

## 《离散数学课程项目文档》

### ——命题逻辑联接词、真值表、主范式

作者姓名: \_\_\_\_\_ 胡峻玮

学 号: \_\_\_\_\_ 2153393

指导教师: \_\_\_\_\_ 唐剑锋

学院、专业: \_\_\_\_\_ 软件学院 软件工程



## 目 录

1 项目分析.....	1
1.1 项目背景.....	1
1.2 项目要求.....	1
2 项目设计.....	1
2.1 数据结构设计.....	1
2.2 算法设计.....	2
2.2.1 算法思路.....	2
2.2.2 性能评估.....	2
2.2.3 流程图表示.....	3
2.2.4 代码实现.....	3
· 求自反闭包.....	3
· 求对称闭包.....	3
· 求传递闭包.....	4
· 项目主体部分.....	6
3 项目测试.....	8
3.1 求关系的自反闭包.....	8
3.2 求关系的对称闭包.....	9
3.3 求关系的传递闭包.....	9
4 心得体会.....	10

## 1. 项目分析

### 1.1 项目背景

假设集合  $A$  上存在非空关系  $R$ ，通常我们希望这种关系具备某些有益特性，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  $R'$ 。目的是使得  $R'$  保持原有的特性，并且希望通过尽可能少的有序对扩展来实现这一点。这样的  $R'$  被称为关系  $R$  的自反闭包、对称闭包或传递闭包。

### 1.2 项目要求

手动输入矩阵阶数，然后手动输入关系矩阵，程序需要能求解该关系的自反、对称和传递闭包。

## 2. 项目设计

### 2.1 数据结构设计

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。鉴于在逻辑推理过程中通常涉及的命题数量不多，常规的数组结构已足够使用，因此本项目不另行设计新的数据结构。

## 2.2 算法设计

### 2.2.1 算法思路

由各类闭包的性质有：

$$\begin{cases} r(R) = R \cup R^0 \\ s(R) = R \cup R^{-1} \\ t(R) = R \cup R^2 \cup R^3 \cup \dots \cup R^n \end{cases}$$

转换成关系矩阵，求解方法如下：（均为逻辑加）

$$\begin{cases} M_r = M + E \\ M_s = M + M^T \\ M_t = M + M^2 + M^3 + \dots + M^n \end{cases}$$

### 2.2.2 性能评估

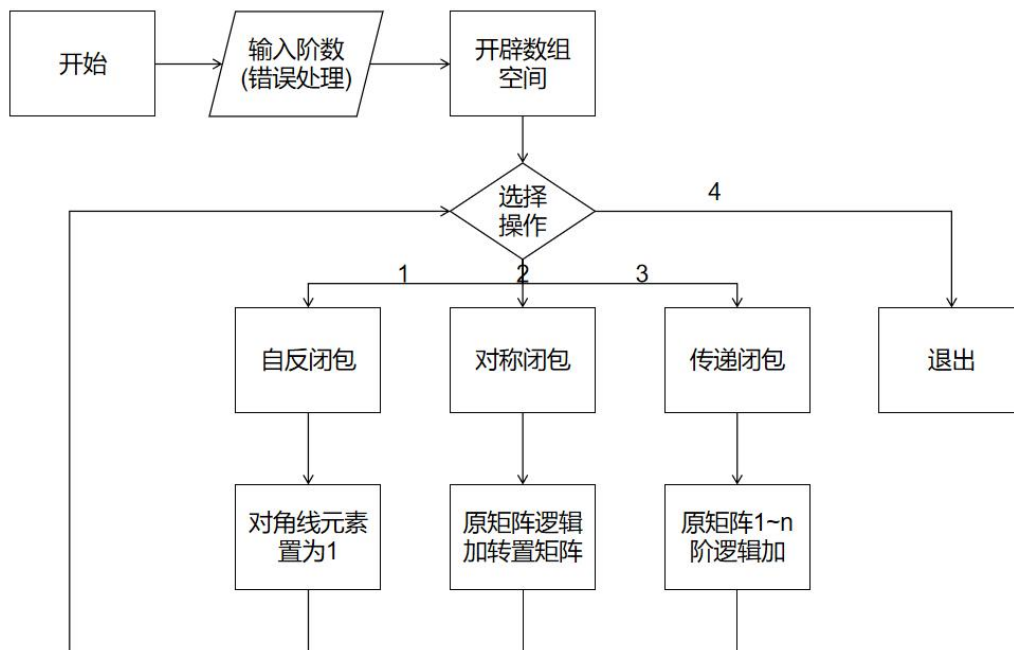
设  $R$  是集合  $A$  上的二元关系， $A$  中有  $n^2$  个元素。

自反闭包只需让对角线元素都为 1，仅需  $n$  次，时间复杂度为  $O(n)$ 。

对称闭包是原矩阵加上原矩阵的转置矩阵，转置需要  $O(n^2)$ ，相加需要  $O(n^2)$ ，总时间复杂度仍为  $O(n^2)$ 。

传递闭包需要完成原矩阵 1 次到  $n$  次方的累加，第  $k$  次中已经得到原矩阵  $k-1$  次方，与原矩阵相乘得原矩阵  $k$  次方，需要  $O(n^3)$ ，累加到结果上需要  $O(n^2)$ ，故一次仍需  $O(n^3)$ 。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

### · 求自反闭包

```

void Reflexive(bool** Matrix, const int num)
{
    cout << "自反闭包:" << endl;
    //对角线上元素都置为1
    for (int i = 0; i < num; i++)
        Matrix[i][i] = true;
}
  
```

### · 求对称闭包

```

void Symmetric(bool** Matrix, const int num)
{
    bool** rev;
    Create(rev, num);
    cout << "对称闭包" << endl;
    //取原矩阵的转置
    for (int i = 0; i < num; i++)
    {
        for (int j = 0; j < num; j++)
  
```

```
{
    rev[i][j] = Matrix[j][i];
}

}

//原矩阵与转置相加
for (int i = 0; i < num; i++)
{
    for (int j = 0; j < num; j++)
    {
        Matrix[i][j] = Matrix[i][j] || rev[i][j];
    }
}

Delete(rev, num);
}
```

## · 求传递闭包

```
void Transitive(bool** Matrix, const int num)
{
    //原矩阵
    bool** Default;
    Create(Default, num);

    //原矩阵的 n 次方
    bool** Current;
    Create(Current, num);

    //运算原矩阵 n 次方时，存放临时结果
    bool** Result;
    Create(Result, num);
    cout << "传递闭包" << endl;
    //给 Default 和 Current 赋值，使得 Default 是原矩阵，Current 是单位阵
    for (int i = 0; i < num; i++)
    {
        for (int j = 0; j < num; j++)
        {
            Default[i][j] = Matrix[i][j];

            if (i == j)
                Current[i][j] = true;
            else
                Current[i][j] = false;
        }
    }
}
```

```
}

for (int times = 0; times < num; times++)
{
    //Current 和 Default 相乘，得到 M 的 n 次方
    for (int i = 0; i < num; i++)
    {
        for (int j = 0; j < num; j++)
        {
            bool* primes = new bool[num];

            for (int k = 0; k < num; k++)
                primes[k] = Current[i][k] && Default[k][j];

            Result[i][j] = false;

            for (int k = 0; k < num; k++)
                Result[i][j] = Result[i][j] || primes[k];

            delete[] primes;
        }
    }

    //更新 Current 的值
    for (int i = 0; i < num; i++)
    {
        for (int j = 0; j < num; j++)
        {
            Current[i][j] = Result[i][j];
        }
    }

    //Matrix 和 Current 相加，得到结果
    for (int i = 0; i < num; i++)
    {
        for (int j = 0; j < num; j++)
        {
            Matrix[i][j] = Matrix[i][j] || Current[i][j];
        }
    }
}

Delete(Default, num);
```

```
Delete(Current, num);
Delete(Result, num);
}
```

## · 项目主体部分

```
int main()
{
    int num = 0;
    bool isContinue = true;

    while (1)
    {
        cout << "请输入矩阵阶数: ";
        cin >> num;
        cout << endl;
        if (cin.good() && num > 0 && num <= INT_MAX)
            break;

        cin.clear();
        cin.ignore(INT_MAX, '\n');
        cout << "输入错误! 请重新输入" << endl;
    }

    bool** data;
    Create(data, num);

    for (int i = 0; i < num; i++)
    {
        //输入每个元素
        cout << "请输入矩阵的第" << i << "行元素(元素以空格分隔):";
        for (int j = 0; j < num; j++)
        {
            while (1)
            {
                cin >> data[i][j];

                if (cin.good() && (data[i][j] == true || data[i][j] == false))
                    break;

                cin.clear();
                cin.ignore(INT_MAX, '\n');
                cout << "矩阵的第" << i << "行, 第" << j << "列元素输入错误, 请继续输入:";
            }
        }
    }
}
```



```
}
cout << endl;
}

while (isContinue)
{
    cout << endl;
    cout << "*****" << endl;
    cout << "**      输入对应序号选择算法      **" << endl;
    cout << "*****" << endl;
    cout << "**          1. 自反闭包          **" << endl;
    cout << "**          2. 对称闭包          **" << endl;
    cout << "**          3. 传递闭包          **" << endl;
    cout << "**          4. 退出              **" << endl;
    cout << "*****" << endl;

    char choice = '\0';

    while (1)
    {
        choice = _getch();

        if (choice >= '1' && choice <= '4')
        {
            cout << choice << endl;
            break;
        }
    }

    bool** Matrix;
    Create(Matrix, data, num);

    switch (choice)
    {
        case '1':
            Reflexive(Matrix, num);
            show(Matrix, num);
            Delete(Matrix, num);
            break;
        case '2':
            Symmetric(Matrix, num);
            show(Matrix, num);
            Delete(Matrix, num);
```

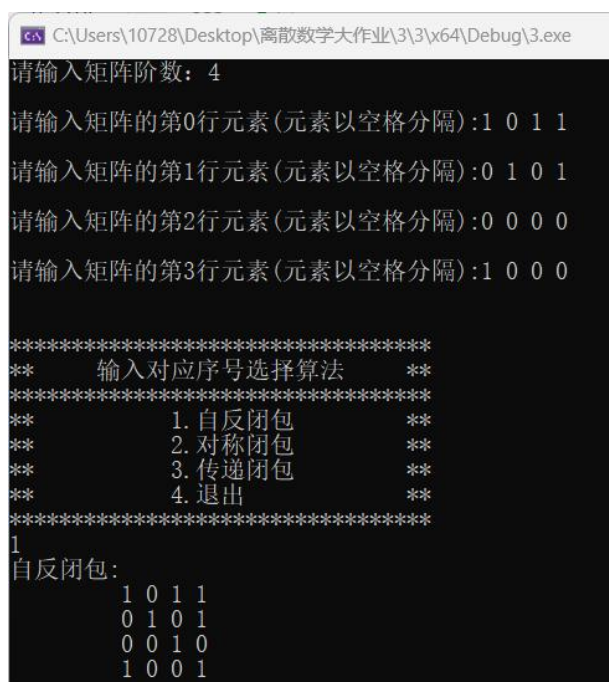
```
        break;
    case '3':
        Transitive(Matrix, num);
        show(Matrix, num);
        Delete(Matrix, num);
        break;
    case '4':
        isContinue = false;
        Delete(Matrix, num);
        break;
    }
}

return 0;
}
```

## 3. 项目测试

### 3.1 求关系的自反闭包

测试结果：



```
C:\Users\10728\Desktop\离散数学大作业\3\3\x64\Debug\3.exe
请输入矩阵阶数: 4
请输入矩阵的第0行元素(元素以空格分隔): 1 0 1 1
请输入矩阵的第1行元素(元素以空格分隔): 0 1 0 1
请输入矩阵的第2行元素(元素以空格分隔): 0 0 0 0
请输入矩阵的第3行元素(元素以空格分隔): 1 0 0 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:
    1 0 1 1
    0 1 0 1
    0 0 1 0
    1 0 0 1
```

## 3.2 求关系的对称闭包

测试结果：

```
C:\Users\10728\Desktop\离散数学大作业\3\3\x64\Debug\3.exe
请输入矩阵阶数：4
请输入矩阵的第0行元素(元素以空格分隔):1 0 1 1
请输入矩阵的第1行元素(元素以空格分隔):0 1 0 1
请输入矩阵的第2行元素(元素以空格分隔):0 0 0 0
请输入矩阵的第3行元素(元素以空格分隔):1 0 0 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
    1 0 1 1
    0 1 0 1
    1 0 0 0
    1 1 0 0
```

## 3.3 求关系的传递闭包

测试结果：

```
C:\Users\10728\Desktop\离散数学大作业\3\3\x64\Debug\3.exe
请输入矩阵阶数：4
请输入矩阵的第0行元素(元素以空格分隔):1 0 1 1
请输入矩阵的第1行元素(元素以空格分隔):0 1 0 1
请输入矩阵的第2行元素(元素以空格分隔):0 0 0 0
请输入矩阵的第3行元素(元素以空格分隔):1 0 0 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
    1 0 1 1
    1 1 1 1
    0 0 0 0
    1 0 1 1
```

## 4. 心得体会

在实施本项目的信息存储环节，鉴于必须使用动态二维数组来储存数据，而这需要较大的连续内存空间，我们选择了动态分配内存的策略。在解决问题的过程中，自反闭包和对称闭包的计算相对顺利，但是在计算传递闭包时却频繁遭遇错误。经过反复检查，我发现问题所在：传递闭包的求解需要将原矩阵从 1 次方加至  $n$  次方，这要求在第  $k$  次迭代计算时，不仅要保存原始矩阵和原矩阵的  $k$

次方，还要保存从 1 到  $k$  次的累积结果，以便进行下一次迭代。这是一开始编码时未能预见的，导致了无数的错误。由此我得到了一个宝贵的经验：在循环计算中，如果每次迭代依赖于前一次的结果，就必须妥善保存每次的计算结果，否则失去了前次的结果，当前的计算也就失去了意义。