离散数学

项目说明文档

Warshall 算法求传递闭包

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，利用 Warshall 算法高效求解传递闭包。

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：

- ✓ 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。
- ✓ 执行效率高
  与之前实现的根据定义朴素算法求解传递闭包相比，本题使用的 `Warshall` 算法具有较低的时间复杂度。

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

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，使用 Warshall 算法实现了传递闭包的求解。

1.3.2 输入格式

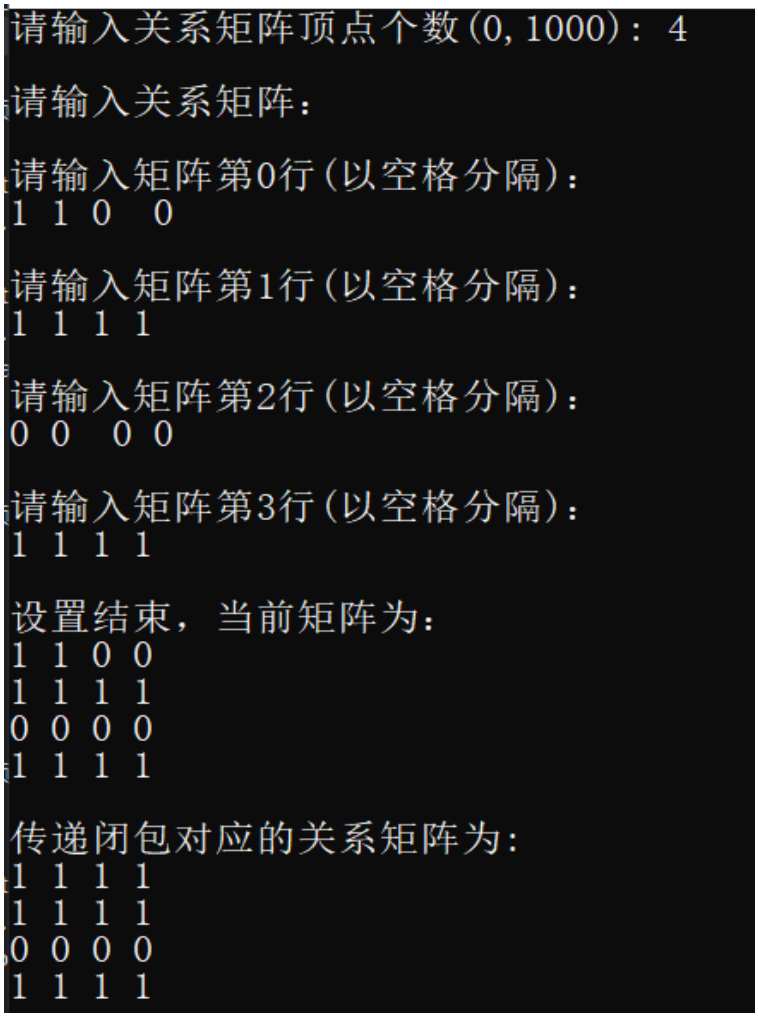
以下内容利用 input_tools 函数集实现健壮性

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



2 项目实现

本项目核心部分：

- ✓ 健壮性实现
- ✓ 利用 Warshall 算法求解传递闭包

下面将对本项目核心的部分进行介绍，部分代码如下：

2.1 健壮性

通过 input_tools 函数集以及 Matrix 函数集，实现了提示用户建立关系矩阵的功能，同时保证了健壮性。这部分内容已经在之前的报告里体现，故在此不再展示，请老师查阅。

2.2 Warshall 求传递闭包

2.2.1 运算原理

以下为我通过课件与网络学习后得到的 Warshall 算法求解传递闭包的过程：

（1）置新矩阵 A=M;

（2）置 i=1;

（3）对所有 j 如果 A[j,i]=1，则对 k=1…n 执行：

A[j,k]=A[j,k]∨A[i,k];

（4）i 加 1;

（5）如果 i≤n，则转到步骤（3），否则停止。

2.2.2 代码实现

在上述代码中，按照运算原理中的步骤实现了传递闭包的求解。

```
// Warshall算法O（n^3）求传递闭包
RelationMatrix RelationMatrix::Warshall()
{
    RelationMatrix ret = *this;

    FOR(j, 0, vertexNumber)             // 第j列
        FOR(i, 0, vertexNumber)         // 第i行
        if (ret.nearArray[i][j])        // 若第j列第i行元素为真
        {
            FOR(k, 0, vertexNumber)     // 使得任意k在[0,vertexNumber)上，做下面的运算
                ret.nearArray[i][k] = ret.nearArray[i][k] | ret.nearArray[j][k];
        }

    return ret;
}
```

2.2.3 时间复杂度对比

```
// 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;
}
```

根据定义用朴素算法求传递闭包时，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（可通过矩阵快速幂的方法来优化，在此题中不是重点，故仅仅提到，未实现），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。

```
// Warshall算法O（n^3）求传递闭包
RelationMatrix RelationMatrix::Warshall()
{
    RelationMatrix ret = *this;

    FOR(j, 0, vertexNumber)             // 第j列
        FOR(i, 0, vertexNumber)         // 第i行
        if (ret.nearArray[i][j])        // 若第j列第i行元素为真
        {
            FOR(k, 0, vertexNumber)     // 使得任意k在[0,vertexNumber)上，做下面的运算
                ret.nearArray[i][k] = ret.nearArray[i][k] | ret.nearArray[j][k];
        }

    return ret;
}
```

可以得到，利用 Warshall 算法来进行传递闭包的求解所需要的时间复杂度为 $O(n^3)$，当关系矩阵规模较大时，Warshall 比朴素算法的时间优秀程度非常可观。

3 项目测试

```
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 0 0

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：
1 0 0 1
1 1 1 1
0 0 0 0
1 1 1 0

传递闭包对应的关系矩阵为：
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1
```

健壮性测试示例已经在之前的项目中展示过，故在此不再展示，请老师查阅。

4 心得与总结

在 Warshall 算法中，我们规避了求矩阵合成运算这个高时间复杂度的步骤，从而降低了求传递闭包所需的时间。在算法设计过程中，时间复杂度是需要高度考虑和注意的一个因素。在高中的信息学竞赛学习过程中，曾经设计过闭包的概念和求法，在当时采用了矩阵快速幂的做法，来实现矩阵运算的高效。在大学离散数学的学习过程中，Warshall 算法则从另一个全新的角度给出了根据关系矩阵求解传递闭包的方法，给人以新鲜感和空间思考回味。