

**实 验 报 告**

**（2019 / 2020 学年 第 一 学期）**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 离散数学 | | | | | |
| 实验名称 | 利用真值表法求取主析取范式以及主合取范式的实现 | | | | | |
| 实验时间 | 2021 | 年 | 9 | 月 | 20 | 日 |
| 指导单位 | 计算机学院计算机科学与技术系 | | | | | |
| 指导教师 |  | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 学生姓名 |  | 班级学号 |  |
| 学院(系) |  | 专 业 |  |

**实 验 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验名称** | 利用真值表法求取主析取范式以及主合取范式的实现 | | | **指导教师** |  |
| **实验类型** | **验证** | **实验学时** | **4** | **实验时间** | 2021-09-20 |
| 1. **实验目的和要求**   利用真值表法求取主析取范式以及主合取范式，通过上机实践进一步理解析取、合取范式的求法以及范式与二进制一一对应的关系。 | | | | | |
| 二、**实验环境(实验设备)**  硬件：微型计算机  软件：Arch Linux操作系统，GCC 11.1.0编译器，C语言代码编辑器等 | | | | | |
| **三、实验原理及内容**  传统表达式计算所使用方式是，将变量的数值与对应字母建立映射关系，之后利用栈或构造表达式树，将变量代入，完成计算。对于本次实验，每一次计算把命题变元分别赋值为0或1，计算结果为1，真值表对应行为真。计算2n次后，完成整个真值表的构造。这种方法虽然可行，但**针对逻辑表达式的计算**比较泛化，有以下两个较大问题：  1. 逻辑表达式里，变量取值只有零一两种。然而计算机最大一次取地址可取64位，即其中最多63位被浪费了。  2. 逻辑表达式里，仅有一种单目运算符“非”，剩下的“合取”“析取”“条件”“双条件”优先级相同且为了表达式的可读性，不同种二元运算符绝大多数情况需要加括号。  针对问题1，把整数的每一位都用来存储一种真值指派。这样，32位整数可以为log232=5个不同的命题变元提供所有不同的指派。定义5个整数，每一个整数代表一个变元的32种指派。它们的最低位到最高为的组合各个不同。即：  A：0b01010101010101010101010101010101 0xaaaaaaaa  B: 0b11001100110011001100110011001100 0xcccccccc  C: 0b11110000111100001111000011110000 0xf0f0f0f0  D: 0b11111111000000001111111100000000 0xff00ff00  E: 0b11111111111111110000000000000000 0xffff0000  针对问题2，由于逻辑表达式优先级由括号确定，本程序求值采用可读性高的从左至右递归求值法。即将式子拆成若干个数字、运算符，从左至右依次求。若遇到括号，先递归地把括号的内容当作子表达式求值，求出来的值当作父表达式的数字。  获取了计算结果后，从最低位向最高位遍历。若对应位为1，代表该命题指派为真。以此输出整个真值表。范式的输出过程类似，通过取出的位是否为1判断是否需要加非符号。  复杂度方面，一般方法每一次计算，每个运算式都被计算一遍，有m个变元就有2m种指派，复杂度为O(n\*2m);改进后若使用\_\_int128\_t，7个命题及以下运算时间相同，复杂度O(n)。7个命题变元已经覆盖了绝大部分命题。但是面对更多命题变元计算机无法同时对更多位的整数同时运算，需要使用数组，复杂度仍然为O(n\*2m)，但计算次数是传统方法的1/128。  由于本程序仅为实验，使用32位整数，支持5个命题变元。完整源代码与注释如下：  #ifdef \_\_cplusplus  #error C CODE, NO C++ COMPILERS  #endif  #include <stdlib.h>  #include <stdio.h>  #include <ctype.h>  #include <stdbool.h>  // 五个基本命题变元 32位整数的每一位对应一种真假组合  const unsigned baseProp[5] = {  0xaaaaaaaa, 0xcccccccc, 0xf0f0f0f0,  0xff00ff00, 0xffff0000  };  // 所有要用的运算符  const char oprands[] = {  '&', '|', '>', '<', '~',  94, 118, 26, 29, 170, '(', ')', '\0'  };  // 前置声明  unsigned evalExpr(const char\* src, const char\*\* end, const char\* letter);  // 在str里寻找c所在的下标 未找到返回-1  int charAt(char c, const char\* str) {  int res = 0;  while (str[res] != c && str[res])  res++;  return str[res] == c ? res : -1;  }  // 提取src里的命题变元所使用字母放进dest里面，返回个数  int getLetter(const char\* src, char\* dest) {  char\* const begin = dest;  \*begin = '\0';  while (\*src) {  // 不是运算符且还没有被写进字母表， 即在运算符和字母表里都找不到  if (charAt(\*src, oprands) == -1 && charAt(\*src, begin) == -1)  \*dest++ = \*src;  src++;  }  \*dest = '\0';  return dest - begin;  }  // 计算两个操作数的运算结果  unsigned calcOne(unsigned lhs, unsigned rhs, char op) {  switch (charAt(op, oprands)) {  case 0: case 5: // 合取&  return lhs & rhs;  case 1: case 6: // 析取|  return lhs | rhs;  case 2: case 7: // 条件->  return (~lhs) | rhs;  case 3: case 8: // 双条件<->  return ~(lhs ^ rhs);  default:  printf("ERROR: %c is not an operator\n", op);  exit(EXIT\_FAILURE);  }  return 0x44f8a1ef; //不会走到这里的  }  // 从src里获取一个运算数的值，获取后的位置放进end里，letter为命题变元的字母  unsigned getNum(const char\* src, const char\*\* end, const char\* letter) {  bool notTag = 0; unsigned num;  while (\*src == oprands[4] || \*src == oprands[9])  notTag = !notTag, ++src;  if (\*src == '(')  num = evalExpr(src + 1, &src, letter); //遇到括号当作完整表达式求出运算数的值  else if (charAt(\*src, letter) == -1) {  printf("ERROR: found %c when expecting a proposition\n", \*src);  exit(EXIT\_FAILURE);  } else num = baseProp[charAt(\*src++, letter)];  \*end = src;  return notTag ? ~num : num;  }  // 求表达式src的值，求取后的位置放在end里，letter为命题变元的字母  unsigned evalExpr(const char\* src, const char\*\* end, const char\* letter) {  unsigned lhs, rhs; char op;  lhs = getNum(src, &src, letter);  while (\*src != ')' && \*src) {  op = \*src++;  rhs = getNum(src, &src, letter);  lhs = calcOne(lhs, rhs, op);  }  if (end)  \*end = (\*src == ')' ? src + 1 : src);  return lhs;  }  // 根据获取的值（真值表），把主析取范式写入dest里  char\* orFormula(unsigned res, const char\* letters, int letterNum, char\* dest) {  for (int i = 0; i < (1 << letterNum); i++) {  if (!(res & (1 << i))) continue;  \*dest++ = '(';  for (int j = 0; j < letterNum; j++) {  if (!(baseProp[j] & (1 << i)))  \*dest++ = oprands[4];  \*dest++ = letters[j]; \*dest++ = oprands[5];  }  \*(dest - 1) = ')'; \*dest++ = oprands[6];  }  \*(dest - 1) = '\n'; return dest;  }  // 根据获取的值（真值表），把主合取范式写入dest里  char\* andFormula(unsigned res, const char\* letters, int letterNum, char\* dest) {  for (int i = 0; i < (1 << letterNum); i++) {  if (res & (1 << i)) continue;  \*dest++ = '(';  for (int j = 0; j < letterNum; j++) {  if (baseProp[j] & (1 << i))  \*dest++ = oprands[4];  \*dest++ = letters[j]; \*dest++ = oprands[6];  }  \*(dest - 1) = ')'; \*dest++ = oprands[5];  }  \*(dest - 1) = '\n'; return dest;  }  // 判断输入的命题是否合法，并去除空格  bool initProposition(char\* prop) {  static const char lBraces[] = "([{", rBraces[] = ")]}";  const char\* inIter = prop; char\* outIter = prop;  int braceDepth = 0;  while (\*inIter) {  if (isspace(\*inIter)) {  inIter++; continue;  }  if (charAt(\*inIter, lBraces) != -1)  ++braceDepth;  else if (charAt(\*inIter, rBraces) != -1)  --braceDepth;  if (braceDepth < 0) // 右括号大于左括号数目  return false;  \*outIter++ = \*inIter++;  }  \*outIter = '\0';  return braceDepth == 0; // 左括号大于右括号数目  }  // 把获取的值打印为真值表  char\* trueValTable(unsigned res, const char\* letters, int letterNum, char\* dest) {  for (int i = 0; i < letterNum; i++)  \*dest++ = letters[i], \*dest++ = '\t';  \*dest++ = '\*', \*dest++ = '\n';  for (int i = 0; i < (1 << letterNum); i++) {  for (int j = 0; j < letterNum; j++)  \*dest++ = (baseProp[j] & (1 << i)) ? 'T' : 'F', \*dest++ = '\t';  \*dest++ = (res & (1 << i) ? 'T' : 'F'), \*dest++ = '\n';  }  return dest;  }  // 完整显示命题表达式expr的主合取析取范式  void display(char\* expr) {  char letters[6] = "", buf[1024] = "";  if (!initProposition(expr))  puts("Brackets Do Not Match"), exit(EXIT\_FAILURE);  int letLen = getLetter(expr, letters);  if (letLen > 5)  puts("Too Much Base Propositions"), exit(EXIT\_FAILURE);  unsigned res = evalExpr(expr, NULL, letters);  andFormula(res, letters, letLen,  orFormula(res, letters, letLen, buf));  puts(buf);  }  int main(int argc, char\*\* argv) {  if (argc < 2) {  char exprIn[100];  scanf("%s", exprIn);  display(exprIn);  } else for (int i = 1; i < argc; i++)  display(argv[i]);  }  下面是程序的运行测试：由于带有5个命题的真值表过于庞大，因此分为两个图    图1 不输出真值表时的效率测试    图2 输出真值表时的效率测试  如图，当出现输入错误时，会提示哪里出错，如运算符过多、括号不匹配。由于采用按位存储命题的真假情况，充分利用了32位整数的每个位的内容，即使是常规需要运行32次的五变元命题也能一次运行得出结果，再加上纯C语言环境避免了各种动态容器的内存分配，大大提升了程序运行效率。计算输出3个命题的真值表与范式只需要3毫秒。  离散数学里的逻辑运算是0和1的运算，利用计算机整数的寻址特点，同时进行大量逻辑运算是可行的：各命题变元和最终结果从最低位到最高位，每一位代表真值表上的一行。 | | | | | |

**实 验 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **四、实验小结**（包括问题和解决方法、心得体会、意见与建议等）  **说明：这部分内容主要包括：在编程、调试或测试过程中遇到的问题及解决方法、本次实验的心得体会、进一步改进的设想等。**  (一)实验中遇到的主要问题及解决方法  起初把表达式看作“左操作数 运算符 右操作数”，这样做认为一切算式都必须被括号括起来，遇到A&B&～C就崩溃。后来把获取操作数值和表达式计算分开，代码更可读且解决了问题。  测试程序时，输入命令./a.out P>Q时什么都不输出。最后想到>被当作输出重定向符，输出到了名称为Q的文档里。打开文件Q，里面有单个命题P的真值表与范式，程序本身没有问题。  （二）实验心得  遇到某个特殊的问题，可以结合问题本身的特点，找到使用一般方法时可能造成的资源利用不充分等问题，把一般算法改进成只适用当前问题但是更高效的实例化算法。  对于有更多变元的命题，可以通过增加整数的位数，如把数组看作“大整数”，同样可以做到一次计算求出所有结果。  （三）意见与建议（没有可省略） | | | | | |
| 1. **支撑毕业要求指标点**   支撑毕业要求的指标点为：   * 1-4掌握计算机科学与技术领域的专业知识，能将专业知识用于分析和解决计算机领域复杂工程问题。   √   * 2-1能够应用数学、自然科学和工程科学的基本知识，识别和分析计算机领域复杂工程问题的特征。 | | | | | |
| **六、指导教师评语 (含学生能力达成度的评价)** | | | | | |
| **成 绩** |  | **批阅人** |  | **日 期** |  |

如果不太想写太多字，“指导教师评语”也可以设计为如下的各选择项用打勾形式（仅仅作为一个简单示例，请各课程负责人根据课程和实验情况以及支撑的指标点来自行设定选择项，同一门课程的不同实验评分细则项允许存在不同）：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **评 分 细 则** | **评分项** | **优秀** | **良好** | **中等** | **合格** | **不合格** |
| **遵守实验室规章制度** |  |  |  |  |  |
| **学习态度** |  |  |  |  |  |
| **算法思想准备情况** |  |  |  |  |  |
| **程序设计能力** |  |  |  |  |  |
| **解决问题能力** |  |  |  |  |  |
| **课题功能实现情况** |  |  |  |  |  |
| **算法设计合理性** |  |  |  |  |  |
| **算法效能评价** |  |  |  |  |  |
| **回答问题准确度** |  |  |  |  |  |
| **报告书写认真程度** |  |  |  |  |  |
| **内容详实程度** |  |  |  |  |  |
| **文字表达熟练程度** |  |  |  |  |  |
| **其它评价意见** |  | | | | |
| **本次实验能力达成评价（总成绩）** |  |  |  |  |  |