《离散数学》课程实验报告 1 命题逻辑联接词、真值表、主范式

2254321 陆诚彬

1. 实验内容

1. 从键盘输入两个命题变元 P 和 Q 的真值，求它们的合取、析取、条件和双向条件的真值。（A）。

2. 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：

1.1. 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。

1.2. 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 2^n-1，则结束，否则转（2）。

2. 解题思路

2.1. A 题

输入处理：接收用户输入的命题变元 P 和 Q 的真值（0 或 1）。验证输入的有效性。

逻辑运算实现：使用 C/C++中的逻辑运算符(&& for AND, || for OR, ! for NOT)来计算合取、析取、条件和双向条件的真值。实现条件和双向条件逻辑，可能需要结合逻辑运算符和条件语句。

真值表生成：对于给定的命题公式，生成所有可能的命题变元组合。计算每种组合下公式的真值。输出真值表。

主范式计算：根据真值表，找出公式为真的所有命题变元组合（主析取范式）或为假的组合（主合取范式）。构造主范式表达式。

技术难点：处理复杂的逻辑表达式。确保各种逻辑组合的准确性。

2.2. BC 题

输入和变量解析：接收用户输入的命题公式。解析公式中的变量和操作符。

真值表生成函数：使用递归方法生成命题变元的所有可能组合。用二进制计数模拟器来模拟不同的真值分配。

表达式计算函数：根据操作符优先级，计算表达式的真值。可以使用栈来处理操作符和操作数。

主范式构建：根据生成的真值表，构建主析取范式或主合取范式。使用递归或迭代方法来构建范式表达式。

技术难点：实现高效的递归和迭代算法。精确解析和计算复杂的命题逻辑表达式。

3. 数据结构设计

由于 A 题采用 C/C++自带的逻辑运算符，比较简单，在此处不再赘述，以下内容仅分析 BC 题。

3.1. 映射类型 Map_ci 和 Map_ic

这两个映射类型（map<char, int> 和 map<int, char>）分别用于存储字符到整数的映射和整数到字符的映射。在逻辑运算软件中，这些映射被用来存储命题变量及其对应的索引。

设计:

Map_ci (字符到整数映射):

用途: 存储运算符和它们的优先级。

键: 单个字符，表示一个运算符（如 '!', '&', '|', '^', '~'）。

值: 整数，表示运算符的优先级。

示例: priority['&'] = 4; 表示运算符 '&' 的优先级是 4。

Map_ic (整数到字符映射):

用途: 存储命题变量及其索引。

键: 整数，表示命题变量的索引。

值: 字符，表示一个命题变量。

示例: 在处理逻辑公式时，每个唯一的命题变量都被分配一个整数索引。

3.2. 映射类型 Map_ii

这个映射类型（map<int, int>）用于存储整数到整数的映射。在程序中，它主要用于表示命题变量的二进制值。

用途：存储命题变量的二进制值。

键：整数，表示命题变量的索引。

值：整数（0 或 1），表示命题变量的值。

示例：在生成真值表时，Map_ii 用于表示每个变量在特定行的值。

3.3. 栈结构

程序中使用了两个栈：stack<int> 和 stack<char>，分别用于存储命题变量的值和逻辑运算符。

设计:

stack<int> pvalue:

用途：存储命题变量的值。

操作：在计算逻辑表达式的过程中，将命题变量的值（0 或 1）压入栈中。

stack<char> opter:

用途：存储逻辑运算符。

操作：在处理逻辑表达式时，运算符根据它们的优先级被压入或弹出栈。

4. 项目实现

在这一板块，笔者选择了几个关键函数来详细解释它们的实现。这些函数对于理解和实现逻辑运算软件的整体功能至关重要。

4.1. Map_ic getProposition(string formula)

此函数用于从给定的逻辑公式字符串中提取所有唯一的命题变量，并为每个变量分配一个唯一的整数索引。

参数：string formula - 输入的逻辑公式字符串。

返回值：Map_ic - 包含公式中所有唯一命题变量的映射，每个变量对应一个唯一索引。

流程:

1) 创建一个 Map_ic 类型的映射 proposition。
2) 遍历公式中的每个字符。
3) 检查每个字符是否是一个命题变量（字母）。

4) 如果是，检查它是否已存在于映射中。

5) 如果不存在，将其添加到映射中，并为其分配一个唯一的索引。

```
// 此函数用于提取并返回逻辑公式中所有唯一的命题变量。
// formula: 输入的逻辑公式字符串
// 返回值: 包含公式中所有唯一命题变量的映射，每个变量对应一个唯一索引。
Map_ic getProposition(string formula)
{
    Map_ic proposition; // 创建存储命题变量的映射
    int n_proposition = 0; // 用于给命题变量分配唯一的索引
    for (unsigned int i = 0; i < formula.length(); i++) // 遍历公式每个字符
    {
        char c = formula[i]; // 当前字符
        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
        {
            int r = findProposition(proposition, c); // 检查字符是否在映射中
            if (r == -1)
            {
                // 如果该字符（命题变量）还未在映射中，添加它
                proposition[n_proposition] = c;
                n_proposition++; // 增加索引计数
            }
        }
        else if (!priority.count(c)) // 如果字符不是一个已定义的运算符
        {
            cout << c << " is undefined!" << endl; // 打印错误消息
            exit(2); // 退出程序
        }
    }
    return proposition; // 返回包含所有命题变量的映射
}
```

4.2. Map_ii toBinary(int n_proposition, int index)

该函数将一个整数索引转换为其二进制表示，表示特定命题变量组合的布尔值。

参数:

int n_proposition - 命题变量的总数。

int index - 要转换的整数索引。

返回值: Map_ii - 包含命题变量的二进制值的映射。

流程:

1) 创建一个 Map_ii 类型的映射 result。

2) 使用循环将 index 转换为二进制表示。

3) 将每个位的值存储在 result 中。

```
Map_ii toBinary(int n_proposition, int index)
 //该函数返回命题变项的二进制(0或1)取值
{
```

```
    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;
}
```

4.3. int calculate(string formula, Map_ic pSet, Map_ii value)

参数:

string formula - 要计算的逻辑公式。

Map_ic pSet - 命题变量及其索引的映射。

Map_ii value - 命题变量的布尔值。

返回值: int - 公式的布尔值（0 或 1）。

流程:

1) 使用两个栈 opter 和 pvalue 分别存储运算符和命题变量的值。
2) 遍历公式中的每个字符，根据字符的类型（运算符或命题变量）进行相应的处理。
3) 对于运算符，根据优先级进行运算或将其压入栈中。
4) 对于命题变量，将其对应的布尔值压入栈中。
5) 计算表达式的结果，并返回。

```
// 该函数计算给定的逻辑表达式的值。它处理命题逻辑中的基本运算，如非、与、或等。
// formula: 要计算的逻辑公式字符串
// pSet: 包含公式中所有命题变量的集合
// value: 包含命题变量对应值的映射，每个变量对应一个布尔值(0 或 1)
int calculate(string formula, Map_ic pSet, Map_ii value)
{
    stack<char> opter; // 存储运算符的栈
    stack<int> pvalue; // 存储命题变量值的栈
    opter.push('#'); // 在栈底添加一个特殊字符以标识栈底
    formula = formula + "#"; // 在公式末尾也添加特殊字符以标识结束

    // 遍历整个公式
    for (unsigned int i = 0; i < formula.length(); i++)
    {
        char c = formula[i]; // 当前字符

        // 判断字符是否为命题变量
        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
        {
            // 是命题变量，将其对应的布尔值压入栈
            pvalue.push(value[findProposition(pSet, c)]);
        }
        else
        {
```

```
            // 处理运算符
            char tmp = opter.top(); // 查看栈顶运算符
            // 如果栈顶运算符的优先级高于当前运算符
            if (priority[tmp] > priority[c])
            {
                // 当栈顶运算符的优先级大于当前运算符时，执行运算
                while (priority[tmp] > priority[c] && tmp != '(')
                {
                    check(pvalue, opter); // 执行运算
                    tmp = opter.top(); // 更新栈顶运算符
                    // 如果遇到公式结束标识符，则返回结果
                    if (tmp == '#' && c == '#')
                    {
                        return pvalue.top(); // 返回计算结果
                    }
                }
                // 当前运算符压入栈
                opter.push(c);
            }
            else
            {
                // 如果栈顶运算符优先级不高于当前运算符，直接压栈
                opter.push(c);
            }
        }
    }
    // 如果没有正确计算出结果，则返回-1
    return -1;
}
```

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

此函数用于执行逻辑运算，根据栈顶的运算符和命题变量的值计算结果。

参数:

stack<int> &value - 存储命题变量值的栈。

stack<char> &opter - 存储运算符的栈。

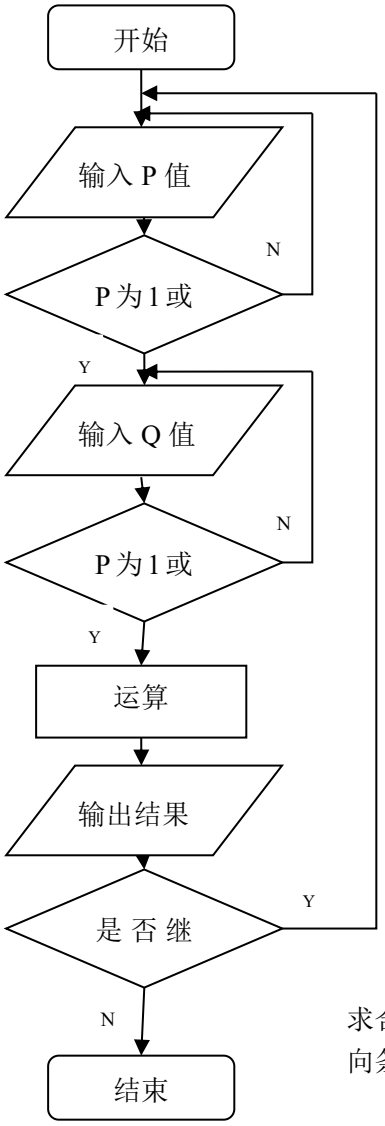
返回值: 无（直接修改栈中的内容）。

流程:

1) 取出栈顶的运算符。
2) 根据运算符类型（如 &, |, !, ^, ~），从 value 栈中取出相应数量的命题变量值。
3) 执行逻辑运算。
4) 将结果压回 value 栈中。
5) 从 opter 栈中弹出已处理的运算符。

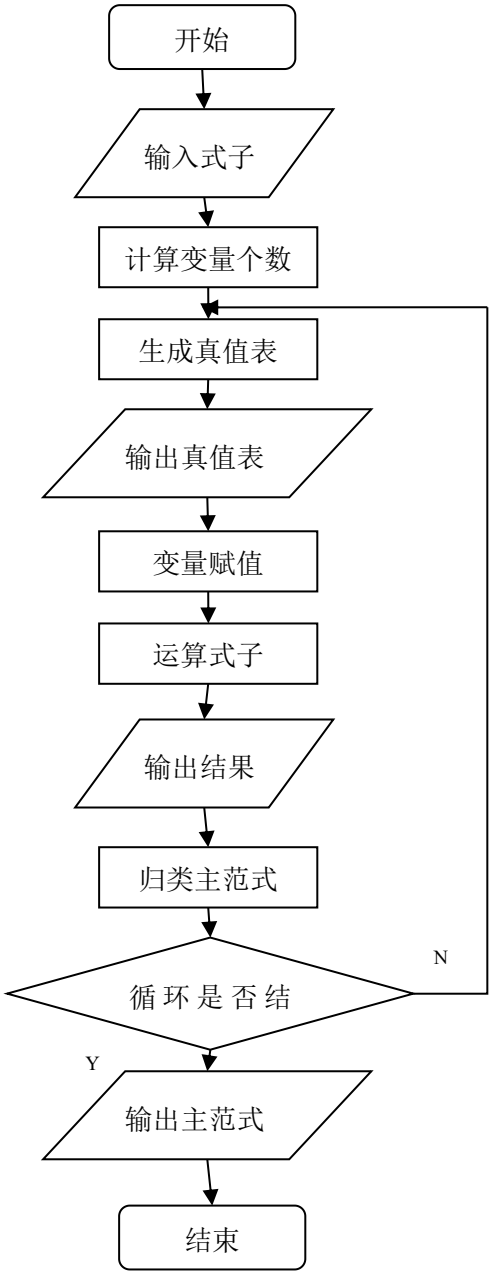
4.5. 项目流程图

4.5.1. A 题



求合取、析取、条件和双向条件的真值流程图

4.5.2. BC 题


开始
输入式子
计算变量个数
生成真值表
输出真值表
变量赋值
运算式子
输出结果
归类主范式
循 环 是 否 结
N
Y
输出主范式
结束

5. 设计小结

5.1. 逻辑联接词的实现与应用

实验首先通过C/C++语言实现了基本的逻辑联接词运算：合取、析取、条件和双向条件。通过利用C/C++中内置的逻辑运算符（如 &&, ||, !），实现了这些逻辑运算。这部分的设计关键在于理解各逻辑运算符的运算规则及其在程序中的应用。

5.2. 真值表的生成和处理

真值表生成是实验的核心部分，需要根据输入的命题公式计算出所有可能的命题变元组合下的真值。利用二进制加法模拟器的概念，实现了一个高效的真值表生成方法。实现了 Map_ci 和 Map_ic 映射类型，以及相关的函数，如 getProposition 和 toBinary，这些都是为了有效处理命题变量和它们的真值。

5.3. 主范式的构建

根据生成的真值表，实验中实现了主析取范式和主合取范式的构建。主范式的构建涉及到从真值表中提取特定的组合，并转换为相应的逻辑表达式。这部分的难点在于如何从真值表中准确地提取和表示所需的逻辑组合。

5.4. 程序结构与数据处理

使用了映射和栈这两种数据结构来存储和处理逻辑表达式中的元素。栈结构用于处理逻辑运算符和命题变量的值，映射结构用于存储命题变量及其对应的布尔值和索引。实现了几个关键函数（如 calculate 和 check），这些函数对于逻辑运算的正确执行至关重要。

5.5. 技术难点与解决方案

处理复杂逻辑表达式和确保逻辑运算的准确性是实验的主要挑战。通过有效的数据结构设计和算法实现，解决了这些技术难点。程序代码的模块化和函数的细粒度设计有助于提高代码的可读性和可维护性。

综上所述，本实验在理论和实践层面都对离散数学中的命题逻辑运算有深入的探讨和实现，展示了从基础逻辑运算到复杂逻辑表达式处理的完整过程。通过这个实验，不仅加深了对离散数学理论的理解，也提高了编程实践能力。