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 по лабораторной работе №2 по дисциплине «Дискретная математика»

«Вычисление значения булевой функции по БДР, СДНФ, СКНФ и полиному Жегалкина.» Вариант 16

Студент,
группы 5130201/20001 _____ Якунин Д. Д.

Преподаватель _____ Востров А. В.

«_____»_____ 2023г.

Санкт-Петербург, 2023

Содержание

Введение

Задан вектор-столбец значений булевой функции 4-х переменных (порядок определен по возрастанию элементов).

Необходимо:

1) Построить по таблице истинности дерево решений и бинарную диаграмму решений, а также синтаксическое дерево для минимальной формулы с критерием минимизации по количеству входящих в нее переменных (вручную). Реализовать программно хранение полученной бинарной диаграммы решений и вычисление ее значения (по пользовательскому вводу).

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 булева функция f = (0000111011011010).

1 Математическое описание

1.1 Функции алгебры логики

Функции $f : E_2^n \to E_2$, где $_2 = \{0, 1\}$, называются функциями алгебры логики, или булевыми функциями от n переменных, по имени Дж. Буля. Булеву функцию от n переменных можно задать таблицей истинности:

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

Таблица истинности исходной функции приведена на Рис.1.

a	b	c	d	func
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Рис. 1. Исходная функция

1.2 Совершенная дизъюнктивная нормальная форма

Всякая булева функция имеет единственную совершенную дизъюнктивную нормальную форму (СДНФ):

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

Для исходной функции СДНФ f = $\bar{a}b\overline{cd} \vee \bar{a}b\bar{c}d \vee \bar{a}bc\bar{d} \vee a\overline{bcd} \vee a\bar{b}cd \vee a\bar{b}\overline{cd} \vee abc\bar{d}$

1.3 Совершенная конъюктивная нормальная форма

Всякая булева функция имеет единственную совершенную конъюктивную нормальную форму (СКНФ):

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

Для исходной функции СКНФ f = $(a \vee b \vee c \vee d) \wedge (a \vee b \vee c \vee \overline{d}) \wedge (a \vee b \vee \overline{c} \vee d) \wedge (a \vee b \vee \overline{c} \vee \overline{d}) \wedge (a \vee \overline{b} \vee \overline{c} \vee \overline{d}) \wedge (\overline{a} \vee b \vee \overline{c} \vee d) \wedge (\overline{a} \vee \overline{b} \vee c \vee \overline{d}) \wedge (\overline{a} \vee \overline{b} \vee \overline{c} \vee \overline{d})$

1.4 Полином Жегалкина

Представление булевой функции над базисом $\{0, 1, \wedge, +\}$ называется полиномом Жегалкина. В общем виде он является полиномом вида

$$P = a \oplus \bigoplus_{1 \le i_1 \le ... \le i_k \le n; k \in \overline{1,n}} a_{i_1,...,i_k} \wedge x_{i_1} \wedge ... \wedge x_{i_k}, \quad a, a_{i_1,...,i_k} \in \{0, 1\}$$

Для исходной функции полином следующий: $f = b + bcd + a + ac + acd + ab + abd + abc$.

1.5 Семантическое и синтаксическое деревья решений, бинарная диаграмма решений

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

Дерево решений можно сократить, если заменить корень каждого поддерева, все листья которого имеют одно и то же значение, этим значением. Иногда такое сокращение значительно уменьшает объём дерева.

Семантическое дерево решений заданной функции приведено на Рис.2. Пунктирная линия означает 0, сплошная - 1.

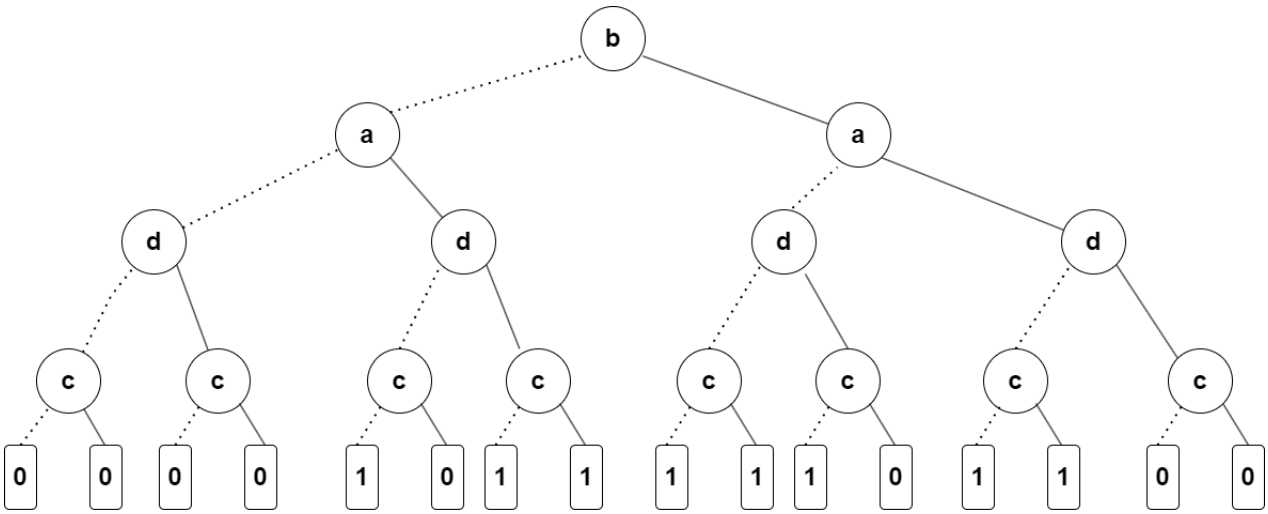

Рис. 2. Семантическое дерево решений

Дерево решений можно сделать ещё 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 заданной функции приведена на Рис.3. Пунктирная линия означает 0, сплошная - 1.

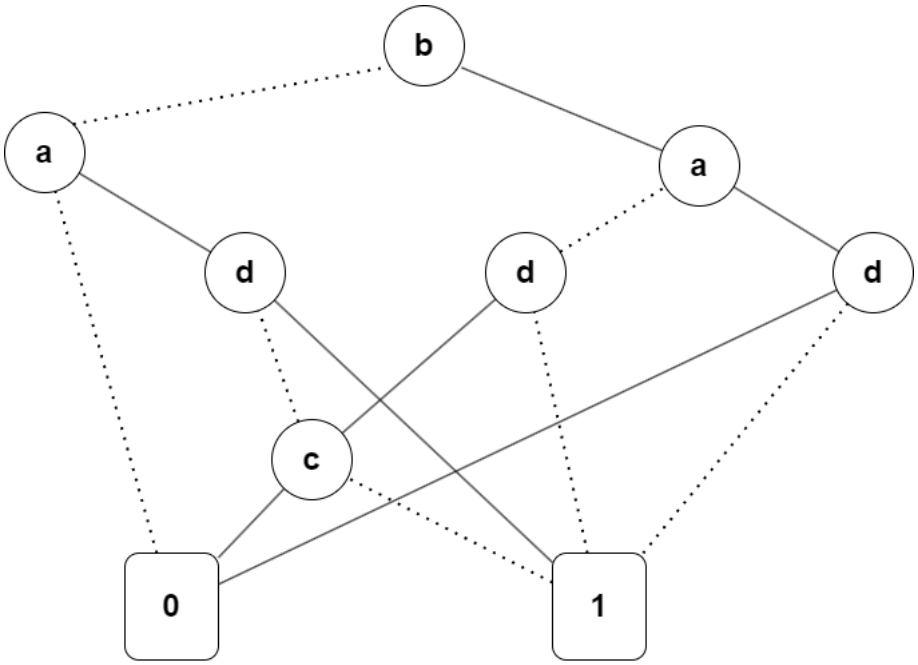

Рис. 3. Бинарная диаграмма решений

Синтаксическое дерево решений построенно на основе формулы $f = b\overline{d} \vee a\overline{b}d \vee \overline{c}(a+b)$, полученной путем упрощения СДНФ, и приведено на Рис.4.

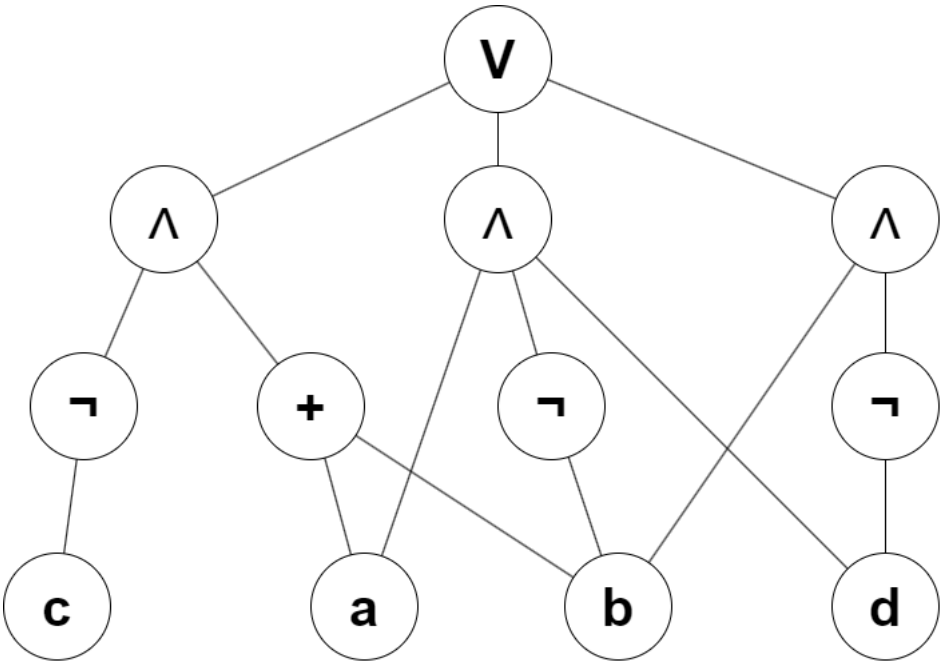

Рис. 4. Синтаксическое дерево решений

2 Особенности реализации

2.1 Вспомогательный класс Branch

Для хранения БДР используется класс Branch, каждый элемент которого хранит значение типа bool и два указателя на следующие узлы дерева.

Реализация класса приведена в Листинге 1.

Листинг 1. Вспомогательный класс Branch

```
1  class Branch {
2    public:
3    bool Data;
4    Branch* one;
5    Branch* zero;
6  };
```

2.2 Класс class boolFunc

Класс предназначен для хранения таблицы истинности, значений функции и БДР, создания СКНФ, СДНФ и полинома Жегалкина и последующих вычислений значений на их основе.

2.2.1 Поля класса

- std::vector<bool> a - вектор, хранящий значения переменной «a» для таблицы истинности. Аналогичные поля есть для переменных «b», «c», «d»
- std::vector<bool> f - вектор, хранящий значения переменной функции
- std::vector<std::vector<bool> > SDNF - вектор векторов, хранящий такие наборы значений переменных, что функция при них равна 1
- std::vector<std::vector<bool> > SKNF - вектор векторов, хранящий такие наборы значений переменных, что функция при них равна 0
- std::vector<bool> Zhegalkin - вектор, хранящий коэффициэнты полинома Жегалкина
- Branch *BDDRoot - указатель на корень БДР, из которого далее идут ветки для вычисления значений

2.2.2 Конструктор класса

При создании объекта типа boolFunc вызывается конструктор, записывающий значения таблицы истинности и функции, а после выстраивающий связи для БДР.

Реализация приведена в Листинге 2.

Листинг 2. Конструктор класса

```
1  boolFunc::boolFunc() {
```

```
    a = { 0,0,0,0, 0,0,0,0, 1,1,1,1, 1,1,1,1 };
    b = { 0,0,0,0, 1,1,1,1, 0,0,0,0, 1,1,1,1 };
    c = { 0,0,1,1, 0,0,1,1, 0,0,1,1, 0,0,1,1 };
    d = { 0,1,0,1, 0,1,0,1, 0,1,0,1, 0,1,0,1 };
    f = { 0,0,0,0, 1,1,1,0, 1,1,0,1, 1,0,1,0 };

    BDDRoot = new Branch;
    // b
    BDDRoot->one = new Branch;
    BDDRoot->zero = new Branch;
    // a0
    BDDRoot->zero->one = new Branch;
    BDDRoot->zero->zero = new Branch;

    Branch* zeroEnd = BDDRoot->zero->zero;
    zeroEnd->Data = 0;
    // a1
    BDDRoot->one->zero = new Branch;
    BDDRoot->one->one = new Branch;
    // d0
    BDDRoot->zero->one->zero = new Branch; // c
    BDDRoot->zero->one->one = new Branch;

    Branch* oneEnd = BDDRoot->zero->one->one;
    oneEnd->Data = 1;
    // d1
    BDDRoot->one->zero->one = BDDRoot->zero->one->zero; // c
    BDDRoot->one->zero->zero = oneEnd;
    // d2
    BDDRoot->one->one->zero = oneEnd;
    BDDRoot->one->one->one = zeroEnd;
    // c
    BDDRoot->zero->one->zero->one = zeroEnd;
    BDDRoot->zero->one->zero->zero = oneEnd;

    oneEnd->one = oneEnd;
    oneEnd->zero = oneEnd;
    zeroEnd->one = zeroEnd;
    zeroEnd->zero = zeroEnd;
}
```

2.2.3 Метод valBDD

Вход: вектор, хранящий значения bool

Выход: значение типа bool

Метод принимает значения, для которых нужно посчитать значение, ставит их в порядке, в котором они в БДР и далее идет по указателям в зависимости от значения каждой переменной, если значение 0, то идем по указателю zero, иначе по указателю

one.

Реализация приведена в Листинге 3.

Листинг 3. Метод valBDD

```
bool boolFunc::valBDD(std::vector<bool> inp) {
  std::vector<bool> newInp = { inp[1], inp[0], inp[3], inp[2] };
  Branch* tmp = BDDRoot;
  for (int i = 0; i < inp.size(); i++) {
    if (newInp[i] == 0) tmp = tmp->zero;
    else tmp = tmp->one;
  }

  return tmp->Data;
}
```

2.2.4 Методы createSDNF и createSKNF

Вход обеих методов: вектор значений функции f

Выход createSDNF: вектор наборов значений, при которых функция f равна 1, SDNF

Выход createSKNF: вектор наборов значений, при которых функция f равна 0, SKNF

Метод createSDNF проходит по всем значениям функции f и сохраняет в переменную SDNF все наборы переменных, при которых значение f = 1.

Метод createSKNF аналогичен методу createSDNF, но он сохраняет наборы, при которых значение f = 0 в переменную SKNF.

Реализации приведены в Листингах 4 и 5.

Листинг 4. Метод createSDNF

```
void boolFunc::createSDNF() {

  for (int i = 0; i < f.size(); i++) {
    if (f[i] == 1) {
      SDNF.push_back({ a[i], b[i], c[i], d[i] });
    }
  }
}
```

Листинг 5. Метод createSKNF

```
void boolFunc::createSKNF() {

  for (int i = 0; i < f.size(); i++) {
    if (f[i] == 0) {
      SKNF.push_back({ a[i], b[i], c[i], d[i] });
    }
  }
}
```

2.2.5 Методы valSDNF и valSKNF

Вход valSDNF: вектор векторов SDNF и вектор, хранящий значения bool

Вход valSKNF: вектор векторов SKNF и вектор, хранящий значения bool

Выход: значение типа bool

Метод valSDNF принимает вектор значений, для которых нужно посчитать значение, а потом сравнивает этот вектор с векторами в поле SDNF. Если совпадение найдено, то метод вернет 1, иначе - 0.

Метод valSKNF делает аналогичные сравнения с векторами внутри вектора SKNF. Если совпадение найдено, то метод вернет 0, иначе - 1.

Реализации приведены в Листингах 6 и 7.

Листинг 6. Метод valSDNF

```
bool boolFunc::valSDNF(std::vector<bool> inp) {
  for (int i = 0; i < SDNF.size(); i++) {
    if (SDNF[i] == inp) {
      return 1;
    }
  }
  return 0;
}
```

Листинг 7. Метод valSKNF

```
bool boolFunc::valSKNF(std::vector<bool> inp) {
  for (int i = 0; i < SKNF.size(); i++) {
    if (SKNF[i] == inp) {
      return 0;
    }
  }
  return 1;
}
```

2.2.6 Методы printSDNFstring и printSKNFstring

Вход printSDNFstring: вектор векторов SDNF

Вход printSKNFstring: вектор векторов SKNF

Выход обоих методов: строка

Метод printSDNFstring пробегается по всем векторам переменной SDNF и печатает в консоль СДНФ в виде строки.

Метод printSKNFstring пробегается по всем векторам переменной SKNF и печатает в консоль СКНФ в виде строки.

Реализации приведены в Листингах 8 и 9.

Листинг 8. Метод printSDNFstring

```
void boolFunc::printSDNFstring() {
  std::cout << "\nSDNF:\n";
  for (int i = 0; i < SDNF.size(); i++) {
```

```
    if (i != 0 and i != SDNF.size()) std::cout << " V ";

    if (SDNF[i][0] == 1) std::cout << "a";
    else std::cout << "(~a)";
    if (SDNF[i][1] == 1) std::cout << "b";
    else std::cout << "(~b)";
    if (SDNF[i][2] == 1) std::cout << "c";
    else std::cout << "(~c)";
    if (SDNF[i][3] == 1) std::cout << "d";
    else std::cout << "(~d)";
  }
  std::cout << std::endl;
}
```

Листинг 9. Метод printSKNFstring

```
void boolFunc::printSKNFstring() {
  std::cout << "\nSKNF:\n";
  for (int i = 0; i < SKNF.size(); i++) {
    std::cout << "(";

    if (SKNF[i][0] == 0) std::cout << "a V ";
    else std::cout << "(~a) V ";
    if (SKNF[i][1] == 0) std::cout << "b V ";
    else std::cout << "(~b) V ";
    if (SKNF[i][2] == 0) std::cout << "c V ";
    else std::cout << "(~c) V ";
    if (SKNF[i][3] == 0) std::cout << "d";
    else std::cout << "(~d)";

    std::cout << ")";
    if (i != (SKNF.size()-1)) std::cout << "*";
  }
}
```

2.2.7 Метод createZhegalkin

Вход: вектор значений функции f

Выход: вектор коэффициентов полинома Жегалкина Zhegalkin

Метод createZhegalkin берет значения функции f и методом треугольника считает коэффициенты полинома. Для этого создается новый вектор, значения которого являются результатом сложения по модулю 2 соседних элементов изначального вектора. Далее аналогичная операция делается с каждым новым вектором до тех пор, пока вектор не будет состоять из одного элемента. Коэффициентами же полинома Жегалкина будут крайние левые значения каждого полученного вектора. Все коэффициенты добавляются в вектор Zhegalkin.

Реализация приведена в Листинге 10.

Листинг 10. Метод createZhegalkin

```
void boolFunc::createZhegalkin() {
  Zhegalkin.push_back(f[0]);
  std::vector<bool> prev = f;
  std::vector<bool> next;
  for (int i = 0; i < f.size() - 1; i++) {
    for (int j = 0; j < prev.size() - 1; j++) {
      next.push_back(prev[j] ^ prev[j + 1]);
    }
    Zhegalkin.push_back(next[0]);
    prev = next;
    next.resize(0);
  }
}
```

2.2.8 Метод valZhegalkin

Вход: вектор, хранящий значения bool

Выход: значение типа bool

Метод valZhegalkin пробегается по всем значениям переменной Zhegalkin. Для всех ненулевых коэфициентов считается, какое бы было значение с входными данными, а после оно складывается по модулю 2 с другими такими значениями. В итоге получается значение функции для входного набора данных.

Реализация приведена в Листинге 11.

Листинг 11. Метод valZhegalkin

```
bool boolFunc::valZhegalkin(std::vector<bool> inp) {
  bool result = 0;
  for (int i = 0; i < Zhegalkin.size(); i++) {
    if (Zhegalkin[i] == 1) {
      bool tmp = 1;
      if (a[i]) tmp = tmp && inp[0];
      if (b[i]) tmp = tmp && inp[1];
      if (c[i]) tmp = tmp && inp[2];
      if (d[i]) tmp = tmp && inp[3];

      result ^= tmp;
    }
  }

  return result;
}
```

2.2.9 Метод printZhegalkinString

Вход: вектор коэффициентов полинома Жегалкина Zhegalkin

Выход: строка

Метод printZhegalkinString пробегается по всем значениям переменной printZhegalkinString и печатает в консоль полином Жегалкина в виде строки.

Реализация приведена в Листинге 12.

Листинг 12. Метод printZhegalkinString

```
void boolFunc::printZhegalkinString() {
  std::cout << "\n\nZhegalkin Polynom:\n";
  bool flag = 0;
  for (int i = 0; i < Zhegalkin.size(); i++) {
    if (Zhegalkin[i] == 1) {
      if (flag && i != Zhegalkin.size()) std::cout << " + ";
      if (a[i]) std::cout << "a";
      if (b[i]) std::cout << "b";
      if (c[i]) std::cout << "c";
      if (d[i]) std::cout << "d";
      flag = 1;
    }
  }
}
```

2.2.10 Метод checkFunc

Вход: векторы SDNF, SKNF, Zhegalkin и указатель на корень БДР BDDRoot

Выход: строка

Метод считает значения СКНФ, СДНФ, БДР и полинома Жегалкина с помощью вышеприведенных функций и сравнивает полученные значения с значениями f. Если все значения правильные, то метод выведет на консоль сообщение «test is successful!», иначе - «test failed!».

Реализация приведена в Листинге 13.

Листинг 13. Метод checkFunc

```
void boolFunc::checkFunc() {
  bool flagRes = 1;
  std::vector<bool> tmp;
  for (int i = 0; i < f.size(); i++) {
    tmp = { a[i], b[i], c[i], d[i] };
    if ((valSDNF(tmp) != f[i]) or (valSKNF(tmp) != f[i]) or
    (valBDD(tmp) != f[i]) or (valZhegalkin(tmp) != f[i]))
       flagRes = 0;
  }

  if (flagRes) std::cout << "\n\ntest is successful!";
  else std::cout << "\n\ntest failed!";
}
```

2.3 Функция main

Вход: константный вектор значений функции f

Выход: int значение

При запуске программы создается переменная типа boolFunc, после чего для нее вычисляются СКНФ, СДНФ и полином Жегалкина.

После этого идет вывод таблицы истинности и значений СКНФ, СДНФ, БДР, полинома Жегалкина и исходной функции для всех наборов входных значений.

После этого СКНФ, СДНФ и полином Жегалкина выводятся в виде строк.

После этого через метод checkFunc автоматически проверяются все значения.

Реализация приведена в Листинге 14.

Листинг 14. Функция main

```
int main()
{
  boolFunc myFunc;
  myFunc.createSDNF();
  myFunc.createSKNF();
  myFunc.createZhegalkin();

  std::vector<bool> tmp = {0,0,0,0};
  std::cout << "a  b  c  d\tfunc\tSDNF\tSKNF\tPolyZ\tBDD" << std
     ::endl;
  for (int i = 0; i < 16; i++) {
    tmp = { myFunc.a[i], myFunc.b[i], myFunc.c[i], myFunc.d[i]
       };
    std::cout << tmp[0] << "  " << tmp[1] << "  " << tmp[2] << "
         " << tmp[3] << "\t";
    std::cout << myFunc.f[i] << '\t';

    std::cout << myFunc.valSDNF(tmp) << '\t';
    std::cout << myFunc.valSKNF(tmp) << '\t';
    std::cout << myFunc.valZhegalkin(tmp) << '\t';
    std::cout << myFunc.valBDD(tmp) << std::endl;
  }


  myFunc.printSDNFstring();
  myFunc.printSKNFstring();
  myFunc.printZhegalkinString();
  myFunc.checkFunc();
}
```

3 Результаты работы программы

После запуска программы в консоль выводится таблица истинности, все значения для СКНФ, СДНФ, БДР, полинома Жегалкина. Далее выводятся СКНФ, СДНФ и полином Жегалкина в виде строк. После этого выводится сообщение об успешности всех проверок (Рис. 5).

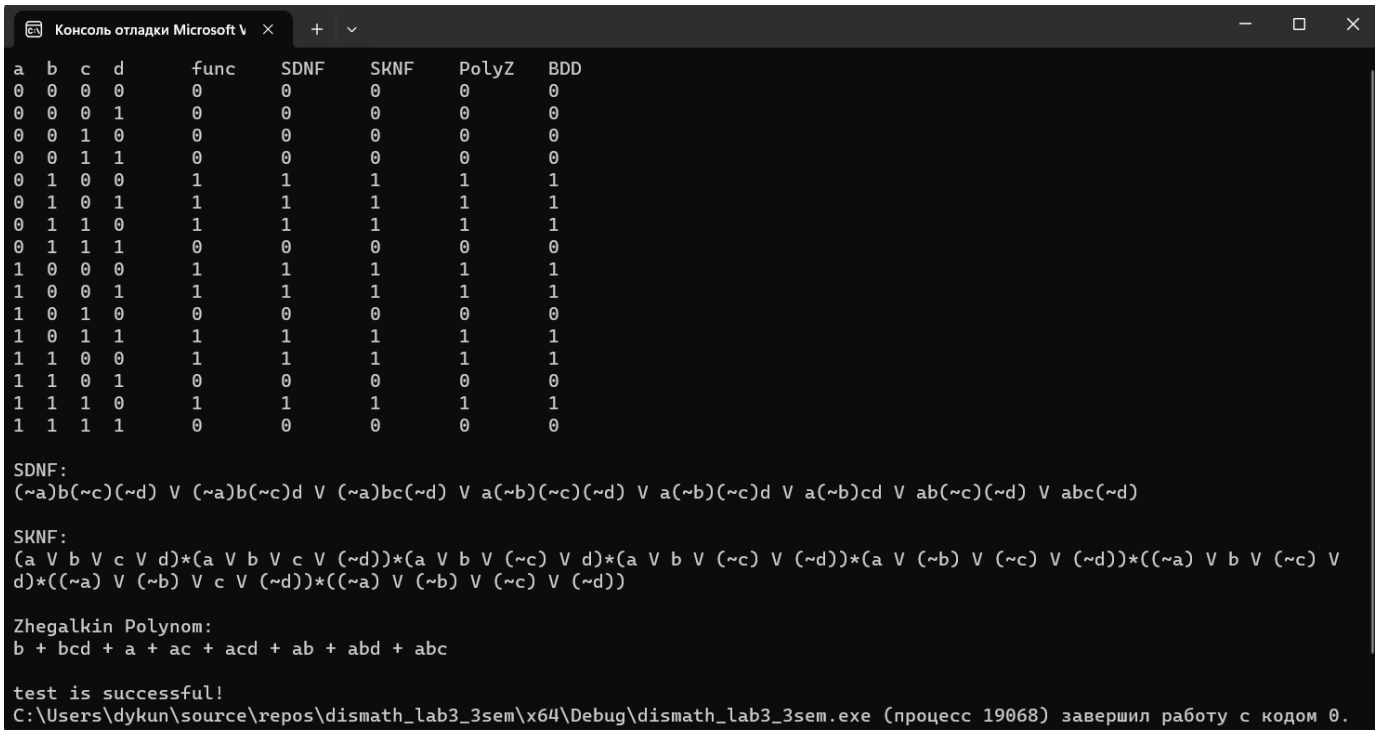

Рис. 5. Результат работы программы

При вводе значений программа выведет значение функции, посчитанное при помощи любого выбранного способа. На Рис.6 приведен пример с подсчетом через полином Жегалкина.

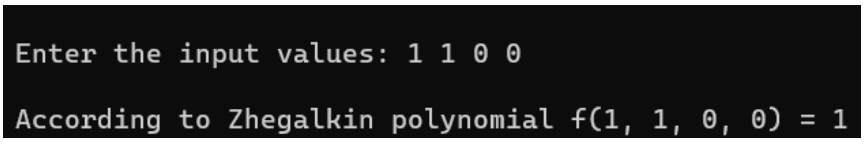

Рис. 6. Подсчет значение функции через полином Жегалкина

Заключение

В процессе выполнения работы была реализована программа, позволяющая получить для функции СКНФ, СДНФ и полином Жегалкина. Так же возможно получить значения функции разными способами: через СКНФ, СДНФ, полином Жегалкина и БДР.

Вручную построены по таблице истинности деревья решений и бинарная диаграмма решений.

Достоинства программы:

- использование ООП упрощает написание программы и увеличивает читаемость кода;
- в качестве массивов данных используются контейнеры vector, что позволяет избежать утечек памяти и повысить удобство написания программы.

Недостатки программы:

- хранение БДР реализовано вручную, что неудобно задавать и изменять в будущем;
- очистка памяти указателей БДР производится вручную и выполняется при удалении объекта myFunc, деструктор которого вызывает деструктор всей БДР, что неэффективно.

Масштабирование: На основе реализованного класса можно продолжать реализовывать новые операции, например, логические and и or между объектами класса. Также можно сделать ввод функции с консоли, а не задавать её в конструкторе, тем самым расширив возможности программы.

Список литературы

[1] Новиков, Ф.А. ДИСКРЕТНАЯ МАТЕМАТИКА ДЛЯ ПРОГРАММИСТОВ / 3-е издание. - Питер: ПитерПресс, 2009, (дата обращения - 09.12.2023)

[2] Секция «Телематика» - https://tema.spbstu.ru/dismath/, (дата обращения - 09.12.2023)

[3] Бинарные деревья в C++ - https://purecodecpp.com/archives/2483, (дата обращения - 09.12.2023)