Министерство образования и науки РФ

Пермский национальный исследовательский политехнический университет

Электротехнический факультет

Кафедра информационных технологий и автоматизированных систем

Дискретная математика

Лабораторная работа № 4

Тема: «Таблица классов»

Выполнил: студентка группы ИВТ-23-1б

Хорошилова К. П.

Проверил: ст. преподаватель

Рустамханова Г. И.

г. Пермь – 2024**Цель работы**

Изучить свойства булевых функций, включая их принадлежность к различным классам, и определить полноту системы функций на основе заданных входных данных.

**Задачи работы**

1. Реализовать возможность ввода нескольких булевых функций.
2. Обеспечить поддержку функций с разной длиной вектора истинности (2, 4, 8 и т.д.).
3. Построить таблицу, отображающую принадлежность функций к классам (сохраняющая ноль, сохраняющая единицу, самодвойственная, монотонная, линейная).
4. Проверить, является ли система заданных функций полной, и отобразить результат на экране.
5. Оформить отчет.

# Код программы

using System.Numerics;

using System.Runtime.CompilerServices;

using System.Runtime.Intrinsics;

using System.Text.RegularExpressions;

namespace Дискретная\_математика\_\_\_Классы\_функций

{

internal class Program

{

//Обработка меню

void SpisokMenu(ref string[] mas)

{

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

{

Console.WriteLine((i + 1) + ") " + mas[i]);

}

}

int SelectMenu(int length)

{

bool isChek = false;

int element = 0;

string str;

do

{

str = Console.ReadLine();

Int32.TryParse(str, out element);

isChek = element > 0 && element < length + 1;

} while (!isChek);

return element;

}

//Конец обработки меню

//Работа с вводом

void InputVector(ref string[] function)

{

string func;

Console.WriteLine("Введите количество функций");

int str;

Int32.TryParse(Console.ReadLine(), out str);

function = new string[str];

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

{

Console.WriteLine("Введите " + (i + 1) + " функцию");

func = Console.ReadLine();

func = Assembling(func);

function[i] = func;

}

}

string Assembling(string str)

{

string vector = "";

Regex regex = new Regex(@"\w+");

MatchCollection matches = regex.Matches(str);

foreach (Match match in matches)

{

vector += match.Value;

}

return vector;

}

void InputFileVector(ref string[] function)

{

StreamReader file = new StreamReader("D:\\Мои проекты Vethyal studia\\Дискретная математика - Классы функций\\Function.txt");

string str = "";

function = new string[System.IO.File.ReadAllLines("D:\\Мои проекты Vethyal studia\\Дискретная математика - Классы функций\\Function.txt").Length];

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

{

str = file.ReadLine();

str = Assembling(str);

function[i] = str;

}

}

void MenuInpyt(ref string[] function)

{

bool isEnd = false;

string[] mas = new string[3];

mas[0] = "Ввод функций в ручную";

mas[1] = "Ввод функций из файла";

mas[2] = "Отмена работы";

do

{

SpisokMenu(ref mas);

switch (SelectMenu(mas.Length))

{

case 1:

{

InputVector(ref function);

break;

}

case 2:

{

InputFileVector(ref function);

break;

}

}

isEnd = true;

} while (!isEnd);

}

//Конец работы с вводом

//Работа с классами

//М0нoтонность функции

bool Manatonic(string function, string[][] tree)

{

bool isM = true;

int a = 1, b = 0, c = 0;

int count = 0;

while (a < tree.Length && isM)

{

while (b < tree[a - 1].Length && isM)

{

while (c < tree[a].Length && isM)

{

for (int i = 0; i < tree[a][c].Length; i++)

{

if (tree[a - 1][b][i] != tree[a][c][i])

{

count++;

}

}

if (count == 1)

{

char One = function[Convert.ToInt32(tree[a - 1][b], 2)];

char Two = function[Convert.ToInt32(tree[a][c], 2)];

if (One > Two)

{

isM = false;

}

}

c++;

count = 0;

}

b++;

c = 0;

}

a++;

b = 0;

}

return isM;

}

bool FunctionManatonic(string function, ref string[][] tree)

{

bool isM = true;

int n = 1;

int N = 2;

while (N != function.Length)

{

n++;

N \*= 2;

}

string[] tableTrue = new string[function.Length];

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

{

string str = "";

string strTable = "";

str = Convert.ToString(i, 2);

for (int j = 0; j < n - str.Length; j++)

{

strTable += '0';

}

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

{

strTable += str[j];

}

tableTrue[i] = strTable;

}

int[] counter = new int[n + 1];

foreach (string strTable in tableTrue)

{

int count = 0;

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

{

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

{

count++;

}

}

counter[count]++;

}

tree = new string[n + 1][];

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

{

tree[i] = new string[counter[i]];

}

foreach (string strTable in tableTrue)

{

int count = 0;

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

{

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

{

count++;

}

}

tree[count][counter[count] - 1] = strTable;

counter[count]--;

}

isM = Manatonic(function, tree);

return isM;

}

//Линейная

// Проверка линейности функции

bool FunctionLinearity(string function, string[][] tree)

{

int n = tree.Length;

char[][] polinomJigal = new char[n][];

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

{

polinomJigal[i] = new char[tree[i].Length];

}

// Инициализация первой строки полинома Жигалкина

polinomJigal[0][0] = function[0];

// Построение полинома Жигалкина

for (int level = 1; level < n; level++)

{

for (int j = 0; j < tree[level].Length; j++)

{

string element = tree[level][j];

List<char> terms = new List<char> { function[Convert.ToInt32(element, 2)] };

for (int k = 0; k < level; k++)

{

for (int m = 0; m < tree[k].Length; m++)

{

if (IsSubset(tree[k][m], element))

{

terms.Add(polinomJigal[k][m]);

}

}

}

polinomJigal[level][j] = CalculateParity(terms) ? '1' : '0';

}

}

// Проверка наличия нелинейных членов

for (int level = 2; level < n; level++)

{

if (polinomJigal[level].Any(coef => coef == '1'))

{

return false;

}

}

return true;

}

// Проверяет, является ли "subset" подмножеством "set" в двоичном представлении

private bool IsSubset(string subset, string set)

{

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

{

if (subset[i] == '1' && set[i] != '1')

{

return false;

}

}

return true;

}

// Вычисляет чётность множества битов

private bool CalculateParity(List<char> terms)

{

int count = terms.Count(bit => bit == '1');

return count % 2 != 0;

}

//Запуск общёта классов

void CalculationClasses(string function, ref bool[] functionClass)

{

if (function.Length != 1)

{

string[][] tree = new string[0][];

bool isS = true;

bool isM = true;

bool isL = true;

if (function[0] == '0')

{

functionClass[0] = true;

}

if (function[function.Length - 1] == '1')

{

functionClass[1] = true;

}

int n = 0;

while ((n < function.Length / 2 + 1) && isS)

{

if (function[n] == function[function.Length - 1 - n])

{

isS = false;

}

n++;

}

if (isS)

{

functionClass[2] = true;

}

isM = FunctionManatonic(function, ref tree);

functionClass[3] = isM;

isL = FunctionLinearity(function, tree);

functionClass[4] = isL;

}

else

{

if (function == "0")

{

functionClass[0] = true;

}

else

{

functionClass[1] = true;

}

functionClass[3] = true;

functionClass[4] = true;

}

}

//Конец работы с классами

//Полнота функции

void CompletenessFunction(string[] function, bool[,] functionClass)

{

bool isCompleteness = true;

Console.Write('\t' + "|");

Console.Write("T0" + "\t|");

Console.Write("T1" + "\t|");

Console.Write("S" + "\t|");

Console.Write("M" + "\t|");

Console.Write("L" + "\t|");

Console.WriteLine();

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

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

{

Console.Write(function[i] + "|");

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

{

Console.Write(functionClass[i, j] + "\t|");

}

Console.WriteLine();

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

}

int[] completeness = { -1, -1, -1, -1, -1 };

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

{

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

{

if (functionClass[i, j])

{

if (completeness[j] == -1)

{

completeness[j] = 1;

}

}

else

{

completeness[j] = 0;

}

}

}

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

{

if (completeness[i] == 1)

{

isCompleteness = false;

}

}

if (isCompleteness)

{

Console.WriteLine("Система функций полная");

}

else

{

Console.WriteLine("Система функций неполная");

}

}

void ClassFunction(string[] function, bool[,] functionClass)

{

bool[] temp = new bool[5];

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

{

CalculationClasses(function[i], ref temp);

for (int j = 0; j < 5; j++)

{

functionClass[i, j] = temp[j];

temp[j] = false;

}

}

CompletenessFunction(function, functionClass);

}

//Меню

void Menu()

{

int lengthClass = 0;

string[] function = new string[0];

bool[,] functionClass = new bool[0, 0];

bool isEnd = false;

string[] mas = new string[3];

mas[0] = "Ввод функций";

mas[1] = "Расчёт полноты системы";

mas[2] = "Завершить работу";

do

{

SpisokMenu(ref mas);

switch (SelectMenu(mas.Length))

{

case 1:

{

MenuInpyt(ref function);

functionClass = new bool[function.Length, 5];

break;

}

case 2:

{

ClassFunction(function, functionClass);

break;

}

case 3:

{

isEnd = true;

break;

}

}

} while (!isEnd);

}

static void Main(string[] args)

{

Program main = new Program();

main.Menu();

}

}

}

**Анализ кода**

Данный код представляет собой программу для работы с булевыми функциями, включающую ввод, анализ и проверку их свойств.

Функция SpisokMenu отображает список пунктов меню. Она принимает массив строк mas, где каждая строка соответствует пункту меню. Функция проходит по массиву с помощью цикла и выводит на экран индекс пункта (сдвинутый на единицу для удобства пользователя) и его текстовое описание. Это позволяет пользователю видеть все доступные действия.

Функция SelectMenu обрабатывает выбор пункта меню пользователем. Она принимает количество доступных пунктов length и ожидает от пользователя ввода номера пункта. С помощью цикла проверяется корректность ввода: введенное число должно быть в диапазоне от 1 до length. Если ввод некорректен, пользователь вводит значение повторно. Возвращает функция корректно выбранный номер пункта.

Функция InputVector реализует ручной ввод функций. Сначала пользователь указывает количество функций. Затем в цикле для каждой функции происходит ввод строки. Эта строка очищается от лишних символов (с помощью функции Assembling) и сохраняется в массиве function. Это позволяет пользователю вручную ввести функции в удобном формате.

Функция Assembling удаляет из строки все лишние символы, оставляя только буквы, цифры и знаки. Она принимает строку, использует регулярное выражение для поиска нужных символов и возвращает новую строку, собранную из найденных элементов. Это позволяет упрощать обработку пользовательского ввода.

Функция InputFileVector считывает функции из файла. Путь к файлу указан в коде. Функция читает каждую строку файла, очищает её с помощью Assembling и добавляет в массив function. Это позволяет пользователю быстро загрузить функции из заранее подготовленного файла.

Функция MenuInpyt предоставляет выбор способа ввода функций: ручной ввод, загрузка из файла или отмена. Она выводит меню, ожидает выбора пользователя и вызывает соответствующие функции (InputVector или InputFileVector). Если выбран пункт отмены, завершает работу. Это делает ввод гибким и удобным для пользователя.

Функция Manatonic проверяет монотонность функции, используя дерево битовых строк tree. Внутри функция анализирует пары строк в дереве, различающиеся только одним битом, и сравнивает значения функции для этих строк. Если значение функции уменьшается, то функция не является монотонной. Возвращает булевое значение, указывающее на монотонность.

Функция FunctionManatonic строит дерево строк для проверки монотонности. Сначала вычисляется количество переменных функции. Затем для каждой строки значений функции формируется бинарное представление, добавляемое в массив tableTrue. Далее создаётся дерево строк, где строки группируются по количеству единиц в бинарном представлении. Затем вызывается Manatonic, чтобы определить, является ли функция монотонной. Возвращает результат проверки.

Функция FunctionLinearity проверяет линейность функции с использованием полинома Жигалкина. Она строит полином Жигалкина на основе дерева строк. Сначала вычисляются коэффициенты, проверяется чётность битов, и определяются линейные и нелинейные члены. Если в полиноме обнаружены нелинейные члены, функция не является линейной. Возвращает булевое значение.

Функция CalculationClasses определяет принадлежность функции к определённым классам: сохраняющая ноль, сохраняющая единицу, самодвойственная, монотонная, линейная. Проверки выполняются по правилам для каждого класса, включая использование функций FunctionManatonic и FunctionLinearity. Результаты сохраняются в массиве functionClass.

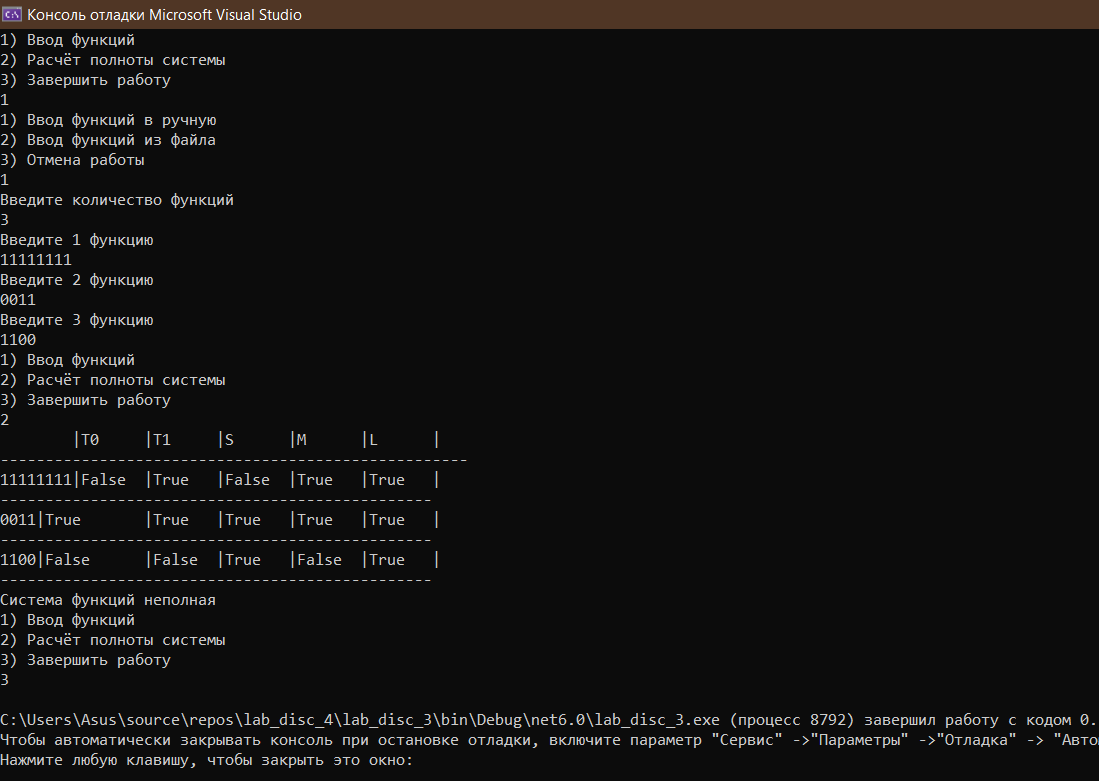
Функция CompletenessFunction проверяет полноту системы функций. Она выводит таблицу принадлежности функций к классам и анализирует, покрывают ли все функции необходимые классы для полноты. Если хотя бы один из классов не покрыт, система функций считается неполной. Результат проверки выводится на экран.

Функция ClassFunction вызывает CalculationClasses для каждой функции, заполняя массив functionClass. Затем передаёт результаты в CompletenessFunction для проверки полноты системы.

Функция Menu обеспечивает основную логику программы. Она предоставляет пользователю меню выбора действий: ввод функций, проверка полноты или завершение работы. В зависимости от выбора вызываются соответствующие функции. Меню работает в цикле, пока пользователь не выберет пункт завершения.

Ссылка на Github: <https://github.com/ksinph/disc_math>

**Тесты**



# Заключение

В ходе лабораторной работы я изучила свойства булевых функций, включая их принадлежность к различным классам, и определить полноту системы функций на основе заданных входных данных.