Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное учреждение высшего образования  
«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

Дискретная математика

Семестр 3

Лабораторная работа 3

Выполнил  
студент группы РИС-22-2б  
Баяндин К.С.

Проверила  
старший преподаватель кафедры ИТАС  
Рустамханова Г.И.

Пермь, 2023

Постановка задачи

Создать консольное приложение, которое вычисляет минимальную ДНФ функции.

Для этого разработать:

1. Ввод системы функций векторным способом – 16-значный
2. Создание таблицы истинности
3. Запись функции в СДНФ (! – отрицание)
4. Неполное склеивание методом Квайна
5. Минимальное покрытие импликантной матрицы
6. Результат минимизации

Алгоритмы работы программы

Для взаимодействия пользователя с программой, был разработан массив из возможных команд, вывод этих команд на экран, а также возможность выбора одной из них. Листинг представлен ниже:

static void Main()

{

string[] menu =

{

"1. Создание вектора вручную",

"2. Создание вектора из списка",

"3. Выполнить задание",

"4. Выход",

};

string vector = "";

int command;

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

Console.WriteLine(menu[i]);

do

{

command = InputInt("Введите номер команды: ");

switch (command)

{

case 1:

vector = InputVector();

break;

case 2:

vector = CreateSelectedVector(vector);

break;

case 3:

Task(vector);

break;

case 4:

Console.WriteLine("Спасибо что пользуетесь нашей программой!");

break;

default:

Console.WriteLine("Неверная команда");

break;

}

} while (command != 4);

}

Как можно заметить, программа позволяет создать вектор вручную или выбрать из предложенных, а также выполнить получить минимизированную ДНФ Методом Квайна.

Ввод вектора вручную осуществляется с помощью функции *InputVector()* (код представлен ниже). Вектор должен быть 16-значным, а также состоять только из 0 и 1. Чтобы введеный вектор удовлетворял данным условиям, созданы функции *IsValidVector()* – проверка что вектор существует и длина равна 16, *ContainsOnlyZeroAndOne()* – проверка что вектор имеет только 0 и 1, *FullVectorZerosOrOnes()* – проверка что весь вектор состоит из 0 или 1.

static string InputVector()

{

string inputVector;

Console.Write("Введите вектор функции: ");

bool isCorrectSize = false;

do

{

inputVector = Console.ReadLine() ?? "";

if (!IsValidVector(inputVector))

Console.Write("Ошибка, введите вектор заново: Вектор должен состоять из 16 значений 0 и 1\n");

else

isCorrectSize = true;

} while (!isCorrectSize);

return inputVector;

}

Создание вектора из предложенного списка осуществляет функция *CreateSelectedVector()*, в которой заранее сформирован массив векторов и пользователь может выбрать любой из них:

public static string CreateSelectedVector(string vector)

{

string[] arrRows = {

"1. 1001100110011001",

"2. 1111000011110000",

"3. 1111111100000000",

"4. 1010101010101010"

};

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

Console.WriteLine(arrRows[i]);

int command = InputInt("Введите номер вектора: ");

switch (command)

{

case 1:

vector = "1001100110011001";

break;

case 2:

vector = "1111000011110000";

break;

case 3:

vector = "1111111100000000";

break;

case 4:

vector = "1010101010101010";

break;

default:

Console.WriteLine("Нет такого вектора в списке. Возвращен пустой вектор");

vector = "";

break;

}

return vector;

}

Чтобы получить минимизированную ДНФ, сначала нужно получить таблицу истинности, по ней построить СДНФ, далее выполнить неполные склеивания методом Квайна и в конце составить импликантную матрицу, минимальное покрытие импликантной матрицы и будет результатом.

Все данные этапы происходят в отдельных функциях (*CreateTruthTable()*, *GluingConstituents()*…) с выводом промежуточных результатов (вывод таблицы истинности, вывод СДНФ…). Листинг данных функций можно посмотреть в конце отчета.

static void Task(string vector)

{

if (string.IsNullOrEmpty(vector))

{

Console.WriteLine("Вектор не создан");

return;

}

var truthTable = new List<string>();

var constituents = new List<string>();

CreateTruthTable(vector, ref truthTable, ref constituents);

Console.WriteLine();

PrintTruthTable(truthTable);//таблица истинности

Console.WriteLine("СДНФ");

PrintDNF(constituents);//сднф

var gluedConstituents = new List<string>();

gluedConstituents.AddRange(constituents);

GluingConstituents(ref gluedConstituents);//склеивание

AbsorpSDNF(ref gluedConstituents);//поглощение

Console.WriteLine("Сокращенное ДНФ");

PrintDNF(gluedConstituents);

Console.WriteLine("Таблица импликант");

implicantTable = new List<List<bool>>();

CreateImplicantTable(constituents, gluedConstituents);

PrintImplicantTable(constituents, gluedConstituents);

ProcessAndPrintResults(gluedConstituents);

}

**Результаты работы программы:**

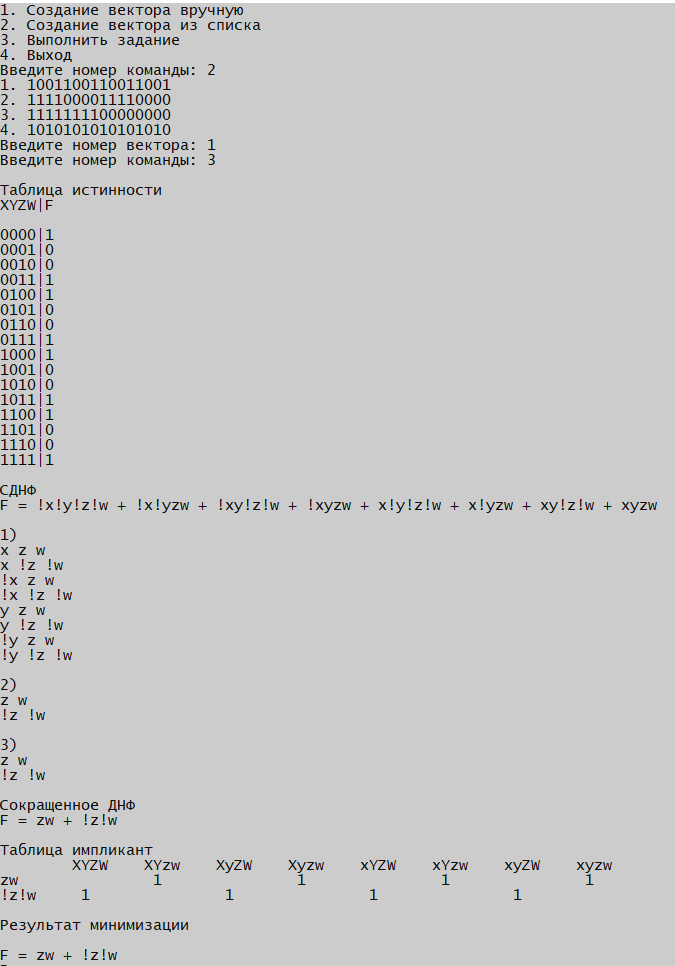


Рисунок 1. Вектор из списка

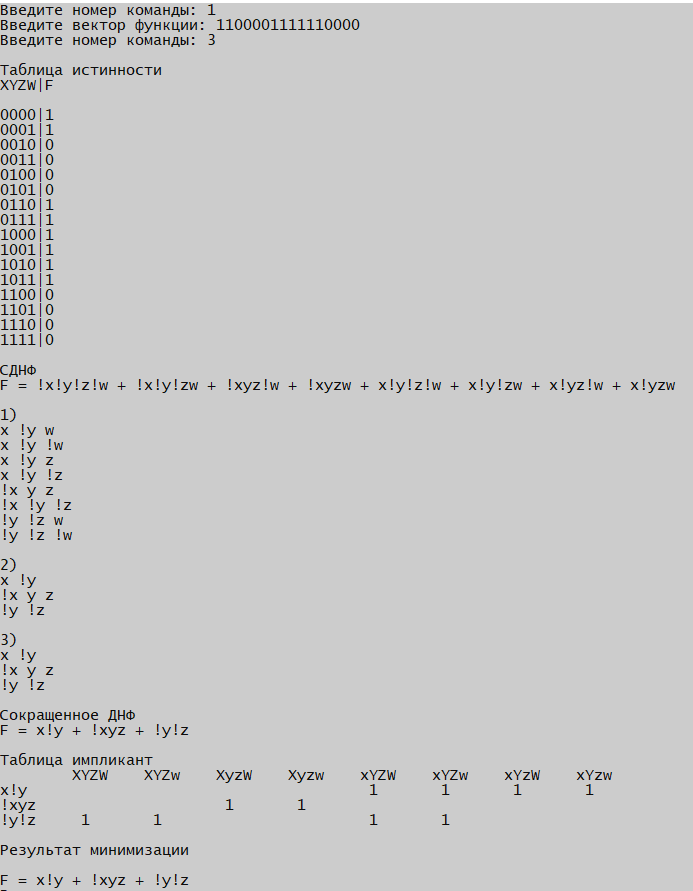


Рисунок 2. Вектор, веденный вручную

Листинг

using System;

using System.Collections.Generic;

using System.Text;

using System.Linq;

namespace Calculator

{

class Program

{

static List<List<bool>> implicantTable;

static void Main()

{

string[] menu =

{

"1. Создание вектора вручную",

"2. Создание вектора из списка",

"3. Выполнить задание",

"4. Выход",

};

string vector = "";

int command;

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

Console.WriteLine(menu[i]);

do

{

command = InputInt("Введите номер команды: ");

switch (command)

{

case 1:

vector = InputVector();

break;

case 2:

vector = CreateSelectedVector(vector);

break;

case 3:

Task(vector);

break;

case 4:

Console.WriteLine("Спасибо что пользуетесь нашей программой!");

break;

default:

Console.WriteLine("Неверная команда");

break;

}

} while (command != 4);

}

//проверка на ввод целого числа

static int InputInt(string message)

{

int number;

Console.Write(message);

if (!int.TryParse(Console.ReadLine(), out number) || (number < 0))

{

do

{

Console.WriteLine("Ошибка!");

if (number < 0)

Console.WriteLine("Число не может быть отрицательным");

else

Console.WriteLine("Введите число типа int");

Console.Write(message);

} while (!int.TryParse(Console.ReadLine(), out number) || (number < 0));

}

return number;

}

static string InputVector()

{

string inputVector;

Console.Write("Введите вектор функции: ");

bool isCorrectSize = false;

do

{

inputVector = Console.ReadLine() ?? "";

if (!IsValidVector(inputVector))

Console.Write("Ошибка, введите вектор заново: Вектор должен состоять из 16 значений 0 и 1\n");

else

isCorrectSize = true;

} while (!isCorrectSize);

return inputVector;

}

static bool IsValidVector(string inputVector)

{

return !(inputVector is null || inputVector.Length != 16 || !ContainsOnlyZeroAndOne(inputVector)

|| FullVectorZerosOrOnes(inputVector));

}

static bool ContainsOnlyZeroAndOne(string inputVector)

{

return inputVector.All(c => c == '1' || c == '0');

}

static bool FullVectorZerosOrOnes(string inputVector)

{

return inputVector == new String('1', 16) || inputVector == new String('0', 16);

}

public static string CreateSelectedVector(string vector)

{

string[] arrRows = {

"1. 1001100110011001",

"2. 1111000011110000",

"3. 1111111100000000",

"4. 1010101010101010"

};

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

Console.WriteLine(arrRows[i]);

int command = InputInt("Введите номер вектора: ");

switch (command)

{

case 1:

vector = "1001100110011001";

break;

case 2:

vector = "1111000011110000";

break;

case 3:

vector = "1111111100000000";

break;

case 4:

vector = "1010101010101010";

break;

default:

Console.WriteLine("Нет такого вектора в списке. Возвращен пустой вектор");

vector = "";

break;

}

return vector;

}

static void Task(string vector)

{

if (string.IsNullOrEmpty(vector))

{

Console.WriteLine("Вектор не создан");

return;

}

var truthTable = new List<string>();

var constituents = new List<string>();

CreateTruthTable(vector, ref truthTable, ref constituents);

Console.WriteLine();

PrintTruthTable(truthTable);//таблица истинности

Console.WriteLine("СДНФ");

PrintDNF(constituents);//сднф

var gluedConstituents = new List<string>();

gluedConstituents.AddRange(constituents);

GluingConstituents(ref gluedConstituents);//склеивание

AbsorpSDNF(ref gluedConstituents);//поглощение

Console.WriteLine("Сокращенное ДНФ");

PrintDNF(gluedConstituents);

Console.WriteLine("Таблица импликант");

implicantTable = new List<List<bool>>();

CreateImplicantTable(constituents, gluedConstituents);

PrintImplicantTable(constituents, gluedConstituents);

ProcessAndPrintResults(gluedConstituents);

}

static void ProcessAndPrintResults(List<string> gluedConstituents)

{

var minRows = new List<int>();

minRows = FindRowsMinNum(minRows, 0, 0);

minRows.Sort();

Console.WriteLine("Результат минимизации");

PrintResult(minRows, gluedConstituents);

}

static List<string> CreateTruthTable(string vector, ref List<string> truthTable, ref List<string> constituents)

{

truthTable = new List<string>();

constituents = new List<string>();

string alphabet = "xyzw";

string invertedAlphabet = "XYZW";

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

{

string binaryNum = Convert.ToString(i, 2).PadLeft(4, '0');

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

{

constituents.Add(ConvertToLetters(binaryNum, alphabet, invertedAlphabet));

}

truthTable.Add($"{binaryNum}|{vector[i]}");

}

return truthTable;

}

static string ConvertToLetters(string binaryNum, string alphabet, string invertedAlphabet)

{

var result = new StringBuilder();

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

{

result.Append(binaryNum[i] == '1' ? alphabet[i] : invertedAlphabet[i]);

}

return result.ToString();

}

static void PrintTruthTable(List<string> truthTable)

{

Console.WriteLine("Таблица истинности\nXYZW|F\n");

foreach (var row in truthTable)

{

Console.WriteLine(row);

}

Console.WriteLine();

}

static void PrintDNF(List<string> constituents)

{

string arbitraryFormula = string.Join(" + ", constituents.Select(c => PrintConstituent(c)));

Console.WriteLine($"F = {arbitraryFormula}\n");

}

static string PrintConstituent(string constituent)

{

string variables = "xyzw";

var res = new StringBuilder();

foreach (char c in constituent)

{

int index = variables.IndexOf(char.ToLower(c));

if (index != -1)

{

if (char.IsLower(c))

{

res.Append(variables[index]);

}

else

{

res.Append("!" + variables[index]);

}

}

}

return res.ToString();

}

//склеивание

static void GluingConstituents(ref List<string> constituentsToGlue)

{

int changesCount, count = 1;

do

{

changesCount = 0;

var usedPairNumbers = new List<int>();

var tempCorrectPairs = new List<string>();

for (int i = 0; i < constituentsToGlue.Count - 1; i++)

{

var pair1 = constituentsToGlue[i];

for (int j = i + 1; j < constituentsToGlue.Count; j++)

{

string pair2 = constituentsToGlue[j], resPair;

if (MightBeGlued(pair1, pair2))

{

resPair = Glue(pair1, pair2);

tempCorrectPairs.Add(resPair);

usedPairNumbers.Add(i);

usedPairNumbers.Add(j);

changesCount++;

}

}

}

usedPairNumbers = usedPairNumbers.Distinct().ToList();

usedPairNumbers.Sort();

var notUsedPairs = new List<int>();

if (usedPairNumbers.Count != constituentsToGlue.Count)

{

for (int i = usedPairNumbers.Count; i < constituentsToGlue.Count; i++)

{

if (!usedPairNumbers.Contains(i))

notUsedPairs.Add(i);

}

}

notUsedPairs.Sort();

notUsedPairs = notUsedPairs.Distinct().ToList();

AddUnusedConstituents(usedPairNumbers, ref constituentsToGlue, ref tempCorrectPairs, notUsedPairs);

DeleteDublicates(ref tempCorrectPairs);

if (changesCount > 0)

{

constituentsToGlue.Clear();

constituentsToGlue.AddRange(tempCorrectPairs);

}

Console.Write($"{count++})\n");

PrintConstituentsList(constituentsToGlue);

} while (changesCount > 0);

}

static bool MightBeGlued(string pair1, string pair2)

{

if (pair1.Length != pair2.Length)

return false;

int diffCount = 0;

int length = pair1.Length;

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

{

if (pair1[i] != pair2[i])

{

if (!AreEqualLetters(pair1[i], pair2[i]))

return false;

else

diffCount++;

}

}

if (diffCount > 1)

return false;

return true;

}

static bool AreEqualLetters(char first, char second)

{

if (first == 'x' && second == 'X' || first == 'X' && second == 'x')

return true;

if (first == 'y' && second == 'Y' || first == 'Y' && second == 'y')

return true;

if (first == 'z' && second == 'Z' || first == 'Z' && second == 'z')

return true;

if (first == 'w' && second == 'W' || first == 'W' && second == 'w')

return true;

return false;

}

static string Glue(string pair1, string pair2)

{

var resPair = new StringBuilder();

for (int i = 0; i < pair1.Length; i++)

{

if (pair1[i] == pair2[i])

resPair.Append(pair1[i]);

}

return resPair.ToString();

}

static void AddUnusedConstituents(List<int> usedPairNumbers, ref List<string> correctPairs, ref List<string> tempCorrectPairs, List<int> notUsedPairs)

{

int counter = 0, size = usedPairNumbers.Count;

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

{

if (counter != usedPairNumbers[i])

{

tempCorrectPairs.Add(correctPairs[counter]);

usedPairNumbers.Add(counter);

--i;

}

counter++;

}

foreach (int item in notUsedPairs)

tempCorrectPairs.Add(correctPairs[item]);

}

static void DeleteDublicates(ref List<string> correctPairs)

{

correctPairs.Sort();

correctPairs = correctPairs.Distinct().ToList();

}

static void PrintConstituentsList(List<string> pairs)

{

string al = "xyzw";

for (int i = 0; i < pairs.Count; i++)

{

for (int j = 0; j < pairs[i].Length; j++)

{

if (pairs[i][j] == 'X')

Console.Write($"!{al[0]} ");

if (pairs[i][j] == 'Y')

Console.Write($"!{al[1]} ");

if (pairs[i][j] == 'Z')

Console.Write($"!{al[2]} ");

if (pairs[i][j] == 'W')

Console.Write($"!{al[3]} ");

if (pairs[i][j] == 'x')

Console.Write($"{al[0]} ");

if (pairs[i][j] == 'y')

Console.Write($"{al[1]} ");

if (pairs[i][j] == 'z')

Console.Write($"{al[2]} ");

if (pairs[i][j] == 'w')

Console.Write($"{al[3]} ");

}

Console.WriteLine();

}

Console.WriteLine();

}

static void AbsorpSDNF(ref List<string> constituentsToAbsorb)

{

for (int i = 0; i < constituentsToAbsorb.Count - 1; i++)

{

int index = i + 1;

if (constituentsToAbsorb[i] != constituentsToAbsorb[index])

{

int size = (constituentsToAbsorb[i].Length > constituentsToAbsorb[index].Length) ?

constituentsToAbsorb[i].Length

: constituentsToAbsorb[index].Length;

if (constituentsToAbsorb[i].Contains(constituentsToAbsorb[index])

|| constituentsToAbsorb[index].Contains(constituentsToAbsorb[i]))

{

if (constituentsToAbsorb[i].Length == size)

constituentsToAbsorb.RemoveAt(i);

else

constituentsToAbsorb.RemoveAt(index);

}

}

}

}

//создание таблицы импликант

static void CreateImplicantTable(List<string> constituents, List<string> gluedConstituents)

{

for (int i = 0; i < gluedConstituents.Count; i++)

{

var tmpMatr = new List<bool>();

for (int j = 0; j < constituents.Count; j++)

{

bool elem = FindSubsrtInStr(gluedConstituents[i], constituents[j]);

tmpMatr.Add(elem);//implicantTable[i][j] = elem;

}

implicantTable.Add(tmpMatr);

}

}

static bool FindSubsrtInStr(string subStr, string str)

{

int foundCounter = 0;

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

{

if (str.Contains(subStr[i]))

foundCounter++;

}

if (foundCounter == subStr.Length)

return true;

return false;

}

//таблица импликант

static void PrintImplicantTable(List<string> constituents, List<string> gluedConstituents)

{

Console.Write(" ");

for (int i = 0; i < constituents.Count; i++)

{

Console.Write("\t" + constituents[i] + " ");

}

Console.WriteLine();

for (int i = 0; i < gluedConstituents.Count; i++)

{

Console.Write(PrintConstituent(gluedConstituents[i]) + " ");

for (int j = 0; j < implicantTable[i].Count; j++)

{

if (implicantTable[i][j])

Console.Write("\t 1 ");

else

Console.Write("\t ");

}

Console.WriteLine();

}

Console.WriteLine();

}

static List<int> FindRowsMinNum(List<int> solution, int str, int col)

{

if (str < implicantTable.Count && col < implicantTable[str].Count && str >= 0 && col >= 0)

{

if (implicantTable[str][col])

{

if (!solution.Contains(str))

solution.Add(str);

return FindRowsMinNum(solution, str, col + 1);

}

else

{

bool isFirst = true;

for (var i = 0; i < implicantTable.Count; i++)

{

if (implicantTable[i][col])

{

var tempList = new List<int>(solution);

if (!tempList.Contains(i))

tempList.Add(i);

tempList = FindRowsMinNum(tempList, i, col + 1);

if (isFirst)

solution = tempList;

else

{

if (tempList.Count < solution.Count)

solution = tempList;

}

isFirst = false;

}

}

return solution;

}

}

else

return solution;

}

//Результат минимизации

static void PrintResult(List<int> answer, List<string> gluedConstituents)

{

Console.Write("\nF = " + PrintConstituent(gluedConstituents[answer[0]]));

for (int i = 1; i < answer.Count; i++)

{

Console.Write(" + " + PrintConstituent(gluedConstituents[answer[i]]));

}

Console.WriteLine();

}

}

}