离散数学

项目说明文档

**求关系的自反、对称和传递闭包**



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

**1.实验介绍**

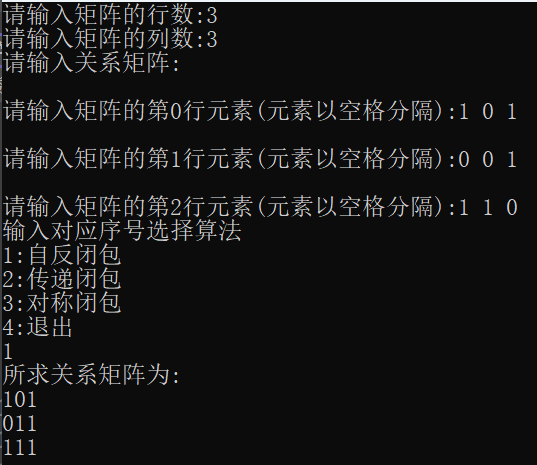
本题是要以C++程序的方式实现自反闭包、对称闭包和传递闭包的构建。对以矩阵表示的关系，其自反闭包只要将矩阵的主对角线全部置为1，对称闭包则由关系矩阵加上其转置矩阵得到（逻辑加），传递闭包则是需要通过构造一些中间矩阵，通过一系列算法和逻辑加的计算，得到最终的传递闭包。

**2.实验要求**

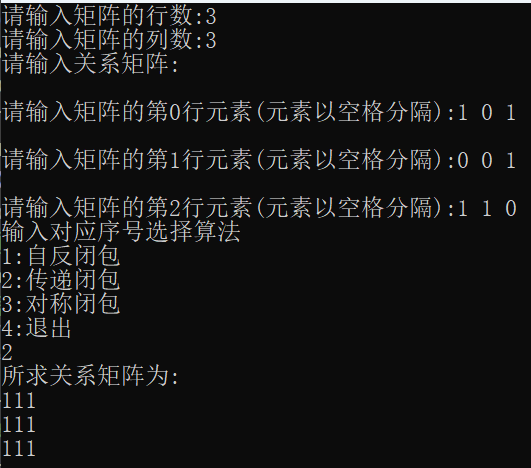
**2.1 输入要求**

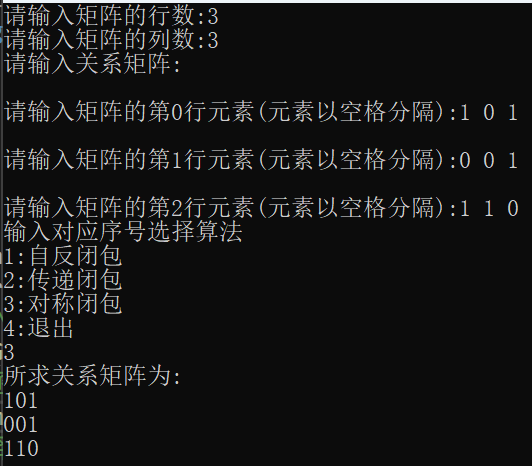
要求用户输入关系矩阵的行数和列数，再通过用户指定的矩阵大小提示用户输入某一行的矩阵具体内容(在输入的每一个过程中都需要做相应的错误处理来保证代码的健壮性)。然后用户可以输入想要的菜单选项数。

**2.2 输出要求**

按用户的要求输出原始关系矩阵通过自反、对称或传递之后的关系矩阵。

**2.3 项目示例**





**3.数据结构设计**

对于存储关系矩阵，我们利用二维数组这一数据结构。通过宏定义#define MAX 100将数组的大小限制，原始的关系矩阵和之后需要操作的暂时数组都使用二维数组这一数据结构。arr2[i][j]表示关系矩阵中第i行第j列的元素。

同时，还需要设计row,col,i,j等循环变量来充当计算闭包这一过程中循环操作的计数器功能。

**4.项目实现**

**4.1输入部分的实现**

cout << "请输入矩阵的行数:"; //输入行数

while (1)

{

cin >> row;

if (cin.good() == 0 || row < 1)

{

cin.clear();

cin.ignore(1024, '\n');

cout << "输入行数错误！请重新输入！" << endl;

continue;

}

break;

}

cout << "请输入矩阵的列数:"; //输入列数

while (1)

{

cin >> col;

if (cin.good() == 0 || col < 1)

{

cin.clear();

cin.ignore(1024, '\n');

cout << "输入列数错误！请重新输入！" << endl;

continue;

}

break;

}

cout << "请输入关系矩阵:" << endl; //输入关系矩阵

for (i = 0; i < row; i++)

{

cout << endl << "请输入矩阵的第" << i << "行元素(元素以空格分隔):";

for (j = 0; j < col; j++)

arr[i][j] = Get\_element();

}

在输入部分首先用户需要输入该关系矩阵的列数和行数，在输入列数和行数的时候进行了错误判断：是否输入的时正确的数据类型，输入的整数类型是否符合规范（不能是小于1的整数）；其次就是要输入关系矩阵的具体内容，在输入关系矩阵具体内容的时候依据题目要求需要逐行输入关系矩阵的element，因此也要对输入关系矩阵的element进行错误处理，故我想到了再写一个行数叫Get\_element（如下）对输入关系矩阵的element进行统一的代码健壮性判断。

int Get\_element()

{

int retint;

while (1)

{

cin >> retint;

if (cin.good() == 0 || (retint != 0 && retint != 1))

{

cin.clear();

cin.ignore(1024, '\n');

cout << "输入错误！请重新输入！";

continue;

}

break;

}

return retint;

}

**4.2输出部分的实现**

输出部分由Output函数实现，通过遍历矩阵中的每一行每一列，输出关系矩阵内的具体内容。

void Output(int arr[][MAX])

{

cout << "所求关系矩阵为:" << endl; //给出输出提示

for (i = 0; i < row; i++)

{

for (j = 0; j < col; j++)

cout << arr[i][j]; //输出关系矩阵

cout << endl;

}

cout << endl;

}

**4.3自反闭包的实现**

自反闭包指的是在一个关系中，如果对于所有的元素a，都存在一个a与自己相关联的关系，则称该关系是自反的。

在本程序中，为了得到一个关系的自反闭包关系矩阵，需要将原始关系矩阵中的每个元素arr2[i][j]，如果i等于j，则将该元素设置为1，表示a与自己之间存在关系。如果i不等于j，则将该元素保持不变即可，这样自反闭包关系矩阵便构建出来了。

void Zifan(int arr2[][MAX])

{

for (i = 0; i < row; i++)

arr2[i][i] = 1; //自反闭包的构建：对角线上的元素都设为1

Output(arr2); //输出对应的的关系矩阵

Select(); //重新选择操作

}

**4.4对称闭包的实现**

对称闭包是指在一个关系中，如果对于任意的元素a与b，如果a与b相关联，则必然存在b与a相关联的关系。

在本程序中，为了得到一个关系的对称闭包关系矩阵，需要将原始关系矩阵中的每个元素arr2[i][j]，如果arr2[i][j]为1，则将arr2[j][i]也设置为1，表示b与a也存在关系。如果arr2[i][j]为0，则将arr2[j][i]保持不变。

void Duichen(int arr2[][MAX])

{

int arr1[MAX][MAX];

for (i = 0; i < row; i++)

for (j = 0; j < col; j++)

arr1[j][i] = arr2[i][j]; //获取关系矩阵的转置矩阵

for (i = 0; i < row; i++)

for (j = 0; j < col; j++)

{

arr2[i][j] = arr2[i][j] + arr1[i][j]; //对称闭包的构造：矩阵元素与其转置元素相加

if (arr2[i][j] > 1)

arr2[i][j] = 1; //超过1的元素全部设为1

}

Output(arr2); //输出相应的的关系矩阵

Select(); //重新选择操作

}

**4.5传递闭包的实现**

传递闭包是指在一个关系中，如果对于任意的元素a、b和c，如果a与b相关联，且b与c相关联，则必然存在a与c相关联的关系。

在本程序中，为了得到一个关系的传递闭包关系矩阵，我们除了原始数组还需要三个临时数组，初始化三个数组时结果数组置为原数组内容，然后通过计算：如果temparr2里面的某个位置为1，则要遍历原数组的对应行，如果存在1，则将temparr2中的相应位置置为1；前面操作完成之后，遍历整个关系矩阵，将result数组和temparr2数组进行相加，得出最后的传递闭包关系矩阵。

void Chuandi(int arr2[][MAX])

{

int temparr1[MAX][MAX], temparr2[MAX][MAX], k, h;

//设置两个临时数组temparr1和temparr2，循环变量k和h

int result\_arr[MAX][MAX]; //结果数组，存放结果

//初始化三个数组

for (i = 0; i < row; i++)

{

for (j = 0; j < col; j++)

{

temparr2[i][j] = 0; //temparr2数组赋值为0

result\_arr[i][j] = arr2[i][j]; //result\_arr和temparr1数组赋值为原数组内容

temparr1[i][j] = arr2[i][j];

}

}

//计算结果矩阵内容

for (h = 0; h < row; h++)

{

for (i = 0; i < row; i++)

for (j = 0; j < col; j++)

if (temparr1[i][j] == 1) //如果temparr1里面某个位置为1

{

for (k = 0; k < row; k++)

//则遍历原数组的对应行，如果存在1，则将temparr2相应位置置为1

if (arr2[j][k] == 1)

temparr2[i][k] = 1;

}

for (i = 0; i < row; i++)

for (j = 0; j < col; j++)

{

temparr1[i][j] = temparr2[i][j];

result\_arr[i][j] += temparr2[i][j]; //将result\_arr和temparr2矩阵相加

temparr2[i][j] = 0;

if (result\_arr[i][j] > 1) //超过1的元素全部设为1

result\_arr[i][j] = 1;

}

}

Output(result\_arr); //输出最终结果矩阵

Select();

}

**5.实验心得**

此次项目是离散数学的理论知识和其相关程序实现的结合部分。我们首先需要将离散数学的理论知识通过数学矩阵化变为可以编程实现的关系矩阵，再通过程序编写的方式将结果显示出来。再计算相关矩阵的过程中，需要理解每一种闭包的实质自反闭包和对称闭包的构建过程并不复杂，传递闭包的构造需要我们创建几个中间数组，过程相对比较复杂，但是经过逐步分析，可以大致理解程序构建的过程，同时加深了对于各种闭包理论知识的掌握。

附件(源代码)：

#include <iostream>

#define MAX 100 //使用宏定义代替数字

using namespace std;

void Output(int arr[][MAX]); //输出矩阵内容的函数

void Zifan(int arr2[][MAX]); //计算自反闭包的算法

void Duichen(int arr2[][MAX]); //计算对称闭包的算法

void Chuandi(int arr2[][MAX]); //计算传递闭包的算法

void Select(); //选择相应操作的函数

void Exitprogram(); //退出程序的函数

int arr[MAX][MAX], choice; //设置两个全局变量：关系矩阵和选择操作的变量

int row, col, i, j; //行数、列数、循环变量

int Get\_element() //输入矩阵函数（保证输入的矩阵元素只有0和1）

{

int retint;

while (1)

{

cin >> retint;

if (cin.good() == 0 || (retint != 0 && retint != 1))

{

cin.clear();

cin.ignore(1024, '\n');

cout << "输入错误！请重新输入！";

continue;

}

break;

}

return retint;

}

int main() //主函数：调用选择函数

{

Select();

return 0;

}

//选择函数

void Select()

{

cout << "请输入矩阵的行数:"; //输入行数

while (1)

{

cin >> row;

if (cin.good() == 0 || row < 1)

{

cin.clear();

cin.ignore(1024, '\n');

cout << "输入行数错误！请重新输入！" << endl;

continue;

}

break;

}

cout << "请输入矩阵的列数:"; //输入列数

while (1)

{

cin >> col;

if (cin.good() == 0 || col < 1)

{

cin.clear();

cin.ignore(1024, '\n');

cout << "输入列数错误！请重新输入！" << endl;

continue;

}

break;

}

cout << "请输入关系矩阵:" << endl; //输入关系矩阵

for (i = 0; i < row; i++)

{

cout << endl << "请输入矩阵的第" << i << "行元素(元素以空格分隔):";

for (j = 0; j < col; j++)

arr[i][j] = Get\_element();

}

cout << "输入对应序号选择算法" << endl << "1:自反闭包" << endl << "2:传递闭包" << endl << "3:对称闭包" << endl << "4:退出" << endl;

cin >> choice;

switch (choice)

{

case 1:

Zifan(arr); //调用自反闭包算法函数

break;

case 2:

Chuandi(arr); //调用传递闭包算法函数

break;

case 3:

Duichen(arr); //调用对称闭包算法函数

break;

case 4:

Exitprogram(); //退出程序

break;

default:

cout << "选择错误！请重新选择！" << endl; //在输入错误时给出提示

Select(); //重新选择操作

}

}

//输出矩阵内容的函数

void Output(int arr[][MAX])

{

cout << "所求关系矩阵为:" << endl; //给出输出提示

for (i = 0; i < row; i++)

{

for (j = 0; j < col; j++)

cout << arr[i][j]; //输出关系矩阵

cout << endl;

}

cout << endl;

}

//计算自反闭包的算法

void Zifan(int arr2[][MAX])

{

for (i = 0; i < row; i++)

arr2[i][i] = 1; //自反闭包的构建：对角线上的元素都设为1

Output(arr2); //输出对应的的关系矩阵

Select(); //重新选择操作

}

//计算对称闭包的算法

void Duichen(int arr2[][MAX])

{

int arr1[MAX][MAX];

for (i = 0; i < row; i++)

for (j = 0; j < col; j++)

arr1[j][i] = arr2[i][j]; //获取关系矩阵的转置矩阵

for (i = 0; i < row; i++)

for (j = 0; j < col; j++)

{

arr2[i][j] = arr2[i][j] + arr1[i][j]; //对称闭包的构造：矩阵元素与其转置元素相加

if (arr2[i][j] > 1)

arr2[i][j] = 1; //超过1的元素全部设为1

}

Output(arr2); //输出相应的的关系矩阵

Select(); //重新选择操作

}

//计算传递闭包的算法

void Chuandi(int arr2[][MAX])

{

int temparr1[MAX][MAX], temparr2[MAX][MAX], k, h;

//设置两个临时数组temparr1和temparr2，循环变量k和h

int result\_arr[MAX][MAX]; //结果数组，存放结果

for (i = 0; i < row; i++) //初始化三个数组

{

for (j = 0; j < col; j++)

{

temparr2[i][j] = 0; //temparr2数组赋值为0

result\_arr[i][j] = arr2[i][j]; //result\_arr和temparr1数组赋值为原数组内容

temparr1[i][j] = arr2[i][j];

}

}

for (h = 0; h < row; h++) //计算结果矩阵内容

{

for (i = 0; i < row; i++)

for (j = 0; j < col; j++)

if (temparr1[i][j] == 1) //如果temparr1里面某个位置为1

{

for (k = 0; k < row; k++)

//则遍历原数组的对应行，如果存在1，则将temparr2相应位置置为1

if (arr2[j][k] == 1)

temparr2[i][k] = 1;

}

for (i = 0; i < row; i++)

for (j = 0; j < col; j++)

{

temparr1[i][j] = temparr2[i][j];

result\_arr[i][j] += temparr2[i][j]; //将result\_arr和temparr2矩阵相加

temparr2[i][j] = 0;

if (result\_arr[i][j] > 1) //超过1的元素全部设为1

result\_arr[i][j] = 1;

}

}

Output(result\_arr); //输出最终结果矩阵

Select();

}

void Exitprogram() //退出程序的函数

{

cout << "欢迎下次再次使用!" << endl; //输出欢迎提示

exit(0); //退出程序

}