МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт Компьютерных наук и кибербезопастности

Направление 02.03.01 Математика и компьютерные науки

Высшая школа технологий искусственного интеллекта

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ 2 ПО ДИСКРЕТНОЙ МАТЕМАТИКЕ:

Программная реализация алгоритмов работы с булевыми функциями Вариант 30

Обучающийся: _____ Санько В. В.

Руководитель: _____ Востров А. В.

«_____»_____ 20___г.

Санкт-Петербург, 2024

Содержание

Введение **3**

1 Математическое описание **4**

1.1 СДНФ 4

1.2 СКНФ 4

1.3 БДР 4

1.4 Полином Жегалкина 4

1.5 Схемы, деревья, диаграммы 5

2 Особенности реализации **7**

2.1 Структура data и класс BulIst 7

2.2 BulIst() 9

2.3 Вывод таблицы истинности 13

2.4 Полином Жегалкина в реализации 14

2.5 Поиск элемента по Полиному Жегалкина 15

2.6 СКНФ и СДНФ 15

2.7 Поиск элемента по СДНФ 16

2.8 Поиск элемента по БДР 19

2.9 Визуализация БДР 20

2.10 Защита от некорректного ввода 21

2.11 Основное меню 22

3 Результаты работы **24**

Заключение **28**

Список материалов **29**

Введение

Цель данного исследования заключается в разработке программы для эффективной работы с булевыми функциями. Программа включает в себя реализацию основных алгоритмов, таких как вычисление значений функций, представление их в виде СДНФ и СКНФ, в виде Полинома Жегалкина, а также построение Бинарного Дерева Решений (БДР). Также есть возможность найти значение по полиному Жегалкина, БДР и по СДНФ. Номер варианта: 30, функция 0011010000011111.

1 Математическое описание

1.1 СДНФ

СДНФ (Совершенная Дизъюнктивная Нормальная Форма) – способ написания функции алгебры логики в качестве логического выражения. СДНФ представляет собой математическое выражение, описывающее булеву функцию как дизъюнкцию элементарных конъюнкций. К СДНФ возможно привести любую формулу алгебры логики. Исключение составляет только тождественно ложная формула. СДНФ можно получить как используя таблицы истинности, так и через равносильные преобразования.

$$\text{СДНФ}(f) = \bar{x}\bar{y}z\bar{h} \vee \bar{x}\bar{y}zh \vee \bar{x}y\bar{z}h \vee x\bar{y}zh \vee xy\bar{z}\bar{h} \vee xy\bar{z}h \vee xyz\bar{h} \vee xyzh$$

1.2 СКНФ

СКНФ (Совершенная Конъюнктивная Нормальная Форма) – совершенная конъюнктивная нормальная форма. Формулу можно назвать таковой, когда она — конъюнкция неповторяющихся элементарных дизъюнкций.

$$\text{СКНФ}(f) = (x \vee y \vee z \vee h) \wedge (x \vee y \vee z \vee \bar{h}) \wedge (x \vee \bar{y} \vee z \vee h) \wedge (x \vee \bar{y} \vee \bar{z} \vee h) \wedge$$

$$\wedge (x \vee \bar{y} \vee \bar{z} \vee \bar{h}) \wedge (\bar{x} \vee y \vee z \vee h) \wedge (\bar{x} \vee y \vee z \vee \bar{h}) \wedge (\bar{x} \vee y \vee \bar{z} \vee h)$$

1.3 БДР

БДР (Булева Диаграмма Решения) является формой представления булевой функции в виде направленного ациклического графа, состоящего из внутренних узлов решений, каждый из которых имеет по два потомка, и двух терминальных узлов (помеченных 0 и 1), каждый из которых соответствует одному из двух значений булевой функции(см.рис3).

1.4 Полином Жегалкина

Полином Жегалкина — это логическая функция, использующая две операции: конъюнкцию и разделительную дизъюнкцию. Это полином с коэффициентами вида 0 и 1, где в качестве произведения берётся конъюнкция, а в качестве сложения — исключающее или(см.рис5). Общая формула для 4 переменных:

$$P(x,y,z,h) = a_0 + a_1x + a_2y + a_3z + a_4h + a_{12}xy + a_{13}xz + a_{14}xh + a_{23}yz + a_{24}yh + a_{34}zh + a_{123}xyz +$$

$$a_{124}xyh + a_{134}xzh + a_{234}yzh + a_{1234}xyzh$$

Формула для заданной функции:

$$F = z + xy + xz + yz + yh + xyz + xyh + xzh + yzh$$

1.5 Схемы, деревья, диаграммы

Дерево решений — разновидность схемы, где показаны возможные последствия принятия серии связанных между собой решений(см.рис1). Его упрощенной версией называется сокращенное дерево решений (см.рис2) - это дерево решений, которое было оптимизировано с целью уменьшения размера и сложности. Также существует синтаксическое дерево - это прдеставление синтаксической структуры минимизированной СДНФ, узел соответствует оператору или операнду, а листья - переменные выражения(см.рис4). Формула синтаксического дерева:

СДНФ$(f) = \bar{x}\bar{y}z\bar{h} \vee \bar{x}\bar{y}zh \vee \bar{x}y\bar{z}h \vee x\bar{y}zh \vee xy\bar{z}\bar{h} \vee xy\bar{z}h \vee xyz\bar{h} \vee xyzh = \bar{x}\bar{y}z \vee y\bar{z}h \vee xy\bar{h} \vee xzh$

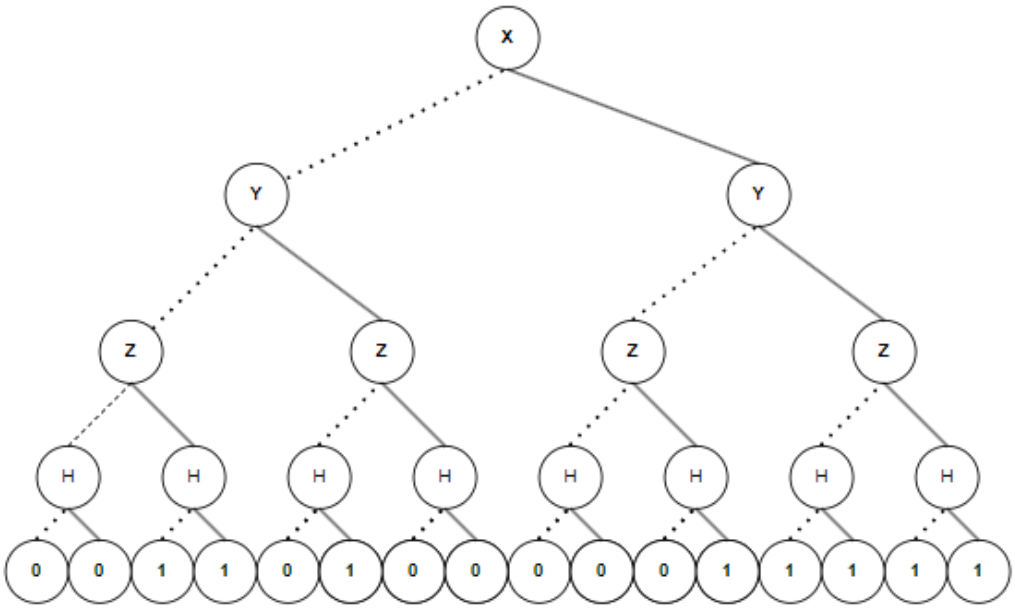

Рис. 1: дерево решений

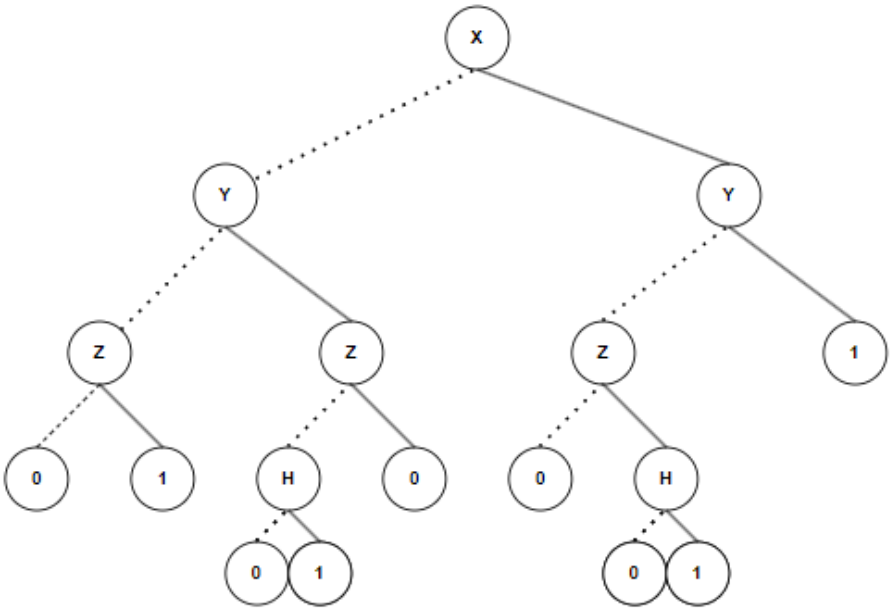

Рис. 2: сокращенное дерево решений

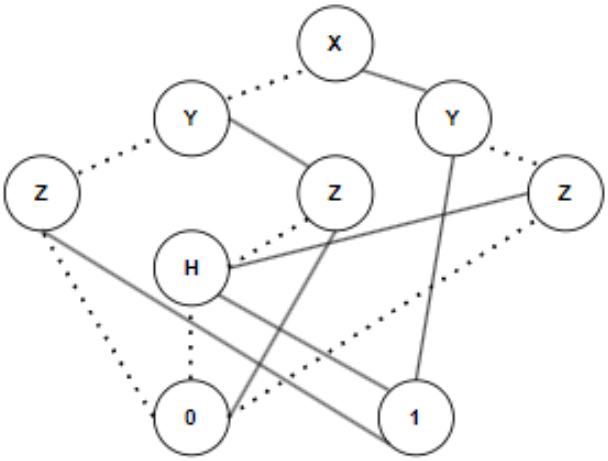

Рис. 3: бинарная диаграмма решений

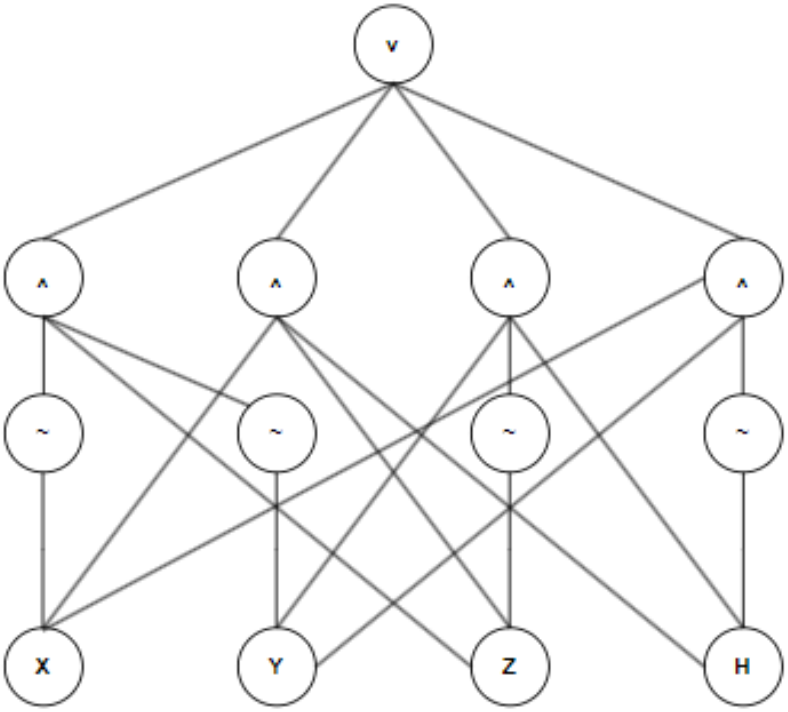

Рис. 4: синтаксическое дерево решений

2 Особенности реализации

2.1 Структура data и класс BulIst

Структура содержит в себе методы void AddTrue(data* d) и void AddFalse(data* d), в которых происходит создание указателя buf и проверка не указывает ли этот указатель на себя и инициализируется значением True или False. После это значение передается входному параметру.

Класс содержит в себе вектора типа bool которые сразу инициализируются значениями. Вектора соответствуют столбцам x,y,z,h,f таблицы истинности. Также класс содержит указатель, для того чтобы работать с БДР и строки SDNF, SKNF, Zhegalkin_str, Zhegalkin_str2 и вектора альфа-коэффициентов полинома и вектор указателей для того, чтобы хранить промежуточные шаги перемещения по дереву.

```
struct data {
char name;
data* False;
data* True;

data(const char s) {
name = s;
```

```
False = nullptr;
True = nullptr;
}
void AddTrue(data* d) {
data* buf = this;
while (buf->True != nullptr) {
buf = buf->True;
}
buf->True = d;
}

void AddFalse(data* d) {
data* buf = this;
while (buf->False != nullptr) {
buf = buf->False;
}
buf->False = d;
}
};

class BulIst
{
std::vector<bool> X{0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1};
std::vector<bool> Y{0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1};
std::vector<bool> Z{0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1};
std::vector<bool> H{0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1};
std::vector<bool> F{0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1};
data* BDR;
std::vector<data*> steps;
std::string SDNF;
std::string SKNF;
std::vector <std::vector<bool>> Zhegalkin;
std::vector <int> Alpha_index;
std::string Zhegalkin_str = "F = ";
std::string Zhegalkin_str2 = "F = ";
```

```
public:
BulIst();
void PrintIst();
void Print_SDNF();
void Print_SKNF();
void Print_Zhegalkin();
void Search_SDNF(std::string s);
void SearchBDR(std::string s);
void Search_Zhegalkin(std::string s);
void VisualizeBDR(data* node, int level);
};
```

2.2 BulIst()

При создании элемента типа BulIst автоматически происходит создание БДР при помощи функций AddFalse и AddTrue. Исходя из таблицы истинности происходит заполнение строк СКНФ и СДНФ. Вместе с этим происходит заполнение полинома Жегалкина полагаясь на заданный f-вектор и создание альфа-коэфициентов.

Вход: вектора X, Y, Z, H, F, ссылку на структуру БДР, четыре строки и булевый вектор.

Выход: заполненная структкра БДР, сформированные строки СКНФ и СДНФ, полином Жегалкина, 2 одинаковые строки Жегалкина.

```
BulIst::BulIst()
{
    BDR = new data('x');
    data* BDR_yF = new data('y');
    data* BDR_yT = new data('y');
    BDR->AddFalse(BDR_yF);
    BDR->AddTrue(BDR_yT);

    data* BDR_z0 = new data('z');
    data* BDR_z1 = new data('z');
    data* BDR_z2 = new data('z');
    data* BDR_z3 = new data('z');

    BDR_yF->AddFalse(BDR_z3);
    BDR_yF->AddTrue(BDR_z2);
```

```
BDR_yT->AddFalse(BDR_z1);
BDR_yT->AddTrue(BDR_z0);

data* BDR_h0 = new data('h');
data* BDR_h1 = new data('h');
data* BDR_h2 = new data('h');
data* BDR_h3 = new data('h');
data* BDR_h4 = new data('h');
data* BDR_h5 = new data('h');
data* BDR_h6 = new data('h');
data* BDR_h7 = new data('h');

BDR_z0->AddFalse(BDR_h1);
BDR_z0->AddTrue(BDR_h0);

BDR_z1->AddFalse(BDR_h3);
BDR_z1->AddTrue(BDR_h2);

BDR_z2->AddFalse(BDR_h5);
BDR_z2->AddTrue(BDR_h4);

BDR_z3->AddFalse(BDR_h7);
BDR_z3->AddTrue(BDR_h6);

data* BDR_F = new data('0');
data* BDR_T = new data('1');

BDR_h0->AddFalse(BDR_T);
BDR_h0->AddTrue(BDR_T);

BDR_h1->AddFalse(BDR_T);
BDR_h1->AddTrue(BDR_T);

BDR_h2->AddFalse(BDR_F);
BDR_h2->AddTrue(BDR_T);
```

```
BDR_h3->AddFalse(BDR_F);
BDR_h3->AddTrue(BDR_F);

BDR_h4->AddFalse(BDR_F);
BDR_h4->AddTrue(BDR_F);

BDR_h5->AddFalse(BDR_F);
BDR_h5->AddTrue(BDR_T);

BDR_h6->AddFalse(BDR_T);
BDR_h6->AddTrue(BDR_T);

BDR_h7->AddFalse(BDR_F);
BDR_h7->AddTrue(BDR_F);



SDNF = "СДНФ(f) = ";
SKNF = "СКНФ(f) = ";

for (int i = 0; i < 16; i++) {
    if (F[i] == 1) {
        SDNF += X[i] == 1 ? "x " : "!x ";
        SDNF += Y[i] == 1 ? "y " : "!y ";
        SDNF += Z[i] == 1 ? "z " : "!z ";
        SDNF += H[i] == 1 ? "h " : "!h ";
        SDNF += "  V  ";
    }
    else {
        SKNF += "(";
        SKNF += X[i] == 0 ? "x" : "!x";
        SKNF += " V ";
        SKNF += Y[i] == 0 ? "y" : "!y";
        SKNF += " V ";
        SKNF += Z[i] == 0 ? "z" : "!z";
        SKNF += " V ";
```

```
        SKNF += H[i] == 0 ? "h" : "!h";
        SKNF += ")";
    }
}
if (SDNF.length() >= 3) {
    SDNF.erase(SDNF.length() - 3);
}
SDNF += "\0";
SKNF += "\0";


for (int i = 16; i > 0; i--) {
    std::vector<bool> buf(i);
    Zhegalkin.push_back(buf);
}
for (int i = 0; i < 16; i++) {
    Zhegalkin[0][i] = F[i];
}
for (int i = 1; i < 16; i++) {
    for (int j = 0; j < Zhegalkin[i].size(); j++) {
        Zhegalkin[i][j] = Zhegalkin[i - 1][j] == Zhegalkin[i - 1][j + 1] ? 0 : 1;
    }
}

Alpha_index.push_back(0);
Alpha_index.push_back(1);
Alpha_index.push_back(2);
Alpha_index.push_back(3);
Alpha_index.push_back(4);
Alpha_index.push_back(5);
Alpha_index.push_back(6);
Alpha_index.push_back(7);
Alpha_index.push_back(8);
Alpha_index.push_back(9);
Alpha_index.push_back(10);
Alpha_index.push_back(11);
```

```
    Alpha_index.push_back(12);
    Alpha_index.push_back(13);
    Alpha_index.push_back(14);
    Alpha_index.push_back(15);



    for (int i = 0; i < 16; i++) {
        if (Zhegalkin[i][0] == 1) {
            Zhegalkin_str += "&_";
            Zhegalkin_str += std::to_string(Alpha_index[i]);
            Zhegalkin_str += " ";
            Zhegalkin_str += X[i] == 1 ? "x" : "";
            Zhegalkin_str += Y[i] == 1 ? "y" : "";
            Zhegalkin_str += Z[i] == 1 ? "z" : "";
            Zhegalkin_str += H[i] == 1 ? "h" : "";
            Zhegalkin_str += i < 15 ? " + " : "\0";
            Zhegalkin_str2 += X[i] == 1 ? "x" : "!x";
            Zhegalkin_str2 += Y[i] == 1 ? "y" : "!y";
            Zhegalkin_str2 += Z[i] == 1 ? "z" : "!z";
            Zhegalkin_str2 += H[i] == 1 ? "h" : "!h";
            Zhegalkin_str2 += i < 15 ? " + " : "\0";
        }
    }
    if (Zhegalkin_str.length() >= 3) {
        Zhegalkin_str.erase(Zhegalkin_str.length() - 3);
        Zhegalkin_str2.erase(Zhegalkin_str2.length() - 3);
    }
}
```

2.3 Вывод таблицы истинности

В этой функции формируется таблица состоящая из 5 столбцов и при помощи цикла for происходит поэлементный вывод всех значний. В каждой итерации происходит обращение к элементам векторов и они выводятся на экран через пробел. Когда все пять элементов одной итерации выведены цикл повторяется до тех пор, пока элементы не закончаться.

Вход: вектора X, Y, Z, H, F.

Выход: отображение таблицы истинности.

```
void BulIst::PrintIst() {
    std::cout << "  ~             Таблица истинности: " << std::endl;
    std::cout << "  ~              -------------- " << std::endl;
    std::cout << "  ~              | x y z h | f |" << std::endl;
    std::cout << "  ~              |-------------|\n";
    for (int i = 0; i < 16; i++) {
        std::cout << "  ~              | " << X[i] << " " << Y[i] << " " << Z[i] << " " <<
        H[i] << " | " << F[i] << " |" << std::endl;
    }
    std::cout << "  ~              |-------------| " << std::endl;
}
```

2.4 Полином Жегалкина в реализации

Данная функция *Print_Zhegalkin* предназначена для вывода на экран полинома Жегалкина булевой функции. Она формирует и отображает таблицу, где каждая строка представляет собой множитель в полиноме, а столбцы соответствуют переменным и их значениям в различных комбинациях. Для этого используются данные, предварительно рассчитанные в элементе типа BulIst который создается по умолчанию.

Вход: двумерный массив, иначе полиномом Жегалкина.

Выход: аккуратное отображение полинома Жегалкина, строка решений.

```
void BulIst::Print_Zhegalkin() {
    std::string s = " ";
    std::cout << "                Полином Жегалкина" << std::endl;
    std::cout << "                -----------------" << std::endl;
    for (int i = 0; i < 16; i++) {
        std::cout << "&_" << Alpha_index[i] << '\t' << s;
        s += " ";
        for (int j = 0; j < Zhegalkin[i].size(); j++) {
            std::cout << Zhegalkin[i][j] << " ";
        }
        std::cout << std::endl;
    }
    std::cout << std::endl << Zhegalkin_str << std::endl;
}
```

2.5 Поиск элемента по Полиному Жегалкина

Эта функция отвечает за вычисление полинома Жегалкина для заданной комбинации переменных s. С каждой итерацией цикла происходит оценка значений. Если существует решение под таким индексом, вычислияется значение термы при помощи операции И. Если хотя бы один элемент равен 0 - терма будет равнятся нулю, что соответствует конъюнкции. И при помощи операции ИЛИ, для каждой термы, вычисляется конечное значение.

Вход: строка, вектора X, Y, Z, H.

Выход: число равное 1 или 0.

```
void BulIst::Search_Zhegalkin(std::string s) {
    int result = 0;
    for (int i = 0; i < 16; i++) {
        if (Zhegalkin[i][0] == 1) {
            bool termResult = true;
            termResult &= X[i] == 1 ? s[0] == '1' : s[0] == '0';
            termResult &= Y[i] == 1 ? s[1] == '1' : s[1] == '0';
            termResult &= Z[i] == 1 ? s[2] == '1' : s[2] == '0';
            termResult &= H[i] == 1 ? s[3] == '1' : s[3] == '0';
            result ^= termResult;
        }
    }
    std::cout << " x = " << s[0] << "; y = " << s[1] << "; z = " << s[2] << "; h = " << s[3] <<s
    std::cout << Zhegalkin_str2 << " = " << result << std::endl;
    std::cout << "    Ответ: " << result << std::endl;
}
```

2.6 СКНФ и СДНФ

При помощи тернартых операторов происходит инициализация строк СНКФ и СДНФ. Если F[i] == 1, то-есть является решением тогда происходит заполнение СДНФ, в противном случае заполняется строка СКНФ. Программа поэлементно сравниват значение каждого элемента таблицы истинности с единицей. Если элемент равет единице то строка дополнится необходимым символом, к примеру x или z. Так происходит для каждой термы. Между термами ставится знак V.

СКНФ заполняется по обратной логике. Если элемент в таблице равен 1, то строка дополнится символави !x или !z. Между каждым элементам ставится знак V, а также каждая терма отделяется скобками. Эта логика связана с формулами которые символьно представляют, чему

равно СКНФ и СДНФ.

Вход: строки СКНФ и СДНФ, вектора X, Y, Z, H.

Выход: грамотно заполненные строки СКНФ и СДНФ.

```
SDNF = "СДНФ(f) = ";
SKNF = "СКНФ(f) = ";


    for (int i = 0; i < 16; i++) {
        if (F[i] == 1) {
            SDNF += X[i] == 1 ? "x " : "!x ";
            SDNF += Y[i] == 1 ? "y " : "!y ";
            SDNF += Z[i] == 1 ? "z " : "!z ";
            SDNF += H[i] == 1 ? "h " : "!h ";
            SDNF += "  V  ";
        }
        else {
            SKNF += "(";
            SKNF += X[i] == 0 ? "x" : "!x";
            SKNF += " V ";
            SKNF += Y[i] == 0 ? "y" : "!y";
            SKNF += " V ";
            SKNF += Z[i] == 0 ? "z" : "!z";
            SKNF += " V ";
            SKNF += H[i] == 0 ? "h" : "!h";
            SKNF += ")";
        }
    }
    if (SDNF.length() >= 3) {
        SDNF.erase(SDNF.length() - 3);
    }
    SDNF += "\0";
    SKNF += "\0";
```

2.7 Поиск элемента по СДНФ

Создается вспомогательный элемент типа string «s_out» и заполняется значениями противоположными тем, что были введены пользователем. Здесь используется цикл, которный повторя-

ется пока не закончится строка SDNF. Согласно формуле СДНФ строка buf заполняется так, что вместо переменных в строке будут хранится их значения, вместо x 1 и вместо !x 0.

Разрядность термы всегда равна 4, так как переменных 4 поэтомы зная это можно точно сказать сколько нужно итераций для следующей операции. Логика следущего цикла заключается в том, чтобы сравнивать первые 2 элемента каждой термы. Если хотя бы один из двух элементов равен нулю, эти два символа заменяются на ноль, если оба элементы 1, два символа заменяются еденицей. Формульно между каждым элементом стоит знак пересечения, что равносильно И. когда строка закончилась она выводится на экран, затем копируется в строку buf и после str снова приравнивается пустой строке для того, чтобы перейти к следующей итерации однако для строки с термами равными 3 символам. С каждой итерацией происходит усеньшение термы на один элемент, это происходит пока на экране не будут выводится термы состоящие из одной цифры, которая будет соответсвовать результату пересечения всех 4 переменных. Для вывода конечного результата, используется условие if, в котором происходит поэлементное сравнение строки с символом 1. Если в строке содержится хотя бы одна 1, конечный ответ равен 1. Эта логика соответсвует логическому ИЛИ, что формульно выражается как знак «+».

Вход: строку, вводимую пользователем и вектора X, Y, Z, H.

Выход: строка вычисления, строка СДНФ, результат поиска.

```
void BulIst::Search_SDNF(std::string s) {
    std::string s_out = "";
    for (int i = 0; i < 4; i++) {
        s_out += s[i] == '0' ? "1" : "0";
    }

    std::cout << " x = " << s[0] << "; y = " << s[1] << "; z = " << s[2] << "; h = " << s[3] <<

    std::string buf = "";
    for (int i = 0; SDNF[i] != '\0'; i++) {
        buf += SDNF[i] == 'x' ? SDNF[i - 1] == '!' ?
        std::string(1, s_out[0]) : std::string(1, s[0]) : "";
        buf += SDNF[i] == 'y' ? SDNF[i - 1] == '!' ?
        std::string(1, s_out[1]) : std::string(1, s[1]) : "";
        buf += SDNF[i] == 'z' ? SDNF[i - 1] == '!' ?
        std::string(1, s_out[2]) : std::string(1, s[2]) : "";
        buf += SDNF[i] == 'h' ? SDNF[i - 1] == '!' ?
        std::string(1, s_out[3]) : std::string(1, s[3]) : "";
        buf += SDNF[i] == 'V' ? "+" : "";
```

```
        buf += SDNF[i + 1] == '\0' ? "\0" : "";
    }

    std::cout << std::endl << " СДНФ(f) = " << buf;

    std::vector<int> result;
    int iterator = 5;
    std::string str = "";
    for (int j = 0; iterator > 2; iterator--) {
        for (int i = 0; i < buf.size() && buf[i] != '\0'; i += iterator) {
            str += buf[i] == '1' ? buf[i + 1] == '1' ? "1" : "0" : "0";
            str += buf.substr(i + 2, iterator - 3);// buf.substr(i + 2, 3), i + 2 - это
            //начальный индекс, и iterator - 3 - это количество символов для включения.
            str += "+";
        }
        if (str.length() >= 1) {
            str.erase(str.length() - 1);
        }
        std::cout << " = " << str;
        buf = str;
        str = "";
    }

    bool res = false;

    for (char c : buf) {
        if (c == '1') {
            res = true;
            break;
        }
    }
    std::cout << " = " << (res ? 1 : 0) << std::endl;
    std::cout << "   Ответ: " << res << std::endl;


}
```

2.8 Поиск элемента по БДР

Поиск осуществляется за счет прохода по пути бинарного решения. Каждый элемент проходит сравнение с 1 и 0, если значение равняется 1, buf указывает на true, а затем происходит переход на следующее звено диаграмы. Аналогично для 0. По итогу программа проходит по необходимым элементам дерева и в конце buf указывает на решение, для вводимого пользователем решения. Каждый шаг отображается на консоль для простоты понимания.

Вход: строка, введенная пользователем, структура БДР и вектора X, Y, Z, H.

Выход: построение БДР и значение 0 или 1.

```
void BulIst::SearchBDR(std::string s) {
    data* buf = this->BDR;
    std::string bdr = "Решение: X";
    int answer = 0;
    for (int i = 0; i < s.size(); i++) {
        if (s[i] == '0') {
            buf = buf->False;
            if (i == 0) {
                bdr += " --f--> Y";
            }
            else if (i == 1) {
                bdr += " --f--> Z";
            }
            else if (i == 2) {
                bdr += " --f--> H";
            }
            else{
                bdr += " --f--> 0";
            }

        }
        if (s[i] == '1') {
            buf = buf->True;
            if (i == 0) {
                bdr += " --t--> Y";
            }
            else if (i == 1) {
```

```
                bdr += " --t--> Z";
            }
            else if (i == 2) {
                bdr += " --t--> H";
            }
            else {
                bdr += " --t--> 1";
                answer = 1;
            }
        }
    }
    std::cout << bdr << std::endl;
    std::cout << "    Ответ: " << answer << std::endl;
    std::cout << std::endl;
    std::cout << std::endl;
    std::cout << "Построение БДР: " << std::endl;
    VisualizeBDR(BDR, 0);

}
```

2.9 Визуализация БДР

Визуализация происходит от последнего элемента дерева к первому сверху вниз с необходимым количеством пробелом. Функция рекурсивна. Это дает возможность вывести каждый узел дерева, не расписывая длинного кода.

Вход: указатель на структуру данных и начальный уровень, с которого начнется построение.

Выход: изображение БДР.

```
void BulIst::VisualizeBDR(data* node, int level) {
    if (node != nullptr) {
        VisualizeBDR(node->True, level + 1);
        for (int i = 0; i < level; i++) {
            std::cout << "    ";
        }
        std::cout << "            " << node->name << std::endl;
        VisualizeBDR(node->False, level + 1);
    }
```

```
}
```

2.10 Защита от некорректного ввода

Метод представленный ниже, является очень эффективным, однако довольно длинным. Пользователь может ввести комбинацию цифр букв и символов, но значение будет читаться только еесли пользователь введет число от 0 до 7 включительно, в остальных случаях программа выдаст ошибку и будет просить повторного ввода до момента корректного ввода.

Вход: строка, введенная пользователем.

Выход: сообщение об ошибке, если ввод некорректный.

```
char choice[5];
cout << endl;
cout << "   Номер: ";
cin >> choice;
cout << endl;
while (true) {
bool validInput = true;

for (char* c = choice; *c != '\0'; ++c) {
if (!isdigit(*c)) {
validInput = false;
break;
}
}

if (validInput) {
int n = atoi(choice);
if (n >= 0 && n < 8) {
break;
}
}

std::cerr << "Некорректный ввод. Число должно быть неотрицательным и не больше 7." << endl;
cout << "Введите число: " << endl;
cin >> choice;
cout << endl;
}
```

Здесь представлен другой способ защиты ввода. Он заметно короче и при этом не менее эффективнее. Использование его возможно только при подключении директивы «regex». В первой строке кода создается условие при котором верным будет значение состоящее только из 0 и 1 с разрядностью 4. $regex\_match(perem, ZNCH)$ это строка сравнивает введенное значение переменной с условием указаным ранее $std :: regexZNCH("[0-1]4\$")$; . Программа будет требовать повторного ввода до тех пор, пока значение не будет удовлетворять условию.

Вход: регулярное выражение, вводимая строка.

Выход: сообщение об ошибке, если ввод некорректный.

```
std::regex ZNCH("^[0-1]{4}$");
...
case 2:
cout << endl << "Введите значения переменных x, y, z, h в однустрочку без пробелов
   (пример : 0011)" << endl;
while (true) {
cout << "   Значение: ";
cin >> perem;
if (regex_match(perem, ZNCH))
break;
else
cout << "Некорректный ввод, число должно содержать четыре цифры по значению равным только
     0 и 1" << endl;
}
tab.Search_Zhegalkin(perem);
break;
```

2.11 Основное меню

Для большего удобства создается элемент типа BulIst, а замет вызывается метод PrintIst() для вывода таблицы истинности на экран. Далее этот метод рассмотрится наглядно.

После таблицы истинности пользователю предлагается выбрать 1 из 8 операций, в их число входит: вывести полином Жегалкина, найти значение по полиному, показать СДНФ и СКНФ заданной функции, а также произвести поиск элемента по СДНФ, найти значение по БДР, вывести повторно на экран таблицу истинности и выйти из программы. Реализация такого меню осуществляется за счет switch case. Значение, которое вводит пользователь проходит проверку и затем конвертируется в значение типа int. Затем выполняется операция с выбранным номером путем вызова соответсвующей функции.

Вход: номер операции.

Выход: отображение операции.

```
        BulIst tab;
tab.PrintIst();


while (true) {
cout << endl << "Укажите номер операции:" << endl;
cout << " 1. Полином Жегалкина" << endl;
cout << " 2. Найти значение по полиному Жегалкина" << endl;
cout << " 3. СДНФ" << endl;
cout << " 4. СКНФ" << endl;
cout << " 5. Найти значение по БДР" << endl;
cout << " 6. Вывести на экран таблицу истинности" << endl;
cout << " 7. Найти значение по СДНФ" << endl;
cout << " 0. Выход" << endl;
```

3 Результаты работы

В результате работы было реализовано 8 различных операций. Каждая операция визуализируется на консоли в приемлимом для восприятии виде. Пользователь имеет возможность испытать любые операции друг за другом, до момента пока пользователь сам не захочет закончить. Для надежности и корректной работы программы была реализована защита на ввод любых значений(см.рис5, рис6, рис7, рис8).

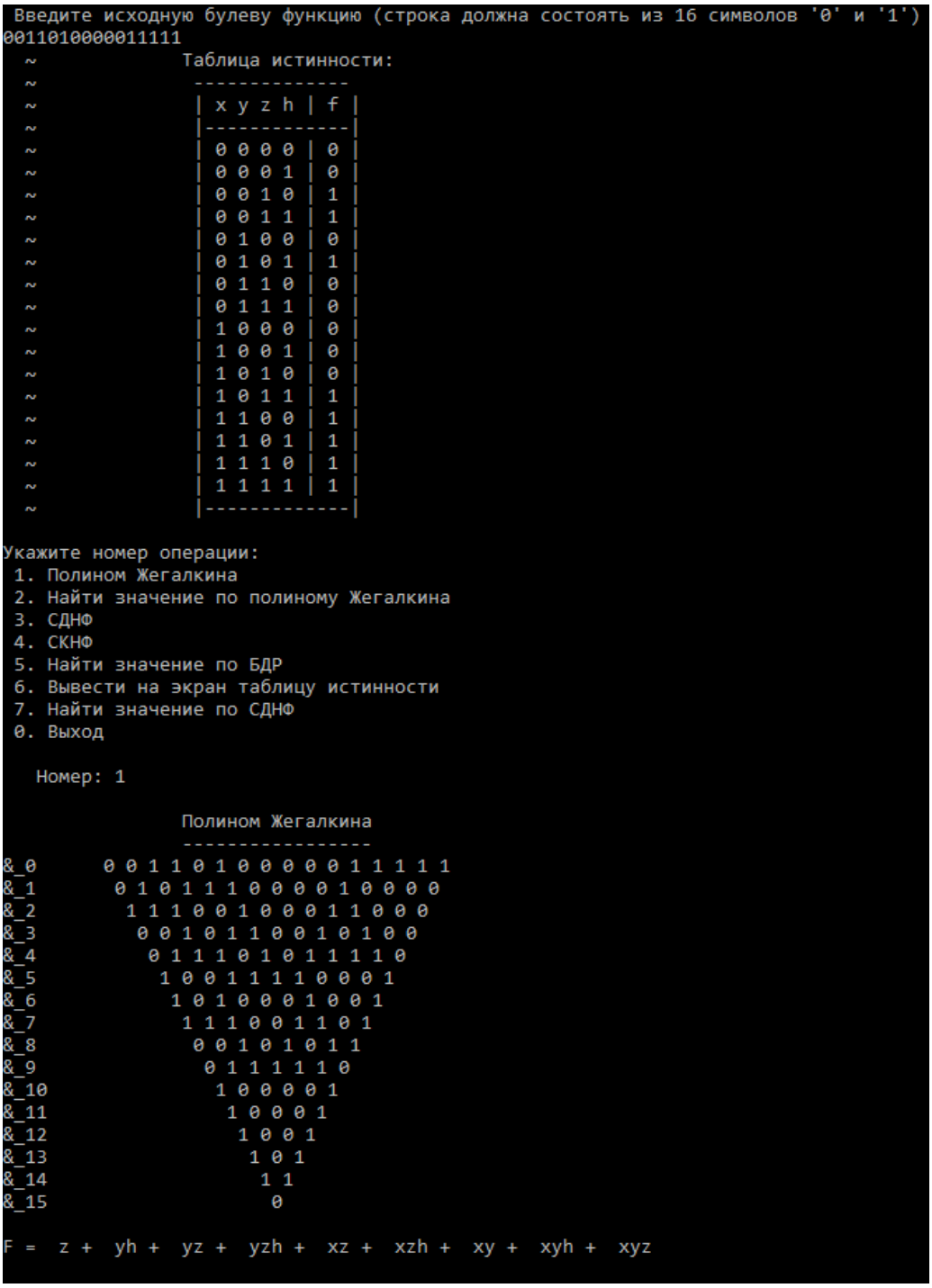

Рис. 5: результаты первых 2 операций

```
Укажите номер операции:
 1. Полином Жегалкина
 2. Найти значение по полиному Жегалкина
 3. СДНФ
 4. СКНФ
 5. Найти значение по БДР
 6. Вывести на экран таблицу истинности
 7. Найти значение по СДНФ
 0. Выход

   Номер: 2


Введите значения переменных x, y, z, h в однустрочку без пробелов(пример : 0011)

   Значение: 0011

 x = 0; y = 0; z = 1; h = 1

F = !x!yz!h + !xy!zh + !xyz!h + !xyzh + x!yz!h + x!yzh + xy!z!h + xy!zh + xyz!h  = 0
    Ответ: 0

Укажите номер операции:
 1. Полином Жегалкина
 2. Найти значение по полиному Жегалкина
 3. СДНФ
 4. СКНФ
 5. Найти значение по БДР
 6. Вывести на экран таблицу истинности
 7. Найти значение по СДНФ
 0. Выход

   Номер: 3


СДНФ(f) = !x !y z !h   V  !x !y z h   V  !x y !z h   V  x !y z h   V  x y !z !h   V  x y !z h   V  x y z !h   V  x y z h

Укажите номер операции:
 1. Полином Жегалкина
 2. Найти значение по полиному Жегалкина
 3. СДНФ
 4. СКНФ
 5. Найти значение по БДР
 6. Вывести на экран таблицу истинности
 7. Найти значение по СДНФ
 0. Выход

   Номер: 4


СКНФ(f) = (x V y V z V h)(x V y V z V !h)(x V !y V z V h)(x V !y V !z V h)(x V !y V !z V !h)(!x V y V z V h)(!x V y V z V !h)(!x V y V !z V h)
```

Рис. 6: результаты 2, 3, 4 операций

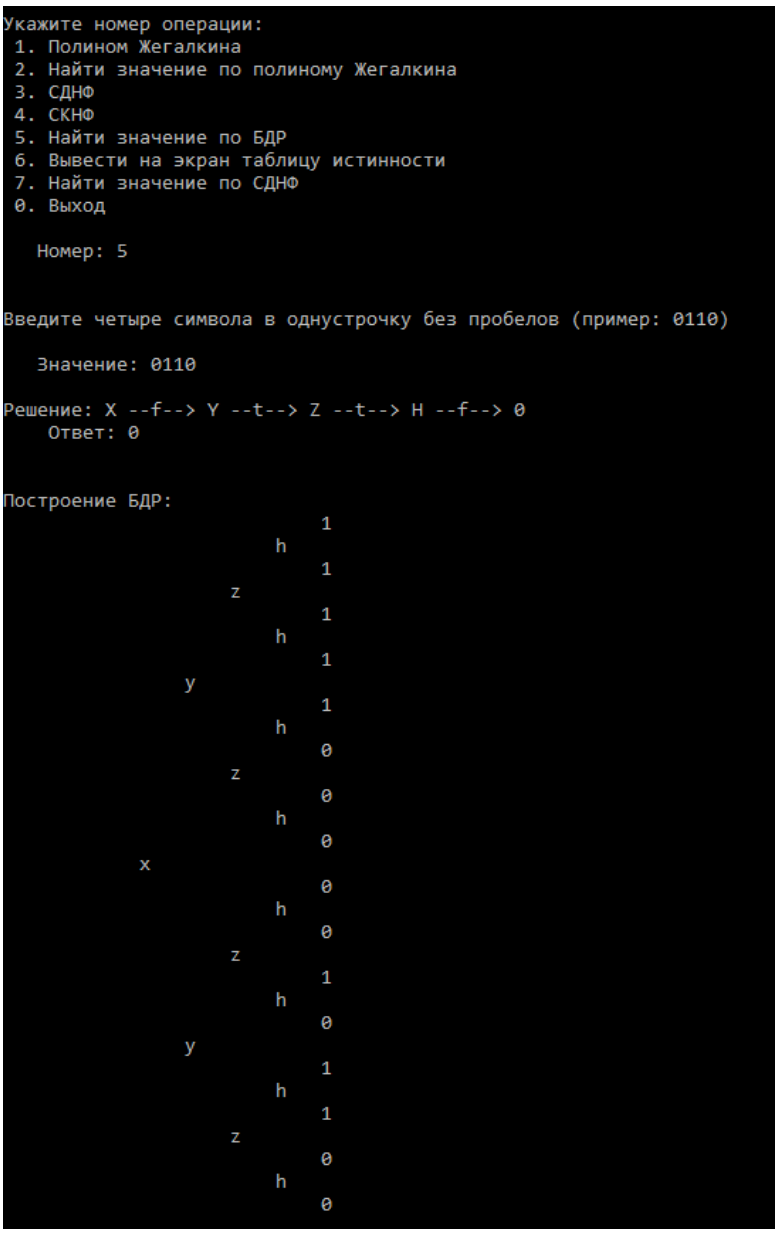

Рис. 7: результат 5 операции

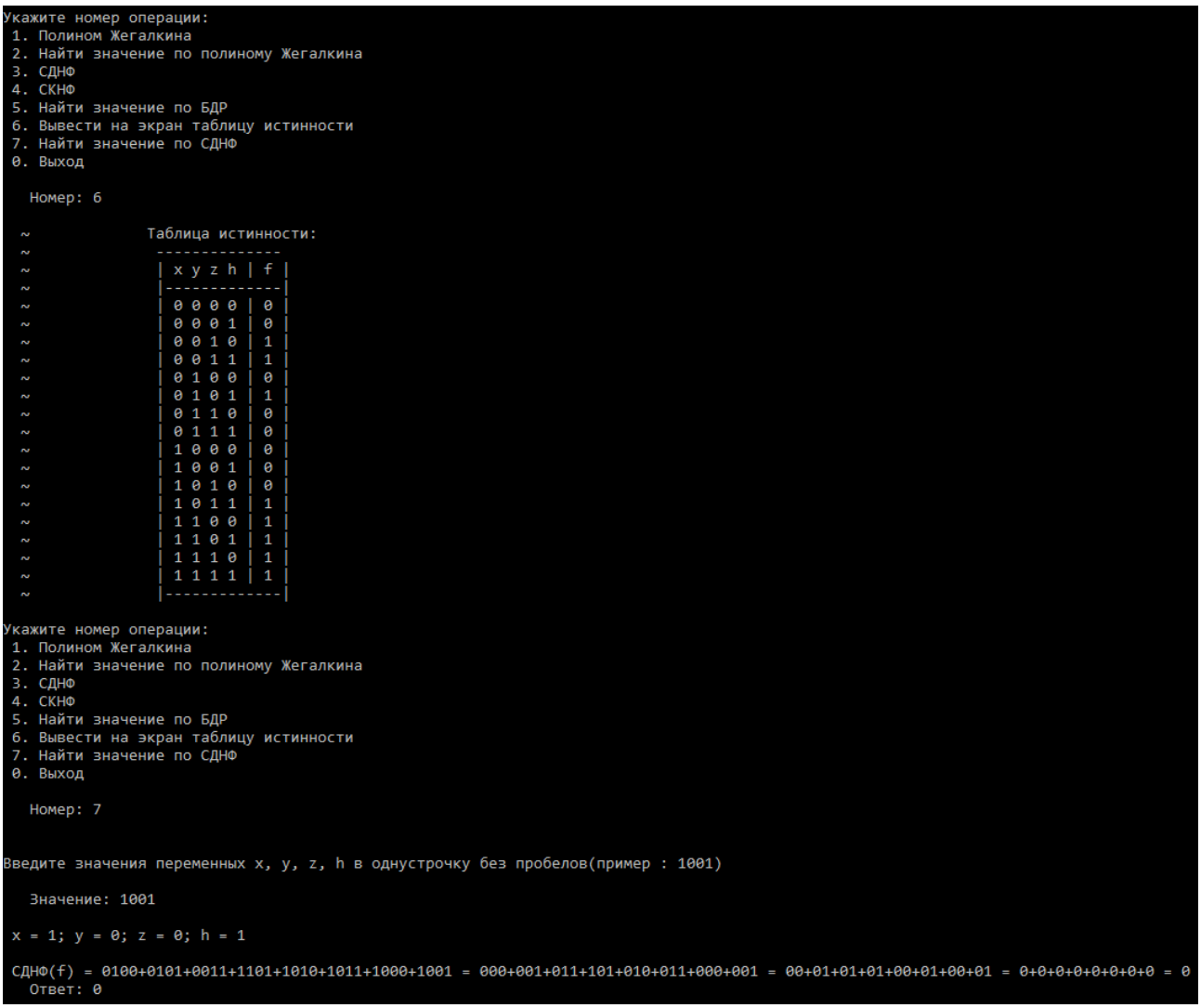

Рис. 8: результаты последних операций

Заключение

В рамках выполнения лабораторной работы №2 была разработана программа, выполняющая ряд операций связанных с булевыми функциями. Эта программа включает в себя реализацию алгоритмов работы с СДНФ и СКНФ, реализация полинома Жегалкина и БДР. Также есть возможнность найти значение по полиному Жегалкина, БДР и по СДНФ.

Главные достоинства этой программы - это возможность применить все операции, включая БДР, для заданного пользователем вектора, удобный интерфейс и довольно простой в понимании код. Минусом можно назвать объем кода,

К возможным доработкам, относиться возможность очищать экран после каждой операции для простоты чтения.

Список материалов

[1] Новиков Ф.А. Дискретная математика для программистов (начиная с 3-его издания). https://stugum.files.wordpress.com/2014/03/novikov.pdf

[2] wiki.fenix - сайт справочник по дисциплине математика(последнее открытие 12.02.2024) https://wiki.fenix.help/informatika/sdnf?ysclid=lr6bu660ud240614897