функции, определенной в таблице истинности, для заданных переменных. Она берет строку из четырёх бит (vars), вычисляет индекс строки в таблице истинности, а затем выводит соответствующее значение функции из вектора f.

Листинг 11: void Bin::check(string vars)

```
int index = 0;
    for (int i = 0; i < 4; ++i) {
        index = (vars[i] - '0') « (3 - i);if (index >= 0  index < 16) if (f[index]
== 1) cout « "Ответ по таблице (проверка): 1«< endl;else cout « "Ответ по
таблице (проверка): 0«< endl;
```

2.2.12 evaluateSDNF()

Вход: vars - пользовательский ввод, sdnf.

Выход: вывод result.

Функция Bin::evaluateSDNF вычисляет значение булевой функции на основе заданной пользователем строки из четырех бит (vars) и представления функции в виде сокращенной дизъюнктивной нормальной формы (СДНФ). Затем она сверяет результат с таблицей истинности.

Каждый терм (выражение в скобках) извлекается из строки СДНФ. Терм разбивается на отдельные литералы, все литералы внутри терма соединены через логическое И, и результат вычисляется с использованием логического И. Результаты всех термов соединяются через логическое ИЛИ, чтобы получить итоговый результат СДНФ.

Листинг 12: void Bin::evaluateSDNF()

```
    string vars;
    if (!checkBinaryInput(vars)) {
        return;
    }
    map<char, int> varValues;
    for (size_t i = 0; i < vars.size(); ++i) {
        char var = 'a' + i;
        varValues[var] = vars[i] == '1' ? 1 : 0;
    }
    string sdnfNoSpaces = "";
    for (char c : sdnf) {
        if (c != ' ') {
```

```
            sdnfNoSpaces += c;
        }
    }
    bool result = false;
    size_t pos = 0;

    while (pos < sdnfNoSpaces.size()) {
        size_t start = sdnfNoSpaces.find("(", pos);
        size_t end = sdnfNoSpaces.find(")", pos);
        if (start == string::npos  end == string::npos) break;

string term = sdnfNoSpaces.substr(start + 1, end - start - 1);

        bool termResult = true;
        size_t andPos = 0;

        while (andPos < term.size()) {
            size_t nextAnd = term.find("AND", andPos);
string literal = (nextAnd == string::npos) ? term.substr(andPos) :
term.substr(andPos, nextAnd - andPos);

            if (literal[0] == '!') {
                char var = literal[1];
                if (varValues.find(var) == varValues.end()) {
cerr << "Ошибка: Переменная " << var << " не найдена!" << endl;
                    return;
                }
                termResult &= (varValues[var] == 0);
            }
            else {
                char var = literal[0];
                if (varValues.find(var) == varValues.end()) {
cerr << "Ошибка: Переменная " << var << " не найдена!" << endl;
                    return;
                }
                termResult &= (varValues[var] == 1);
            }

andPos = (nextAnd == string::npos) ? term.size() : nextAnd + 3;
        }

        result = termResult;pos = end + 1;cout « "Ответ по СДНФ: «<
result « endl;check(vars);
```

2.2.13 evaluatePolynomial()

Вход: polynomial, string vars - пользовательский ввод.

Выход: вывод результата.

Метод вычисляет значение булевой функции, представленное полиномом Жегалкина, на основе пользовательского ввода. Затем сверяет результат с таблицей истинности.

Термы в полиноме Жегалкина отделены "+ если терм = a*b, то termValue = varValues['a'] & varValues['b']. Результат вычисляется с использованием операции XOR.

Листинг 13: void Bin::evaluatePolynomial()

```
    string vars;
    if (!checkBinaryInput(vars)) {
        return;
    }

    map<char, int> varValues;
    for (size_t i = 0; i < vars.size(); ++i) {
        char var = 'a' + i;
        varValues[var] = vars[i] == '1' ? 1 : 0;
    }

    int result = 0;
    size_t pos = 0;

    while (pos < polynomial.size()) {
        size_t termEnd = polynomial.find(" + ", pos);
if (termEnd == string::npos) termEnd = polynomial.size();

string term = polynomial.substr(pos, termEnd - pos);
        int termValue = 1;

        for (char c : term) {
            if (isalpha(c)) {
                termValue &= varValues[c];
            }
        }

        result ^= termValue;
        pos = termEnd + 3;
    }

cout << "Ответ по полиному Жегалкина: " << result << endl;
```

```
check(vars);
```

2.2.14 evaluateBDRFromInput()

Вход: string input пользовательский ввод.

Выход: вывод ответа.

Метод для вычисления значения булевой функции с использованием бинарной диаграммы решений (БДР) на основе пользовательского ввода. После вычисления результат сверяется с таблицей истинности.

Пользоватльский ввод разбивается на a,b,c,d, создается БДР, получаем результат.

Листинг 14: void Bin::evaluateBDRFromInput()

```
string input;
if (!checkBinaryInput(input)) {
    return;
}

bool a = input[0] == '1';
bool b = input[1] == '1';
bool c = input[2] == '1';
bool d = input[3] == '1';

auto root = createBDR();
int result = evaluateBDR(root, a, b, c, d);

cout << "Ответ по БДР: "<< result << endl;
check(input);
```

2.3 Функции

2.3.1 createBDR()

Структура BDRNode представляет узел бинарной диаграммы решений:
variable - имя переменной.
zero - указатель на следующий узел для значения переменной 0.
one - указатель на следующий узел для значения переменной 1.
value - значение узла (0 или 1).

Листинг 15: Структура BDRNode

```
std::string variable;
std::shared_ptr<BDRNode> zero;
std::shared_ptr<BDRNode> one;
```

```
int value = -1;
```

Ввод: структура BDRNode.

Вывод: root.

Функция создаёт диаграмму решений вручную, задавая связи между узлами. Диаграмма соответствует булевой функции с упрощением до минимальной структуры.

Для начала создаются листы, затем узлы уровня d, c, b, a. Диаграмма начинается с корневого узла root, который ссылается на следующие узлы в зависимости от значений переменных a, b, c, и d.

Листинг 16: std::shared_ptr<BDRNode> inline createBDR()

```
    auto leaf1 = std::make_shared<BDRNode>();
    leaf1->value = 1;

    auto leaf0 = std::make_shared<BDRNode>();
    leaf0->value = 0;

    auto node_d1 = std::make_shared<BDRNode>();
    node_d1->variable = "d";
    node_d1->zero = leaf1;
    node_d1->one = leaf0;

    auto node_d2 = std::make_shared<BDRNode>();
    node_d2->variable = "d";
    node_d2->zero = leaf0;
    node_d2->one = leaf1;

    auto node_c1 = std::make_shared<BDRNode>();
    node_c1->variable = "c";
    node_c1->zero = node_d1;
    node_c1->one = leaf1;

    auto node_c2 = std::make_shared<BDRNode>();
    node_c2->variable = "c";
    node_c2->zero = leaf1;
    node_c2->one = node_d2;

    auto node_b1 = std::make_shared<BDRNode>();
    node_b1->variable = "b";
    node_b1->zero = node_d1;
    node_b1->one = node_c2;

    auto root = std::make_shared<BDRNode>();
```

```
    root->variable = "a";
    root->zero = node_c1;
    root->one = node_b1;

    return root;
```

2.3.2 evaluateBDR()

Вход: структура <BDRNode>, bool a, bool b, bool c, bool d - переменные булевой функции.

Выход: значении функции.

Функция evaluateBDR вычисляет значение булевой функции, закодированной в бинарной диаграмме решений, используя заданные значения переменных (a, b, c, d). Она рекурсивно проходит дерево, начиная с корневого узла, до тех пор, пока не достигнет листа. Лист содержит конечное значение функции: 0 или 1.

Листинг 17: const std::shared_ptr<BDRNode>& node bool a bool b bool c bool d

```
        if (node->value != -1) {
                return node->value;
        }
        if (node->variable == "a") {
return evaluateBDR(a ? node->one : node->zero, a, b, c, d);
        }
        else if (node->variable == "b") {
return evaluateBDR(b ? node->one : node->zero, a, b, c, d);
        }
        else if (node->variable == "c") {
return evaluateBDR(c ? node->one : node->zero, a, b, c, d);
        }
        else if (node->variable == "d") {
return evaluateBDR(d ? node->one : node->zero, a, b, c, d);
        }
```

2.3.3 void Start()

Выводит приветственное окно.

Вход: параметры инициализации консоли.

Выход: вывод текста.

Листинг 18: void Start()

```
    cout << endl << "********************" << endl
```

```
        << "* ДОБРО ПОЖАЛОВАТЬ! *" << endl <<
        "**********************" << endl;
```

2.3.4 void Menu()

Выводит заголовок меню.
Вход: параметры инициализации консоли.
Выход: вывод текста.

Листинг 19: void Menu()

```
 cout << endl << endl << endl << "***********"
 << endl
     << "* МЕНЮ *" << endl <<
         "***********";
```

2.3.5 int OperationsMenu()

Выводит меню. Инициализирует 2 переменные int op - для ввода номера операции, bool validInput = false - для обработки некорректного ввода (меньше 0 больше 7). Возращает op и передает в метод класса.
Вход: op - выбор номера действия.
Выход: вывод текста, op.

Листинг 20: int OperationsMenu()

```
    cout << endl << "1) Вывести таблицу истинности" << endl
        << "2) СДНФ f" << endl
        << "3) СКНФ f" << endl
        << "4) Полином Жегалкина f" << endl << endl
        << "5) Вычислить по СДНФ значение булевой функции"
        << endl
        << "6) Вычисление значения булевой функции по полиному
Жегалкина " << endl
        << "7) Вычисление значения булевой функции по БДР"
        << endl << endl;

    int op;
    bool validInput = false;
    while (!validInput) {
        cout << "Ввод: ";
        cin >> op;
        if (cin.fail()  op < 0  op > 7) {
            cin.clear();
```

```
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Ошибка ввода! Введите число от 0 до 7
            включительно!" << endl;
        }
        else if (op == 0) {
            return 0;
        }
        else return op;
    }
```

2.3.6 void Bye()

Выводит сообщение "До новых встреч!"
Вход: 0, пользовательский ввод для выхода из приложения.
Выход: вывод текста.

Листинг 21: void Bye()

```
cout << endl << "********************" << endl
   << "* ДО НОВЫХ ВСТРЕЧ! *" << endl <<
    "********************" << endl << endl;
```

2.4 Порядок вызова в Main.cpp

Для начала выводим приветственное окно, затем создаётся объект bin класса Bin. Выводим заголовок меню и запускаем цикл while. В начале каждого итерации вызывается функция OperationsMenu(), которая отображает меню операций и запрашивает у пользователя выбор. Значение выбора записывается в переменную tmp. Затем вызывается метод Operations(tmp) объекта bin, который выполняет соответствующую операцию на основе выбора пользователя. Если пользователь ввёл 0, программа завершает цикл с помощью break.

Листинг 22: int Continue()

```
{
        setlocale(LC_ALL, "Russian");

        Start();
        Bin bin;
        int tmp = 1;
        Menu();

        while (tmp) {
                tmp = OperationsMenu();
                bin.Operations(tmp);
```

```
            if (tmp == 0) break;
        };
        Bye();
        return 0;
```

3 Результаты работы программы

При запуске приложения мы видим меню (см. [Рис. 8]).

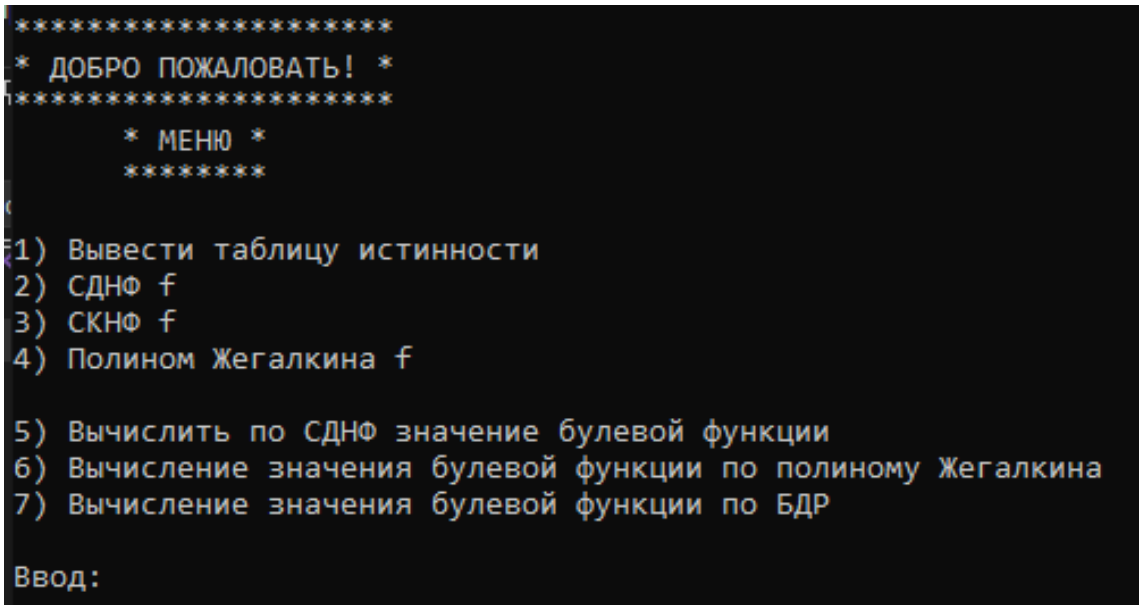

Рис. 8

Вывод СДНФ (см. [Рис. 9]).

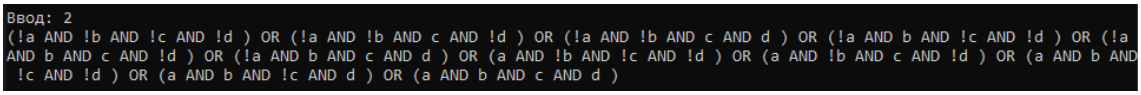

Рис. 9

Вывод СКНФ (см. [Рис. 10]).

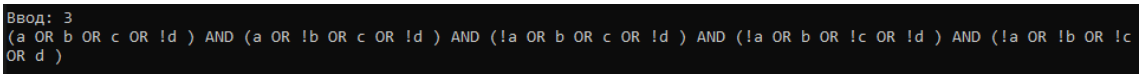

Рис. 10

Вывод полинома Жегалкина (см. [Рис. 11]).

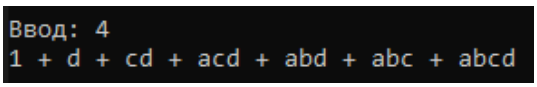

Рис. 11

Вывод таблицы истинности (см. [Рис. 12]).

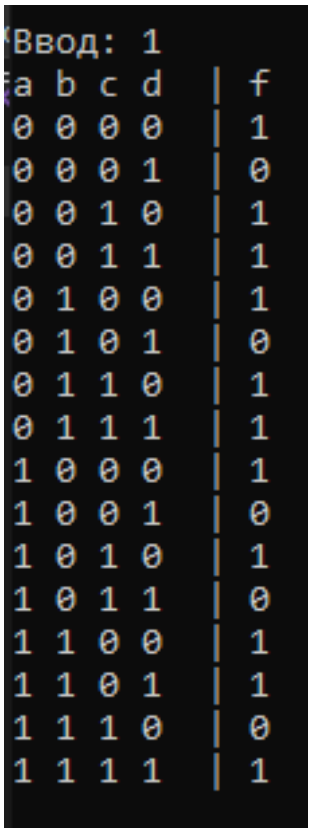

Рис. 12

При некорректном вводе в режиме меню - ошибка ввода. От пользователя будет требоваться ввод до тех пор пока он верно не введет двоичное четырехзначное число (см. [Рис. 13]).

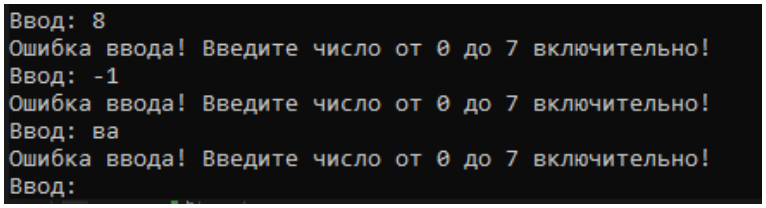

Рис. 13

Вычисление по СДНФ значения булевой функции (см. [Рис. 14]).

```
Ввод: 5
Введите 4 бита (например, 1010): 0011
Ответ по СДНФ: 1
Ответ по таблице (проверка): 1
```

Рис. 14

Вычисление по полиному Жегалкина значения булевой функции (см. [Рис. 15]).

```
Ввод: 6
Введите 4 бита (например, 1010): 1111
Ответ по полиному Жегалкина: 1
Ответ по таблице (проверка): 1
```

Рис. 15

Вычисление по БДР значения булевой функции (см. [Рис. 16]).

```
Ввод: 7
Введите 4 бита (например, 1010): 1010
Ответ по БДР: 1
Ответ по таблице (проверка): 1
```

Рис. 16

При некорректном вводе для метода - ошибка (см. [Рис. 17]).

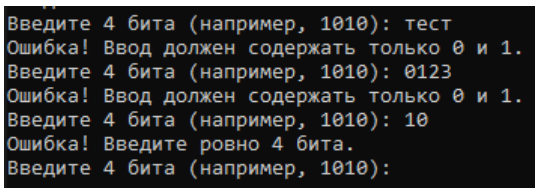

Рис. 17

При вводе 0 для завершения работы приложения выводится прощальное окно (см. [Рис. 18]).

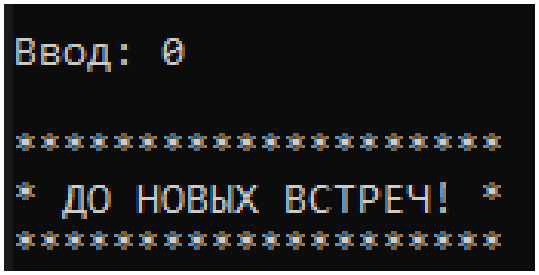

Рис. 18

Заключение

В результате работы была вручную составлена таблица истинности для заданной функции, построено дерево решений, синтаксическое дерево. Была создана бинарная диаграмма решений. Полученная диаграмма реализована программно, добавлена возможность вычислять значение функции, используя пользовательский ввод. Построена СДНФ и СКНФ. Была реализована функция вычисления значения булевой функции по СДНФ на основе пользовательского ввода. Результаты вычислений были сверены с исходной таблицей истинности.

Для функции программно был построен полином Жегалкина. Была реализована функция, которая позволяла вычислять значение булевой функции по полиному Жегалкина на основе пользовательского ввода.

Присутствует огранчение на вводимую разрядность для вычисления функции по СДНФ, полиному Жегалкина, БДР, что обусловлено работой с булевой функция 4-х переменных.

Недостатком данной программы является то, что в СДНФ и полиноме Жегалкина операция поиска и обработки термов осуществляется напрямую, что может быть неэффективным при больших объемах данных. В методах вычислений используется ручной анализ строк, что приводит к увеличению сложности. Это не самый оптимальный вариант.

Преимуществом программы является то, что применяются умные указатели, обеспечивающие автоматическое управление памятью, контейнеры (map, vector) упрощают дальнейшую модернизацию кода. Проверка на корректность ввода вынесена отдельным методом.

Дополнительно может быть реализован вывод БДР. Может быть расширен функционал, можно давать пользователю возможность выбрать количество переменных (от 1 до 10) для булевой функции и вектор значений функции.

Список литературы

[1] Понятие мультимножества. Код Грея. Ф. А. Новиков. *Дискретная математика для программистов.*. СПб : Питер Пресс.

[2] Булева функция.

// URL: https://bigenc.ru/c/buleva-funktsiia-71ad40

(дата обращения: 08.12.2024)

[3] ДНФ, КНФ.

//URL: https://neerc.ifmo.ru/wiki/

(дата обращения: 08.12.2024)

[4] Синтаксические деревья.

//URL: https://studfile.net/preview/7048602/page:45/

(дата обращения: 08.12.2024)