# 求关系的自反、对称和传递闭包

2153726 罗宇翔

(经济与管理学院, 信息管理与信息系统)

# 目录

# 1 实验目的

利用 $C++$ 代码实现求关系矩阵的自反、对称和传递闭包。

# 2 实验内容

给定一个关系矩阵，求其对应的自反、对称和传递闭包。

# 3 实验环境

## 3.1 Visual Studio Code

Version: 1.89.1

Browser:

Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)

AppleWebKit/537.36 (KHTML, like Gecko) Code/1.89.1

Chrome/120.0.6099.291 Electron/28.2.8 Safari/537.36

## 3.2 g++

Apple clang version 14.0.0 (clang-1400.0.29.202)

Target: x86_64-apple-darwin21.6.0

Thread model: posix

# 4 实验原理和方法

对以矩阵表示的关系，其自反闭包只要将矩阵的主对角线全部置为 1，对称闭包则由关系矩阵加上其转置矩阵得到（逻辑加）。本题中传递闭包由如下公式计算得到：

$$t(R) = R \cup R^2 \cup R^3 \cdots$$

# 5 实验代码

```
#include <iostream>
#include <sstream>
#include <vector>

using namespace std;

typedef vector< vector< int > > Matrix;

/**
    * @brief Get the matrix object
    *
    * @return Matrix
    */
Matrix get_matrix();

/**
    * @brief Print the matrix object
    *
    * @param Matrix
    */
void print_matrix(Matrix &matrix);

/**
    * @brief check the legality of the matrix
    *
    * @param matrix
    * @return true
    * @return false
    */
bool validation(Matrix &matrix);

Matrix reflexive_closure(Matrix matrix);
```

```
Matrix symmetry_closure(Matrix matrix);

Matrix transitive_closure(Matrix matrix);

/**
    * @brief calculate A[i][:]*B[:][j]
    *
    * @param A
    * @param B
    * @param i
    * @param j
    * @return int
    */
int product(Matrix &A, Matrix &B, int i, int j);

/**
    * @brief calculate AB
    *
    * @param A
    * @param B
    * @return Matrix
    */
Matrix multi(Matrix &A, Matrix &B);

/**
    * @brief calculate A+B
    *
    * @param A
    * @param B
    * @return Matrix
    */
```

```
Matrix add(Matrix &A, Matrix &B);

int main() {
    while (true) {
        cout << "any to start, q to quit\n";
        string expression;
        cin >> expression;
        if (expression == "q") {
            cout << "thanks for using\n";
            break;
        }
        Matrix a;
        a = get_matrix();
        if (validation(a)) {  // if valid, print the result
            cout << "reflexivity closure:\n";
            Matrix rMatrix = reflexive_closure(a);
            print_matrix(rMatrix);
            cout << "symmetry closure:\n";
            Matrix sMatrix = symmetry_closure(a);
            print_matrix(sMatrix);
            cout << "transitivity closure:\n";
            Matrix tMatrix = transitive_closure(a);
            print_matrix(tMatrix);
        } else {  // else raise exception
            cout << "invalid input, please check\n";
        }
    }
    return 0;
}

Matrix get_matrix() {
    Matrix matrix;
```

```
    string line;
    cout << "Enter the matrix elements (separate elements with
   spaces, and rows with newlines). Enter EOF to finish:\n";
    while (getline(cin, line))  // split the stream by spaces
   and newlines
    {
        if (line == "EOF")
            break;
        istringstream iss(line);
        vector< int > row;
        int num;
        while (iss >> num)
            row.push_back(num);
        matrix.push_back(row);
    }
    matrix.erase(matrix.begin());
    return matrix;
}

void print_matrix(Matrix &matrix) {
    for (int i = 0; i < matrix.size(); i++) {
        for (int j = 0; j < matrix[i].size(); j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }
}

bool validation(Matrix &matrix) {
    int rows = matrix.size();
    for (int row = 0; row < rows; row++) {
        if (matrix[row].size() != rows) {
```

```
            return false;
        }
        for (int col = 0; col < matrix[row].size(); col++) {
            if (matrix[row][col] != 0 && matrix[row][col] != 1)
    {
                printf("invalid input in (%d,%d) %d\n", row,
   col, matrix[row][col]);
                return false;
            }
        }
    }
    return true;
}

Matrix reflexive_closure(Matrix matrix) {
    for (int i = 0; i < matrix.size(); i++)
        matrix[i][i] = 1;
    return matrix;
}

Matrix symmetry_closure(Matrix matrix) {
    for (int i = 0; i < matrix.size(); i++)
        for (int j = i + 1; j < matrix[i].size(); j++)
            matrix[j][i] = matrix[i][j] = matrix[i][j] | matrix
   [j][i];
    return matrix;
}

Matrix transitive_closure(Matrix matrix) {
    Matrix sumMat = matrix;
    Matrix mulMat = matrix;
    for (int i = 0; i < matrix.size() - 1; i++) {
```

```
        mulMat = multi(mulMat, matrix);
        sumMat = add(sumMat, mulMat);
    }
    return sumMat;
}

int product(Matrix &A, Matrix &B, int i, int j) {
    int result = 0;
    for (int k = 0; k < A.size(); k++) {
        result = result || (A[i][k] && B[k][j]);
    }
    return result;
}

Matrix multi(Matrix &A, Matrix &B) {
    Matrix result;
    int size = A.size();
    for (int i = 0; i < size; i++) {
        vector< int > row;
        for (int j = 0; j < size; j++) {
            row.push_back(product(A, B, i, j));
        }
        result.push_back(row);
    }
    return result;
}

Matrix add(Matrix &A, Matrix &B) {
    Matrix result;
    int size = A.size();
    for (int i = 0; i < size; i++) {
        vector< int > row;
```

```
        for (int j = 0; j < size; j++) {
            row.push_back(A[i][j] || B[i][j]);
        }
        result.push_back(row);
    }
    return result;
}
```

# 6 实验数据及结果分析

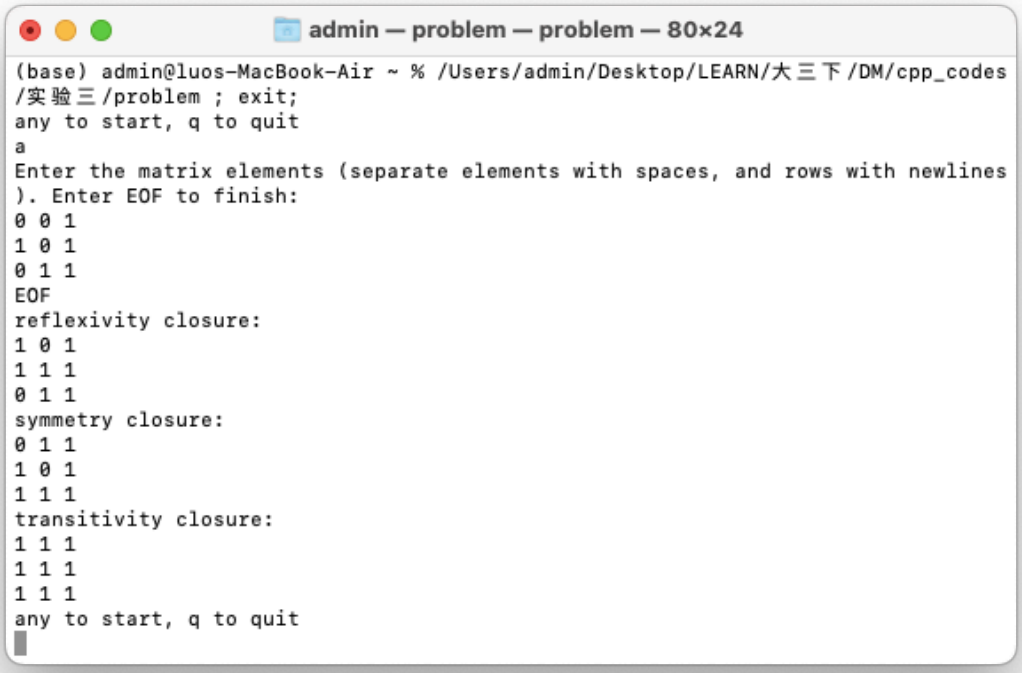


Figure 1：正确输入

正确输入下，根据关系矩阵计算自反、对称和传递闭包。

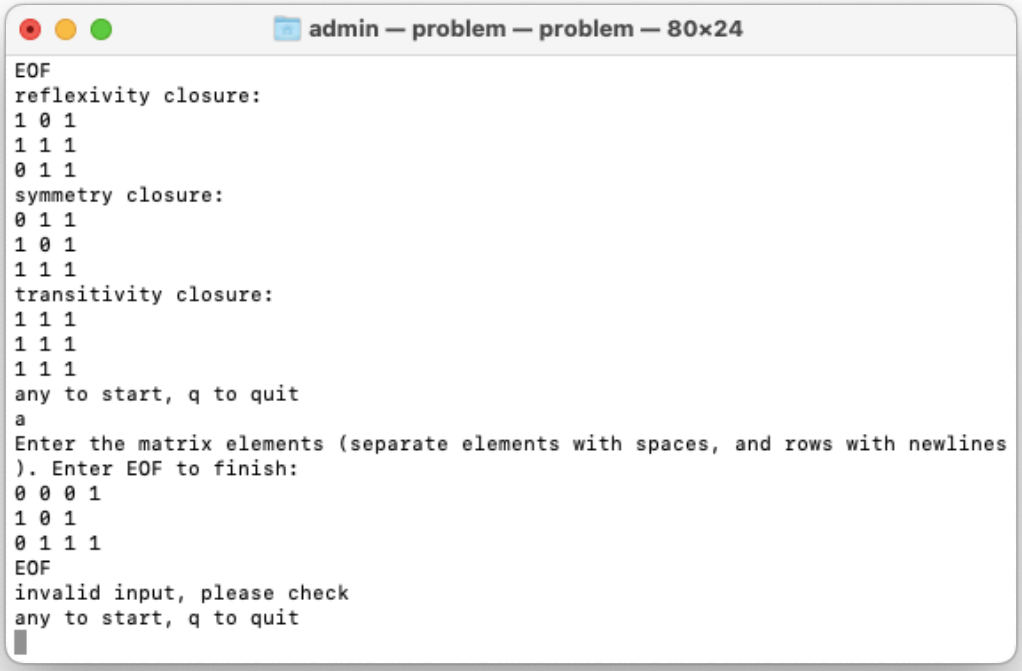


Figure 2: 形状错误

矩阵形状错误，显示错误信息。

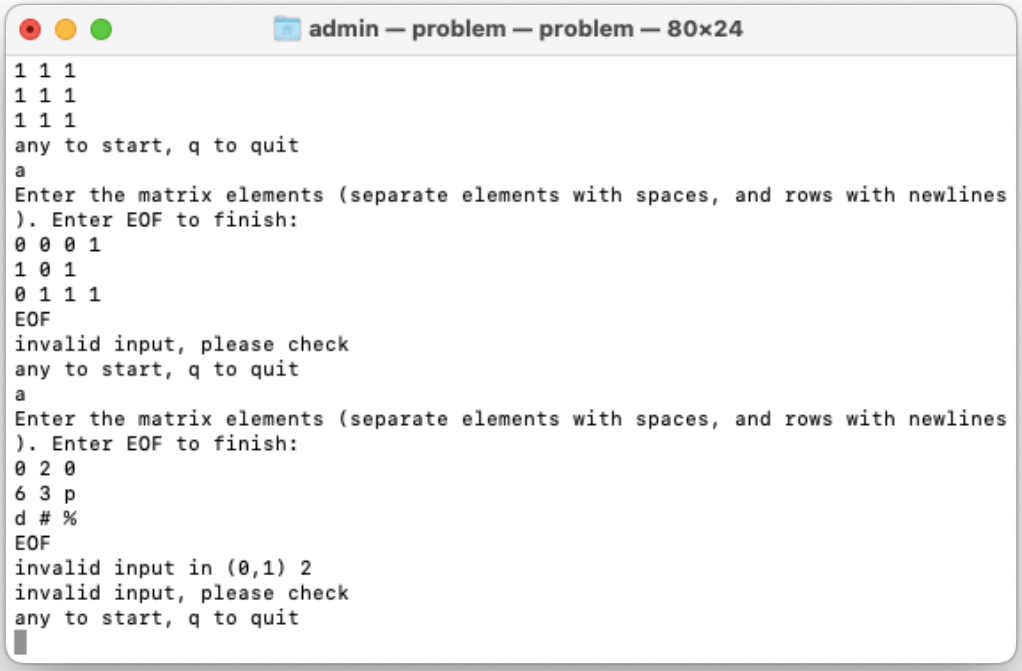


Figure 3：元素错误

矩阵元素错误，显示第一个错误的元素的位置和错误信息。