***Министерство образования Республики Беларусь***

***Учреждение образования***

***«Брестский государственный технический университет»***

***Кафедра ИИТ***

**Лабораторная работа №4**

**По дисциплине Дискретная математика за II семестр**

**Тема: «Булевы функции»**

**Выполнил:**

Студент группы ИИ-15 (1)

1-го курса

Волк И. А.

**Проверил:**

Глущенко Т.А.

Брест 2018

Цель: познакомиться с булевыми функциями.

**Задание:**

Для заданной булевой функции:

1. построить таблицу истинности;
2. представить функцию в совершенной дизъюнктивной нормальной форме, построить таблицу истинности;
3. представить функцию в совершенной конъюнктивной нормальной форме, построить таблицу истинности;
4. представить функцию в виде полинома Жегалкина, построить таблицу истинности;
5. исследовать функцию на принадлежность пяти замкнутым классам;

Вариант 7. 

Код программы:

#include "stdafx.h"

#include <Windows.h>

#include <iostream>

#include <cmath>

#include <bitset>

using namespace std;

bool function(bool[3]);

void truthTable(bool(\*)(bool[3]));

bool \* truthVec(bool(\*)(bool[3]), int\*);

//PDNF

bool PDNF(bool[3]);

void PDNFShow(bool(\*)(bool[3]));

bool \*\* disjunc(bool(\*)(bool[3]), int\*);

//PCNF

bool PCNF(bool [3]);

void PCNFShow(bool(\*)(bool[3]));

bool \*\* conjunc(bool(\*)(bool[3]), int\*);

bool classT0(bool(\*)(bool[3]));

bool classT1(bool(\*)(bool[3]));

bool classS(bool(\*)(bool[3]));

bool classM(bool(\*)(bool[3]));

bool classL(bool(\*)(bool[3]));

bool (\*classes[])(bool(\*)(bool[3])) = { classT0,

classT1,

classS,

classM,

classL };

void zhegalkinVec(bool\*, bool\*, int, int);

int main()

{

cout << "Function: " << endl;

truthTable(\*function);

cout << endl;

cout << "PDNF: " << endl;

PDNFShow(\*function);

cout << endl;

truthTable(\*PDNF);

cout << endl;

cout << "PCNF: " << endl;

PCNFShow(\*function);

cout << endl;

truthTable(\*PCNF);

cout << endl;

char classesStr[5][3] = {"T0", "T1", "S ", "M ", "L "};

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

if (classes[i](\*function))

cout << "Class" << classesStr[i] << endl;

system("pause");

return 0;

}

bool \* truthVec(bool(\*func)(bool[3]), int\*size)

{

\*size = (int)pow(2.0, 3.0);

bool \* vec = new bool[(\*size)];

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

vec[i] = false;

for (int x = 0; x < 2; x++)

for (int y = 0; y < 2; y++)

for (int z = 0; z < 2; z++)

{

bool xyz[3] = { x, y, z };

if (function(xyz))

{

int num = x\*(int)pow(2.0, 2.0) + y\*(int)pow(2.0, 1.0) + z\*(int)pow(2.0, 0.0);

vec[num] = true;

}

}

return vec;

}

void truthTable(bool(\*func)(bool[3]))

{

int sizeOfVec = 0;

bool \* vec = truthVec(func, &sizeOfVec);

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

cout << bitset<3>(i) << " " << vec[i] << endl;

}

// PDNF

void PDNFShow(bool(\*func)(bool[3]))

{

char vars[4] = "xyz";

int sizeOfArr;

bool \*\* dis = disjunc(func, &sizeOfArr);

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

{

for (int j = 0; j < 3; j++)

cout << ((!dis[i][j]) ? ("!") : ("")) << vars[j] << ((j != 2) ? ("\*"):(""));

if(i!=sizeOfArr-1)

cout << "+";

}

}

bool PDNF(bool var[3])

{

bool(\*func)(bool[3]) = \*function;

int sizeOfArr;

bool \*\* dis = disjunc(func, &sizeOfArr);

bool result = false;

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

{

bool con = true;

for (int j = 0; j < 3; j++)

con &= ((!dis[i][j]) ? (!var[j]) : (var[j]));

result |= con;

}

return result;

}

bool \*\* disjunc(bool(\*func)(bool[3]), int\* size)

{

int sizeOfVec = 0;

bool \* vec = truthVec(func, &sizeOfVec);

\*size = 0;

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

if (vec[i])

(\*size)++;

bool \*\* dis = new bool\*[(\*size)];

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

{

dis[i] = new bool[3];

for (int j = 0; j < 3; j++)

dis[i][j] = false;

}

for (int i = 0, j = 0; i < sizeOfVec; i++)

{

if (!vec[i])

continue;

int iCopy = i;

if (iCopy / 4)

{

dis[j][0]++;

iCopy -= 4;

}

if (iCopy / 2)

{

dis[j][1]++;

iCopy -= 2;

}

if (iCopy / 1)

{

dis[j][2]++;

iCopy -= 1;

}

j++;

}

return dis;

}

// PCNF

void PCNFShow(bool(\*func)(bool[3]))

{

char vars[4] = "xyz";

int sizeOfArr;

bool \*\* con = disjunc(func, &sizeOfArr);

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

{

cout << "(";

for (int j = 0; j < 3; j++)

cout << ((!con[i][j]) ? ("!") : ("")) << vars[j] << ((j != 2) ? ("+") : (""));

cout << ")";

if (i != sizeOfArr - 1)

cout << "\*";

}

}

bool PCNF(bool var[3])

{

bool(\*func)(bool[3]) = \*function;

int sizeOfArr;

bool \*\* con = conjunc(func, &sizeOfArr);

bool result = true;

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

{

bool dis = false;

for (int j = 0; j < 3; j++)

dis |= ((!con[i][j]) ? (!var[j]) : (var[j]));

result &= dis;

}

return result;

}

bool \*\* conjunc(bool(\*func)(bool[3]), int\* size)

{

int sizeOfVec = 0;

bool \* vec = truthVec(func, &sizeOfVec);

\*size = 0;

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

if (!vec[i])

(\*size)++;

bool \*\* con = new bool\*[(\*size)];

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

{

con[i] = new bool[3];

for (int j = 0; j < 3; j++)

con[i][j] = false;

}

for (int i = 0, j = 0; i < sizeOfVec; i++)

{

if (!vec[i])

continue;

int iCopy = i;

if (iCopy / 4)

iCopy -= 4;

else

con[j][0]++;

if (iCopy / 2)

iCopy -= 2;

else

con[j][1]++;

if (iCopy / 1)

iCopy -= 1;

else

con[j][2]++;

j++;

}

return con;

}

bool classT0(bool(\*func)(bool[3]))

{

int sizeOfVec = 0;

bool \* vec = truthVec(func, &sizeOfVec);

return (vec[0])?0:1;

}

bool classT1(bool(\*func)(bool[3]))

{

int sizeOfVec = 0;

bool \* vec = truthVec(func, &sizeOfVec);

return (vec[sizeOfVec-1]) ? 1 : 0;

}

bool classS(bool(\*func)(bool[3]))

{

int sizeOfVec = 0;

bool \* vec = truthVec(func, &sizeOfVec);

for (int i = sizeOfVec / 2 - 1, j = sizeOfVec / 2; i != 0; i--, j++)

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

return false;

return true;

}

bool classM(bool(\*func)(bool[3]))

{

int sizeOfVec = 0;

bool \* vec = truthVec(func, &sizeOfVec);

bool now = 0;

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

{

if (vec[i] < now)

return false;

now = vec[i];

}

return true;

}

bool classL(bool(\*func)(bool[3]))

{

int sizeOfVec = 0;

bool \* vec = truthVec(func, &sizeOfVec);

bool \* zhegVec = new bool[sizeOfVec];

zhegalkinVec(vec, zhegVec, sizeOfVec, 1);

for (int i = 3; i < sizeOfVec; i++)

if (zhegVec[i] == 1)

return false;

return true;

}

void zhegalkinVec(bool\* vec, bool\* resVec, int size, int iteration = 1)

{

int sizeNew = size - iteration;

bool\* vecNew = new bool[sizeNew];

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

vecNew[i] = (vec[i]!=vec[i+1]);

resVec[iteration - 1] = vecNew[0];

if (iteration == (size - 1))

return;

zhegalkinVec(vecNew, resVec, size, iteration+1);

return;

}

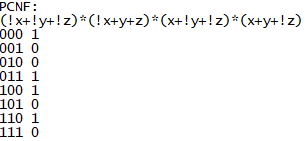
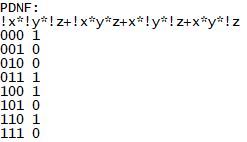
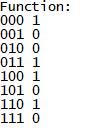
bool function(bool var[3])

{

return (var[0] || (!var[1])) != var[2];

}

Результат выполнения программы:



Вывод: познакомился с булевыми функциями.