同济大学

TONGJI UNIVERSITY

离散数学课程设计

项目名称	求关系的自反、传递和对称闭包
学　　院	计算机科学与技术学院
专　　业	软件工程
学生姓名	杨瑞晨
学　　号	2351050
指导教师	唐剑锋
日　　期	2024 年 12 月 1 日

目 录

1 项目分析

1.1 项目背景

在离散数学中，关系是一种重要的概念，它描述了集合中元素之间的某种联系。对于给定的集合 A 上的非空关系 R，我们经常希望这种关系具备某些有益的特性，如自反性、对称性或传递性等。为了赋予关系 R 这些特性，需要在其基础上添加若干有序对，形成新的关系 R'。这样的 R' 被称为关系 R 的自反闭包、对称闭包或传递闭包。本项目旨在通过编程实现这些闭包的计算。具体而言：

- **自反闭包**：若 $a \in A$，则 $< a, a > \in R$。
- **对称闭包**：若 $< a, b > \in R$，则 $< b, a > \in R$。
- **传递闭包**：若 $< a, b > \in R$ 且 $< b, c > \in R$，则 $< a, c > \in R$。

自反闭包确保矩阵的对角线元素为 1。

对称闭包确保矩阵关于主对角线对称。

传递闭包确保若 aRb 且 bRc，则 aRc。

通过编写程序自动求解这些闭包，能够直观展示离散数学中的逻辑关系。

1.2 项目要求

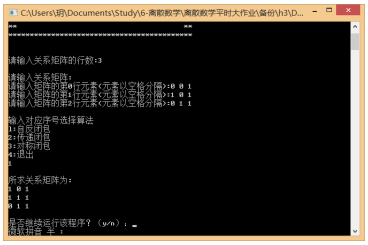
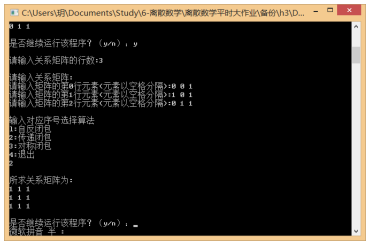
本项目要求实现一个程序，该程序能够：

（1）手动输入矩阵阶数

（2）手动输入关系矩阵

（3）求解关系的自反、对称和传递闭包

1.3 项目示例


1.4 项目环境

使用 C++ 语言实现，开发环境为 Linux 下的 gcc 编译器。

2 项目设计

2.1 数据结构应用

根据项目的分析结果，明确了需要完成关系闭包的计算任务。因为计算过程中需要频繁地直接访问和赋值元素，所以选用二维数组来存储关系矩阵，以表达这种二元关系。

为了支持矩阵运算并便于扩展，使用了通用模板类 Matrix<T>。该类以二维指针数组形式存储数据，提供矩阵基本操作（如加法、乘法、转置等）。

设计重点包括：

- 动态内存分配：实现矩阵的灵活大小定义。
- 深拷贝：通过拷贝构造函数和赋值运算符确保数据安全性。
- 重载运算符：实现矩阵加法、乘法等运算符功能。

数据结构优点：

- 抽象化：支持多种数据类型（如 int、double、bool 等）。本例中主要为 bool
- 易维护：通过封装减少代码重复，提高可读性

2.2 算法设计

2.2.1 算法思路

根据闭包的性质：$\begin{cases} r(R) = R \cup R^0 \\ s(R) = R \cup R^{-1} \\ t(R) = R \cup R^2 \cup R^3 \cup ... \cup R^n \end{cases}$

我们可以将闭包的计算转化为矩阵运算：

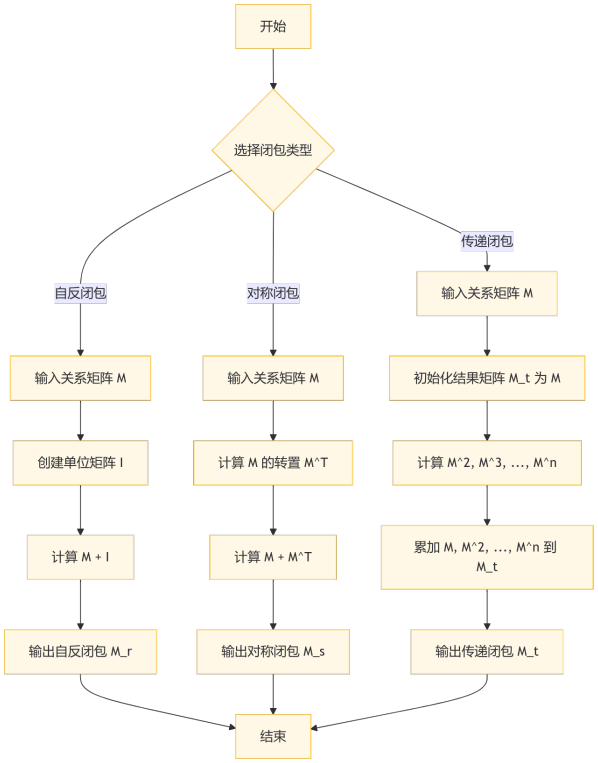
- 自反闭包：$M_r = M + I$，其中 I 是单位矩阵
- 对称闭包：$M_s = M + M^T$，其中 M^T 是 M 的转置矩阵
- 传递闭包：$M_t = M + M^2 + M^3 + ... + M^n$，其中 n 是矩阵的阶数

2.2.2 性能评估

设 R 是集合 A 上的二元关系，A 中有 n 个元素：

- 自反闭包：只需让对角线元素都为 1，时间复杂度为 $O(n)$
- 对称闭包：转置需要 $O(n^2)$，相加需要 $O(n^2)$，总时间复杂度为 $O(n^2)$
- 传递闭包：需要完成原矩阵 1 次到 n 次方的累加，每次需要 $O(n^3)$，总时间复杂度为 $O(n^4)$

2.2.3 流程图表示



2.2.4 代码实现

闭包计算

```cpp
// 自反闭包
Matrix<bool> reflexive(Matrix<bool> &m) // M_r = M + I
{
    Matrix<bool> temp = m ^ 0; // 单位矩阵 I
    Matrix<bool> result = m + temp;
    return result;
}

// 对称闭包
Matrix<bool> symmetric(Matrix<bool> &m) // M_s = M + M^T
{
    Matrix<bool> temp = m.transpose(); // 转置矩阵 M^T
    Matrix<bool> result = m + temp;
    return result;
}

// 传递闭包
Matrix<bool> transitive(Matrix<bool> &m) // M_t = M + M^2 + M^3 + ... + M^n
{
    Matrix<bool> result = m; // 结果矩阵 M + M^2 + M^3 + ... + M^n
    Matrix<bool> temp = m;   // 用于存储 M^2, M^3, ..., M^n
    for (int i = 1; i <= m.getRows(); ++i)
```

```
    {
        temp = temp * m; // M^i
        //cout << "M^" << i << " = " << endl << temp << endl;
        result = result + temp; // M + M^2 + ... + M^i
    }
    return result;
}
```

Matrix<T> 类

```
// 定义模板类 Matrix，可适用于任意类型的矩阵操作
template <class T>
class Matrix
{
private:
    T **matrix; // 矩阵数据，使用二维指针数组
    int rows;   // 行数
    int cols;   // 列数
public:
    // 构造函数
    Matrix(int rows, int cols)
    {
        this->rows = rows;
        this->cols = cols;
        matrix = new T *[rows]; // 分配行指针数组
        for (int i = 0; i < rows; ++i)
        {
            matrix[i] = new T[cols]; // 分配每一行的列数组
            for (int j = 0; j < cols; ++j)
            {
                matrix[i][j] = T(); // 初始化为默认值
            }
        }
    }

    // 拷贝构造函数，确保深拷贝
    Matrix(const Matrix<T> &m)
    {
        rows = m.rows;
        cols = m.cols;
        matrix = new T *[rows];
        for (int i = 0; i < rows; ++i)
        {
            matrix[i] = new T[cols];
            for (int j = 0; j < cols; ++j)
            {
                matrix[i][j] = m.matrix[i][j];
            }
        }
    }

    // 默认构造函数，初始化为空矩阵
    Matrix()
    {
        rows = 0;
        cols = 0;
        matrix = nullptr;
```

```
    }

    // 赋值运算符，防止自我赋值，并保证深拷贝
    Matrix<T> &operator=(const Matrix<T> &m)
    {
        if (this == &m)
            return *this; // 防止自我赋值
        // 释放原有矩阵内存
        for (int i = 0; i < rows; ++i)
            delete[] matrix[i];
        delete[] matrix;
        // 重新分配并复制新矩阵
        rows = m.rows;
        cols = m.cols;
        matrix = new T *[rows];
        for (int i = 0; i < rows; ++i)
        {
            matrix[i] = new T[cols];
            for (int j = 0; j < cols; ++j)
            {
                matrix[i][j] = m.matrix[i][j];
            }
        }
        return *this;
    }

    // 析构函数，释放内存并置空指针
    ~Matrix()
    {
        for (int i = 0; i < rows; ++i)
        {
            delete[] matrix[i];
        }
        delete[] matrix;
        matrix = nullptr; // 防止悬空指针
    }

    void set(int i, int j, T value) // 设置矩阵元素
    {
        matrix[i][j] = value;
    }

    T get(int i, int j) const // 获取矩阵元素
    {
        return matrix[i][j];
    }

    int getRows() const // 获取行数
    {
        return rows;
    }

    int getCols() const // 获取列数
    {
        return cols;
    }

    void print() // 打印矩阵
    {
```

```
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                cout << matrix[i][j] << " ";
            }
            cout << endl;
        }
    }

    // 重载运算符 + (矩阵加法)
    Matrix<T> operator+(const Matrix<T> &m) const
    {
        if (rows != m.rows || cols != m.cols)
        {
            throw invalid_argument(" 矩阵维度不匹配");
        }
        Matrix<T> result(rows, cols);
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                if constexpr (is_same<T, bool>::value)
                {
                    result.set(i, j, matrix[i][j] || m.get(i, j)); // 布尔矩阵使用逻辑或
                }
                else
                {
                    result.set(i, j, matrix[i][j] + m.get(i, j)); // 其他类型使用加法
                }
            }
        }
        return result;
    }

    // 重载运算符 * (矩阵乘法)
    Matrix<T> operator*(const Matrix<T> &m) const
    {
        if (cols != m.rows)
        {
            throw invalid_argument(" 矩阵维度不匹配");
        }
        Matrix<T> result(rows, m.cols);
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < m.cols; j++)
            {
                if constexpr (is_same<T, bool>::value)
                {
                    bool sum = false;
                    for (int k = 0; k < cols; k++)
                    {
                        sum = sum || (matrix[i][k] && m.get(k, j)); // 布尔矩阵使用逻辑与
                    }
                    result.set(i, j, sum);
                }
                else
                {
                    T sum = 0;
```

```
                    for (int k = 0; k < cols; k++)
                    {
                        sum += matrix[i][k] * m.get(k, j); // 其他类型使用乘法
                    }
                    result.set(i, j, sum);
                }
            }
        }
        return result;
    }

    Matrix<T> transpose() // 矩阵转置
    {
        Matrix<T> result(cols, rows);
        for (int i = 0; i < cols; i++)
        {
            for (int j = 0; j < rows; j++)
            {
                result.set(i, j, matrix[j][i]);
            }
        }
        return result;
    }

    // 重载运算符 ==
    bool operator==(Matrix<T> &m)
    {
        if (rows != m.rows || cols != m.cols)
        {
            return false;
        }
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < cols; j++)
            {
                if (matrix[i][j] != m.get(i, j))
                {
                    return false;
                }
            }
        }
        return true;
    }

    // 重载输出运算符
    friend ostream &operator<<(ostream &os, Matrix<T> &m)
    {
        for (int i = 0; i < m.rows; i++)
        {
            for (int j = 0; j < m.cols; j++)
            {
                os << m.get(i, j) << " ";
            }
            os << endl;
        }
        return os;
    }

    // 重载幂运算符
```

```
    Matrix<T> operator^(int n)
    {
        if (rows != cols)
        {
            cout << "Error: Matrix is not square." << endl;
            return *this;
        }
        if (n == 0)
        {
            Matrix<T> result(rows, cols);
            for (int i = 0; i < rows; i++)
            {
                result.set(i, i, 1);
            }
            return result;
        }
        if (n < 0)
        {
            return inv() ^ (-n);
        }
        Matrix<T> result = *this;
        for (int i = 1; i < n; i++)
        {
            result = result * (*this);
        }
        return result;
    }
};
```

其他实现

```
#include <iostream>
#include <limits>
#include <cmath>
#include <string>
#include <sstream>

using namespace std;
Matrix<bool> readMatrix()
{
    int n;

    while (true)
    {
        cout << " 请输入矩阵的阶数 : ";
        cin >> n;
        if (cin.fail() || n <= 0)
        {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << " 请输入正整数! " << endl;
        }
        else { break; }
    }

    int rows = n, cols = n;
    cout << " 请输入关系矩阵 : " << endl;
```

```
27
28      Matrix<bool> m(rows, cols);
29      for (int i = 0; i < rows; i++)
30      {
31          cout << " 请输入矩阵的第 " << i + 1 << " 行元素（元素以空格分割）: ";
32
33          for (int j = 0; j < cols; j++)
34          {
35              while (true)
36              {
37                  int value;
38                  cin >> value;
39                  if (cin.fail() || (value != 0 && value != 1))
40                  {
41                      cin.clear();
42                      cin.ignore(numeric_limits<streamsize>::max(), '\n');
43                      cout << " 矩阵的第 " << i + 1 << " 行输入错误，请重新以合法的布尔值输入该行
                        ↪  : " << endl;
44                      j = 0;
45                  }
46                  else
47                  {
48                      m.set(i, j, value);
49                      break;
50                  }
51              }
52          }
53      }
54      return m;
55  }
56
57  bool askToContinue()
58  {
59      char ch;
60      while (true)
61      {
62          cout << " 是否继续运算（Y/N）？" << endl;
63          cin >> ch;
64          if (ch == 'Y' || ch == 'y')
65          {
66              cin.clear();
67              cin.ignore(numeric_limits<streamsize>::max(), '\n');
68              return true;
69          }
70          else if (ch == 'N' || ch == 'n')
71          {
72              return false;
73          }
74          else
75          {
76              cout << " 输入错误，请重新输入" << endl;
77              cin.clear();
78              cin.ignore(numeric_limits<streamsize>::max(), '\n');
79          }
80      }
81  }
82
83  void printMenu()
84  {
```

```
85      cout << endl;
86      cout << "**********************************" << endl;
87      cout << "**      输入对应序号选择算法      **" << endl;
88      cout << "**********************************" << endl;
89      cout << "**           1. 自反闭包          **" << endl;
90      cout << "**           2. 对称闭包          **" << endl;
91      cout << "**           3. 传递闭包          **" << endl;
92      cout << "**           4. 退出              **" << endl;
93      cout << "**********************************" << endl;
94      cout << endl;
95      cout << " 请输入序号 : ";
96  }
97
98  int main()
99  {
100     Matrix<bool> relationshipMatrix = readMatrix();
101     cout << " 输入的关系矩阵为 : " << endl
102          << relationshipMatrix << endl;
103
104     while (true)
105     {
106         printMenu();
107         int choice;
108         while (true)
109         {
110             cin >> choice;
111             if (cin.fail() || choice < 1 || choice > 4)
112             {
113                 cin.clear();
114                 cin.ignore(numeric_limits<streamsize>::max(), '\n');
115                 cout << " 请输入正确的序号! " << endl;
116             }
117             else
118             {
119                 break;
120             }
121         }
122
123         switch (choice)
124         {
125         case 1:
126         {
127             cout << " 自反闭包 : " << endl;
128             Matrix<bool> reflexiveClosure = reflexive(relationshipMatrix);
129             cout << reflexiveClosure << endl;
130             break;
131         }
132         case 2:
133         {
134             cout << " 对称闭包 : " << endl;
135             Matrix<bool> symmetricClosure = symmetric(relationshipMatrix);
136             cout << symmetricClosure << endl;
137             break;
138         }
139         case 3:
140         {
141             cout << " 传递闭包 : " << endl;
142             Matrix<bool> transitiveClosure = transitive(relationshipMatrix);
143             cout << transitiveClosure << endl;
```

```
144             break;
145         }
146         case 4:
147         {
148             return 0;
149         }
150         default:
151             break;
152         }
153 
154         if (!askToContinue())
155         {
156             break;
157         }
158     }
159     return 0;
160 }
```

3 项目测试

3.1 正常测试

```
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(元素以空格分割) : 1 0 1 1
请输入矩阵的第 2 行元素(元素以空格分割) : 0 1 0 1
请输入矩阵的第 3 行元素(元素以空格分割) : 0 0 0 0
请输入矩阵的第 4 行元素(元素以空格分割) : 1 0 0 0
输入的关系矩阵为 :
1 0 1 1
0 1 0 1
0 0 0 0
1 0 0 0
```

3.1.1 求关系的自反闭包

```
**********************************
**      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              **
**********************************

请输入序号 : 1
自反闭包 :
1 0 1 1
0 1 0 1
0 0 1 0
1 0 0 1
```

3.1.2 求关系的对称闭包

```
************************************
**      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              **
************************************

请输入序号 : 2
对称闭包 :
1 0 1 1
0 1 0 1
1 0 0 0
1 1 0 0
```

3.1.3 求关系的传递闭包

```
************************************
**      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              **
************************************

请输入序号 : 3
传递闭包 :
1 0 1 1
1 1 1 1
0 0 0 0
1 0 1 1
```

3.2 健壮性测试

```
请输入矩阵的阶数 : -
请输入正整数!
请输入矩阵的阶数 : -2
请输入正整数!
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(元素以空格分割) : 1 3 q
矩阵的第 1 行输入错误, 请重新以合法的布尔值输入该行 :
1 0 1
请输入矩阵的第 2 行元素(元素以空格分割) : 2 sjkhbhkba nj
矩阵的第 2 行输入错误, 请重新以合法的布尔值输入该行 :
1 2 0
矩阵的第 2 行输入错误, 请重新以合法的布尔值输入该行 :
0 0 1
请输入矩阵的第 3 行元素(元素以空格分割) :
```

```
是否继续运算(Y/N)?
9
输入错误, 请重新输入
是否继续运算(Y/N)?
x
输入错误, 请重新输入
是否继续运算(Y/N)?
y

**********************************
**      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               **
**********************************

请输入序号 : 5
请输入正确的序号!
`
请输入正确的序号!
```

4 心得体会

在实施本项目的过程中，我们深刻体会到了数据结构和算法设计的重要性。特别是在计算传递闭包时，我们遇到了一些挑战。我们发现，为了正确地计算传递闭包，必须妥善保存每次迭代的结果，否则会丢失前次的计算结果，导致错误的输出。这个经验教训告诉我们，在处理循环计算时，如果每次迭代依赖于前一次的结果，就必须小心保存每次的计算结果。

此外，通过本项目的实现，我们也加深了对离散数学中关系闭包概念的理解，并通过编程实践提高了我们的编程技能。