Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Дискретная математика и математическая логика»

Тема: «Метод Квайна»

Семестр 2

Выполнил работу

Студент группы ИВТ-23-2б

Меновщиков Глеб Николаевич

Проверил

старший преподаватель кафедры ИТАС

Рустамханова Гульшат Ильдаровна

**Цель программы**

Цель данной программы — реализовать метод Квайна для минимизации логических функций, представленных в виде таблицы истинности. Программа позволяет загружать вектор значений из текстового файла, строить таблицу истинности, находить стандартную дизъюнктивную нормальную форму (СДНФ) и минимизировать её с помощью метода Квайна, а также выводить результаты в удобочитаемом формате.

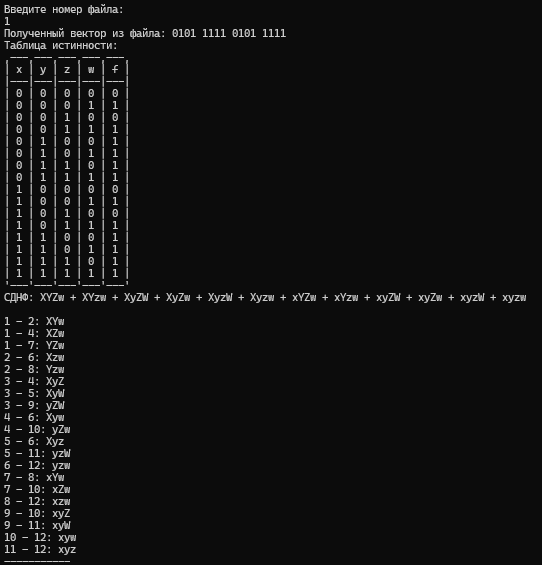
**Описание работы программы**

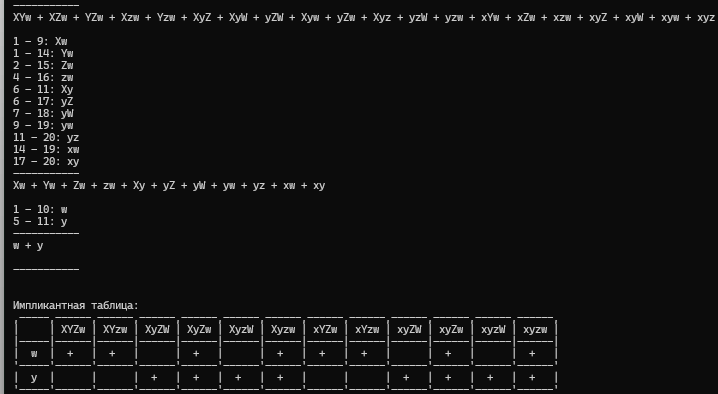
1. **Ввод номера файла**: Программа запрашивает у пользователя номер файла, из которого будет загружен вектор значений. Ввод проверяется на корректность, и программа продолжает запрашивать номер до тех пор, пока не будет введён корректный целочисленный номер.
2. **Загрузка вектора из файла**: После получения номера файла программа пытается загрузить вектор из текстового файла, который должен содержать 16 символов (0 или 1). Если файл не существует, программа уведомляет об этом и повторяет запрос.
3. **Построение таблицы истинности**: Программа создает таблицу истинности на основе загруженного вектора. Она выводит значения переменных (x, y, z, w) и соответствующий результат функции f для всех возможных комбинаций входных переменных.
4. **Получение стандартной дизъюнктивной нормальной формы (СДНФ)**: Программа анализирует таблицу истинности и формирует массив строк, представляющих СДНФ, которая включает все комбинации переменных, для которых функция f равна 1.
5. **Минимизация с помощью метода Квайна**: Программа применяет метод Квайна для минимизации СДНФ. Она ищет пары термов, которые могут быть объединены, и создает новый массив, содержащий минимизированные термы. Процесс повторяется до тех пор, пока не останется термов, которые можно объединить.
6. **Вывод результатов**: Программа выводит полученный вектор, таблицу истинности, СДНФ и минимизированные термы. Также она формирует и отображает импликантную таблицу, которая показывает, какие термы покрывают какие значения функции.
7. **Проверка на наличие термов**: Программа включает функции для проверки наличия термов в массивах, что позволяет избежать дублирования при формировании минимизированной формы.
8. **Обработка ошибок**: Программа обрабатывает возможные ошибки, такие как неверный ввод номера файла или отсутствие файла, и информирует пользователя о проблемах.

**Структура программы**

* **Методы**:
  + **GetNumberInt()**: Запрашивает и возвращает номер файла.
  + **GetVectorFromFile(ref int[] vector)**: Загружает вектор из файла.
  + **GetTruthtable(ref int[,] truthTable, ref int[] vector)**: Строит таблицу истинности.
  + **GetSDNF(ref int[,] truthTable)**: Формирует СДНФ на основе таблицы истинности.
  + **ShowSDNF(ref string[] SDNF)**: Выводит СДНФ на экран.
  + **Quine(ref string[] SDNF, ref string[] notGlued)**: Минимизирует СДНФ с помощью метода Квайна.
  + **ShowImplicantTable(ref string[] SDNF, ref string[] final)**: Выводит импликантную таблицу.
  + **IsFound(ref string[] SDNF, string str)**: Проверяет наличие терма в массиве.
* **Основной метод**: **Main(string[] args)** — точка входа в программу, где происходит последовательный вызов всех вышеописанных методов для выполнения логической минимизации.

**Скриншоты работы программы**:





**Программа:**

[**https://github.com/Gpushak/CsharpUnProj/tree/7d357b4e79d29a5e293faec5bbd90852e5c73cee/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%B0%D1%8F%20%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0/3labaDiskMat**](https://github.com/Gpushak/CsharpUnProj/tree/7d357b4e79d29a5e293faec5bbd90852e5c73cee/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%B0%D1%8F%20%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0/3labaDiskMat)

using System;

using System.Collections.Generic;

using System.Collections.Specialized;

using System.IO;

using System.Linq;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Linq;

namespace Метод\_квайна

{

internal class Program

{

static int GetNumberInt()

{

int number;

string inputNumber;

bool isRightType;

do

{

Console.WriteLine("Введите номер файла: ");

inputNumber = Console.ReadLine();

isRightType = Int32.TryParse(inputNumber, out number);

if (!isRightType)

Console.Write("Неверный ввод данных. Пожалуйста, попробуйте ещё раз.\n");

}

while (!isRightType);

return number;

}

static void GetVectorFromFile(ref int [] vector)

{

string fileVector;

FileInfo fileinf;

do

{

fileinf = new FileInfo($@"C:\Users\igleb\OneDrive\Рабочий стол\CsharpUnProj\3labaDiskMat\3labaDiskMat\bin\Debug\net8.0\{ GetNumberInt()}.txt");

if (!fileinf.Exists)

{

Console.WriteLine("Файла не существует!");

}

}

while (!fileinf.Exists);

string path = fileinf.FullName;

fileVector = File.ReadAllText(path);

int checkNumber = 0;

for (int i = 0; i < 16; i++)

{

if (Int32.TryParse(fileVector[i].ToString(), out checkNumber))

{

vector[i] = Convert.ToInt32(fileVector[i]) - 48;

}

}

}

static void GetTruthtable(ref int[,] truthTable, ref int[] vector)

{

Console.WriteLine("Таблица истинности:");

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

Console.WriteLine("| x | y | z | w | f |");

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

bool checkY = false;

for (int i = 0; i < 16; i++)

{

int x = (i & 8) >> 3;

int y = (i & 4) >> 2;

int z = (i & 2) >> 1;

int w = i & 1;

int result = vector[i];

Console.WriteLine($"| {x} | {y} | {z} | {w} | {result} |");

truthTable[i, 0] = x;

truthTable[i, 1] = y;

truthTable[i, 2] = z;

truthTable[i, 3] = w;

truthTable[i, 4] = result;

}

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

}

static string [] GetSDNF(ref int[,] truthTable)

{

int size = 0;

for (int i = 0; i < 16; i++)

{

if (truthTable[i, 4] == 1)

{

size++;

}

}

string[] SDNF = new string[size];

int k = 0;

for (int i = 0; i < 16; i++)

{

if (truthTable[i, 4] == 1)

{

if (truthTable[i, 0] == 0) SDNF[k] += 'X';

else SDNF[k] += 'x';

if (truthTable[i, 1] == 0) SDNF[k] += 'Y';

else SDNF[k] += 'y';

if (truthTable[i, 2] == 0) SDNF[k] += 'Z';

else SDNF[k] += 'z';

if (truthTable[i, 3] == 0) SDNF[k] += 'W';

else SDNF[k] += 'w';

k++;

}

}

Console.Write("СДНФ: ");

ShowSDNF(ref SDNF);

return SDNF;

}

static void ShowSDNF(ref string [] SDNF)

{

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

{

Console.Write(SDNF[i]);

if (i != SDNF.Length - 1)

{

Console.Write(" + ");

}

}

Console.WriteLine();

Console.WriteLine();

}

static string[] Quine(ref string[] SDNF, ref string[] notGlued)

{

string[] gluing = new string[0];

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

{

bool same, f = false;

int counter = 0;

string element = SDNF[i];

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

{

string tempString = "";

same = false;

for (int k = 0; k < element.Length; k++)

{

if (element[k] == SDNF[j][k])

{

tempString += element[k];

}

else if (Math.Abs(Convert.ToInt32(element[k]) - Convert.ToInt32(SDNF[j][k])) == 32)

{

same = true;

}

}

if (tempString.Length == element.Length - 1 && same && !IsFound(ref gluing, tempString))

{

Array.Resize(ref gluing, gluing.Length + 1);

gluing[gluing.Length - 1] = tempString;

Console.WriteLine($"{i + 1} - {j + 1}: {gluing[gluing.Length - 1]}");

}

if (tempString.Length != element.Length - 1 || !same)

{

counter++;

if (counter == SDNF.Length)

{

Array.Resize(ref notGlued, notGlued.Length + 1);

notGlued[notGlued.Length - 1] = SDNF[i];

}

}

}

}

return gluing;

}

static void ShowImplicantTable(ref string[] SDNF, ref string[] final)

{

for (int i = 0; i < SDNF.Length + 1; i++)

{

if (i == 0)

{

Console.Write(",-----,");

}

else

{

Console.Write("------,");

}

}

Console.WriteLine();

Console.Write($"| |");

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

{

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

}

Console.WriteLine();

for (int i = 0; i < SDNF.Length + 1; i++)

{

if (i == 0)

{

Console.Write("|-----|");

}

else

{

Console.Write("------|");

}

}

Console.WriteLine();

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

{

Console.Write("| ");

if (final[i].Length == 1) Console.Write($" {final[i]}");

else if (final[i].Length == 2) Console.Write(final[i]);

else if (final[i].Length >= 3) Console.Write(final[i]);

if (final[i].Length < 3) Console.Write(" |");

else if (final[i].Length == 3) Console.Write(" |");

else Console.Write("|");

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

{

bool check = true;

for (int k = 0; k < final[i].Length; k++)

{

if (!SDNF[j].Contains(final[i][k])) check = false;

}

if (check) Console.Write(" + |");

else Console.Write(" |");

}

Console.WriteLine();

for (int j = 0; j < SDNF.Length + 1; j++)

{

if (j == 0)

{

Console.Write("'-----'");

}

else

{

Console.Write("------'");

}

}

Console.WriteLine();

}

}

static bool IsFound(ref string[] SDNF, string str)

{

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

{

if (SDNF[i] == str) return true;

}

return false;

}

static void Main(string[] args)

{

int[] vector = new int[16];

int[,] truthTable = new int[16, 5];

string[] notGlued = new string[0];

GetVectorFromFile(ref vector);

Console.Write("Полученный вектор из файла: ");

for (int i = 1; i <= 16; i++)

{

if (i / 4 > 0 && i % 4 == 1)

{

Console.Write(" ");

}

Console.Write(vector[i - 1]);

}

Console.WriteLine();

GetTruthtable(ref truthTable, ref vector);

Console.ResetColor();

string[] SDNF = GetSDNF(ref truthTable);

if (SDNF.Length == 0) Console.WriteLine("Для введённых данных не существует МДНФ!");

else

{

string[] temp = SDNF;

string[] gluing;

do

{

gluing = Quine(ref temp, ref notGlued);

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

ShowSDNF(ref gluing);

if (gluing.Length != 0) temp = gluing;

}

while (gluing.Length != 0);

if (temp[0] != "")

{

Console.WriteLine("Импликантная таблица: ");

int k = 0, amount = 0;

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

{

if (IsFound(ref temp, notGlued[i]))

{

amount++;

}

}

int oldLength = temp.Length;

Array.Resize(ref temp, temp.Length + notGlued.Length - amount);

for (int i = oldLength; i < temp.Length; i++) // если не повторяются

{

if (!IsFound(ref temp, notGlued[k]))

{

temp[i] = notGlued[k];

}

k++;

}

ShowImplicantTable(ref SDNF, ref temp);

}

else

{

Console.WriteLine("Таблица импликантности пуста!");

}

}

}

}

}