Министерство образования и науки РФ
Пермский национальный исследовательский политехнический университет
Электротехнический факультет
Кафедра информационных технологий и автоматизированных систем

Дискретная математика и математическая логика

Лабораторная работа № 3

Тема: «СДНФ, СКНФ. Минимизация»

Выполнил: студент группы
ИВТ-23-1Б
Долганова Диана Евгеньевна
Проверил: ст. пр.
Рустамханова Г. И.

г. Пермь – 2024

Оглавление

Цель работы

Разработать программу, которая будет принимать на вход 16-значный вектор, выводить таблицу истинности, СДНФ и СКНФ, импликантную матрицу, все короткие покрытия.

Задачи работы

Реализовать следующий функционал:

1. Вводится вектор из 16 символов (функция 4 переменных).
2. Таблица истинности.
3. СДНФ и СКНФ.
4. Импликантная матрица.
5. Вывести все короткие покрытия.

Этапы выполнения

Программа была разработана на языке C++ в программной среде Microsoft Visual Studio 2022.

1. **Определение функций**

Определим функции, которые будут реализованы позже. Каждая функция отвечает за определенную задачу:

- askForVectorInput — спрашивает пользователя, хочет ли он ввести вектор вручную или сгенерировать случайный.
- createVectorManually — создает вектор вручную.
- generateRandomVector — генерирует случайный вектор.
- displayMenu — отображает меню.
- displayVector — выводит вектор на экран.
- displayTruthTable — формирует и отображает таблицу истинности.
- getDNF и getCNF — предполагаемые функции получения ДНФ и КНФ, но они не реализованы в коде.
- displayIntermediateResults, displayMinimalCovers, и другие — функции, связанные с минимизацией логических функций, отображением промежуточных результатов и т.д. (рис. 1).

```cpp
#include <algorithm>
#include <bitset>
#include <cstdlib>
#include <ctime>
#include <iomanip>
#include <iostream>
#include <sstream>
#include <string>
#include <unordered_set>
#include <vector>

using namespace std;

string askForVectorInput();
string createVectorManually();
string generateRandomVector();
void displayMenu();
void displayVector(const string& vector);
void displayTruthTable(const string& vector);
string getDNF(const vector<bool>& boolVector);
string getCNF(const vector<bool>& boolVector);
void displayIntermediateResults(const string& vector);
void displayMinimalCovers(const string& vector);
vector<string> quineMcCluskey(const vector<string>& minterms);
vector<vector<string>> groupByOnesCount(const vector<string>& minterms);
bool canCombine(const string& a, const string& b);
string combine(const string& a, const string& b);
void displayImplicantMatrix(const vector<string>& minterms,
    const vector<string>& primeImplicants);
bool covers(const string& primeImplicant, const string& minterm);
vector<string> getMinimalCovers(const vector<string>& minterms,
    const vector<string>& primeImplicants);
bool coversAll(const vector<int>& cover,
    const vector<vector<bool>>& coverageMatrix,
    const vector<string>& minterms);
string getMintermName(const string& minterm);
string getImplicantName(const string& implicant);
std::vector<std::vector<int>> beforePCNF;
std::vector<std::vector<int>> beforePDNF;
void createThuthTable(const string& vector);
void displayFormulas();
```

Рисунок 1 - Определение функций

2. Главная функция

askForVectorInput(); — вызывает функцию для ввода вектора.

while (!exit) — запускает цикл, который будет выполняться, пока пользователь не выберет выход.

Внутри цикла программа отображает меню и считывает выбор пользователя. switch определяет, какая функция должна быть вызвана в зависимости от выбора:

1. Выводит текущий вектор.
2. Формирует таблицу истинности.
3. Создает таблицу истинности и выводит формулы.
4. Отображает промежуточные результаты.
5. Отображает минимальные покрытия.
6. Позволяет пользователю ввести новый вектор.
7. Выход из программы.

Если пользователь вводит некорректный выбор, программа сообщает об этом (рис. 2).

```
int main() {
    setlocale(LC_ALL, "RU");
    string vector = askForVectorInput();
    bool exit = false;

    while (!exit) {
        displayMenu();
        string choice;
        cin >> choice;

        switch (stoi(choice)) {
        case 1:
            displayVector(vector);
            break;
        case 2:
            displayTruthTable(vector);
            break;
        case 3:
            createThuthTable(vector);
            displayFormulas();
            break;
        case 4:
            displayIntermediateResults(vector);
            break;
        case 5:
            displayMinimalCovers(vector);
            break;
        case 6:
            vector = askForVectorInput();
            break;
        case 7:
            exit = true;
            break;
        default:
            cout << "Неверный выбор. Пожалуйста, попробуйте снова." << endl;
            break;
        }
    }

    return 0;
}
```

Рисунок 2 - Главная функция

3. Функции работы с вектором

3.1 Запрос ввода вектора

Функция спрашивает пользователя, как он хочет создать вектор. Если ввод 1, вызывается createVectorManually() для ручного ввода, иначе вызывается generateRandomVector() для генерации случайного вектора. (рис. 3).

```
string askForVectorInput() {
    cout << "Хотите создать вектор вручную или случайным образом? (1 - вручную/2 "
        "- случайно)"
        << endl;
    string choice;
    cin >> choice;

    if (choice == "1") {
        return createVectorManually();
    }
    else {
        return generateRandomVector();
    }
}
```

Рисунок 3 - Функция askForVectorInput()

3.2 Создание вектора вручную

Здесь пользователь вводит вектор. Если вектор не соответствует требованиям (должен содержать ровно 16 символов и только '0' или '1'), выводится сообщение об ошибке, и функция запрашивает ввод снова (рис. 4).

```
string createVectorManually() {
    cout << "Введите вектор из 16 символов (0 или 1):" << endl;
    string input;
    cin >> input;

    if (input.length() != 16 || !all_of(input.begin(), input.end(), [](char c) {
        return c == '0' || c == '1';
        })) {
        cout
            << "Неверный ввод. Вектор должен содержать ровно 16 символов (0 или 1)."
            << endl;
        return createVectorManually();
    }

    return input;
}
```

Рисунок 4 - Функция createVectorManually()

3.3 Генерация случайного вектора

Эта функция создает строку длиной 16 символов, заполненную случайными '0' и '1'. srand(time(0)) инициализирует генератор случайных чисел, а цикл заполняет вектор случайными значениями (рис. 5).

```
string generateRandomVector() {
    srand(static_cast<unsigned int>(
        time(0)));
    string vector(16, '0');

    for (int i = 0; i < 16; i++) {
        vector[i] = rand() % 2 == 0 ? '0' : '1';
    }

    return vector;
}
```

Рисунок 5 - Функция generateRandomVector()

4. Отображение меню

Данная функция выводит список доступных опций для пользователя (рис. 5).

```
void displayMenu() {
    cout << "\nМеню:" << endl;
    cout << "1. Вывести созданный вектор" << endl;
    cout << "2. Таблица истинности" << endl;
    cout << "3. СДНФ и СКНФ в виде формул" << endl;
    cout << "4. Промежуточные результаты склеек и импликантная матрица методом "
        "Квайна"
        << endl;
    cout << "5. Все минимальные покрытия (минимизация)" << endl;
    cout << "6. Создать новый вектор" << endl;
    cout << "7. Выход" << endl;
    cout << "Выберите опцию: ";
}
```

Рисунок 6 - Функция displayMenu()

5. Отображение вектора

Эта функция принимает вектор в виде строки и выводит его на экран (рис. 6).

```
void displayVector(const string& vector) {
    cout << "\nСозданный вектор:" << endl;
    cout << vector << endl;
}
```

Рисунок 6 - Функция displayVector()

6. Таблица истинности

Функция displayTruthTable() создает таблицу истинности, выводя значения переменных x1, x2, x3, и x4, а также значения функции F из вектора. bitset<4>(i).to_string() преобразует индекс i в бинарную строку длиной 4. Функция createTruthTable() создает таблицы, используемые для получения СКНФ и СДНФ, на основе значения вектора. Она заполняет вектор tmp значениями переменных для каждого индекса, а затем добавляет tmp в соответствующий вектор в зависимости от значения F (рис. 7).

```
void displayTruthTable(const string& vector) {
    cout << "\nТаблица истинности:" << endl;
    cout << "x1 x2 x3 x4 | F" << endl;
    for (int i = 0; i < 16; i++) {
        string binary = bitset<4>(i).to_string();
        cout << binary[0] << "  " << binary[1] << "  " << binary[2] << "  "
            << binary[3] << "  | " << vector[i] << "   " << endl;
    }
}

void createThuthTable(const string& vector) {
    std::vector<int> tmp;

    for (int i(0), tableIdx(0); i < vector.size(); ++i) {
        for (int j(log2(vector.size()) - 1); j >= 0; --j) {
            tmp.push_back((i >> j) & 1);
        }
        (vector[tableIdx++] == '0') ? beforePCNF.push_back(tmp)
            : beforePDNF.push_back(tmp);
        tmp.clear();
    }
}
```

Рисунок 7 - Функции displayTruthTable() и createTruthTable()

7. Отображение формул

Функция выводит как СКНФ, так и СДНФ на основе результатов, сохраненных в beforePCNF и beforePDNF. Она формирует строки формул на

основе значений, выводя соответствующие переменные и их отрицания (рис. 8).

```
void displayFormulas() {
    std::cout << "СКНФ" << std::endl;
    for (int i = 0; i < beforePCNF.size(); ++i) {
        for (int j = 0; j < beforePCNF[i].size(); ++j) {
            if (beforePCNF[i][j] > 0) {
                std::cout << "x" << j + 1;
            }
            else {
                std::cout << "¬x" << j + 1;
            }
            if (j != 3)
                std::cout << "∧";
        }
        if (i != beforePCNF.size() - 1) {
            std::cout << " * ";
        }
    }
    std::cout << '\n';

    std::cout << "СДНФ" << std::endl;
    for (int i = 0; i < beforePDNF.size(); ++i) {
        for (int j = 0; j < beforePDNF[i].size(); ++j) {
            if (beforePDNF[i][j] > 0) {
                std::cout << "x" << j + 1;
            }
            else {
                std::cout << "¬x" << j + 1;
            }
        }
        if (i != beforePDNF.size() - 1) {
            std::cout << " + ";
        }
    }
    std::cout << '\n';

    beforePDNF.clear();
    beforePCNF.clear();
}
```

Рисунок 8 - Функция displayFormulas()

8. Промежуточные результаты склеек

Функция displayIntermediateResults выводит промежуточные результаты минимизации. Проверяется длина таблицы истинности. Она должна быть 16 бит (для 4 переменных). Преобразуются все индексы, соответствующие единичным значениям в таблице истинности, в строки, представляющие минтермы. Далее вызывается функция quineMcCluskey для нахождения первичных импликантов. Результаты отображаются, включая импликантную матрицу (рис. 9).

```
void displayIntermediateResults(const string& truthTable) {
    cout << "\nПромежуточные результаты склеек и импликантная матрица методом "
        "Квайна:"
        << endl;
    vector<string> minterms;

    if (truthTable.length() != 16) {
        cerr << "Ошибка: неверная длина таблицы истинности. Ожидалось 16 бит."
            << endl;
        return;
    }

    for (size_t i = 0; i < truthTable.length(); ++i) {
        if (truthTable[i] == '1') {
            minterms.push_back(bitset<4>(i).to_string());
        }
    }

    vector<string> primeImplicants = quineMcCluskey(minterms);

    if (primeImplicants.empty()) {
        cout << "Не найдено первичных импликантов." << endl;
        return;
    }

    cout << "Промежуточные результаты:" << endl;
    for (const auto& implicant : primeImplicants) {
        cout << implicant << endl;
    }

    cout << "\nИмпликантная матрица:" << endl;
    displayImplicantMatrix(minterms, primeImplicants);
}
```

Рисунок 9 - Функция displayIntermediateResults

9. Минимальные покрытия

Функция displayMinimalCovers выводит минимальные покрытия для заданной таблицы истинности. Проверяется длина таблицы истинности (16 бит). Из таблицы истинности извлекаются минтермы. Вызывается функция quineMcCluskey для нахождения первичных импликантов, и затем вызывается getMinimalCovers для нахождения минимальных покрытий. Результат минимизации выводится в виде ДНФ (рис. 10).

```
void displayMinimalCovers(const string& truthTable) {
    cout << "\nВсе минимальные покрытия (минимизация):" << endl;
    vector<string> minterms;

    if (truthTable.length() != 16) {
        cerr << "Ошибка: неверная длина таблицы истинности. Ожидалось 16 бит."
            << endl;
        return;
    }

    for (size_t i = 0; i < truthTable.length(); ++i) {
        if (truthTable[i] == '1') {
            minterms.push_back(bitset<4>(i).to_string());
        }
    }

    vector<string> primeImplicants = quineMcCluskey(minterms);
    vector<string> minimalCovers = getMinimalCovers(minterms, primeImplicants);

    if (minimalCovers.empty()) {
        cout << "Не найдено минимальных покрытий." << endl;
        return;
    }

    string result;
    for (const auto& cover : minimalCovers) {
        if (!result.empty())
            result += " v ";
        result += cover;
    }
    cout << "ДНФmin: " << result << endl;
}
```

Рисунок 10 - Функция displayMinimalCovers

10. Алгоритм Квайна-МакКласки

Функция quineMcCluskey реализует основную логику алгоритма Квайна-МакКласки. Митермы группируются по количеству единичных бит с помощью функции groupByOnesCount. Далее происходит попарное объединение минтермов, если они могут быть объединены (функция canCombine). В процессе объединения создаются новые группы, и первичные импликанты добавляются в список (рис. 11).

```
vector<string> quineMcCluskey(const vector<string>& minterms) {
    vector<vector<string>> groups = groupByOnesCount(minterms);
    vector<string> primeImplicants;
    unordered_set<string> uniqueImplicants;

    while (groups.size() > 1) {
        vector<vector<string>> newGroups;
        vector<string> usedMinterms;

        for (size_t i = 0; i < groups.size() - 1; i++) {
            vector<string> combined;
            for (const auto& a : groups[i]) {
                for (const auto& b : groups[i + 1]) {
                    if (canCombine(a, b)) {
                        string combinedTerm = combine(a, b);
                        combined.push_back(combinedTerm);
                        usedMinterms.push_back(a);
                        usedMinterms.push_back(b);
                    }
                }
            }
            if (!combined.empty()) {
                newGroups.push_back(combined);
            }
        }

        for (const auto& group : groups) {
            for (const auto& minterm : group) {
                if (find(usedMinterms.begin(), usedMinterms.end(), minterm) ==
                    usedMinterms.end()) {
                    if (uniqueImplicants.insert(minterm).second) {
                        primeImplicants.push_back(minterm);
                    }
                }
            }
        }

        groups = newGroups;
    }

    if (groups.size() == 1) {
        for (const auto& implicant : groups[0]) {
            if (uniqueImplicants.insert(implicant).second) {
                primeImplicants.push_back(implicant);
            }
        }
    }

    return primeImplicants;
}
```

Рисунок 11 - Функция quineMcCluskey

11. Группировка минтерм

Функция groupByOnesCount группирует минтермы по количеству единичных бит. Для каждого минтерма считается количество единиц, и минтерм добавляется в соответствующую группу. Пустые группы удаляются (рис. 12)

```
vector<vector<string>> groupByOnesCount(const vector<string>& minterms) {
    vector<vector<string>> groups(5);

    for (const auto& minterm : minterms) {
        int onesCount = count(minterm.begin(), minterm.end(), '1');
        groups[onesCount].push_back(minterm);
    }

    groups.erase(
        remove_if(groups.begin(), groups.end(),
            [](const vector<string>& group) { return group.empty(); }),
        groups.end());
    return groups;
}
```

Рисунок 12 - Функция groupByOnesCount

12. Объединение минтерм

Функция canCombine проверяет, могут ли два минтерма быть объединены (различие должно быть только в одном бите). Функция combine объединяет два минтерма, заменяя различные биты на знак - (рис. 13).

```
bool canCombine(const string& a, const string& b) {
    int diffCount = 0;
    for (size_t i = 0; i < a.length(); i++) {
        if (a[i] != b[i]) {
            diffCount++;
        }
    }
    return diffCount == 1;
}

string combine(const string& a, const string& b) {
    string result(a.length(), '-');
    for (size_t i = 0; i < a.length(); i++) {
        if (a[i] == b[i]) {
            result[i] = a[i];
        }
    }
    return result;
}
```

Рисунок 13 - Функции canCombine и combine

13.Импликантная матрица

Функция отображает импликантную матрицу, которая показывает, какие минтермы покрываются каждым импликантом. Создается матрица покрытия и выводится на экран, где каждый элемент матрицы представляет, покрывает ли импликант минтерм (рис. 14).

```
void displayImplicantMatrix(const vector<string>& minterms,
    const vector<string>& primeImplicants) {
    vector<vector<bool>> coverageMatrix(primeImplicants.size(),
        vector<bool>(minterms.size(), false));

    for (size_t i = 0; i < primeImplicants.size(); i++) {
        for (size_t j = 0; j < minterms.size(); j++) {
            coverageMatrix[i][j] = covers(primeImplicants[i], minterms[j]);
        }
    }

    cout << "       ";
    for (const auto& minterm : minterms) {
        cout << setw(5) << minterm;
    }
    cout << endl;

    for (size_t i = 0; i < primeImplicants.size(); i++) {
        cout << setw(5)
            << primeImplicants[i];
        for (size_t j = 0; j < minterms.size(); j++) {
            string cellValue = coverageMatrix[i][j] ? "+" : " ";
            cout << setw(5) << cellValue;
        }
        cout << endl;
    }
}
```

Рисунок 14 - Функция displayImplicantMatrix

14. Вспомогательные функции

Функция covers проверяет, покрывает ли импликант минтерм

Алгоритм:

1. Для каждой позиции в строках primeImplicant и minterm выполняется проверка:
    1.1. Если символ в primeImplicant не равен '-' (заполнитель, означающий "не важно") и не равен соответствующему символу в minterm, то возврат false (импликант не покрывает минтерм).
    1.2. Если символы совпадают или в импликанте стоит '-', продолжаем проверку.
2. Если все проверки прошли успешно, возвращаем true, что импликант покрывает минтерм (рис. 15).

Функция getMinimalCovers находит минимальные покрытия (минимальные ДНФ выражения), используя матрицу покрытия и первичные

импликанты. Функция coversAll проверяет, покрывает ли набор импликантов все минтермы (рис. 16-17).

```
bool covers(const string& primeImplicant, const string& minterm) {
    for (size_t i = 0; i < primeImplicant.length(); i++) {
        if (primeImplicant[i] != '-' && primeImplicant[i] != minterm[i]) {
            return false;
        }
    }
    return true;
}
```

Рисунок 15 - Функция covers

```
vector<string> getMinimalCovers(const vector<string>& minterms,
    const vector<string>& primeImplicants) {
    vector<vector<bool>> coverageMatrix(primeImplicants.size(),
        vector<bool>(minterms.size(), false));

    for (size_t i = 0; i < primeImplicants.size(); i++) {
        for (size_t j = 0; j < minterms.size(); j++) {
            coverageMatrix[i][j] = covers(primeImplicants[i], minterms[j]);
        }
    }

    vector<vector<int>> covers;
    for (size_t i = 0; i < primeImplicants.size(); i++) {
        covers.push_back({ static_cast<int>(i) });
    }

    for (size_t i = 0; i < covers.size(); i++) {
        for (size_t j = i + 1; j < covers.size(); j++) {
            if (coversAll(covers[i], coverageMatrix, minterms) &&
                coversAll(covers[j], coverageMatrix, minterms)) {
                if (covers[i].size() > covers[j].size()) {
                    covers.erase(covers.begin() + i);
                    i--;
                    break;
                }
                else if (covers[i].size() < covers[j].size()) {
                    covers.erase(covers.begin() + j);
                    j--;
                }
            }
        }
    }

    vector<string> minimalCovers;
    for (const auto& cover : covers) {
        string result;
        for (int index : cover) {
            if (!result.empty())
                result += " + ";
            result += primeImplicants[index];
        }
        minimalCovers.push_back(result);
    }

    return minimalCovers;
}
```

Рисунок 16 - Функция getMinimalCovers

```
bool coversAll(const vector<int>& cover,
    const vector<vector<bool>>& coverageMatrix,
    const vector<string>& minterms) {
    vector<bool> covered(minterms.size(), false);
    for (int index : cover) {
        for (size_t j = 0; j < minterms.size(); j++) {
            if (coverageMatrix[index][j]) {
                covered[j] = true;
            }
        }
    }
    return all_of(covered.begin(), covered.end(), [](bool c) { return c; });
}
```

Рисунок 17 - Функция coversAll

Функция getMintermName преобразует минтерм (в виде строки) в его "имя", представляя каждую переменную в виде x1, !x1 и т.д (рис. 18).

```
string getMintermName(const string& minterm) {
    string result;
    for (size_t i = 0; i < minterm.length(); ++i) {
        result += (minterm[i] == '1' ? "x" : "!x") + to_string(i + 1);
    }
    return result;
}
```

Рисунок 18 - Функция getMintermName

Функция getImplicantName преобразует импликант (в виде строки) в его "имя" в формате, где каждый 1 становится x<номер переменной>, каждый 0 — !x<номер переменной>, а - (заполнитель) остается как есть (рис. 19).

```
string getImplicantName(const string& implicant) {
    string result;
    for (size_t i = 0; i < implicant.length(); ++i) {
        if (implicant[i] == '1') {
            result += "x" + to_string(i + 1);
        }
        else if (implicant[i] == '0') {
            result += "!x" + to_string(i + 1);
        }
        else {
            result += "-";
        }
    }
    return result;
}
```

Рисунок 19 - Функция getImplicantName

Тестирование программы

Для проверки программы введем случайный вектор и проверим работу всех функций на корректность (рис. 20 - 13).

```
Хотите создать вектор вручную или случайным образом? (1 - вручную/2 - случайно)
2

Меню:
1. Вывести созданный вектор
2. Таблица истинности
3. СДНФ и СКНФ в виде формул
4. Промежуточные результаты склеек и импликантная матрица методом Квайна
5. Все минимальные покрытия (минимизация)
6. Создать новый вектор
7. Выход
Выберите опцию: 1

Созданный вектор:
0111100001010011
```

Рисунок 20 - Получение случайного вектора

```
Выберите опцию: 2

Таблица истинности:
x1 x2 x3 x4 | F
0  0  0  0  | 0
0  0  0  1  | 1
0  0  1  0  | 1
0  0  1  1  | 1
0  1  0  0  | 1
0  1  0  1  | 0
0  1  1  0  | 0
0  1  1  1  | 0
1  0  0  0  | 0
1  0  0  1  | 1
1  0  1  0  | 0
1  0  1  1  | 1
1  1  0  0  | 0
1  1  0  1  | 0
1  1  1  0  | 1
1  1  1  1  | 1
```

Рисунок 21 - Получение таблицы истинности

```
СКНФ
¬x1?¬x2?¬x3?¬x4 * ¬x1?x2?¬x3?x4 * ¬x1?x2?x3?¬x4 * ¬x1?x2?x3?x4 * x1?¬x2?¬x3?¬x4 * x1?¬x2?x3?¬x4 * x1?x2?¬x3?¬x4 * x1?x2?
¬x3?x4
СДНФ
¬x1¬x2¬x3x4 + ¬x1¬x2x3¬x4 + ¬x1¬x2x3x4 + ¬x1x2¬x3¬x4 + x1¬x2¬x3x4 + x1¬x2x3x4 + x1x2x3¬x4 + x1x2x3x4
```

Рисунок 22 - Получение СКНФ и СДНФ

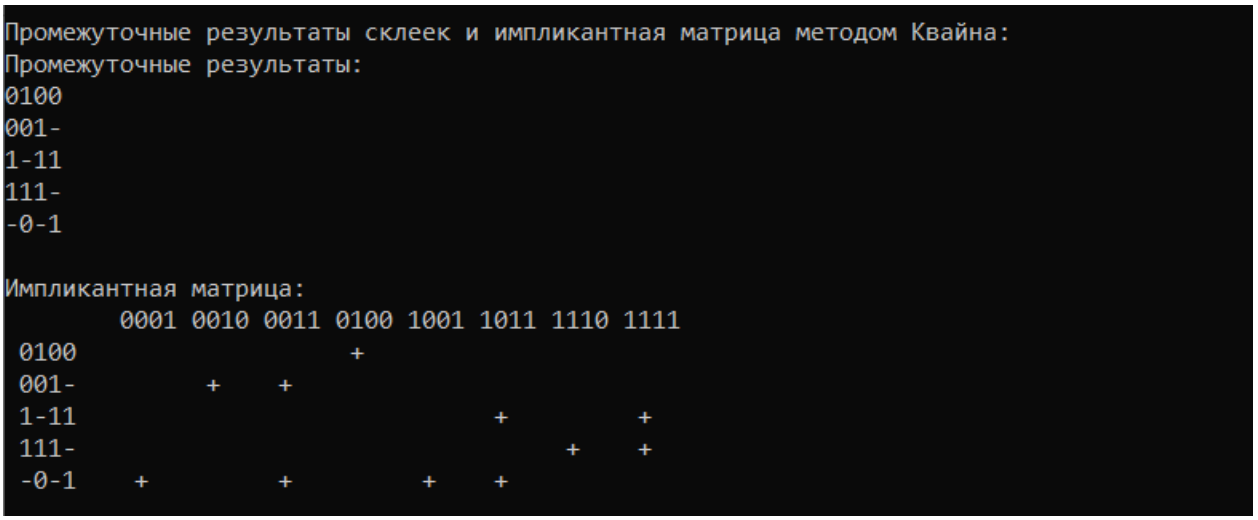

Рисунок 23 - Получение промежуточных результатов и импликантной матрицы

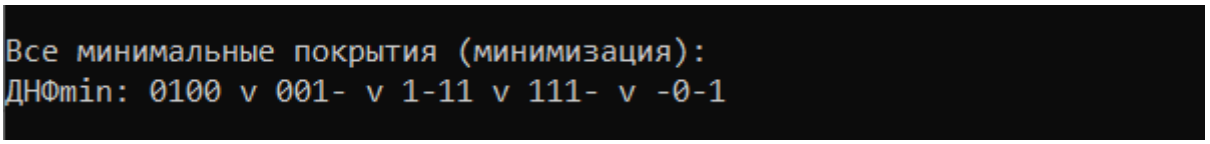

Рисунок 24 - Получение всех минимальных покрытий

Заключение

В ходе работы была разработана программа, которая позволяет ввести вектор из 16 символов, вывести его таблицу истинности, СКНФ и СДНФ, промежуточные результаты и импликантную матрицу и все минимальные покрытия.

Список используемой литературы

1. Построение СКНФ и СДНФ по таблице истинности URL: https://spravochnick.ru/informatika/algebra_logiki_logika_kak_nauka/postroenie_sknf_i_sdnf_po_tablice_istinnosti/ (дата обращения: 04.12.2024).
2. Минимизация логических функций методом Квайна URL: https://function-x.ru/minimizacija_logicheskih_funkcij_quine.html (дата обращения: 05.12.2024)
3. 8.2. Метод Квайна и импликантные матрицы URL: https://studfile.net/preview/996149/page:29/ (дата обращения: 05.12.2024)
4. 11. Минимизация булевых функций URL: https://ido.tsu.ru/iop_res/bulevfunc/text/g11_2_4.html (дата обращения: 06.12.2024)