离散数学

项目说明文档

**命题逻辑连接词、真值表、主范式BC**



同济大学

Tongji University

姓名： 林觉凯

学号： 2253744

指导老师： 李冰

学院专业： 软件学院 软件工程

# 一.实验目的；

本实验课程训练学生掌握命题逻辑中的联接词、真值表、主范式等，进一步能用它们来解决实际问题。通过实验提高学生编写实验报告、总结实验结果的能力；使学生具备程序设计的思想，能够独立完成简单的算法设计和分析。

# 二.实验内容；

求任意一个命题公式的真值表（B），并根据真值表求主范式（C）。

**详细说明：**

## 1. 逻辑联接词的运算

本实验要求利用C/C＋＋语言，实现二元合取、析取、条件和双向条件表达式的计算。充分利用联接词和逻辑运算符之间的相似性来实现程序功能。

## 2. 求任意一个命题公式的真值表

本实验要求利用C/C＋＋语言，实现任意输入公式的真值表计算。一般将公式中的命题变元放在真值表的左边，将公式的结果放在真值表的右边。命题变元可用数值变量表示，合式公式的表示及求真值表转化为逻辑运算结果；可用一维数表示合式公式中所出现的n个命题变元，同时它也是一个二进制加法器的模拟器，每当在这个模拟器中产生一个二进制数时，就相当于给各个命题变元产生了一组真值指派。算法逻辑如下：

（1）将二进制加法模拟器赋初值0。

（2）计算模拟器中所对应的一组真值指派下合式公式的真值。

（3）输出真值表中对应于模拟器所给出的一组真值指派及这组真值指派所对应的一行真值。

（4）产生下一个二进制数值，若该数值等于2n-1，则结束，否则转（2）。

# 三.实验环境；

采用C或C＋＋编程语言，MS Visual Studio 2019实验环境实现。

# 四. 实验原理和实现过程（算法描述）；

## 1.实验原理

（5）**真值表**:表征逻辑事件输入和输出之间全部可能状态的表格。列出命题公式真假值的表。通常以1表示真，0 表示假。命题公式的取值由组成命题公式的命题变元的取值和命题联结词决定，命题联结词的真值表给出了真假值的算法。 真值表是在逻辑中使用的一类数学表，用来确定一个表达式是否为真或有效。

（6）**主范式**：

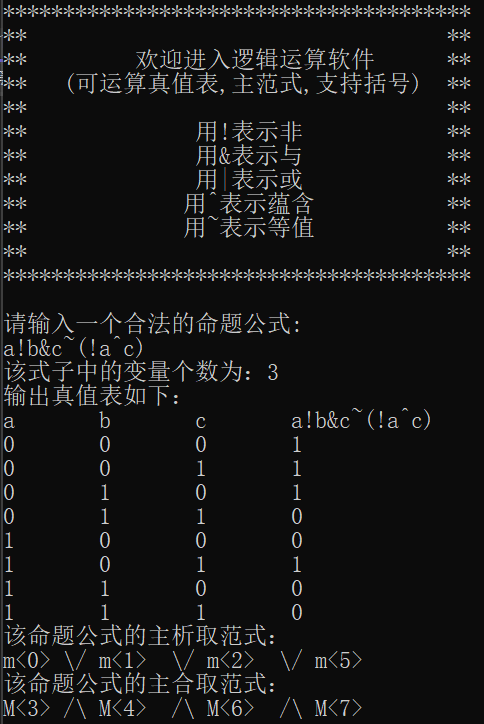
**主析取范式**：在含有n个命题变元的简单合取式中,若每个命题变元与其否定不同时存在,而两者之一出现一次且仅出现一次,则称该简单合取式为极小项。由若干个不同的极小项组成的析取式称为主析取范式;与A等价的主析取范式称为A的主析取范式。任意含n个命题变元的非永假命题公式A都存在与其等价的主析取范式,并且是惟一的。

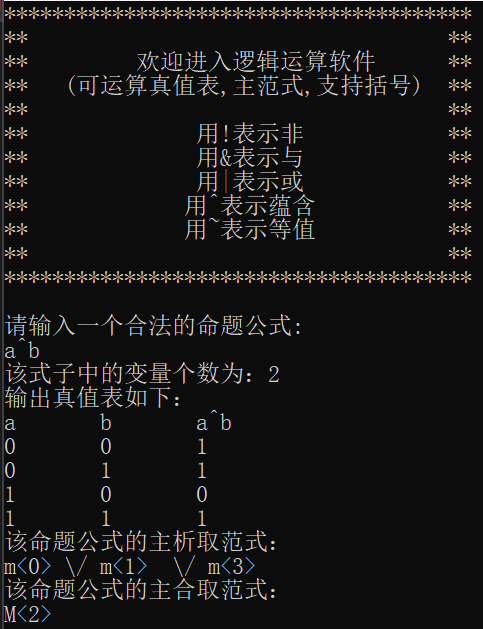
**主合取范式**：在含有n个命题变元的简单析取式中，若每个命题变元与其否定不同时存在，而两者之一出现一次且仅出现一次，称该简单析取式为极大项。由若干个不同的极大项组成的合取式称为主合取范式；与A等价的主合取范式称为A的主合取范式。任意含n个命题变元的非永真命题公式A都存在与其等价的主合取范式，并且是惟一的。

## 2.实验过程

（2）B，C题部分，首先是输入一个合理的式子，然后从式子中查找出变量的个数，开辟一个二进制函数，用来生成真值表，然后用函数运算，输出结果，并根据结果归类给范式，最后输出范式。

函数部分，主要是3个函数，一个为真值表递加函数，通过二进制的加法原理递进产生，一个为分级运算函数，这个函数是通过判断括号，选出最内级括号的内容执行运算函数，这样一级级向外运算，最后得出最终结果，剩下一个为主运算函数，按照运算符号的优先级按顺序进行运算。

**五.项目示例**



1. **项目实现**

项目输入的健壮性实现：本次项目的健壮性主要是要判断输入的命题公式是否合法，在先前代码的基础上，我加了一个输入命题公式的健壮性的函数：

int Judgecorrect(string formula)

{

stack<char> Judge\_Stack; //设置一个栈来暂存符号

size\_t i = 0, length = formula.length(); //循环变量i，表达式的长度length

//检查括号是否匹配

while (i < length)

{

if (formula[i] == '(')//如果是左括号

Judge\_Stack.push(formula[i]);

else if (formula[i] == ')' && !Judge\_Stack.empty())//如果有右括号并且栈内存在左括号

Judge\_Stack.pop();

else if (formula[i] == ')')//如果栈内不存在左括号但是存在右括号

return 1;

i++;

}

if (!Judge\_Stack.empty())

return 1;

//return 1说明括号检测不匹配

i = 0;

//检查在操作符后是否都有命题变项

if (formula[length - 1] == '!' || formula[length - 1] == '|' ||

formula[length - 1] == '&' || formula[length - 1] == '~' ||

formula[length - 1] == '^')

return 2;

//return 2说明在连接词后不存在命题变项

//检查开头是否存在非法操作符

if (formula[0] == '|' || formula[0] == '&' || formula[0] == '~' || formula[0] == '^')

return 3;

//return 3表示开头的连接词输入是否正确

//检查命题公式中是否存在非法符号

while (i < length)

{

if (formula[i] == '(' || formula[i] == ')' ||

(formula[i] >= 'a' && formula[i] <= 'z') ||

(formula[i] >= 'A' && formula[i] <= 'Z') ||

formula[i] == '!' || formula[i] == '|' ||

formula[i] == '&' || formula[i] == '^' ||

formula[i] == '~')

i++;

else

return 4;

}

//return 4说明在公式中存在非法字符

//检查合法字符是否出现连续的情况

i = 0;

while (i < length)

{

if ((formula[i] >= 'a' && formula[i] <= 'z') ||

(formula[i] >= 'A' && formula[i] <= 'Z'))

if ((formula[i + 1] >= 'a' && formula[i + 1] <= 'z') ||

(formula[i + 1] >= 'A' && formula[i + 1] <= 'Z'))

return 5;

if (formula[i] == '!' || formula[i] == '|' ||

formula[i] == '&' || formula[i] == '^' ||

formula[i] == '~')

if (formula[i + 1] == '!' || formula[i + 1] == '|' ||

formula[i + 1] == '&' || formula[i + 1] == '^' ||

formula[i + 1] == '~')

return 5;

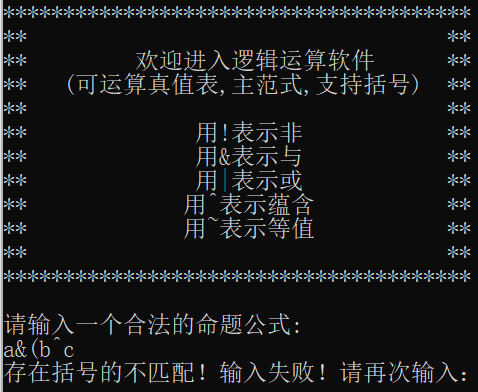
i++;

}

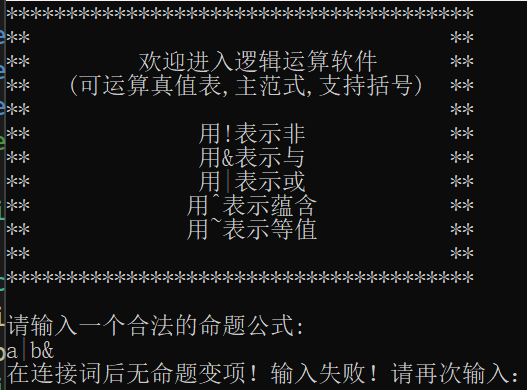
//return 5说明存在合法字符出现连续的情况

return 0;

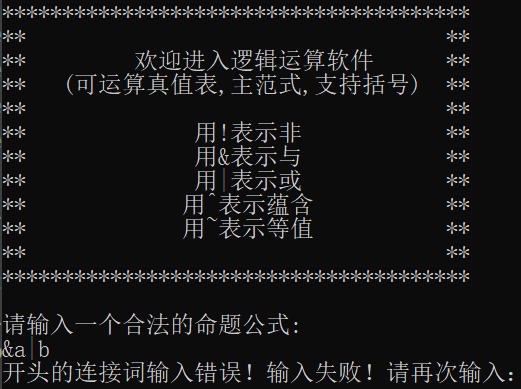
}

return不同的数值表示输入的命题变项中存在不同的错误：return 1说明括号检测不匹配。return 2说明在连接词后不存在命题变项，return 3表示开头的连接词输入错误。return 4说明在公式中存在非法字符，return 5说明存在合法字符出现连续的情况。通过通过暂存辅助栈来判断输入的命题变项中的括号等是否正确，并且通过遍历的方式来判断该命题变项中是否存在非法字符。

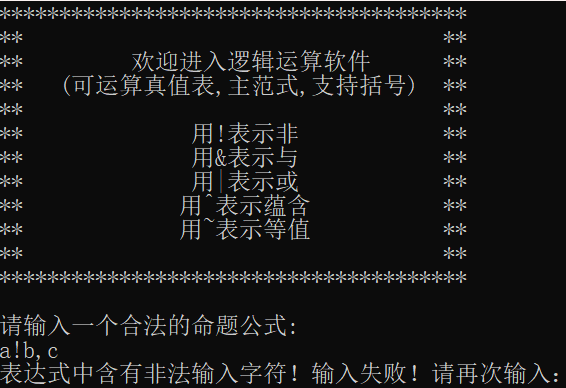
括号检测不匹配：

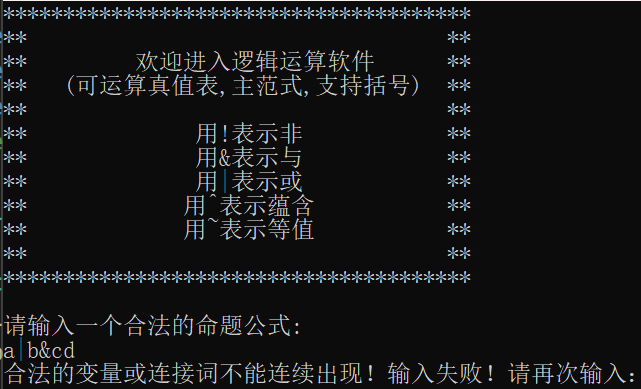


操作符后不存在命题变项：



开头的符号非法：

公式中存在非法字符：



存在合法字符出现连续：

计算真值表、主范式功能的实现：主要是由三个函数完成，一个为真值表递加函数，通过二进制的加法原理递进产生，一个为分级运算函数，这个函数是通过判断括号，选出最内级括号的内容执行运算函数，这样一级级向外运算，最后得出最终结果，剩下一个为主运算函数，按照运算符号的优先级按顺序进行运算。同时在每一个函数中都构建了暂时栈数据结构来完成运算中间步骤的处理。此部分代码较长，放在源代码部分展示。

1. **项目心得**

此次项目是对之前项目的拓展，变得更加得复杂。想要完整地完成此题需要细心处理每一种运算符号，同时要对真值表、主析取范式和主合取范式等理论知识有深刻的理解。通过这次项目，我对离散数学这部分的理论知识有了更加深入的理解，怎么构造主析取范式、怎么构造主合取范式，要补上哪些项，有了更加熟练的运用；同时这次项目极大地提高了我的编程水平和使用stack、map等数据结构的能力，以及细心分部解决问题的能力。

附件：源代码

#include <iostream>

#include <string>

#include <map>

#include <stack>

using namespace std;

typedef map<char, int> Map\_ci;

typedef map<int, char> Map\_ic;

typedef map<int, int> Map\_ii;

//typedef map<int, bool> Map\_ib;

Map\_ci priority;

Map\_ic getProposition(string formula);

int findProposition(Map\_ic, char p);

int pow2(int n);

Map\_ii toBinary(int n\_proposition, int index);

int calculate(string formula, Map\_ic pSet, Map\_ii value);

void check(stack<int>& value, stack<char>& opter);

int Judgecorrect(string str); //检查输入的公式是否合法的函数

//主函数的实现

int main()

{

priority['('] = 6;

priority[')'] = 6;

priority['!'] = 5;

priority['&'] = 4;

priority['|'] = 3;

priority['^'] = 2;

priority['~'] = 1;

priority['#'] = 0;

//运算符优先级

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n";

cout << "\*\* \*\*\n";

cout << "\*\* 欢迎进入逻辑运算软件 \*\*\n";

cout << "\*\* (可运算真值表,主范式,支持括号) \*\*\n";

cout << "\*\* \*\*\n";

cout << "\*\* 用!表示非 \*\*\n";

cout << "\*\* 用&表示与 \*\*\n";

cout << "\*\* 用|表示或 \*\*\n";

cout << "\*\* 用^表示蕴含 \*\*\n";

cout << "\*\* 用~表示等值 \*\*\n";

cout << "\*\* \*\*\n";

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n";

cout << "请输入一个合法的命题公式: " << endl;

string formula;//检查输入的命题公式是否合法

enter: cin >> formula;

int Flag = Judgecorrect(formula);

switch (Flag)

{

case 1:

cout << "存在括号的不匹配！输入失败！请再次输入：" << endl;

goto enter;

break;

case 2:

cout << "在连接词后无命题变项！输入失败！请再次输入：" << endl;

goto enter;

break;

case 3:

cout << "开头的连接词输入错误！输入失败！请再次输入：" << endl;

goto enter;

break;

case 4:

cout << "表达式中含有非法输入字符！输入失败！请再次输入：" << endl;

goto enter;

break;

case 5:

cout << "合法的变量或连接词不能连续出现！输入失败！请再次输入：" << endl;

goto enter;

break;

default:

break;

}

Map\_ic proposition\_set = getProposition(formula);

cout << "该式子中的变量个数为：" << proposition\_set.size() << endl << "输出真值表如下：" << endl;

for (unsigned int i = 0; i < proposition\_set.size(); i++) {

cout << proposition\_set[i] << "\t";

}

cout << formula << endl;

int\* m;

m = (int\*)malloc(sizeof(int) \* pow2(proposition\_set.size())); //该数组依次存放命题公式的各行(0或1)的运算结果的值

// int z = pow2(proposition\_set.size());

// cout << z << endl;

for (int i = 0; i < pow2(proposition\_set.size()); i++)

{

Map\_ii bina\_set = toBinary(proposition\_set.size(), i);

for (unsigned int j = 0; j < bina\_set.size(); j++)

{

cout << bina\_set[j] << "\t";

}

int result = calculate(formula, proposition\_set, bina\_set);

//m[i] = result;

\*(m + i) = result;

cout << result << endl;

}

int n\_m = 0, n\_M = 0;

cout << "该命题公式的主析取范式：" << endl;

for (int i = 0; i < pow2(proposition\_set.size()); i++)

{

if (\*(m + i) == 1)//主析取范式，值为真

{

if (n\_m == 0)//第一个范式

{

cout << "m<" << i << ">";

}

else

{

cout << " \\/ m<" << i << "> ";

}

n\_m++;

}

}

if (n\_m == 0)//没有范式

{

cout << "0";

}

cout << endl;

cout << "该命题公式的主合取范式：" << endl;

for (int i = 0; i < pow2(proposition\_set.size()); i++)

{

if (\*(m + i) == 0)//主合取范式，值为假

{

if (n\_M == 0)//第一个范式

{

cout << "M<" << i << ">";

}

else

{

cout << " /\\ M<" << i << "> ";

}

n\_M++;

}

}

if (n\_M == 0)//没有范式

{

cout << "0";

}

cout << endl;

return 0;

}

//判断错误输入

int Judgecorrect(string formula)

{

stack<char> Judge\_Stack; //设置一个栈来暂存符号

size\_t i = 0, length = formula.length(); //循环变量i，表达式的长度length

//检查括号是否匹配

while (i < length)

{

if (formula[i] == '(')//如果是左括号

Judge\_Stack.push(formula[i]);

else if (formula[i] == ')' && !Judge\_Stack.empty())//如果有右括号并且栈内存在左括号

Judge\_Stack.pop();

else if (formula[i] == ')')//如果栈内不存在左括号但是存在右括号

return 1;

i++;

}

if (!Judge\_Stack.empty())

return 1;

//return 1说明括号检测不匹配

i = 0;

//检查在操作符后是否都有命题变项

if (formula[length - 1] == '!' || formula[length - 1] == '|' ||

formula[length - 1] == '&' || formula[length - 1] == '~' ||

formula[length - 1] == '^')

return 2;

//return 2说明在操作符后不存在命题变项

//检查开头是否存在非法操作符

if (formula[0] == '|' || formula[0] == '&' || formula[0] == '~' || formula[0] == '^')

return 3;

//return 3表示开头的符号非法

//检查命题公式中是否存在非法符号

while (i < length)

{

if (formula[i] == '(' || formula[i] == ')' ||

(formula[i] >= 'a' && formula[i] <= 'z') ||

(formula[i] >= 'A' && formula[i] <= 'Z') ||

formula[i] == '!' || formula[i] == '|' ||

formula[i] == '&' || formula[i] == '^' ||

formula[i] == '~')

i++;

else

return 4;

}

//return 4说明在公式中存在非法字符

//检查合法字符是否出现连续的情况

i = 0;

while (i < length)

{

if ((formula[i] >= 'a' && formula[i] <= 'z') ||

(formula[i] >= 'A' && formula[i] <= 'Z'))

if ((formula[i + 1] >= 'a' && formula[i + 1] <= 'z') ||

(formula[i + 1] >= 'A' && formula[i + 1] <= 'Z'))

return 5;

if (formula[i] == '!' || formula[i] == '|' ||

formula[i] == '&' || formula[i] == '^' ||

formula[i] == '~')

if (formula[i + 1] == '!' || formula[i + 1] == '|' ||

formula[i + 1] == '&' || formula[i + 1] == '^' ||

formula[i + 1] == '~')

return 5;

i++;

}

//return 5说明存在合法字符出现连续的情况

return 0;

}

//返回-1，表示该命题变项尚未被遍历过，可计数；否则说明该命题变项已被遍历过，则不重复计数。另外，还可以返回指定命题变项的下标

int findProposition(Map\_ic pSet, char p)

{

Map\_ic::iterator it = pSet.begin();

while (it != pSet.end())

{

if (it->second == p)

{

return it->first;

}

it++;

}

return -1;

}

//该函数返回所输入公式中的命题变项(不包括运算符)

Map\_ic getProposition(string formula)

{

Map\_ic proposition;

int n\_proposition = 0;

for (unsigned int i = 0; i < formula.length(); i++)

{

char c = formula[i];

if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))

{

//遍历所有命题变项

int r = findProposition(proposition, c);

//cout << r << endl;

if (r == -1)

{

//说明该命题变项尚未被遍历过

proposition[n\_proposition] = c;

n\_proposition++;

}

}

else if (!priority.count(c))

{

cout << c << " is undefined!" << endl;

exit(2);

}

}

return proposition;

}

//该函数返回命题变项的二进制(0或1)取值

Map\_ii toBinary(int n\_proposition, int index)

{

Map\_ii result;

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

{

int r = index % 2;

result[n\_proposition - 1 - i] = r;

index = index / 2;

}

return result;

}

//该函数返回指定数字的二次方的值

int pow2(int n)

{

if (n == 0)

return 1;

else

return 2 \* pow2(n - 1);

}

//该函数返回给定，命题变项(值取0或1，可含括号)组合的运算结果

int calculate(string formula, Map\_ic pSet, Map\_ii value)

{

stack<char> opter;

stack<int> pvalue;

opter.push('#');

formula = formula + "#";

for (unsigned int i = 0; i < formula.length(); i++)

{

char c = formula[i];

if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))

{

pvalue.push(value[findProposition(pSet, c)]);

}

else

{

//此时遍历的是运算符

char tmp = opter.top();

if (priority[tmp] > priority[c])

{

while (priority[tmp] > priority[c] && tmp != '(')

{

check(pvalue, opter);

tmp = opter.top();

if (tmp == '#' && c == '#')

{

return pvalue.top();

}

}

opter.push(c);

}

else

opter.push(c);

}

}

return -1;

}

//该函数返回两个命题变项(取值0或1)的各种运算结果(0或1)

void check(stack<int>& value, stack<char>& opter)

{

int p, q, result;

char opt = opter.top();

switch (opt)

{

case '&':

p = value.top();

value.pop();

q = value.top();

value.pop();

result = p && q;

value.push(result);

opter.pop();

break;

case '|':

p = value.top();

value.pop();

q = value.top();

value.pop();

result = p || q;

value.push(result);

opter.pop();

break;

case '!':

p = value.top();

value.pop();

result = !p;

value.push(result);

opter.pop();

break;

case '^':

q = value.top();

value.pop();

p = value.top();

value.pop();

result = !p || q;

value.push(result);

opter.pop();

break;

case '~':

p = value.top();

value.pop();

q = value.top();

value.pop();

result = (!p || q) && (p || !q);

value.push(result);

opter.pop();

break;

case '#':

break;

case '(':

break;

case ')':

opter.pop();

while (opter.top() != '(')

{

check(value, opter);

}

if (opter.top() == '(')

{

opter.pop();

}

break;

default:

break;

}

}