离散数学

项目说明文档

求矩阵的自反、对称、传递闭包

作者姓名:	高逸轩
学　　号:	2053385
指导教师:	唐剑锋
学院专业:	软件学院 软件工程

同济大学

Tongji University

1 功能简介

1.1 题目要求

根据给定的关系矩阵和用户指令，求其自反、对称、传递闭包。

1.2 项目需求分析

本项目在实现的过程中，考虑并且满足了以下的需求：

- ✓ 功能完善

  系统应当能够满足基本需求，即能够正确的给出对应闭包的关系矩阵。
- ✓ 健壮性

  当用户输入的数据不合理时，系统应当给予相应的提示而非直接报错。
- ✓ 代码可读性强

  本项目在实现过程中，将代码根据功能的不同划分为了不同的代码块，同时进行了合理封装。

1.3 项目要求

实现基类一般矩阵的建立和内容展示功能，同时保证健壮性；利用基类一般矩阵得到派生类关系矩阵，实现关系矩阵的建立、实现关系矩阵的合成运算、利用关系矩阵得到自反、对称、传递闭包（其中传递闭包包含了 O(n^4)朴素算法和 O(n^3)Warshall 算法两种求解方法），同时保证健壮性。

1.3.1 功能要求

基类一般矩阵：

首先接受矩阵 row 行、column 列的输入，借助 input_tools 功能集保证健壮性。然后接受 row*column 个整型数据输入，作为矩阵元素，其中在每一行都保证当前行的数据合理，实现健壮性。另外实现函数 PrintMatrix()来实现矩阵内容的展示。

派生类关系矩阵：

由于关系矩阵必为方阵，所以仅需输入一个变量 vertexNumber 顶点数量来建立关系矩阵。然后接受$vertexNumber^2$个整型数据(0/1)输入，作为矩阵元素，其中在每一行都保证当前行的数据合理，实现健壮性。另外，根据闭包与对应关系矩阵的关系，实现了自反、对称、传递闭包的求解。

1.3.2 输入格式

以下内容利用 input_tools 函数集实现健壮性

一般矩阵：
- 矩阵行 row、矩阵列 column （整型数据）
- 矩阵内容 row*column （整型数据）

关系矩阵
- 关系矩阵顶点数目 vertexNumber （整形数据）
- 关系矩阵内容 $vertexNumber^2$ （0/1）

1.3.3 项目简单示例

```
请输入关系矩阵顶点个数(0,1000): 3

请输入关系矩阵:

请输入矩阵第0行(以空格分隔):
0 0 1

请输入矩阵第1行(以空格分隔):
1 1 0

请输入矩阵第2行(以空格分隔):
1 0 0

设置结束, 当前矩阵为:
0 0 1
1 1 0
1 0 0

输入序号选择对应算法:
1.自反闭包
2.对称闭包
3.传递闭包
请输入: 2

所求关系闭包为:
0 1 1
1 1 0
1 0 0
```

2 项目实现

本项目核心共两部分：

- ✓ 菜单界面实现选择
- ✓ 求三种闭包

下面将对本项目核心的部分进行介绍，部分代码如下：

2.1 菜单界面实现选择

```
switch (Menu())
{
case 1:
    m1 = m.ReflexiveClosure(); // 1 自反闭包
    break;
case 2:
    m1 = m.SymmetricClosure(); // 2 对称闭包
    break;
case 3:
    m1 = m.TransitiveClosure();// 3 传递闭包
    break;
default:
    cout << "程序运行错误！";
    exit(-1);
}
```

```
// 进入菜单，选择所求闭包类型，1为自反闭包，2为对称闭包，3为传递闭包，已保证健壮性
const int Menu()
{
    cout << endl << "输入序号选择对应算法：" << endl;
    cout << "1.自反闭包" << endl;
    cout << "2.对称闭包" << endl;
    cout << "3.传递闭包" << endl;
    cout << "请输入：";
    int ret = getint(1, 3, "序号输入错误，请重新输入：");
    return ret;
}
```

通过 input_tools 函数集以及 Menu()函数，实现了提示用户选择所求闭包种类的功能，同时保证了健壮性。

2.2 求关系闭包

2.2.1 运算原理介绍

设 R 为 A 上的关系,

(1) 若 $\forall x(x\in A\rightarrow\langle x, x\rangle\in R)$，则称 R 在 A 上是自反的.

(2) 若 $\forall x\forall y(x, y\in A\wedge\langle x, y\rangle\in R\rightarrow\langle y, x\rangle\in R)$, 则称 R 为 A 上对称的关系.

(3) $\forall x\forall y\forall z(x, y, z\in A\wedge\langle x, y\rangle\in R\wedge\langle y, z\rangle\in R\rightarrow\langle x, z\rangle\in R)$, 则称 R 是 A 上的传递关系.

设 R 是非空集合 A 上的关系, R 的自反（对称或传递）闭包 是 A 上的关系 R', 使得 R' 满足以下条件：

(1) R' 是自反的（对称的或传递的）

(2) $R\subseteq R'$

(3) 对 A 上任何包含 R 的自反（对称或传递）关系 R'' 有 $R'\subseteq R''$.

以上为我们在离散数学理论课程中学习的关于关系的三种性质，以及闭包的定义方式。

2.2.2 闭包与关系矩阵的对应

在本题中，我们采用关系矩阵的方式来描述闭包关系。所以，我们要找到闭包和关系矩阵的对应关系，以及求各种闭包的矩阵表达方式。

一般将 R 的自反闭包记作 $r(R)$, 对称闭包记作 $s(R)$, 传递闭包记作 $t(R)$.

设关系 R, $r(R)$, $s(R)$, $t(R)$ 的关系矩阵分别为 M, M_r, M_s 和 M_t，则

$$M_r = M + E$$

$$M_s = M + M'$$

$$M_t = M + M^2 + M^3 + \cdots$$

其中 E 是和 M 同阶的单位矩阵, M' 是 M 的转置矩阵.

注意：在上述等式中矩阵的元素相加时使用逻辑加.

以上给出了三种闭包的矩阵求解方式。

2.2.3 代码实现介绍

```
// 求自反闭包
RelationMatrix RelationMatrix::ReflexiveClosure()
{
    RelationMatrix ret = *this;

    FOR(i, 0, vertexNumber)
        ret.nearArray[i][i] = 1;    // 求自反闭包，将关系矩阵对角线置为1

    return ret;
}
```

自反闭包求解较为简单,只需在原有关系矩阵的基础上将对角线置为1即可,时间复杂度为O(n).

```
// 求对称闭包
RelationMatrix RelationMatrix::SymmetricClosure()
{
    RelationMatrix ret = *this;

    FOR(i, 0, vertexNumber)
        FOR(j, 0, vertexNumber)
        ret.nearArray[i][j] = ret.nearArray[i][j] | ret.nearArray[j][i];// 求对称闭包，将关系矩阵和其转置矩阵做逻辑加运算

    return ret;
}
```

对称闭包需要将原有矩阵和其转置矩阵做逻辑加运算后得到，易得到时间复杂度为 O(n^2).

```
// 求传递闭包( 朴素算法复杂度O (n^4) , 后续实现了Warshall算法 O (n^3) 求传递闭包 )
RelationMatrix RelationMatrix::TransitiveClosure()
{
    RelationMatrix ret = *this;
    RelationMatrix currentDegree = *this;          // 记录当前关系矩阵R的x次方

    FOR(k, 0, vertexNumber)
    {
        FOR(i, 0, vertexNumber)
            FOR(j, 0, vertexNumber)
            ret.nearArray[i][j] = ret.nearArray[i][j] | currentDegree.nearArray[i][j];// ret = ret + R^k ,其中 + 为逻辑加

        currentDegree = currentDegree * (*this);   // currentDegree =  currentDegree * R ,记录为 R^k , 其中 * 操作为合成操作
    }
    return ret;
}
```

```
// 重载*运算符，实现关系矩阵合成运算
RelationMatrix RelationMatrix::operator*(const RelationMatrix& otherMatrix)
{
    RelationMatrix ret(vertexNumber);

    FOR(i, 0, vertexNumber)
        FOR(j, 0, vertexNumber)
        FOR(k, 0, vertexNumber)
        ret.nearArray[i][j] = ret.nearArray[i][j] | nearArray[i][k] & otherMatrix.nearArray[k][j];

    return ret;
}
```

求传递闭包时，我们需要根据公式 $M_t = M + M^2 + M^3 + \cdots$ 来计算得到，其中共需要进行 n 次逻辑加运算。另外，在计算M^n的过程中，我们需要进行 n 次合成运算操作。其中，合成运算操作通过*运算符的重载来实现，可理解为矩阵的逻辑乘运算。

对时间复杂度进行分析，一次合成运算所需的时间复杂度为 $O(n^3)$。共需进行 n 次运算，故求传递闭包的朴素算法时间复杂度为 $O(n^4)$。受限于合成运算的巨大复杂度，对于阶数较大的关系矩阵来说，这并不是一个优秀的结果。故在后续作业课程中，我通过 Warshall 算法实现了在时间上更为优秀的求传递闭包方法。

3 项目测试

3.1 健壮性测试

```
请输入关系矩阵顶点个数(0,1000): 2413wr gdf
顶点个数输入错误, 请重新输入:
```

```
输入序号选择对应算法:
1.自反闭包
2.对称闭包
3.传递闭包
请输入: r3qfwepdsj0vzio
序号输入错误, 请重新输入: pofjdsvihf
序号输入错误, 请重新输入: 234254
序号输入错误, 请重新输入: 2

所求关系闭包为:
0 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0
```

在 Matrix 函数集的实验报告中，我们以及对矩阵信息设置的健壮性进行了示例，故在此不再展示，请老师查阅。

3.2 求自反闭包

```
请输入关系矩阵顶点个数(0,1000)： 4

请输入关系矩阵：

请输入矩阵第0行(以空格分隔)：
1 0 0 1

请输入矩阵第1行(以空格分隔)：
1 1 1 1

请输入矩阵第2行(以空格分隔)：
0 0 1 1

请输入矩阵第3行(以空格分隔)：
1 1 0 1

设置结束，当前矩阵为：
1 0 0 1
1 1 1 1
0 0 1 1
1 1 0 1

输入序号选择对应算法：
1.自反闭包
2.对称闭包
3.传递闭包
请输入： 1

所求关系闭包为：
1 0 0 1
1 1 1 1
0 0 1 1
1 1 0 1
```

3.3 求对称闭包

```
请输入关系矩阵顶点个数(0,1000): 4

请输入关系矩阵:

请输入矩阵第0行(以空格分隔):
0 0 1 1

请输入矩阵第1行(以空格分隔):
1 1 0 0

请输入矩阵第2行(以空格分隔):
1 0 0 0

请输入矩阵第3行(以空格分隔):
0 0 0 0

设置结束, 当前矩阵为:
0 0 1 1
1 1 0 0
1 0 0 0
0 0 0 0

输入序号选择对应算法:
1.自反闭包
2.对称闭包
3.传递闭包
请输入: 2

所求关系闭包为:
0 1 1 1
1 1 0 0
1 0 0 0
1 0 0 0
```

3.4 求传递闭包

```
请输入关系矩阵顶点个数(0,1000): 4

请输入关系矩阵:

请输入矩阵第0行(以空格分隔):
0 1 0 1

请输入矩阵第1行(以空格分隔):
1 0 1 0

请输入矩阵第2行(以空格分隔):
0 0 0 1

请输入矩阵第3行(以空格分隔):
1 1 1 0

设置结束, 当前矩阵为:
0 1 0 1
1 0 1 0
0 0 0 1
1 1 1 0

输入序号选择对应算法:
1.自反闭包
2.对称闭包
3.传递闭包
请输入: 3

所求关系闭包为:
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
```

4 心得与总结

在本次实验中，我采取关系矩阵的方式（借助了前面自主实现的函数集文件 Matrix），来帮助求解各类闭包。在此之前，我们需要先对课堂知识中自反、对称、传递三种关系性质熟练掌握，同时找到其和关系矩阵的对应关系。另外，我们还应该在求解过程中，对程序的运行效率进行基本的分析。由此可以简单得到，在根据定义求解传递闭包的过程中，我们的朴素算法效率并不优秀，所以引发了后续项目 Warshall 算法的出现。