《离散数学》课程实验报告 3 求关系的自反、对称和传递闭包

2254321 陆诚彬

1. 实验内容

本实验旨在学习如何计算关系的自反闭包、对称闭包和传递闭包，以及如何用矩阵表示这些闭包。这是理解离散数学中关系理论的一个重要实践环节。

2. 解题思路

自反闭包：

理论基础：一个关系的自反闭包是在原有关系的基础上，添加必要的元素以使其成为自反的。

实现方法：将关系矩阵的主对角线上的所有元素置为 1。

对称闭包：

理论基础：对称闭包是在原有关系的基础上，添加最少的元素使其成为对称的。

实现方法：将关系矩阵与其转置矩阵进行逻辑或操作。

传递闭包：

理论基础：传递闭包是在原有关系的基础上，添加最少的元素使其成为传递的。

实现方法：使用深度优先搜索（DFS）算法，递归地检查和更新矩阵中的元素。

3. 数据结构设计

关系矩阵表示：

数据类型：vector<vector<int>>

描述：使用二维向量来存储关系矩阵，其中每个元素代表关系的存在（1）或不存在（0）。

4. 项目实现

4.1. void reflexiveClosure(vector<vector<int>>& s)

功能：计算关系矩阵的自反闭包。

实现细节：遍历关系矩阵的主对角线，将对角线上的元素设置为 1，从而确保关系是自反的。

代码实现：

```
1.    void reflexiveClosure(vector<vector<int>>& s) {
2.        for (int i = 0; i < n; i++) {
3.            s[i][i] = 1;
4.        }
5.        output(s);
6.        selectOption();
7.    }
```

4.2. void symmetricClosure(vector<vector<int>>& s)

功能：计算关系矩阵的对称闭包。

实现细节：创建一个与原矩阵相同的辅助矩阵，遍历原矩阵的每个元素，将它与对应的转置元素进行逻辑或操作，以确保对称性。

代码实现：

```
1.    void symmetricClosure(vector<vector<int>>& s) {
2.        vector<vector<int>> s1 = s;
3.        for (int i = 0; i < n; i++) {
4.            for (int j = 0; j < d; j++) {
5.                s[i][j] = s[i][j] || s1[j][i];
6.            }
7.        }
8.        output(s);
9.        selectOption();
10.   }
```

4.3. void transitiveClosure(vector<vector<int>>& s)

功能：计算关系矩阵的传递闭包。

实现细节：使用深度优先搜索（DFS）遍历矩阵，为每个节点构建可达性路径。如果从节点 i 可以通过节点 current 达到节点 next，则将 t[i][next] 设置为 1。

代码实现：

```
1.    void transitiveClosure(vector<vector<int>>& s) {
2.        vector<vector<int>> t(n, vector<int>(d, 0));
3.        for (int i = 0; i < n; i++) {
4.            dfs(i, i, t);
5.        }
6.        output(t);
7.        selectOption();
8.    }
```

4.4. void dfs(int start, int current, vector<vector<int>>& t)

功能：为 transitiveClosure 函数提供深度优先搜索功能。

实现细节：从起始节点 start 开始递归搜索，更新可达节点信息。

代码实现：

```
1.    void dfs(int start, int current, vector<vector<int>>& t) {
2.        for (int next = 0; next < d; next++) {
3.            if (s[current][next] && !t[start][next]) {
4.                t[start][next] = 1;
5.                dfs(start, next, t);
6.            }
7.        }
8.    }
```

这些函数构成了程序的核心逻辑，分别实现了求解自反闭包、对称闭包和传递闭包的主要功能。通过深入分析这些函数，我们可以清晰地理解它们各自的作用和实现方式。

4.5. 其他辅助函数

(1) void output(const vector<vector<int>>& s)

功能：输出关系矩阵。

实现细节：遍历矩阵的每一行和每一列，打印出每个元素。每打印完一行元素后换行，以保持输出的格式整齐。

代码实现：

```
1.    void output(const vector<vector<int>>& s) {
2.        cout << "所求关系矩阵为:\n";
3.        for (int i = 0; i < n; i++) {
4.            for (int j = 0; j < d; j++) {
5.                cout << s[i][j] << " ";
6.            }
7.            cout << endl;
8.        }
9.    }
```

(2) void selectOption()

功能：提供用户界面，允许用户选择要执行的操作。

实现细节：首先提示用户输入矩阵的大小和元素，然后提供一个选项菜单，让用户选择计算自反闭包、对称闭包、传递闭包，或退出程序。

代码实现：

```
1.    void selectOption() {
2.        cout << "请输入矩阵的行数和列数: ";
3.        cin >> n >> d;
4.        s.resize(n, vector<int>(d));
5.
6.        cout << "请输入关系矩阵:\n";
7.        for (int i = 0; i < n; i++) {
8.            cout << "请输入矩阵的第" << i << "行元素(元素以空格分隔): ";
9.            for (int j = 0; j < d; j++) {
10.               cin >> s[i][j];
11.           }
12.       }
13.
14.       cout << "输入对应序号选择算法\n1:自反闭包\n2:传递闭包\n3:对称闭包\n4:退出
    \n";
15.       char choice;
16.       cin >> choice;
17.       // ... 后续根据选择调用不同的函数
```

这些辅助函数虽然在整个程序中扮演的角色不如核心函数那么直接，但它们为程序提供了必要的交互功能和数据展示方式，是整个程序能够顺利运行的基础。通过对它们的详细分析，我们可以更好地理解程序的整体结构和工作流程。

5. 设计小结

关键实现：

(1) **自反闭包：**通过简单地修改矩阵的对角线元素，我们能够将任何关系转换成自反关系，这是一种高效且直观的方法。

(2) **对称闭包：**这里采用了将原矩阵与其转置矩阵进行逻辑或运算的方法，有效地实现了对称闭包的计算。

(3) **传递闭包：**通过深度优先搜索（DFS），我们能够发现所有间接的关系，从而成功构建出传递闭包。

代码结构：

我们的程序结构清晰，分为几个核心的功能实现函数和辅助函数。核心函数直接处理闭包的计算，而辅助函数则负责与用户的交互和结果展示。这种模块化的设计使得程序易于理解和维护。

学习成果：

通过这次实验，我们不仅学习了关系闭包的计算方法，还练习了如何使用向量（vector）这一数据结构来有效地存储和处理矩阵数据。此外，深度优先搜索的应用也是一个很好的算法实践，加深了我们对递归和图遍历概念的理解。

结论：

总的来说，这次实验是一个成功的学习经历。它不仅提高了我们在理论知识方面的理解，还增强了我们在实际编程和问题解决方面的能力。这些技能和知识对于我们未来的学术和职业生涯都是非常宝贵的资产。