Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №3**

Дисциплина: «Дискретная математика и математическая

логика»

Тема: Минимизация логических функций методом Квайна

Выполнил работу

студент группы ИВТ-22-2б

Казанцев А.В.

Проверила

Старший преподаватель кафедры ИТАС

Рустамханова Г.И.

Пермь 2023

**Цель работы**

1. Создание консольного приложения «Минимизация логических функций методом Квайна»

2. Изучение теории дискретной математики об свойствах логических функций

3. Изучение и отработка навыков минимальных форм

**Постановка задачи**

1. Программа должен иметь возможность задавать вектор функции из файла или через клавиатуру
2. Программа должна выполнять следующие функции:
   1. Создание и вывод на экран таблицы истинности
   2. Получать совершенную дизъюнктивную нормальную форму (СДНФ)
   3. Выполнять неполное склеивание
   4. Создавать и выводить на экран импликантную матрицу
   5. Проводить минимальное покрытие, и выводить минимальную форму.

**Анализ задачи**

1. Определение типа заданных выражений.

|  |  |  |  |
| --- | --- | --- | --- |
| Пункт | Входные данные | Классы входных данных | Выходные данные |
| 1 | Файл содержащий вектор функции | Строка=16 символов | Матрица NxN |
| Количество файлов – целое число | matrixCount>=1 | Проверка нескольких матриц последовательно |

**Ход работы**

Программа имеет возможность открыть файл с подходящий имеем.  
программа считывает файл в строку. Нормализируется к виду из 16 символов. Файл закрывается.

После проходит вычисление минимальной формы.

Первый шаг создание таблицы истинности.

Второй шаг получение совершенной дизъюнктивной нормальной формы(СДНФ).

Третий шаг проведение операции «склейки» и получить сокращенную форму.

Четвертый шаг создать импликантную матрицу.

Пятый шаг получить минимальную форму путем минимального покрытия.

Методы, используемые в программе:

ReadVectorFile(string path)– метод открывает и считывает файл в строку после форматирует строку . Возвращает строку из 16 символов 0 или 1.

ReadVector ()– метод считывает с клавиатуры строку после форматирует строку . Возвращает строку из 16 символов 0 или 1.

WriteTruthTable(string vector) выводит на экран таблицу истиности.

CreateDNF(string vector) -метод вычисляет СДНФ. Возвращает члены СДНФ в массиве строк.

PrintDNF(String[] SDNF) метод выводит СДНФ в консоль.

GluingSDNF(String[] SDNF) – возвращает сокращённую форму. Получает СДНФ, «склеивает» элементы пока это возможно.

CreateCoverageСable(string[] SDNF, string[] MDNF) возвращает импликантную матрицу. Получает СДНФ и сокращённую форму.

ShowCoverageСable(string[,] CoverTab, string[] SDNF, string[] MDNF) выводит на экран импликантную матрицу.

MinimalForm(String[] strings, String[,] table) получает импликантную матрицу и сокращённую форму. Возвращает минимальную форму, которая покрывает импликантную матрицу.

Дополнительные методы, используемые в программе:

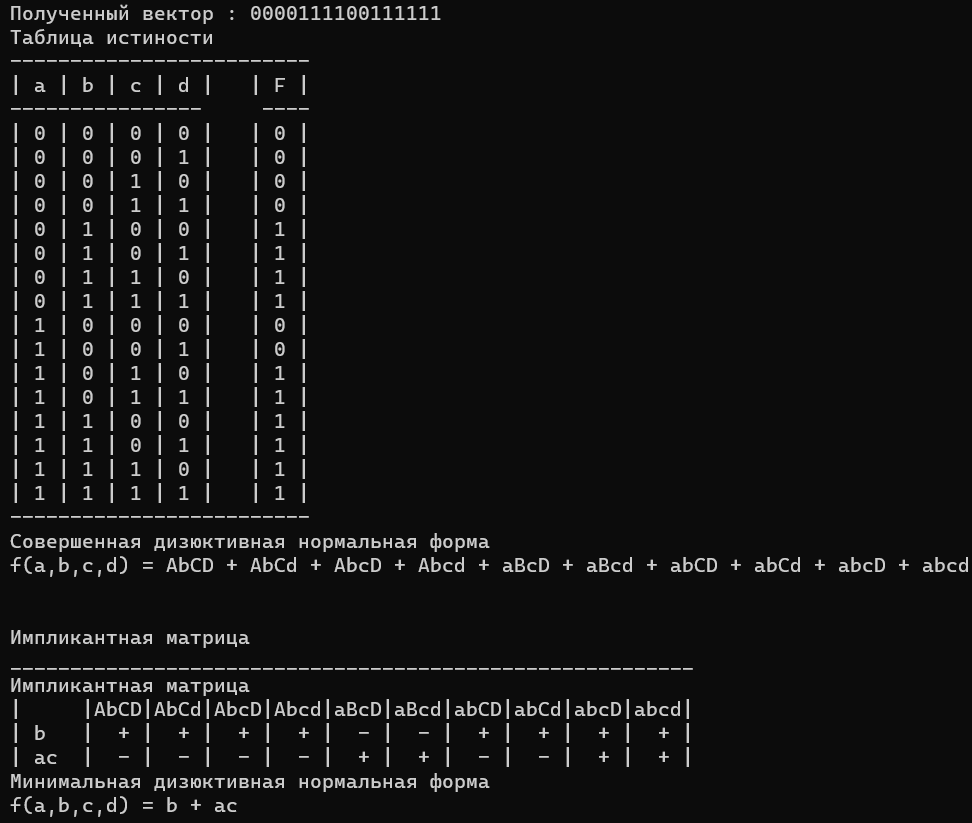
ADDelement(int[] index, int element)- добавляет в конец массива элемент.

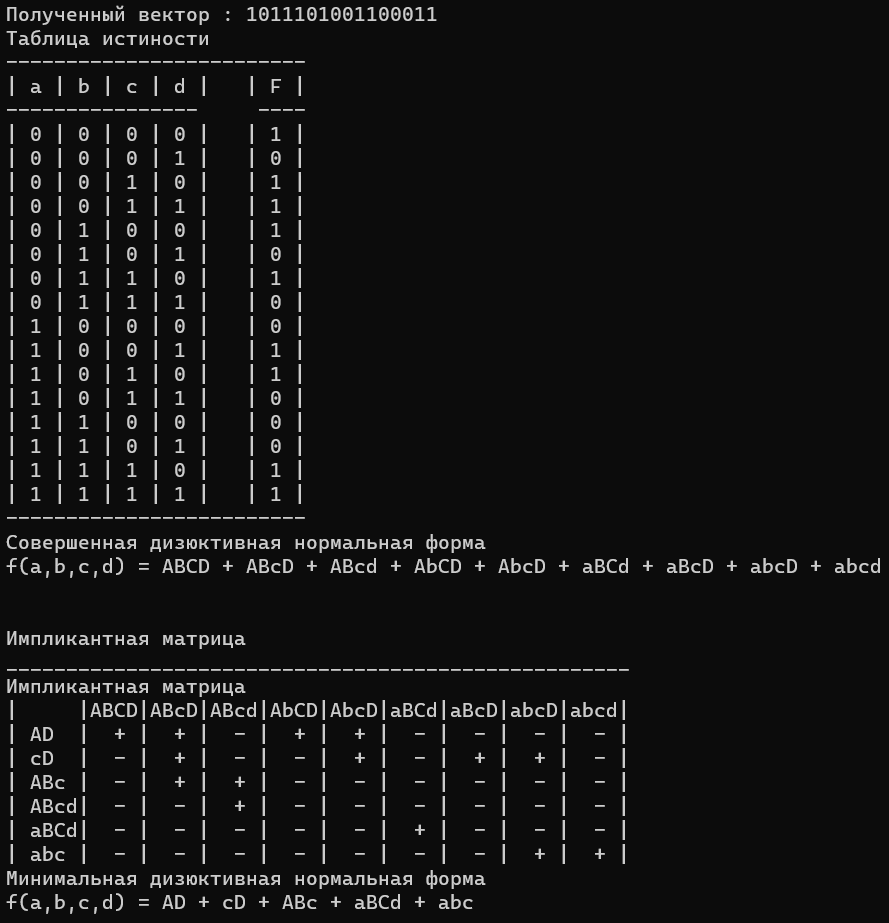
Gluing(string a, string b) проверяет на соответствие и проводит «склеивание» двух слов. Возвращает строку с полученным результатом.

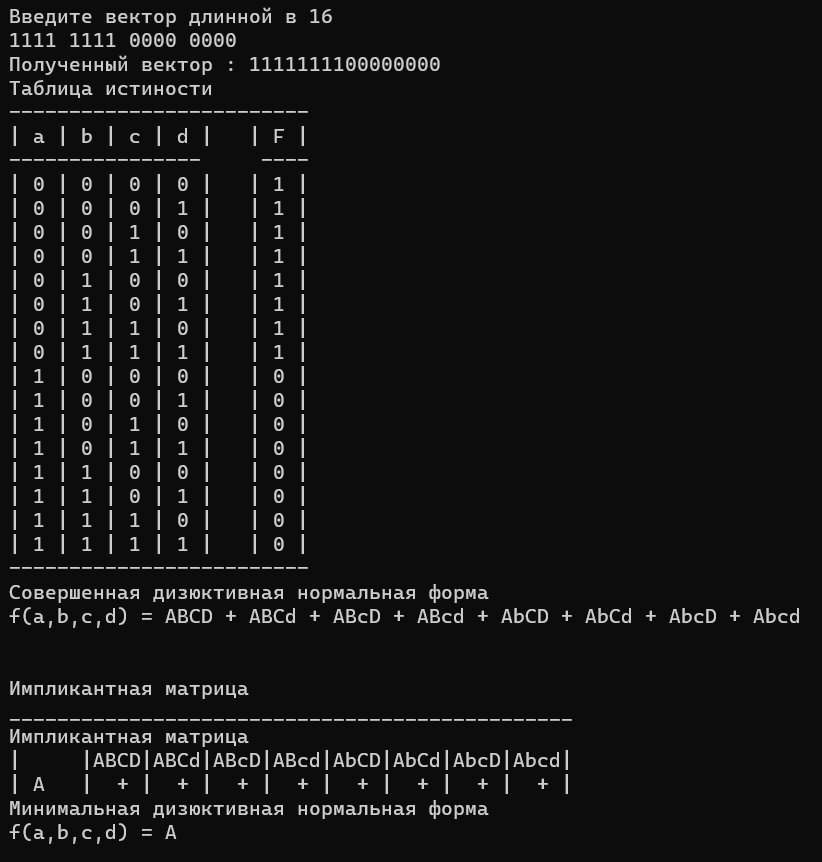
DelElement (String[] strings, params string [] elements) – удаляет из массива все элементы указанные в массиве elements.

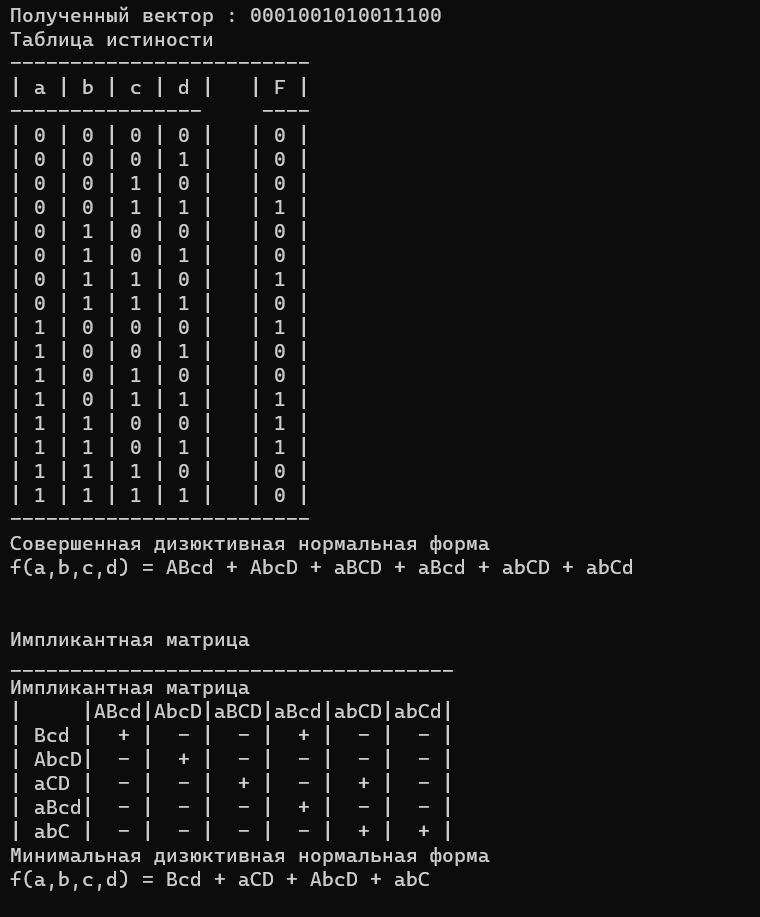
DelRepeatElement(String[] array) удаляет повторы из массива.

Тесты











**Код программы**

using System.Text.RegularExpressions;

internal class Program

{

static string path = "v1.txt";

private static void Main(string[] args)

{

string vector = "0000111100111111";

AnalysesFunction(vector);

vector = "1011101001100011";

AnalysesFunction(vector);

while (true)

{

vector = ReadVector();

AnalysesFunction(vector);

}

}

public static void AnalysesFunction(string vector)

{

Console.WriteLine("Полученный вектор : " + vector);

WriteTruthTable(vector);

string[] SDNF = CreateDNF(vector);

Console.WriteLine("Совершенная дизюктивная нормальная форма");

PrintDNF(SDNF);

string[] SNF = GluingSDNF(SDNF);

string[,] CoverTab = CreateCoverageСable(SDNF, SNF);

Console.WriteLine("Импликантная матрица");

ShowCoverageСable(CoverTab, SDNF, SNF);

Console.WriteLine("Минимальная дизюктивная нормальная форма");

PrintDNF(MinimalForm(SNF, CoverTab));

Console.WriteLine("Нажмите любую кнопку для продолжения.\nДля выхода нажмите esc");

if (Console.ReadKey(true).Key == ConsoleKey.Escape) Environment.Exit(0);

}

public static string[,] CreateCoverageСable(string[] SDNF, string[] MDNF)

{

string[,] CoverTab = new string[MDNF.Length, SDNF.Length];

for (int i = 0; i < CoverTab.GetLength(0); i++)

{

for (int j = 0; j < CoverTab.GetLength(1); j++)

{

int cou = 0;

foreach (char Ca in SDNF[j])

{

bool isCheck = Regex.IsMatch(MDNF[i], Ca.ToString());

if (isCheck) { cou++; }

}

if (cou == MDNF[i].Length)

{

CoverTab[i, j] = "+";

}

else { CoverTab[i, j] = "-"; }

}

}

return CoverTab;

}public static void ShowCoverageСable(string[,] CoverTab, string[] SDNF, string[] MDNF)

{

Console.Write(new string('\_', 5 \* SDNF.Length + 7));

Console.Write($"\nИмпликантная матрица\n| |");

for (int i = 0; i < SDNF.Length; i++)

{

Console.Write($"{SDNF[i]}|");

}

int jm = 0;

for (int i = 0; i < CoverTab.GetLength(0); i++)

{

Console.Write($"\n| {MDNF[jm]}" + new string(' ', 4 - MDNF[jm++].Length) + "|");

for (int j = 0; j < CoverTab.GetLength(1); j++)

{

Console.Write($" {CoverTab[i, j]} |");

}

}

Console.Write("\n");

}

public static string Gluing(string a, string b)

{

int cou = 0;

string chars = "";

if (!String.Equals(a, b, StringComparison.OrdinalIgnoreCase))

return "";

foreach (char Ca in a)

{

bool isCheck = Regex.IsMatch(b, Ca.ToString());

if (isCheck) { cou++; }

else { chars = Ca.ToString(); }

}

if (cou + 1 == a.Length)

{

return Regex.Replace(a, chars, "");

}

return "";

}

public static string ReadVectorFile(string path)

{

string vector = File.ReadAllText(path);

vector = Regex.Replace(vector, "[^01]", "");

while (vector.Length != 16)

{

if (vector.Length < 16)

vector += "0";

if (vector.Length > 16)

vector = vector.Remove(vector.Length - 1);

}

return vector;

}

public static string ReadVector()

{

Console.WriteLine("Введите вектор длинной в 16");

string vector = Console.ReadLine();

vector = Regex.Replace(vector, "[^01]", "");

while (vector.Length != 16)

{

if (vector.Length < 16)

vector += "0";

if (vector.Length > 16)

vector = vector.Remove(vector.Length - 1);

}

return vector;

}

public static void WriteTruthTable(string vector)

{

int i = 0;

Console.WriteLine($"Таблица истиности\n-------------------------");

Console.WriteLine($"| a | b | c | d | | F | ");

Console.WriteLine($"---------------- ----");

for (int a = 0; a <= 1; a++)

{

for (int b = 0; b <= 1; b++)

{

for (int c = 0; c <= 1; c++)

{

for (int d = 0; d <= 1; d++)

{

Console.WriteLine($"| {a} | {b} | {c} | {d} | | {vector[i++]} | ");

}

}

}

}

Console.WriteLine($"-------------------------");

}

public static string[] CreateDNF(string vector)

{

int i = 0;

int vectrue = Regex.Count(vector, "1");

string[] SDNF = new string[vectrue];

int j = 0;

for (int a = 0; a <= 1; a++)

{

for (int b = 0; b <= 1; b++)

{

for (int c = 0; c <= 1; c++)

{

for (int d = 0; d <= 1; d++)

{

if (vector[i++] == '1')

{

if (a == 0) SDNF[j] += "A";

else SDNF[j] += "a";

if (b == 0) SDNF[j] += "B";

else SDNF[j] += "b";

if (c == 0) SDNF[j] += "C";

else SDNF[j] += "c";

if (d == 0) SDNF[j] += "D";

else SDNF[j] += "d";

j++;

}

}

}

}

}

if (j == 0) return new string[0];

return SDNF;

}

public static void PrintDNF(String[] SDNF)

{

if (SDNF.Length == 0) return;

Console.Write($"f(a,b,c,d) = {SDNF[0]}");

for (int k = 1; k < SDNF.Length; k++)

{

Console.Write(" + " + SDNF[k]);

}

Console.WriteLine("\n\n");

}

public static string[] GluingSDNF(String[] SDNF)

{

string[] Mdnf = new string[0];

string[] resalt = new string[0];

bool isNotExit = true;

for (int i = 0; i < SDNF.Length; i++)

{

int iter = 0;

for (int j = i + 1; j < SDNF.Length; j++)

{

string word = Gluing(SDNF[i], SDNF[j]);

if (word.Length != 0)

{

Mdnf = ADDelement(Mdnf, word);

iter++;

}

}

if (iter == 0 && i != SDNF.Length - 1) Mdnf = ADDelement(Mdnf, SDNF[i]);

}

SDNF = arrayClone(Mdnf);

while (isNotExit)

{

isNotExit = false;

resalt = new string[0];

for (int i = 0; i < SDNF.Length; i++)

{

for (int j = i + 1; j < SDNF.Length; j++)

{

string word = Gluing(SDNF[i], SDNF[j]);

if (word.Length != 0)

{

resalt = ADDelement(resalt, word);

isNotExit = true;

Mdnf = DelElement(Mdnf, SDNF[i], SDNF[j]);

}

}

}

foreach (var item in Mdnf)

{

resalt = ADDelement(resalt, item);

}

resalt = DelRepeatElement(resalt);

SDNF = arrayClone(resalt);

Mdnf = arrayClone(resalt);

}

return DelRepeatElement(resalt);

}

public static string[] DelRepeatElement(String[] array)

{

string[] strings1 = arrayClone(array);

for (int i = 0; i < array.Length; i++)

{

for (int j = i + 1; j < array.Length; j++)

{

if (array[i] == array[j])

{

strings1 = DelElement(strings1, false, array[j]);

break;

}

}

}

return strings1;

}

public static string[] DelElement(String[] strings, params string[] elements)

{

string[] strings1 = new string[0];

for (int i = 0; i < strings.Length; i++)

{

bool isCheck = true;

for (int j = 0; j < elements.Length && isCheck; j++)

isCheck = isCheck && strings[i] != elements[j];

if (isCheck)

strings1 = ADDelement(strings1, strings[i]);

}

return strings1;

}

public static string[] DelElement(String[] strings, bool isAll = true, params string[] elements)

{

string[] strings1 = new string[0];

bool iswithoutCheck = false;

foreach (var item in elements)

{

for (int i = 0; i < strings.Length; i++)

{

if (strings[i] != item || iswithoutCheck)

strings1 = ADDelement(strings1, strings[i]);

else if (!isAll) iswithoutCheck = true;

}

}

return strings1;

}

public static string[] ADDelement(String[] strings, string element)

{

string[] strings1 = new string[strings.Length + 1];

for (int i = 0; i < strings.Length; i++)

strings1[i] = strings[i];

strings1[strings1.Length - 1] = element;

return strings1;

}

public static int[] ADDelement(int[] index, int element)

{

int[] index1 = new int[index.Length + 1];

for (int i = 0; i < index.Length; i++)

index1[i] = index[i];

index1[index1.Length - 1] = element;

return index1;

}

public static string[] MinimalForm(String[] strings, String[,] table)

{

int[] index = new int[0];

int[] index1 = new int[0];

int ind = 0; int[] buf = new int[0];

int count = 0; int max = 0;

string[] MinForm = new string[0];

for (int i = 0; i < table.GetLength(0); i++)

{

for (int j = 0; j < table.GetLength(1); j++)

{

if (table[i, j] == "+")

count++;

else buf = ADDelement(buf, j);

}

if (count > max)

{

max = count;

index = arrayClone(buf);

ind = i;

}

count = 0;

buf = new int[0];

}

MinForm = ADDelement(MinForm, strings[ind]);

max = 0;

while (index.Length > 0)

{

for (int i = 0; i < table.GetLength(0); i++)

{

foreach (var j in index)

{

if (table[i, j] == "+")

count++;

else buf = ADDelement(buf, j);

}

if (count > max)

{

max = count;

index1 = arrayClone(buf);

ind = i;

}

count = 0;

buf = new int[0];

}

index = arrayClone(index1);

index1 = new int[0];

max = 0;

MinForm = ADDelement(MinForm, strings[ind]);

}

foreach (int i in index)

{

Console.WriteLine(i);

}

return MinForm;

}

public static int[] arrayClone(int[] array, int element = 0) { return array; }

public static string[] arrayClone(string[] array, int element = 0) { return array; }

}