《离散数学》课程实验报告 3 求关系的自反、对称和传递闭包

2253893 苗君文 软件工程

1. 题目简介

1.1. 背景与目的

在许多计算机科学和离散数学领域，关系矩阵的闭包计算是一个重要的问题。本项目旨在设计一个能够计算关系矩阵的自反闭包、对称闭包和传递闭包的程序。这些闭包在关系理论和图论中具有广泛的应用，包括在数据库系统中的关系优化、网络通信中的路径计算等方面。

1.2. 问题描述

给定一个关系矩阵，其中元素为 0 或 1，表示两个元素之间是否存在关系。程序需要实现以下功能：求解关系矩阵的自反闭包、对称闭包、传递闭包。

1.3. 程序输入输出

1.3.1. 输入

用户通过程序输入关系矩阵的阶数和具体元素。程序需要检查输入的有效性，确保元素为 0 或 1。

1.3.2. 输出

程序将输出计算得到的自反闭包、对称闭包或传递闭包的关系矩阵。

1.4. 功能选择

用户可以选择计算自反闭包、对称闭包、传递闭包，或退出程序。

2. 解题思路

2.1. 明确闭包定义

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 $R' \subseteq R''$.

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).

2.2. 闭包的数学构造方法

2.2.1. 集合表示

设 R 为 A 上的关系，则有

(1) r(R)=R∪R0

(2) s(R)=R∪R-1

(3) t(R)=R∪R2∪R3∪…

说明：对于有穷集合 A (|A|=n) 上的关系，(3)中的并最多不超过 Rn；若 R 是自反的，则 r(R)=R；若 R 是对称的，则 s(R)=R;若 R 是传递的，则 t(R)=R.

2.2.2. 矩阵表示

设关系 R， r(R)， s(R)， t(R)的关系矩阵分别为 M，Mr，Ms 和 Mt ，则

Mr =M+E

Ms =M+M’

Mt =M+M^2+M^3+…

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。

本程序根据矩阵表示的思路来实现。

2.3. 自反闭包计算

2.3.1. 算法简述

自反闭包是在关系矩阵的基础上，确保矩阵中每个元素对角线上的元素都为 1。即，对于矩阵中的每个元素 (i，i)，将其值设为 1。

2.3.2. 实现步骤

a. 遍历关系矩阵的对角线元素。

b. 将对角线元素的值设为 1。

2.4. 对称闭包计算

2.4.1. 算法简述

对称闭包是在关系矩阵的基础上，确保矩阵中如果元素（i, j）为1，则元素（j, i）也为1。

2.4.2. 实现步骤

a. 创建一个临时矩阵，将矩阵的行列互换，得到转置矩阵。

b. 遍历原始矩阵和转置矩阵，将对应位置的元素进行逻辑或运算。

2.5. 传递闭包计算

2.5.1. 算法简述

传递闭包是在关系矩阵的基础上，通过矩阵乘法的方式来实现。若关系矩阵为n阶，则只需要计算 A^1 + A^2 + ... + A^n 的和即可得到传递闭包。

2.5.2. 实现步骤

a. 创建一个临时矩阵，将其初始化为原始矩阵。

b. 迭代计算幂运算，使用矩阵乘法和逻辑加法。

2.6. 界面设计

本程序为方便使用者调试与使用，增设了让用户判断是否需要继续运行程序的环节。使用循环与判断语句，并做了详尽的输入错误处理。

3. 所用数据结构

3.1. 二维动态数组

本程序使用二维动态数组来存放各种矩阵。二维动态数组具有以下优点：

动态大小： 二维动态数组的大小可以在运行时动态确定，而不需要在编译时指定。这使得程序可以根据需要动态分配和释放内存，适应不同大小的输入数据。

灵活性： 动态数组允许在程序执行期间动态调整矩阵的大小。这在处理不同大小的输入或动态变化的问题时非常有用。

内存管理： 动态分配内存的使用允许程序员手动管理内存，而不是依赖于编译器的静态内存分配。这有助于避免内存浪费，并使得程序更具优化性。

避免浪费： 对于大型矩阵，静态数组可能会导致内存浪费，因为必须为可能不被完全使用的最大大小分配内存。动态数组可以根据实际需要调整大小，从

而避免不必要的内存浪费。

便于传递： 动态数组通过指针传递，使得在函数之间传递矩阵变得更加方便。这避免了复制整个数组的开销，提高了程序的效率。

解决大小未知的问题： 当矩阵的大小在编译时未知或在运行时可变时，使用动态数组是解决这类问题的有效方法。这对于需要适应动态输入的算法和数据结构尤其重要。

3.2. 二维动态数组相关的操作/函数

3.2.1. 动态内存分配

使用 new 运算符在堆上动态分配内存，以创建二维数组。程序通过两个循环，先为每一行分配内存，然后为每一列分配内存。例如：

```
int** result = new int* [n];
for (int i = 0;i < n;i++) {
    result[i] = new int[n];
}  //给结果矩阵分配内存
```

上图的这段代码创建了一个 n x n 的二维数组，并将其地址存储在 result 指针数组中。

3.2.2. 内存释放函数

使用 delete 运算符释放动态分配的内存，以避免内存泄漏。在本程序中，有一个名为 deallocateMatrix 的函数，用于释放矩阵占用的内存。

```
void deallocateMatrix(int** matrix, int n) {
    for (int i = 0;i < n;i++) {
        delete[]matrix[i];
    }
    delete[]matrix;
}  //释放动态分配的二维数组
```

此函数遍历矩阵的每一行，先释放每行的内存，然后释放指向行的指针数组的内存。

4. 核心算法

4.1. 矩阵相关的基本算法/操作

4.1.1. 矩阵逻辑加法函数

此函数 matrixAdd 接受两个二维整数数组 A 和 B，以及它们的阶数 n。函数的主要目的是将这两个矩阵逻辑相加，结果存储在新的动态分配的矩阵中，然后返回这个结果矩阵。

```
int** matrixAdd(int** A, int** B, int n) {
    int** result = new int* [n];
    for (int i = 0;i < n;i++) {
        result[i] = new int[n];
    }  //给结果矩阵分配内存
    for (int i = 0;i < n;i++) {
        for (int j = 0;j < n;j++) {
            result[i][j] = A[i][j] + B[i][j];
            if (result[i][j] > 1)
                result[i][j] = 1;
        }
    }

    return result;
}  //矩阵逻辑加法函数
```

先分配结果矩阵的内存：创建一个新的二维数组 result，其大小为 n x n。通过使用 new 运算符为每一行分配内存。

再做矩阵的逻辑加法：使用双重循环遍历矩阵 A 和 B 的每个元素。将相应位置的元素相加，并将结果存储在 result 矩阵的相应位置。如果结果大于 1，将其截断为 1， 这是逻辑上的“或”运算，确保结果矩阵中的每个元素不超过 1。

最后返回指向逻辑相加后矩阵的指针。

4.1.2 矩阵逻辑乘法函数

此函数 matrixMultiply 接受两个二维整数数组 A 和 B，以及它们的阶数 n。函数的主要目的是将这两个矩阵进行逻辑相乘，结果存储在新的动态分配的矩阵中，然后返回这个结果矩阵。

```
int** matrixMultiply(int** A, int** B, int n) {
    int** result = new int* [n];
    for (int i = 0;i < n;i++) {
        result[i] = new int[n];
    }  //给结果矩阵分配内存
    for (int i = 0;i < n;i++) {
        for (int j = 0;j < n;j++) {
            result[i][j] = 0;//初始化结果矩阵
            for (int k = 0;k < n;k++) {
                result[i][j] += A[i][k] * B[k][j];  //计算结果矩阵
                if (result[i][j] > 1)
                    result[i][j] = 1;
            }
        }
    }

    return result;
}  //矩阵逻辑乘法函数
```

先分配结果矩阵的内存：创建一个新的二维数组 result，其大小为 n x n。通过使用 new 运算符为每一行分配内存。

再做矩阵的逻辑乘法：使用双重循环遍历矩阵 A 和 B 的每个元素。将 result 矩阵的每个元素初始化为 0。使用内层循环进行矩阵乘法操作，将对应位置的元素相乘并累加到结果矩阵的相应位置。如果结果大于 1，则将其截断为 1，也就是逻辑上的“或”运算。

最后返回指向逻辑相乘后矩阵的指针。

4.1.3. 矩阵的输出函数

类似地，用双重循环先行后列遍历每一个元素后输出。

4.2. 计算闭包的相关核心函数

4.2.1. 计算自反闭包

计算自反闭包函数 reflexiveClosure，使用循环遍历关系矩阵的对角线上的元素。将对角线上的每个元素 matrix[i][i] 设为 1，表示每个元素都与自己存在关系。

```
void reflexiveClosure(int** &matrix, int n) {
    for (int i = 0;i < n;i++) {
        matrix[i][i] = 1;
    }
}  //计算自反闭包
```

函数的参数 matrix 是一个对二维数组的引用，也就是对 matrix 的修改

将影响到传递给函数的实际矩阵。

4.2.2. 计算对称闭包

计算对称闭包函数 symmetricClosure，首先创建一个临时矩阵 tmp，其大小与输入矩阵相同。使用两重循环遍历输入矩阵的每个元素。如果原矩阵和转置矩阵对应位置的元素之和大于 0，将对应位置的元素设置为 1。函数通过使用 new 运算符分配了一个临时矩阵，并在函数结束时释放了这个矩阵的内存，以避免内存泄漏。

```
void symmetricClosure(int** &matrix, int n) {
    int** tmp = new int* [n];
    for (int i = 0;i < n;i++) {
        tmp[i] = new int[n];
    }
    for (int i = 0;i < n;i++) {
        for (int j = 0;j < n;j++) {
            tmp[i][j] = matrix[j][i];
        }
    }
    for (int i = 0;i < n;i++) {
        for (int j = 0;j < n;j++) {
            if (matrix[i][j] + tmp[i][j] > 0)
                matrix[i][j] = 1;
        }
    }
}  //计算对称闭包
```

4.2.3. 计算传递闭包

计算传递闭包函数 transitiveClosure，首先创建了指向初始矩阵 matrix 的指针 init，临时矩阵 temp，其大小与输入矩阵相同并将初始矩阵的内容复制到临时矩阵中。计算的部分，其循环从 2 开始，计算矩阵的幂（A^2，A^3，...，A^n）；使用外层循环中的 matrixMultiply 函数进行矩阵乘法，得到 A^i；使用 matrixAdd 函数将计算得到的矩阵与原矩阵相加，累加到 matrix 中。最后，使用 deallocateMatrix 函数释放临时矩阵 temp 的内存。

```
void transitiveClosure(int** &matrix, int size) {
    int** init = matrix;  //存放初始的矩阵A
    int** temp = new int* [size];
    for (int i = 0; i < size; i++) {
        temp[i] = new int[size];
    }

    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            temp[i][j] = matrix[i][j];
        }
    }

    for (int i = 2;i <= size;i++) {
        temp = matrixMultiply(temp, init, size);
        matrix = matrixAdd(matrix, temp, size);//相加
    }//A^1+A^2+...+A^n

    deallocateMatrix(temp, size);// 释放内存
}  //计算传递闭包
```

此函数通过计算矩阵的幂累加，实现了传递闭包的计算。在计算过程中，利用了矩阵乘法和矩阵加法的辅助函数。函数执行完毕后，输入矩阵 matrix 中的元素反映了传递闭包的关系。

4.3. 输入错误处理相关函数

4.3.1.单个参数的输入错误处理

函数 dealInputError 使用 while 循环不断尝试获取用户输入，直到输入满足要求。使用 cin.fail() 及 min、max 来判断输入是否出错，检查输入是否在有效范围内。如果输入无效，则输出错误信息，清除输入缓冲区并忽略之后的字符。通过一个字符来获取输入一个数之后的字符以检查输入字符的个数，若不正确，也输出错误信息，执行相应清除操作。如果输入有效，则跳出循环。

此函数可以用于判断输入的矩阵的阶数的正误，选择算法序号的正误。

```
void dealInputError(int& n, int min = 0, int max = 1)  //处理输入错误
{
    while (1) {
        char c;  //用来获取输入一个数之后的一个字符，若获取一个字符且非换行符则视为输入错误。
        cin >> n;
        if (cin.fail() || n<min || n>max) {
            cout << "输入错误，请重新输入。" << endl;
            cin.clear();
            cin.ignore(9999, '\n');
            continue;
        } //内容错误
        else if (cin.get(c) && c != '\n') {
            cout << "输入错误，请重新输入。" << endl;
            cin.clear();
            cin.ignore(9999, '\n');
            continue;
        } //个数错误
        else
            break;
    }
    return;
}
```

4.3.2. 关系矩阵元素输入的错误处理

使用循环遍历矩阵的每一行，让用户逐行输入矩阵的元素。内部的循环与之前相似，从输入的内容和输入的元素个数两方面入手，若输入错误，则需要重新输入，直到输入正确。

4.3.3. 判断是否继续运行程序的错误处理

此处输入的内容是字符，程序将大小写的 y/n 输入都作为正确输入，而其他情况需要重新输入。类似之前，分为内容错误和个数错误两种，并根据输入正确的内容需不需要继续运行。Y 则 continue，N 则 break。

5. 心得体会

5.1.关于矩阵的存储方式

题目所给出的存储方式使用静态数组，但是在运行时产生了警告(warning C6262)：函数使用堆叠的 "40020" 字节。请考虑将一些数据移动到堆。

这可能是因为局部数组或其他数据结构太大，超过了默认堆栈大小。在 C++ 中，局部变量的内存分配通常发生在栈上，而堆上分配则是使用 new 和 delete 进行的。如果局部数组或数据结构太大，它可能会导致栈溢出或者增加函数的堆栈大小。

为了解决这个问题，考虑将大的数据结构改为动态分配的内存（使用 new

和 delete)，这样就会在堆上分配内存而不是栈上。因此，我选择使用动态内存分配。

5.2. 关于传递闭包的矩阵乘法算法

通过矩阵乘法的方式来计算传递闭包的时间复杂度高达 O(n^4),，其中 n 是矩阵的阶数，因此对于大型矩阵，计算传递闭包会变得非常耗时。不仅时间复杂度高，这种方法的空间复杂度也高，矩阵乘法需要额外的存储空间来保存中间结果。对于传递闭包的计算，需要存储多个中间结果，导致空间复杂度较高。

相较于上面的算法，能清晰感受到 Warshall 算法的优势，可以通过迭代的方式直接在原始矩阵上计算传递闭包，避免了不必要的中间结果和矩阵乘法的高复杂度，这种方法通常更为高效，尤其对于稀疏矩阵和大规模的关系。

6. 测试结果

6.1. 输入关系矩阵的阶数

（1）输入个数及内容错误

```
**********************************************
**                                          **
**          欢迎进入关系的自反、对称        **
**              和传递闭包求解程序          **
**                                          **
**********************************************


请输入矩阵的阶数：5 a
输入错误，请重新输入。
```

（2）输入个数错误

```
3 2
输入错误，请重新输入。
```

（3）输入内容错误

```
a
输入错误，请重新输入。
```

（4）输入的值错误

```
请输入矩阵的阶数：2000
输入错误，请重新输入。
```

（5）输入正确则可以输入关系矩阵的元素

```
4

请输入关系矩阵：
```

6.2. 输入关系矩阵中的元素

（1）输入元素的大小错误

```
请输入关系矩阵：

请输入矩阵的第1行元素(元素为0或1，输入时以空格分隔)：1 2 0 1
输入错误，请重新输入。

请输入矩阵的第1行元素(元素为0或1，输入时以空格分隔)：
```

（2）输入元素的类型错误

```
请输入矩阵的第1行元素(元素为0或1，输入时以空格分隔)：1 a
输入错误，请重新输入。

请输入矩阵的第1行元素(元素为0或1，输入时以空格分隔)：
```

（3）输入元素的个数错误

```
请输入矩阵的第1行元素(元素为0或1，输入时以空格分隔)：1 0 1 1 0
输入错误，请重新输入。

请输入矩阵的第1行元素(元素为0或1，输入时以空格分隔)：
```

（4）输入正确则可以输入选择的算法序号

```
请输入矩阵的第1行元素(元素为0或1，输入时以空格分隔)：1 0 1 1

请输入矩阵的第2行元素(元素为0或1，输入时以空格分隔)：0 0 0 1

请输入矩阵的第3行元素(元素为0或1，输入时以空格分隔)：0 1 0 0

请输入矩阵的第4行元素(元素为0或1，输入时以空格分隔)：0 1 1 0

输入对应序号选择算法
1:自反闭包
2:对称闭包
3:传递闭包
4:退出
```

6.3. 输入对应序号选择算法

（1）输入类型错误

```
输入对应序号选择算法
1:自反闭包
2:对称闭包
3:传递闭包
4:退出
a
输入错误，请重新输入。
```

（2）输入的值错误

```
99
输入错误，请重新输入。
```

（3）输入的个数错误

```
2 5 3
输入错误，请重新输入。
```

（4）输入正确则可以进入对应的算法中

```
输入对应序号选择算法
1:自反闭包
2:对称闭包
3:传递闭包
4:退出
1
自反闭包的关系矩阵为:
1  0  1  1
0  1  0  1
0  1  1  0
0  1  1  1
```

6.4. 求关系矩阵的自反闭包矩阵

```
请输入矩阵的阶数：4

请输入关系矩阵：

请输入矩阵的第1行元素(元素为0或1，输入时以空格分隔)：1 0 1 1

请输入矩阵的第2行元素(元素为0或1，输入时以空格分隔)：0 0 0 1

请输入矩阵的第3行元素(元素为0或1，输入时以空格分隔)：0 1 0 0

请输入矩阵的第4行元素(元素为0或1，输入时以空格分隔)：0 1 1 0

输入对应序号选择算法
1:自反闭包
2:对称闭包
3:传递闭包
4:退出
1
自反闭包的关系矩阵为:
1  0  1  1
0  1  0  1
0  1  1  0
0  1  1  1
```

6.5. 求关系矩阵的对称闭包矩阵

```
请输入矩阵的阶数：4

请输入关系矩阵：

请输入矩阵的第1行元素(元素为0或1，输入时以空格分隔)：1 0 1 1

请输入矩阵的第2行元素(元素为0或1，输入时以空格分隔)：0 0 0 1

请输入矩阵的第3行元素(元素为0或1，输入时以空格分隔)：0 1 0 0

请输入矩阵的第4行元素(元素为0或1，输入时以空格分隔)：0 1 1 0

输入对应序号选择算法
1:自反闭包
2:对称闭包
3:传递闭包
4:退出
2
对称闭包的关系矩阵为：
1   0   1   1
0   0   1   1
1   1   0   1
1   1   1   0
```

6.6. 求关系矩阵的传递闭包矩阵

```
请输入矩阵的阶数：4

请输入关系矩阵：

请输入矩阵的第1行元素(元素为0或1，输入时以空格分隔)：1 0 1 1

请输入矩阵的第2行元素(元素为0或1，输入时以空格分隔)：0 0 0 1

请输入矩阵的第3行元素(元素为0或1，输入时以空格分隔)：0 1 0 0

请输入矩阵的第4行元素(元素为0或1，输入时以空格分隔)：0 1 1 0

输入对应序号选择算法
1:自反闭包
2:对称闭包
3:传递闭包
4:退出
3
传递闭包的关系矩阵为：
1   1   1   1
0   1   1   1
0   1   1   1
0   1   1   1
```

6.7. 选择是否继续运行该程序

（1）输入错误

```
是否继续运行该程序？（y/n）：3
输入无效，请重新输入。

是否继续运行该程序？（y/n）：36
输入无效，请重新输入。

是否继续运行该程序？（y/n）：j6
输入无效，请重新输入。

是否继续运行该程序？（y/n）：a
输入无效，请重新输入。

是否继续运行该程序？（y/n）：
```

（2）输入 y/Y/n/N 均输入正确（下图以 y 为例）

```
是否继续运行该程序？（y/n）：y

请输入矩阵的阶数：
```

6.8. 输入正确时的总览（以传递闭包为例）

```
**********************************************
**                                          **
**          欢迎进入关系的自反、对称        **
**              和传递闭包求解程序          **
**                                          **
**********************************************


请输入矩阵的阶数：3

请输入关系矩阵：

请输入矩阵的第1行元素(元素为0或1，输入时以空格分隔)：1 0 1

请输入矩阵的第2行元素(元素为0或1，输入时以空格分隔)：0 0 1

请输入矩阵的第3行元素(元素为0或1，输入时以空格分隔)：0 1 0

输入对应序号选择算法
1:自反闭包
2:对称闭包
3:传递闭包
4:退出
3
传递闭包的关系矩阵为:
1  1  1
0  1  1
0  1  1

是否继续运行该程序？（y/n）：y

请输入矩阵的阶数：1

请输入关系矩阵：

请输入矩阵的第1行元素(元素为0或1，输入时以空格分隔)：1

输入对应序号选择算法
1:自反闭包
2:对称闭包
3:传递闭包
4:退出
3
传递闭包的关系矩阵为:
1

是否继续运行该程序？（y/n）：n
```