

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

Дискретная математика
ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

Исследование двоичных функций

Вариант 11

Студент,
группы 5130201/30002

_____ Михайлова А. А.

Преподаватель

_____ Востров А. В.

«_____» _____ 2024 г.

Санкт-Петербург, 2024

Содержание

Введение	3
1 Математическое описание	4
1.1 Булева функция	4
1.2 Таблица истинности	4
1.3 Деревья решений	5
1.4 Совершенная дизъюнктивная нормальная форма	7
1.5 Совершенная конъюнктивная нормальная форма	8
1.6 Полином Жегалкина	8
2 Реализация	9
2.1 Структуры данных	9
2.2 Функция printtable	9
2.3 Бинарное дерево решений	10
2.4 Функции getTermSDNF и getTermSKNF	11
2.5 Функции buildSDNF и buildSKNF	12
2.6 Функции evaluateSDNF и evaluateSKNF	13
2.7 Функция traverse	14
2.8 Функция evaluateTree	15
2.9 Функция polynom	15
2.10 Функция printpolynom	16
2.11 Функция checkpolynom	16
2.12 Функция getInputVector	17
2.13 Функция check	18
2.14 Функция main	19
3 Результаты работы программы	22
Заключение	24
Список использованной литературы	25

Введение

Задана булева функция 4-х переменных (порядок определен по возрастанию элементов).

Лабораторная работа представляет собой реализацию следующих пунктов:

1. Вручную построить таблицу истинности и по ней дерево решений и бинарную диаграмму решений, а также синтаксическое дерево для полинома Жегалкина. 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.

Исходная функция: $f = 44011$

Лабораторная работа выполнена на языке C++ в среде разработки XCode.

1 Математическое описание

1.1 Булева функция

Булева функция $f(x_1, x_2, \dots, x_n)$ — это любая функция от n переменных x_1, x_2, \dots, x_n , в которой её аргументы принимают одно из двух значений: либо 0, либо 1, и сама функция принимает значения 0 или 1.

Каждая булева функция может быть представлена в виде таблицы истинности, которая показывает выходные значения для всех возможных комбинаций входных переменных.

Исходную функцию запишем в виде:

$$f(a, b, c, d) = (1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1).$$

1.2 Таблица истинности

Таблица истинности — это таблица, описывающая логическую функцию, а именно отражающая все значения функции при всех возможных значениях её аргументов. Таблица состоит из $n + 1$ столбцов и 2^n строк, где n — число используемых переменных. В первых n столбцах записываются все возможные значения аргументов (переменных) функции, а в $n + 1$ -ом столбце записываются значения функции, которые она принимает на данном наборе аргументов.

Таблицы истинности помогают при проверке правильности логических функций.

Таблица истинности для нашей функции представлена в Таблице 1:

Таблица 1: Таблица истинности

a	b	c	d	$f(a, b, c, d)$
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

1.3 Деревья решений

Дерево решений - это графическое представление логической функции, которое показывает, как значения переменных влияют на её выходное значение (0 или 1). Пунктирная линия означает логический ноль, сплошная линия - логическую единицу. Каждый лист может принимать значение некоторой константы из множества $\{0, 1\}$. Дерево решений для исходной булевой функции показано на рисунке 1.

Если ветки дерева решений ведут к одному и тому же значению функции, его можно упростить. Упрощенное дерево решений для функции показано на рисунке 2

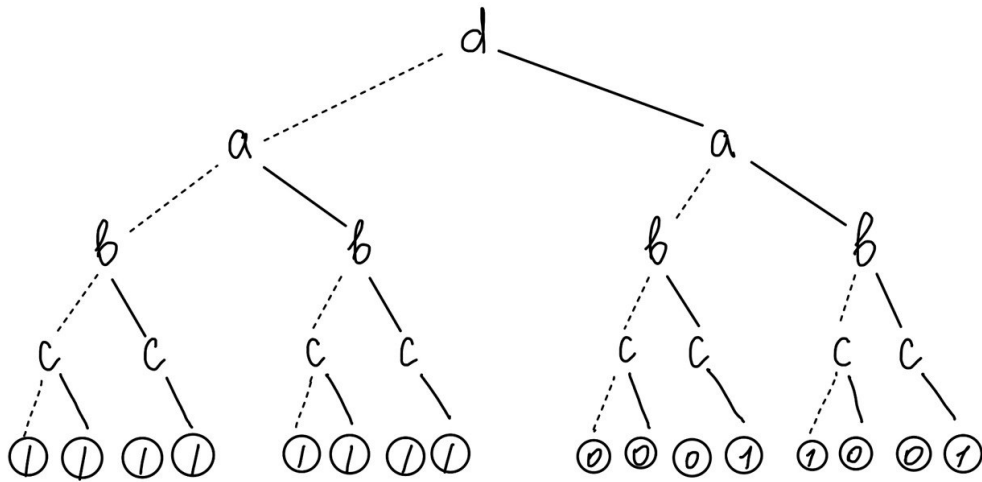


Рис. 1: Дерево решений

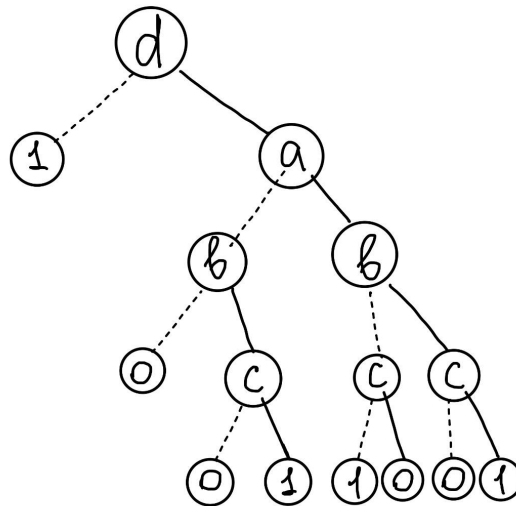


Рис. 2: Упрощённое дерево решений

Если отказаться от древовидности связей, можно получить бинарную диаграмму решений, проделав следующие шаги:

1. Отождествляются листовые узлы, содержащие 0 и содержащие 1.
2. В диаграмме выделяются изоморфные поддиаграммы и заменяются их единственным их экземпляром.
3. Исключаются узлы, обе исходящие дуги которых ведут в один узел.

Бинарная диаграмма решений для исходной булевой функции представлена на рисунке 3.

Синтаксическое дерево для минимальной формы показано на рис 4. Формула, по которой построено синтаксическое дерево:

$$f(a, b, c, d) = \bar{d} \vee dbc \vee dab\bar{c}$$

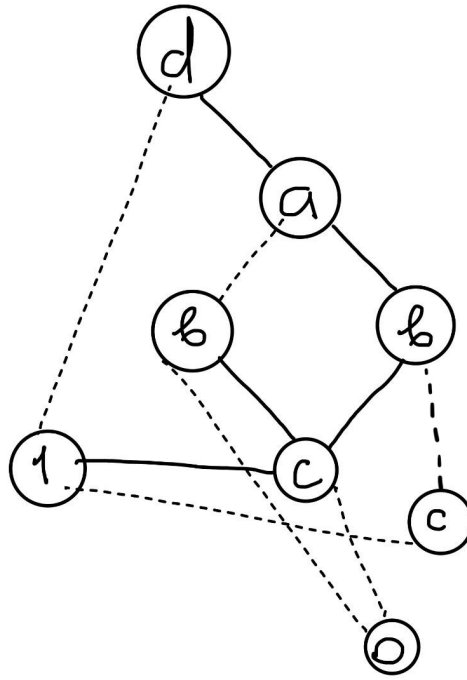


Рис. 3: Бинарная диаграмма решений

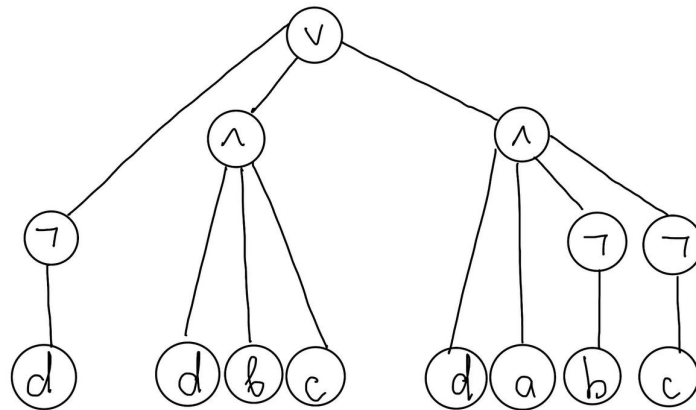


Рис. 4: Синтаксическое дерево

1.4 Совершенная дизъюнктивная нормальная форма

Совершенная дизъюнктивная нормальная форма — это реализация функции $f(x_1, \dots, x_n)$ в виде формулы:

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

где σ_i - значение переменной x_i (0 или 1).

СДНФ может быть легко преобразована из таблицы истинности, так как строки таблицы, соответствующие выходу 1, становятся конъюнкциями.

С помощью теорем упрощения можно оптимизировать СДНФ, уменьшая количество переменных или логических операций.

Если при записи СДНФ используется установленный порядок, то СДНФ однозначно определяет множество наборов значений переменных, на которых функция, реализуемая СДНФ, принимает значение 1. Значит, СДНФ однозначно определяет таблицу истинности реализуемой функции.

СДНФ для заданной функции:

$$\text{СДНФ } f = (\bar{a}\bar{b}\bar{c}\bar{d}) \vee (\bar{a}\bar{b}c\bar{d}) \vee (\bar{a}b\bar{c}\bar{d}) \vee (\bar{a}bcd) \vee (a\bar{b}\bar{c}\bar{d}) \vee (a\bar{b}c\bar{d}) \vee (ab\bar{c}\bar{d}) \vee (abcd)$$

1.5 Совершенная конъюнктивная нормальная форма

Совершенная конъюнктивная нормальная форма — это реализация функции $f(x_1, \dots, x_n)$ в виде формулы:

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

где σ_i - значение переменной x_i (0 или 1) и f^* - функция, которая принимает значение 1 только для тех комбинаций переменных, при которых исходная функция f принимает значение 0.

СКНФ является двойственной функцией для СДНФ, а значит на нее действуют все те же свойства, что и для СДНФ. СКНФ для заданной функции:

$$\text{СКНФ } f = (a \vee b \vee c \vee \bar{d})(a \vee b \vee \bar{c} \vee \bar{d})(a \vee \bar{b} \vee c \vee \bar{d})(\bar{a} \vee b \vee \bar{c} \vee \bar{d})(\bar{a} \vee \bar{b} \vee c \vee \bar{d})$$

1.6 Полином Жегалкина

Представление булевой функции над базисом $\{0, 1, \wedge, +\}$ называется полиномом Жегалкина. Всякая функция представима в виде:

$$P(x_1, \dots, x_n) = a \oplus a_1x_1 \oplus \dots \oplus a_nx_n \oplus a_{12}x_1x_2 \oplus \dots \oplus a_{1\dots n}x_1\dots x_n$$

Полином Жегалкина для нашей функции:

$$f(a, b, c, d) = 1 + d + bcd + ad + acd + abd + abcd$$

2 Реализация

2.1 Структуры данных

Вся программа реализована в файле `main.cpp` с использованием библиотеки `vector`.

1. $vector<int> a = \{0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1\}$ - хранит значения переменной `a` в таблице истинности;
2. $vector<int> b = \{0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1\}$ - хранит значения переменной `b` в таблице истинности;
3. $vector<int> c = \{0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1\}$ - хранит значения переменной `c` в таблице истинности;
4. $vector<int> d = \{0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1\}$ - хранит значения переменной `d` в таблице истинности;
5. $vector<int> f = \{1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1\}$ - хранит значения переменной `f` в таблице истинности;
6. $vector<int> value$ - хранит значения `a`, `b`, `c`, `d`, введенные пользователем;
7. $vector<vector<int>> SDNF$ - хранит термы СДНФ;
8. $vector<vector<int>> SKNF$ - хранит термы СКНФ;
9. $int resbdr$ - хранит результаты значения БДР;
10. $int cp$ - хранить результаты значения полинома;
11. $int eSDNF$ - хранит результаты значения СДНФ;
12. $int eSKNF$ - хранит результаты значения СКНФ;

2.2 Функция `printtable`

Вход: значения `a`, `b`, `c`, `d`, `f`

Выход: таблица истинности

Функция `printtable` выводит на консоль таблицу истинности. Ознакомиться с кодом можно в Листинг 1.

Листинг 1: Функция `printtable`

```
void printtable(vector<int>& a, vector<int> &b, vector<int> &c,  
               vector<int> &d, vector<int> &f){
```

```

        cout << "Таблица_истинности" << '\n';
        cout << "a" << "\t" << "b" << "\t" << "c" << "\t" << "d"
<< "\t" << "f" << '\n';
        for(int i = 0; i < a.size(); i++){
            cout << a[i] << "\t" << b[i] << "\t" << c[i] << "\t" << d[i] << "\t" << f[i] << '\n';
        }
    }
}

```

2.3 Бинарное дерево решений

Класс Usel представляет собой узел бинарного дерева. Функция bdr поэтапно строит бинарное дерево, добавляя левое и правое поддеревья для каждого узла. Ознакомиться с кодом можно в Листинг 2.

Листинг 2: Бинарное дерево решений

```

struct Usel{
    char name;
    int val;
    Usel* left, *right;
    Usel(const char& n){
        name = n;
        left = nullptr;
        right = nullptr;
        val = -1;
    }
};

Usel* root = nullptr;

void bdr(){
    root = new Usel('d');

    root->left = new Usel('n');
    root->left->val = 1;

    root->right = new Usel('a');

    root->right->left = new Usel('b');

    root->right->left->left = new Usel('n');
    root->right->left->left->val = 0;
}

```

```

root->right->left->right = new Usel('c');

root->right->left->right->left = new Usel('n');
root->right->left->right->left->val = 0;

root->right->left->right->right = new Usel('n');
root->right->left->right->right->val = 1;

root->right->right = new Usel('b');

root->right->right->left = new Usel('c');

root->right->right->left->left = new Usel('n');
root->right->right->left->left->val = 1;

root->right->right->left->right = new Usel('n');
root->right->right->left->right->val = 0;

root->right->right->right = new Usel('c');

root->right->right->right->right = new Usel('n');
root->right->right->right->right->val = 1;

root->right->right->right->left = new Usel('n');
root->right->right->right->left->val = 0;
}

```

2.4 Функции `getTermSDNF` и `getTermSKNF`

Вход: вектор значений a b c d

Выход: терм СДНФ или СКНФ

Функции `getTermSDNF` и `getTermSKNF` предназначены для генерации строк, представляющих термы СДНФ и СКНФ на основе заданных значений. Ознакомиться с кодом можно в Листинг 3.

Листинг 3: Функции `getTermSDNF` и `getTermSKNF`

```

string getTermSDNF(const vector<int>& values) {
    string term = "(";
    char variables[] = {'a', 'b', 'c', 'd'};

    for (size_t i = 0; i < values.size(); ++i) {
        if (values[i] == 1) {
            term += variables[i];

```

```

        } else {
            term += "!" + string(1, variables[i]);
        }
    }
    term += ")";
    return term;
}

string getTermSKNF(const vector<int>& values) {
    string term = "(";
    char variables[] = {'a', 'b', 'c', 'd'};

    for (size_t i = 0; i < values.size(); ++i) {
        if (values[i] == 1) {
            term += variables[i];
        } else {
            term += "!" + string(1, variables[i]);
        }
        if (i < values.size() - 1) {
            term += "\/";
        }
    }
    term += ")";
    return term;
}

```

2.5 Функции buildSDNF и buildSKNF

Вход: значения a, b, c, d, f

Выход: СДНФ или СКНФ

Функции buildSDNF и buildSKNF соединяют термы СДНФ(СКНФ) и выводят на экран. Ознакомиться с кодом можно в Листинг 4.

Листинг 4: Функции buildSDNF и buildSKNF

```

vector<vector<int>> buildSDNF(const vector<int>& a, const vector<
    int>& b, const vector<int>& c, const vector<int>& d, const
    vector<int>& f) {
    cout << "СДНФf:\n";
    string sdnf = "";
    for (size_t i = 0; i < f.size(); ++i) {
        if (f[i] == 1) {
            if (!sdnf.empty()) {
                sdnf += "\/";
            }
        }
    }
}

```

```

        sdnf += getTermSDNF({a[i], b[i], c[i], d[i]
    ]});
    }
}
cout << sdnf << endl;

vector<vector<int>> SDNF;
for (int i = 0; i < f.size(); i++) {
    if (f[i] == 1) {
        vector<int> row = {a[i], b[i], c[i], d[i]
    ]};
        SDNF.push_back(row);
    }
}
return SDNF;
}
vector<vector<int>> buildSKNF(const vector<int>& a, const vector<
int>& b, const vector<int>& c, const vector<int>& d, const
vector<int>& f) {
    cout << "CKHΦ1f:\n";
    string sknf = "";
    for (size_t i = 0; i < f.size(); ++i) {
        if (f[i] == 0) {
            sknf += getTermSKNF({1 - a[i], 1 - b[i], 1
- c[i], 1 - d[i]});
        }
    }
    cout << sknf << endl;
    vector<vector<int>> SKNF;
    for (int i = 0; i < f.size(); i++) {
        if (f[i] == 0) {
            vector<int> row = {a[i], b[i], c[i], d[i]
    ]};
            SKNF.push_back(row);
        }
    }
    return SKNF;
}

```

2.6 Функции evaluateSDNF и evaluateSKNF

Вход: СДНФ(СКНФ), значения a, b, c, d введённые пользователем
 Выход: значение СДНФ (СКНФ)

Функции evaluateSDNF и evaluateSKNF подставляют значения a, b, c, d в СНДФ (СКНФ) и возвращают значение. Ознакомиться с кодом можно в Листинг 5.

Листинг 5: Функции evaluateSDNF и evaluateSKNF

```
int evaluateSDNF(const vector<vector<int>>& SDNF, const vector<int>
    & input) {
    for (const vector<int>& term : SDNF) {
        if (term == input) {
            return 1;
        }
    }
    return 0;
}
int evaluateSKNF(const vector<vector<int>>& SKNF, const vector<int>
    & input) {
    for (const vector<int>& term : SKNF) {
        if (term == input) {
            return 0;
        }
    }
    return 1;
}
```

2.7 Функция traverse

Вход: корень дерева, значения, введённые пользователем

Выход: значение узла

Функция traverse осуществляет рекурсивный обход бдр. Ознакомиться с кодом можно в Листинг 6.

Листинг 6: Функция traverse

```
int traverse(Usel* node, const std::vector<int>& values) {
    if (node == nullptr) {
        return -1;
    }
    if (node->name == 'n') {
        return node->val;
    }
    int index = node->name - 'a';
    int value = values[index];
    if (value == 0) {
        return traverse(node->left, values);
    }
}
```

```

    } else {
        return traverse(node->right, values);
    }
}

```

2.8 Функция evaluateTree

Вход: вектор значений, введенных пользователем

Выход: значение узла

Функция evaluateTree предназначена для запуска рекурсивной функции traverse и получения значения узла. Также она проверяет, инициализировано ли дерево. Ознакомиться с кодом можно в Листинг 7.

Листинг 7: Функция evaluateTree

```

int evaluateTree(const std::vector<int>& values) {
    if (root == nullptr) {
        std::cerr << "Error: The tree has not been
initialized.\n";
        return -1;
    }
    return traverse(root, values);
}

```

2.9 Функция polynom

Вход: вектор значений функции, таблица истинности

Выход: коэффициенты полинома

Функция polynom предназначена для преобразования вектора коэффициентов полинома. Ознакомиться с кодом можно в Листинг 8.

Листинг 8: Функция polynom

```

vector<int> polynom(const vector<int>& f, const vector<vector<int>
>>& truthTable) {
    vector<int> coefficients = f;
    for (size_t i = 1; i < coefficients.size(); ++i) {
        for (size_t j = coefficients.size() - 1; j >= i;
--j) {
            coefficients[j] = (coefficients[j] +
coefficients[j - 1]) % 2;
        }
    }

    return coefficients;
}

```

```
}
```

2.10 Функция printpolynom

Вход: коэффициенты полинома, таблица истинности

Выход: полином

Функция printpolynom предназначена для составления полинома из коэффициентов. Ознакомиться с кодом можно в Листинг 9.

Листинг 9: Функция printpolynom

```
string printpolynom(const vector<int>& coefficients, const vector<
    vector<int>>& truthTable) {
    string result;
    bool firstTerm = true;

    for (size_t i = 0; i < coefficients.size(); ++i) {
        if (coefficients[i] == 0) continue;
        string term;
        for (size_t j = 0; j < truthTable[i].size(); ++j)
        {
            if (truthTable[i][j] == 1) {
                term += string(1, 'a' + j);
            }
        }
        if (firstTerm) {
            result += (term.empty() ? "1" : term);
            firstTerm = false;
        } else {
            result += "⊔" + (term.empty() ? "1" :
term);
        }
    }

    return result;
}
```

2.11 Функция checkpolynom

Вход: коэффициенты полинома, таблица истинности, вектор значений a, b, c, d, введенных пользователем

Выход: значение полинома

Функция checkpolynom предназначена для проверки значения полинома по пользовательскому вводу. Ознакомиться с кодом можно в Листинг 10.

Листинг 10: Функция checkpolynom

```
int checkpolynom(const vector<int>& coefficients, const vector<
    vector<int>>& truthTable, const vector<int>& values) {
    int result = 0;

    for (size_t i = 0; i < coefficients.size(); ++i) {
        if (coefficients[i] == 0) continue;
        int termValue = 1;
        for (size_t j = 0; j < truthTable[i].size(); ++j)
        {
            if (truthTable[i][j] == 1) {
                termValue *= values[j];
            }
        }
        result = (result + termValue) % 2;
    }

    return result;
}
```

2.12 Функция getInputVector

Вход: выбор пользователя начать ввод значений a, b, c, d, вводом 'е'

Выход: вектор значений a, b, c, d

Функция getInputVector предназначена для ввода пользователем значений a, b, c, d. Ознакомиться с кодом можно в Листинг 11.

Листинг 11: Функция getInputVector

```
vector<int> getInputVector() {
    vector<int> result;
    int input;
    cout << "Введите 4 числа по одному за раз, каждое из
котрых 0 или 1:" << endl;
    for (int i = 0; i < 4; ++i) {
        while (true) {
            cout << "Введите число #" << (i + 1) << ":
";

            cin >> input;
            if (cin.fail()) {
                cin.clear();
                cin.ignore(numeric_limits<
streamsize>::max(), '\n');
            }
        }
        result.push_back(input);
    }
    return result;
}
```

```

        cout << "Ошибка: Введите целое
число (0 или 1)." << endl;
        continue;
    }
    if (input != 0 && input != 1) {
        cout << "Ошибка: Число должно
быть только 0 или 1. Попробуйте снова." << endl;
        continue;
    }
    result.push_back(input);
    break;
}
}
return result;
}

```

2.13 Функция check

Вход: векторы значений a, b, c, d, вектор значений функции, значения a, b, c, d, введенные пользователем, значения бдр, СДНФ, СКНФ, полинома

Выход: сообщение о проверке значений бдр, СДНФ, СКНФ, полинома со значением функции по пользовательскому вводу

Функция check для проверки значений бдр, СДНФ, СКНФ, полинома со значением функции. Ознакомиться с кодом можно в Листинг 12.

Листинг 12: Функция check

```

void check(vector<int> a, vector<int> b, vector<int> c, vector<int>
> d, vector<int> f, vector<int> value, vector<int> checkres){
    int index =0;
    int numb =0;
    for (int i = 0; i < f.size(); i++)
    {
        if (value [0] == a[i] && value [1] == b[i] &&
value [2] == c[i] &&
        value [3] == d[i])
        {
            index = i;
            break;
        }
    }
    for (int i = 0; i < checkres.size(); i++)
    {
        if (checkres[i] == f[index])

```

```

        numb ++;
    }
    if (numb== checkres.size())
    {
        cout << "Значения_совпадают_с_значением_функции!"
<< endl;
    }
    else
    {
        cout << "Ошибка!" << endl;
    }
}

```

2.14 Функция main

Вход: начало выполнения программы

Выход: результаты программы

Функция реализует логическую обработку булевых функций с использованием четырех логических переменных (a, b, c, d). При запуске она инициализирует значения переменных и создает таблицу истинности, после чего ожидает ввода пользователя для выполнения различных операций. Пользователь может ввести значения переменных для последующего вычисления значений булевой функции по БДР, совершенной дизъюнктивной нормальной форме (СДНФ) и совершенной конъюнктивной нормальной форме (СКНФ). Программа также формирует полином Жегалкина и вычисляет его значение. Результаты вычислений и сообщения о ходе работы выводятся в консоль. Пользователь может выйти из программы, введя соответствующий символ. Ознакомиться с кодом можно в Листинг 13.

Листинг 13: Функция main

```

int main() {
    vector<int> a = {0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
1, 1};
    vector<int> b = {0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
1, 1};
    vector<int> c = {0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
1, 1};
    vector<int> d = {0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
0, 1};
    vector<int> f = {1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0,
1, 1};
    printtable(a, b, c, d, f);
    vector<int> value;
}

```

```

vector<vector<int>> SDNF;
vector<vector<int>> SKNF;
int resbdr;
vector<vector<int>> truthTable(16, vector<int>(4, 0));
for (size_t i = 0; i < 16; ++i) {
    truthTable[i] = {a[i], b[i], c[i], d[i]};
}
vector<int> coefficients;
string zp;
int cp;
vector<int> checkres;
int eSDNF;
int eSKNF;
cout << "Если вы хотите ввести a, b, c, d, введите 'e'\n";
cout << "Если вы хотите выйти из программы, введите 'h'\n";
;
char symbol;
while(true){
    cin >> symbol;
    switch (symbol) {
        case 'e':
            cout << "Введите a, b, c, d\n";
            value = getInputVector();
            bdr();
            resbdr = evaluateTree(value);
            cout << "Значение БДР: " << resbdr << '\n';
;
            SDNF = buildSDNF(a, b, c, d, f);
            eSDNF = evaluateSDNF(SDNF, value);
            cout << "Значение СДНФ: " << eSDNF << '\n';
n';
            SKNF = buildSKNF(a, b, c, d, f);
            eSKNF = evaluateSKNF(SKNF, value);
            cout << "Значение СКНФ: " << eSKNF << '\n';
n';
            coefficients = polynom(f, truthTable);
            zp = printpolynom (coefficients,
truthTable);
            cout << "Полином Жегалкина: " << zp << '\n';
;
            cp = checkpolynom(coefficients, truthTable
, value);
            cout << "Значение полинома: " << cp << '\n';

```

```

';

    checkres.push_back(resbdr);
    checkres.push_back(eSDNF);
    checkres.push_back(eSKNF);
    checkres.push_back(cp);
    check(a, b, c, d, f, value, checkres);
    exit(0);
    break;
    case 'h':
        cout << "Выход_из_программы...\n";
        exit(0);
        break;
    default:
        cout << "Введён_символ_не_из_списка,
попробуйте_снова.\n";
        break;
    }
}
return 0;
}

```

3 Результаты работы программы

После запуска программы выводится таблица истинности и пользователя просят ввести либо 'е' - ввод значений a, b, c, d, либо 'h' - выход из программы (Рисунок 5).

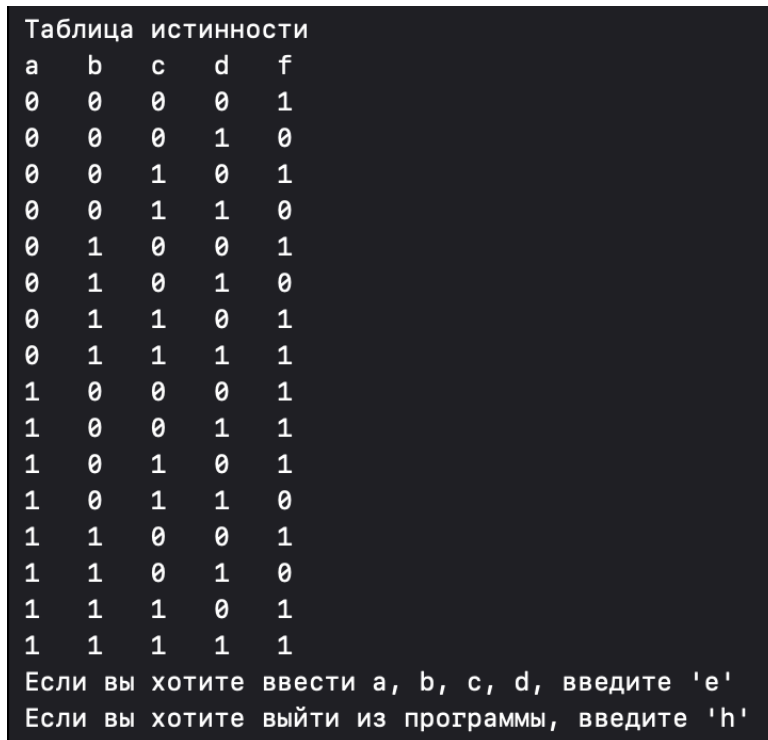


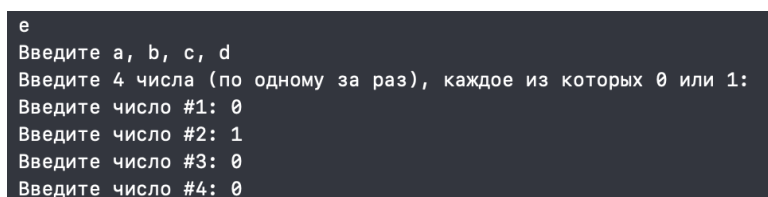
Таблица истинности

a	b	c	d	f
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Если вы хотите ввести a, b, c, d, введите 'е'
Если вы хотите выйти из программы, введите 'h'

Рис. 5: Начальное меню

Если пользователь выбрал 'h' - выход из программы. Если пользователь выбрал 'е' - ему нужно ввести 4 числа - a, b, c, d (Рисунок 6).



е
Введите a, b, c, d
Введите 4 числа (по одному за раз), каждое из которых 0 или 1:
Введите число #1: 0
Введите число #2: 1
Введите число #3: 0
Введите число #4: 0

Рис. 6: Ввод a, b, c, d

```

Значение БДР: 1
СДНФ f:
(!a!b!c!d) v (!a!bc!d) v (!ab!c!d) v (!abc!d) v (!abcd) v (a!b!c!d) v (a!b!cd) v (a!bc!d) v (ab!c!d) v (abc!d) v (abcd)
Значение СДНФ f: 1
СКНФ f:
(a v b v c v !d)(a v b v !c v !d)(a v !b v c v !d)(!a v b v !c v !d)(!a v !b v c v !d)
Значение СКНФ f: 1
Полином Жегалкина: 1 + d + bcd + ad + acd + abd + abcd
Значение полинома: 1
Значения совпадают с значением функции!

```

Несколько вариантов вывода сообщений об ошибке при некорректном вводе:

```
1 Введён символ не из списка, попробуйте снова.
3 Введён символ не из списка, попробуйте снова.
```

```
Введите 4 числа (по одному за раз), каждое из которых 0 или 1:
Введите число #1: 5
Ошибка: Число должно быть только 0 или 1. Попробуйте снова.
Введите число #1: e
Ошибка: Введите целое число (0 или 1).
```

23

Заключение

При выполнении лабораторной работы вручную были построены таблица истинности, дерево решений, бинарная диаграмма решений, синтаксическое дерево для минимальной формы, полином Жегалкин по заданной булевой функции. Программно были построены бинарная диаграмма решений, СДНФ f , СКНФ f , полином Жегалкина.

Достоинства программы:

1. отсутствие утечек памяти из-за использования контейнера STL vector.

Недостатки программы:

1. использование `int`, занимающего 4 байта. Оптимальная замена - тип данных `bool`.

Масштабирование:

1. добавление графического интерфейса;
2. увеличение количества переменных до 5 путём замены бдр и увеличения значений в vector, хранящих значения функции и переменные.

Список литературы

- [1] Новиков, Ф.А. ДИСКРЕТНАЯ МАТЕМАТИКА ДЛЯ ПРОГРАММИСТОВ / Ф.А. Новиков. - 3-е издание. - СПб:Питер Пресс, 2009. - 384 с.