

**实 验 报 告**

**（2019 / 2020 学年 第 一 学期）**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 课程名称 | 离散数学 | | | | | |
| 实验名称 | 集合上二元关系性质的判定 | | | | | |
| 实验时间 | 2021 | 年 | 11 | 月 | 17 | 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** | **集合上二元关系性质的判定** | | | **指导教师** |  |
| **实验类型** | **验证** | **实验学时** | **4** | **实验时间** | 2021.11.17 |
| 1. **实验目的和要求**   目的：描述本次实验的目的和要求。输入集合A及在A的二元关系R，判断二元关系R的几种基本性质。  要求：能正确判断任意二元关系的自反性、对称性、传递性、反自反性和反对称性，同时求出这几种性质的闭包。 | | | | | |
| 二、**实验环境(实验设备)**  中文五号宋体，英文五号Times new roman字体，1.25倍行距  硬件：微型计算机  软件：Ubuntu Linux 20.04, GCC 10.3.0 C语言编译器 | | | | | |
| **三、实验原理及内容**  本实验有两个要求：集合与关系的输入、关系性质的判定。性质判定在内部处理使用由0,1组成的、行列为集合元素个数的关系矩阵，但用户输入的集合元素却不一定是012等连续数字，需要在输入环节建立合适的字符映射。由于程序主体为性质验证，实验里只允许单个英文字母、数字、符号作为集合元素。计算机里的字母数字共有128个，再考虑集合元素的唯一性，为了实现用户输入的字母与内部元素编号的快速对应，建立两个128长度数组，分别是字符->编号与编号->字符。若出现重复输入，给出提醒并忽略第二次输入。字符->编号里，下标为字符的ASCII码的元素为该字符编号。编号->字符反之。如用户输入的集合第二个元素是H(72)，则  symToLbl[72] == 1 && lblToSym[1] == 72  这样在后面使用时，查找字符只需要O(1)的复杂度。  关系输入，首先将关系矩阵初始化为零，再利用建立的映射，根据用户输入的字符把对应位置赋值为1。若出现未记录的数据，即symToLbl[i] == -1，给出提示并忽略。关系读取有两种方法，第一种输入一个字符串，两两搭配组成关系；第二种输入两个字符串，前一个作为各个二元组的第一元素，后一个作为第二元素。  自反性、对称性、反自反性和反对称性四种性质比较相似，都牵扯到矩阵对应位置元素的比较，为了代码的简洁与复用性能，将其作为两个函数。但若只输出“是否”，当矩阵复杂时，程序结果的可信度难以人工检验。因此，该函数返回使某种性质不满足时的下标，结合前述映射，可以方便输出**在哪里**性质未满足。  对称性比较非对角线元素是否相等，而反对称性比较非对角线元素是否不都为1。遍历上三角的所有元素，和对应下三角元素比较判断。若判断失败则写入返回值对应的位置里。由于返回值初值-1,若无判断失败，则该处值为-1，否则存有一个判断失败位置的下标。自反性、反自反性类似，比较的是对角线元素。在判断这些性质时，将不符合的元素“调整”后写入由参数传入的闭包里。  传递闭包利用Warshall算法并加以处理以实现前述功能。Warshall算法内容如下：   1. 置新矩阵 2. 置 3. 对所有*j*如果则对 4. *i*加一 5. 如果则跳转到步骤3，否则停止。   如果在第三步骤里这一表达式里，发生了改变（由于是逻辑加，只能从0变成1），表明这个位置传递闭包与原关系不符，在此处原关系的传递性存在矛盾。此时把返回值数组以*i j k*赋值，即可在外部利用这一对位置方便推出矛盾。  程序的完整源代码及注释如下：  #ifdef \_\_cplusplus  #error "C CODE, NO C++ ALLOWED"  #else  #include <stdio.h>  #include <stdlib.h>  #include <memory.h>  #include <stdint.h>  #include <stdbool.h>  // 结构定义，包含动态分配的关系矩阵和集合元素表  // 为加快元素查找效率，采用由字符到编号、编号到字符的双向数组映射  struct RelationSet {  bool\* relationMat;  int8\_t symToLbl[128];  int8\_t lblToSym[128];  uint8\_t mySize;  };  typedef struct RelationSet RelationSet;  // “公有”函数前置声明  RelationSet\* initSet(RelationSet\* this, const char\* syms);  void addRelation(RelationSet\* this, const char\* from, const char\* to);  void addRelationAlternate(RelationSet\* this, const char\* src);  void freeSet(RelationSet\* this) {free(this->relationMat);}  void printRelation(RelationSet \*this, int8\_t row, int8\_t col);  int32\_t reflectiveClosure(const RelationSet\* this, RelationSet\* dest);  int32\_t symmestricClosure(const RelationSet\* this, RelationSet\* dest);  int32\_t transectiveClosure(const RelationSet\* this, RelationSet\* dest);  void printSet(const RelationSet\* this);  RelationSet\* copySet(const RelationSet\* this, RelationSet\* dest);  // 获取关系表this里行列对应下标  int16\_t indexAt(const RelationSet\* this, uint8\_t row, uint8\_t col) {  // if (row > this->mySize || col > this->mySize)  // return -1;  return row \* this->mySize + col;  }  #pragma GCC diagnostic ignored "-Wchar-subscripts"  // 以syms里的字符作为集合元素初始化集合，分配关系矩阵  RelationSet\* initSet(RelationSet\* this, const char\* syms) {  this->mySize = 0;  memset(this->symToLbl, -1, sizeof(int8\_t) \* 128);  memset(this->lblToSym, -1, sizeof(int8\_t) \* 128);  while (\*syms) {  if (this->symToLbl[\*syms] == -1)  this->symToLbl[\*syms] = this->mySize++;  else printf("Element repeatition: %c, ignoring\n", \*syms);  syms++;  }  memset(this->relationMat = malloc(this->mySize \* this->mySize),  false, sizeof(bool) \* this->mySize \* this->mySize);  for (uint8\_t i = 0; i < 128; i++)  if (this->symToLbl[i] != -1)  this->lblToSym[this->symToLbl[i]] = i;  return this;  }  // 添加从from对应下标字符至to对应下标的字符的关系至this里，  // 如"ab" "bb"建立<a,b> <b,b>两个关系  void addRelation(RelationSet\* this, const char\* from, const char\* to) {  while (\*from && \*to) {  if (this->symToLbl[\*from] != -1 && this->symToLbl[\*to] != -1)  this->relationMat[indexAt(this, this->symToLbl[\*from],  this->symToLbl[\*to])] = true;  else printf("Tuple <%c, %c> contain unrecorded token\n", \*from, \*to);  ++from, ++to;  }  }  // src里元素交替作为二元组里的左、右至this里  // 如"abbb"建立<a,b> <b,b>两个关系  void addRelationAlternate(RelationSet \*this, const char \*src) {  char from[128] = "", to[128] = ""; uint8\_t i = 0;  while (src[i])  if (i & 1) to[i >> 1] = src[i], i++;  else from[i >> 1] = src[i], i++;  addRelation(this, from, to);  }  // 传入行列（可由上面的函数返回值获得）打印一对关系二元组及其存在情况  void printRelation(RelationSet \*this, int8\_t row, int8\_t col) {  printf("<%c, %c> ", this->lblToSym[row], this->lblToSym[col]);  this->relationMat[indexAt(this, row, col)]  ? printf("exists") : printf("does not exist");  };  // 如果两者大小相等，把this的自反闭包放入dest里,否则只正常返回，不修改dest  // 返回一个以int32\_t包装的int8\_t数组，如果无自反性  // 下标为arr[0] arr[1]的元素不满足自反性 [2][3] 不满足反自反性  // 如果有，对应位置是-1  int32\_t reflectiveClosure(const RelationSet \*this, RelationSet \*dest) {  if (dest)  dest = copySet(this, dest);  int32\_t ret = -1; int8\_t\* arr = (int8\_t\*)&ret;  for (uint8\_t i = 0; i < this->mySize; i++) {  dest && (dest->relationMat[indexAt(dest, i, i)] = true);  if (this->relationMat[indexAt(this, i, i)])  arr[2] = arr[3] = i;  else arr[0] = arr[1] = i;  }  return ret;  }  // 如果两者大小相等，把this的自反闭包放入dest里,否则只正常返回，不修改dest  // 返回一个以int32\_t包装的int8\_t数组，如果无对称性  // 下标为arr[0] arr[1]的元素不满足对称性 [2][3] 不满足反对称性  // 如果有，对应位置是-1  int32\_t symmestricClosure(const RelationSet \*this, RelationSet \*dest) {  if (dest)  dest = copySet(this, dest);  int32\_t ret = -1; int8\_t\* arr = (int8\_t\*)&ret;  for (uint8\_t i = 1; i < this->mySize; i++)  for (uint8\_t j = 0; j < i; j++) {  bool up = this->relationMat[indexAt(this, i, j)],  down = this->relationMat[indexAt(this, j, i)];  if (up != down) { // 对称位置不同，这一对位置不满足对称性  arr[0] = i, arr[1] = j;  dest && (dest->relationMat[indexAt(dest, i, j)] = true);  dest && (dest->relationMat[indexAt(dest, j, i)] = true);  } else if (up && down) // 对称位置全为1，这一对位置不满足反对称性  arr[2] = i, arr[3] = j;  }  return ret;  }  // 如果两关系大小相等，把this的邻接矩阵复制给dest，返回dest  // 否则返回NULL，两者不变  RelationSet \*copySet(const RelationSet \*this, RelationSet \*dest) {  if (this && dest && this->mySize == dest->mySize) {  memcpy(dest->relationMat, this->relationMat,  sizeof(bool) \* this->mySize \* this->mySize);  return dest;  }  return NULL;  }  // 打印由关系组成的集合  void printSet(const RelationSet \*this) {  putchar('{');  for (uint8\_t i = 0; i < this->mySize; i++)  for (uint8\_t j = 0; j < this->mySize; j++) {  if (this->relationMat[indexAt(this, i, j)])  printf("<%c, %c>, ", this->lblToSym[i], this->lblToSym[j]);  }  puts("\b\b}\n");  }  // 如果两者大小相等，把this的传递闭包放入dest里，\*否则什么都不做，返回INT32\_MAX\*  // 返回一个以int32\_t包装的int8\_t数组，如果无传递性  // 下标为arr[0] arr[1]的关系成立，[0][2]成立，但[1][2]不成立，可由此推出无传递性  // 如果有传递性，所有位置是-1。  int32\_t transectiveClosure(const RelationSet \*this, RelationSet \*dest) {  dest && (dest = copySet(this, dest)); // 建立新矩阵dest := this  if (!dest) return INT32\_MAX;  int64\_t ret = -1; int8\_t\* arr = (int8\_t\*)&ret;  for (uint8\_t i = 0; i < this->mySize; i++)  for (uint8\_t j = 0; j < this->mySize; j++) { // Warshall: 对所有j  if (dest->relationMat[indexAt(dest, j, i)]) // 如果A[j,i] = 1,  for (uint8\_t k = 0; k < this->mySize; k++) { // 则对k = 1,2,...,n  bool nVal = dest->relationMat[indexAt(dest, j, k)]  | dest->relationMat[indexAt(dest, i, k)]; // A[j,k] = A[j,k] + a[i,k]  if (nVal && !dest->relationMat[indexAt(dest, j, k)]) {  arr[0] = i, arr[1] = j, arr[2] = k; // 到这里出现矛盾，记录矛盾位置  dest->relationMat[indexAt(dest, j, k)] = true; // 并写入闭包  }  }  }  return ret;  }  int main(int argc, char\*\* argv) {  if (argc <= 2 || argc >= 5)  return printf("Usage: %s SET RELATION [RELATIONTO]\n", argv[0]);  RelationSet\* set = initSet(malloc(sizeof(RelationSet)), argv[1]),  \*closure = initSet(malloc(sizeof(RelationSet)), argv[1]);  if (argc == 3)  addRelationAlternate(set, argv[2]);  else addRelation(set, argv[2], argv[3]);  puts("Your Relation: "); printSet(set);  int64\_t res = reflectiveClosure(set, closure);  int8\_t\* symInfo = (int8\_t\*)&res; // 强制指针转换获得使性质矛盾的元素下标  if (symInfo[0] == -1)  puts("Relation is reflective.");  else printRelation(set, symInfo[0], symInfo[1]), puts(", therefore not reflective");  if (symInfo[2] == -1)  puts("Relation is irreflective.");  // （反）自反性只需打印对角线上的一个不满足即可推出  else printRelation(set, symInfo[2], symInfo[3]), puts(", therefore not irreflective");  puts("Reflective Closure: "); printSet(closure);  res = symmestricClosure(set, closure);  if (symInfo[0] == -1)  puts("Relation is symmestric.");  else { // （反）对称性需要打印对称的两个元素推出不成立  printRelation(set, symInfo[1], symInfo[0]); putchar(',');  printRelation(set, symInfo[0], symInfo[1]); puts(", therefore not symmestric");  }  if (symInfo[2] == -1)  puts("Relation is assymmestric.");  else {  printRelation(set, symInfo[3], symInfo[2]); putchar(',');  printRelation(set, symInfo[2], symInfo[3]); puts(", therefore not assymmestric");  }  puts("Symmestric Closure: "); printSet(closure);  res = transectiveClosure(set, closure);  if (symInfo[0] == -1)  puts("Relation is transective.");  else { // 传递性需要三个元素才可推出不成立  printRelation(set, symInfo[1], symInfo[0]); putchar(',');  printRelation(set, symInfo[0], symInfo[2]); putchar(',');  printRelation(set, symInfo[1], symInfo[2]); puts(", therefore not transective");  }  puts("Transective Closure: "); printSet(closure);  freeSet(set); freeSet(closure); free(set); free(closure);  return 0;  }  #endif  程序的运行结果如下（彩色输出为外部工具lolcat，用于突出显示结果）：  程序的输出明确指出，如果关系不符合某种性质，会指示是哪一/哪些二元组推出的。便于查询关系表进行验证程序是否运行正确。运行速度符合预期，对于21\*21的关系表，也在1毫秒内判定完毕。 | | | | | |

**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  整数在计算机存储方法是8位为分割小位在前，大位在后，与日常书写先写大位后小位不同。在强制用小整数指针a指向大整数时，a[0]是大数的低位。第一次运行受常规书写影响，对称性（最低位a[0]）输出的是不符合反自反性（最高位a[7]）的二元组且方向相反。最终依靠输出的“对称性”意识到整数存储问题。  （二）实验心得  自反性、对称性、反自反性和反对称性四种性质比较相似，都牵扯到矩阵对应位置元素的比较，将其作为两个函数。在各个判断函数里，由于性质的判断与闭包的获取方法相同，把这两个功能写入一个函数，大大简洁了代码。而关系的打印、复制与闭包判断相对独立，各作为一个函数，在各个判断函数与主函数里调用，这种“分而治之”的思想使程序具有模块性。  数组在C语言里是不能被“返回”的，但是有些数据类型所需内存较大，而存储的下标由于矩阵较小，1字节即可完成存储。在这个实验里使用返回多字节整数将其当作单字节整数数组的方式，既提升了函数调用传参效率，更避免了传递数组指针而丢弃返回值的混乱。  （三）意见与建议（没有可省略）  可以进一步规定关系的描述方法，为各个学生的判断程序提供统一接口。 | | | | | |
| 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）** |  |  |  |  |  |