

**实 验 报 告**

**（2019 / 2020 学年 第 一 学期）**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 离散数学 | | | | | |
| 实验名称 | 偏序关系中盖住关系的求取及格论中有补格的判定 | | | | | |
| 实验时间 | 2021 | 年 | 11 | 月 | 19 | 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) | 计算机学院 | 专 业 |  |

**实 验 报 告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **实验名称** | **偏序关系中盖住关系的求取及格论中有补格的判定** | | | **指导教师** |  |
| **实验类型** | **验证** | **实验学时** | **4** | **实验时间** | 2021-11-19 |
| 1. **实验目的和要求**   求取集合*A*上的整除关系*R*对应的盖住关系，并判定偏序集<*A*，*R*>是否为格，若是格，判断其是否为有补格。  要求：集合*A*可以是用户任意给定的正整数集合。 | | | | | |
| 二、**实验环境(实验设备)**  硬件：微型计算机  软件：Ubuntu Linux 20.04, gcc-10编译器,代码编辑器 | | | | | |
| **三、实验原理及内容**  本程序采用半关系矩阵存储关系。由于整除关系只有大元素对小元素，半个矩阵一定是0，对半存储节省空间。定义一个函数，方便代替数组下标访问元素内容。  在整除关系里，两元素的上界一定是它们的最小公倍数的倍数，它们的最大公约数一定是下界的倍数。因为格里任意两个元素都有上下界，它们的最大公约数和最小公倍数一定也在这个格里。据此，对于整除关系，可以只使用四则运算实现格的判定。  确定是否为格后，同样利用最小公倍数的方法，求出此整除关系的所有互补元素，根据是否每一个元素都有补，判断其是否是有补格。由于已经确定关系是格，在这个整除关系里，数值最大元素是最大元，数值最小的是最小元。如果两元素最小公倍数（最小上界）是数值最大的元素，且最大公约数是数值最小的元素，它们一定互补。在把输入数据排序之后，可以方便取用最大最小元，因此可以只用四则运算求出有补格。  程序的完整源代码及注释如下：  #include <stdio.h>  #include <stdlib.h>  #include <inttypes.h>  #include <stdbool.h>  // 存储一个覆盖关系  typedef struct covStore {  size\_t mySize;  uint32\_t\* myVals;  bool\*\* covMat;  } covStore;  // 覆盖关系求取的各种操作  covStore\* initCov(covStore\* this, uint32\_t\* vals, size\_t sz);  void genCover(covStore\* this);  void freeCov(covStore\* this);  void printCov(const covStore\* this);  int main(int argc, char\*\* argv) {  if (argc < 2)  return printf("Usage: %s NUMBERS", argv[0]);  // 命令行参数作为整除关系的各个元素  uint32\_t\* vals = malloc((argc - 1) \* sizeof(uint32\_t));  for (int i = 1; i < argc; i++)  vals[i - 1] = (uint32\_t)atoll(argv[i]);  covStore store;  if (!initCov(&store, vals, argc - 1))  return 1;  free(vals);  genCover(&store);  printCov(&store);  freeCov(&store);  return 0;  }  // 排序比较用函数  int \_\_c\_\_(const uint32\_t\* lhs, const uint32\_t\* rhs) {return \*lhs - \*rhs;}  // 辗转相除求最大公约数  uint32\_t gcdUi32(uint32\_t lhs, uint32\_t rhs) {  if (lhs > rhs)  lhs ^= rhs, rhs ^= lhs, lhs ^= rhs;  while (lhs) {  uint32\_t rem = rhs % lhs;  rhs = lhs;  lhs = rem;  }  return rhs;  }  // 长度为sz的数组vals里的数字初始化覆盖关系  covStore\* initCov(covStore\* this, const uint32\_t\* vals, size\_t sz) {  this->myVals = malloc(sz \* sizeof(uint32\_t));  this->mySize = sz;  for (size\_t i = 0; i < sz; i++)  this->myVals[i] = vals[i];  // 后面操作里的取余运算需要保证存放的数据从小到大排列  qsort(this->myVals, this->mySize, sizeof(unsigned),  (int(\*)(const void\*, const void\*))\_\_c\_\_);  if (this->myVals[0] == 0)  return NULL;  // 二维动态数组的分配，求取前全部赋值false  this->covMat = malloc(this->mySize \* sizeof(bool\*));  for (size\_t row = 0; row < this->mySize - 1; row++)  this->covMat[row] = calloc(this->mySize - 1 - row, sizeof(bool));  return this;  }  // 释放覆盖关系的内存  void freeCov(covStore\* this) {  free(this->myVals);  for (size\_t i = 0; i < this->mySize - 1; i++)  free(this->covMat[i]);  free(this->covMat);  }  // 返回一个关系里在row行col列，即<row, col>二元组是否在覆盖关系内  // 此函数只能访问，不能修改  bool valInCovMat(const covStore\* this, size\_t row, size\_t col) {  if (row == col || row >= this->mySize || col >= this->mySize)  return false;  if (row < col)  return this->covMat[row][col - row - 1];  else return this->covMat[col][row - col - 1];  }  // 返回一个关系里在row行col列内容的指针  // 返回的是指针，可以任意修改，因此只能在初始化相关函数调用  bool\* ptrInCovMat(covStore\* this, size\_t row, size\_t col) {  if (row == col || row >= this->mySize || col >= this->mySize)  return NULL;  if (row < col)  return this->covMat[row] + (col - row - 1);  else return this->covMat[col] + (row - col - 1);  }  // 判断一个关系是不是格 如果是返回-1  // 不是返回二元组，两个元素是无界的  uint64\_t isGrid(const covStore\* this) {  for (size\_t row = 0; row < this->mySize; row++) {  for (size\_t col = row + 1; col < this->mySize; col++) {  // gcd最大公约数 lcm最小公倍数 maxElem最大元 minElem最小元  const uint32\_t gcd = gcdUi32(this->myVals[row], this->myVals[col]),  lcm = this->myVals[row] \* this->myVals[col] / gcd;  const uint32\_t maxElem = this->myVals[this->mySize - 1],  minElem = this->myVals[0];  uint64\_t \_; uint32\_t\* res = (uint32\_t\*)&\_; // 64位整数看做32位整数二元组  if (maxElem % lcm != 0) { // 最大元不是lcm的倍数  res[0] = maxElem; // 则最大元和求lcm的一个或两个数是无界的  res[1] = maxElem % this->myVals[row] == 0 ?  this->myVals[col] : this->myVals[row];  return \_;  }  if (gcd % this->myVals[0] != 0) { // gcd不是最小元的倍数  res[0] = minElem; // 则最小元和求gcd的一个或两个数是无界的  res[1] = this->myVals[row] % minElem == 0 ?  this->myVals[col] : this→myVals[row]; // 这个数是无法整除最小元的那个数  return \_;  }  }  }  return ~0; // ～0是两个负一  }  // 是否是补格，返回不符合（没有补元）的元素  // 是补格则返回-1  uint32\_t isComplimentary(const covStore\* this) {  // 辅助数组，临时存储所有数的补元  uint32\_t\* compliments = malloc(this->mySize \* sizeof(uint32\_t));  uint32\_t self = this->myVals[this->mySize - 1];  for (size\_t i = 0; i < this->mySize; i++) {  for (size\_t j = 0; j < this->mySize; j++)  // 最大下界(gcd)为1且最大上界(lcm)为最大数互为补元  // 由于lcm是两数相乘除以gcd而gcd为1,只要相乘即可  if (this->myVals[i] \* this->myVals[j] == self  && gcdUi32(this->myVals[i], this->myVals[j]) == 1) {  compliments[i] = this->myVals[j];  compliments[j] = this->myVals[i];  }  }  for (size\_t i = 0; i < this->mySize; i++)  if (compliments[i] == 0) {  free(compliments);  return this->myVals[i];  }  free(compliments);  return ~0U;  }  // 打印覆盖关系的信息  void printCov(const covStore\* this) {  puts("Covered Relation");  size\_t printed = 0;  for (size\_t r = 0; r < this->mySize; r++)  for (size\_t c = r + 1; c < this->mySize; c++)  if (valInCovMat(this, r, c)) {  printf("<%u, %u> ", this->myVals[r], this->myVals[c]);  if ((++printed & 0b111) == 0)  putchar('\n');  }  putchar('\n');  uint64\_t gridRes = isGrid(this);  if (gridRes == ~0) {  puts("The division relation is a grid.");  uint32\_t cmplRes = isComplimentary(this);  if (cmplRes == ~0)  puts("The relation is complimentary");  else printf("The set is not complimentary:\n"  "Element %u has no compliment.\n", cmplRes);  } else  printf("The division relation is not a grid.\nTuple <%u, %u> is unbounded\n",  ((uint32\_t\*)&gridRes)[0], ((uint32\_t\*)&gridRes)[1]);  }  // 求取覆盖关系  void genCover(covStore\* this) {  // 先求整除关系  for (size\_t row = 0; row < this->mySize; row++) {  for (size\_t col = row + 1; col < this->mySize; col++)  if (this->myVals[col] % this->myVals[row] == 0)  \*ptrInCovMat(this, row, col) = true;  }  // 再筛选掉被覆盖的关系  for (size\_t row = 0; row < this->mySize; row++) {  for (size\_t col = row + 1; col < this->mySize; col++) {  if (valInCovMat(this, row, col))  for (size\_t i = col + 1; i < this->mySize; i++)  if (valInCovMat(this, col, i))  \*ptrInCovMat(this, row, i) = false;  }  }  }  程序的运行结果：  关系是格且是有补格、不是有补格的情况  不是格的情况 | | | | | |

**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）  (一)实验中遇到的主要问题及解决方法  在最大公约数函数已经确定写对后，有补格的判断也出错。后来发现是括号配对把gcd(a,b)！=0写成了gcd(a, b！=0)，而b==0被转换为整形1，变成了gcd(a,1)。  qsort函数和C++ std::sort不同，它的判断函数以返回值大于、等于、小于0判断条件成立，而C++ STL的断言函数是非0与0表明成立与否。C++写法是return lhs < rhs;，qsort写法是return lhs – rhs;。  （二）实验心得  判断一般的偏序关系是否是格，也可以从数值最大的元素开始，往下沿着关系查找，若查找后发现有元素未被覆盖，这个元素和最大元就没有上界，不是格。再沿最小元开始重复。但对于整除关系，利用整除的特性，可以简化常规算法。  （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）** |  |  |  |  |  |