同济大学

TONGJI UNIVERSITY

离散数学课程设计

项目名称	Warshall 算法求关系的对称闭包
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：

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

传递闭包确保若 aRb 且 bRc，则 aRc。

在离散数学二元关系一章的 4.3 节中，学到关系的闭包计算，其中自反闭包及对称闭包都比较容易解决，而对于其中的传递闭包就没有前两者那么容易解决。传统的求传递闭包的算法的时间复杂度是 $O(n^4)$，程序复杂度较高。

Warshall 在 1962 年提出了一种求传递闭包的复杂度更低的 Warshall 算法。Warshall 算法时间复杂度从传统的求传递闭包的算法的 $O(n^4)$ 降到了 $O(n^3)$。是一个伟大的发现，本报告中将详细阐述本算法的实现。

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）利用 Warshall 算法求解传递闭包

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

Warshall 算法是一种用于计算布尔矩阵传递闭包的动态规划算法。其核心思想是通过引入中间节点，不断更新矩阵的可达性，从而计算出图中任意两个节点之间的传递关系。

在图论中，传递闭包对应于图的可达矩阵：如果节点 i 可以通过一系列中间节点到达节点 j，则传递闭包矩阵中 $M[i][j] = 1$（否则为 0）。

算法将传递性递归定义转化为逐步构造的过程。对于矩阵 M，如果 $M[i][j] = 1$ 或 $M[i][k] = 1 \cap M[k][j] = 1$, 则表示节点 i 可达节点 j（直接或间接）。通过将节点 k 逐一作为中间点检查路径，最终构造出完整的传递闭包。

以下是 Warshall 算法的详细步骤：

（1）设 k 为中间节点索引，取值范围为 $[0, n-1]$。

（2）对于每对节点 i 和 j，检查是否存在通过节点 k 的路径：

$$M[i][j] = M[i][j] \cup (M[i][k] \cap M[k][j])$$

•如果 $M[i][k] = 1$ 且 $M[k][j] = 1$，则 $i \to k \to j$ 表示从 i 到 j 可达。

•更新矩阵 M 中对应的值 $M[i][j]$ 为 1。

（3）重复步骤 (2)，直至 k 遍历所有节点。

以求解如下关系（$R^{(\infty)}$就是原矩阵）的传递闭包为例：

$$R^{(0)}=\begin{bmatrix}0&1&0&0\\0&0&0&1\\0&0&0&0\\1&0&1&0\end{bmatrix}\longrightarrow R^{(1)}=\begin{bmatrix}0&1&0&0\\0&0&0&1\\0&0&0&0\\1&1&1&0\end{bmatrix}$$

标记第 0 行第 0 列，检查所有非十字上且为 0 元素。

M[3, 1]=0 && M[3, 0]=1 && M[0, 1]=1，因此把 M[3, 1]置为 1

$$R^{(1)}=\begin{bmatrix}0&1&0&0\\0&0&0&1\\0&0&0&0\\1&1&1&0\end{bmatrix}\longrightarrow R^{(2)}=\begin{bmatrix}0&1&0&1\\0&0&0&1\\0&0&0&0\\1&1&1&1\end{bmatrix}$$

标记第 1 行第 1 列，检查所有非十字上且为 0 元素。

M[0, 4]=0 && M[0, 1]=1 && M[1, 4]=1，因此把 M[0, 4]置为 1

M[4, 4]=0 && M[4, 1]=1 && M[1, 4]=1，因此把 M[4, 4]置为 1

$$R^{(2)}=\begin{bmatrix}0&1&0&1\\0&0&0&1\\0&0&0&0\\1&1&1&1\end{bmatrix}\longrightarrow R^{(3)}=\begin{bmatrix}0&1&0&1\\0&0&0&1\\0&0&0&0\\1&1&1&1\end{bmatrix}$$

标记第 2 行第 2 列，检查所有非十字上且为 0 元素。

没有符合要求的元素

$$R^{(3)}=\begin{bmatrix}0&1&0&1\\0&0&0&1\\0&0&0&0\\1&1&1&1\end{bmatrix}\longrightarrow R^{(4)}=\begin{bmatrix}1&1&1&1\\1&1&1&1\\0&0&0&0\\1&1&1&1\end{bmatrix}$$

标记第 3 行第 3 列，检查所有非十字上且为 0 元素。

M[0, 0]=0 && M[0, 3]=1 && M[3, 0]=1，因此把 M[0, 0]置为 1

M[0, 2]=0 && M[0, 3]=1 && M[3, 2]=1，因此把 M[0, 2]置为 1

M[1, 0]=0 && M[1, 3]=1 && M[3, 0]=1，因此把 M[1, 0]置为 1

M[1, 1]=0 && M[1, 3]=1 && M[3, 1]=1，因此把 M[1, 1]置为 1

M[1, 2]=0 && M[1, 3]=1 && M[3, 2]=1，因此把 M[1, 2]置为 1

所得 $R^{(\infty)}$就是所求传递闭包的矩阵

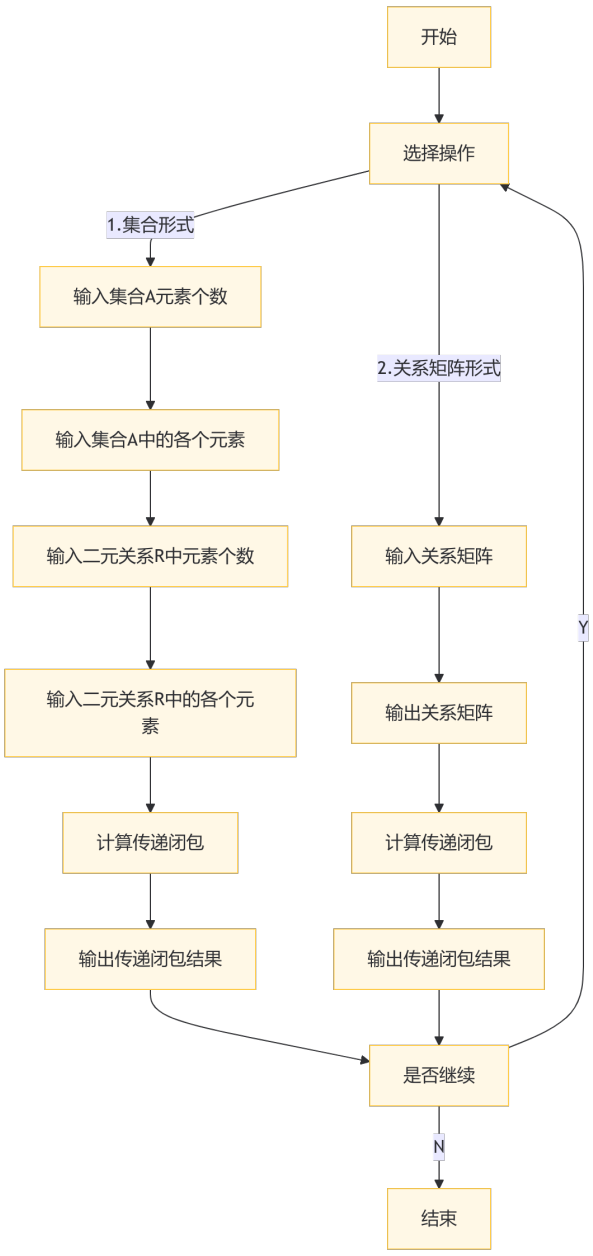
2.2.2 性能评估

Warshall 算法的主要耗时在三重循环中：

- 外层循环 k：中间节点的遍历，需进行 n 次。
- 内层双循环 i 和 j：检查所有起点和终点，需进行 n^2 次。
- 每次更新 $M[i][j]$ 的计算为常数时间 $O(1)$。

综合上述分析，时间复杂度为：$O(n^3)$

2.2.3 流程图表示



2.2.4 代码实现

Warshall 算法传递闭包计算

```cpp
// 传递闭包 Warshall 算法
Matrix<bool> transitiveWarshall(Matrix<bool> &m)
{
    Matrix<bool> result = m;
    int n = m.getRows();
    for (int k = 0; k < n; k++)
    {
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                // // 更新 result[i][j]，如果存在从 i 到 k 和从 k 到 j 的路径，则存在从 i 到 j 的路径
                result.set(i, j, result.get(i, j) || (result.get(i, k) && result.get(k, j)));
            }
        }
    }
    return result;
}
```

Matrix<T> 类

```cpp
// 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;
    }

```

```
90      T get(int i, int j) const // 获取矩阵元素
91      {
92          return matrix[i][j];
93      }
94
95      int getRows() const // 获取行数
96      {
97          return rows;
98      }
99
100     int getCols() const // 获取列数
101     {
102         return cols;
103     }
104
105     void print() // 打印矩阵
106     {
107         for (int i = 0; i < rows; i++)
108         {
109             for (int j = 0; j < cols; j++)
110             {
111                 cout << matrix[i][j] << " ";
112             }
113             cout << endl;
114         }
115     }
116
117     // 重载运算符 + (矩阵加法)
118     Matrix<T> operator+(const Matrix<T> &m) const
119     {
120         if (rows != m.rows || cols != m.cols)
121         {
122             throw invalid_argument(" 矩阵维度不匹配");
123         }
124         Matrix<T> result(rows, cols);
125         for (int i = 0; i < rows; i++)
126         {
127             for (int j = 0; j < cols; j++)
128             {
129                 if constexpr (is_same<T, bool>::value)
130                 {
131                     result.set(i, j, matrix[i][j] || m.get(i, j)); // 布尔矩阵使用逻辑或
132                 }
133                 else
134                 {
135                     result.set(i, j, matrix[i][j] + m.get(i, j)); // 其他类型使用加法
136                 }
137             }
138         }
139         return result;
140     }
141
142     // 重载运算符 * (矩阵乘法)
143     Matrix<T> operator*(const Matrix<T> &m) const
144     {
145         if (cols != m.rows)
146         {
147             throw invalid_argument(" 矩阵维度不匹配");
148         }
```

```
        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;
```

```
208     }
209
210     // 重载输出运算符
211     friend ostream &operator<<(ostream &os, Matrix<T> &m)
212     {
213         for (int i = 0; i < m.rows; i++)
214         {
215             for (int j = 0; j < m.cols; j++)
216             {
217                 os << m.get(i, j) << " ";
218             }
219             os << endl;
220         }
221         return os;
222     }
223
224     // 重载幂运算符
225     Matrix<T> operator^(int n)
226     {
227         if (rows != cols)
228         {
229             cout << "Error: Matrix is not square." << endl;
230             return *this;
231         }
232         if (n == 0)
233         {
234             Matrix<T> result(rows, cols);
235             for (int i = 0; i < rows; i++)
236             {
237                 result.set(i, i, 1);
238             }
239             return result;
240         }
241         if (n < 0)
242         {
243             return inv() ^ (-n);
244         }
245         Matrix<T> result = *this;
246         for (int i = 1; i < n; i++)
247         {
248             result = result * (*this);
249         }
250         return result;
251     }
252 };
```

其他实现

```
1 #include <iostream>
2 #include <limits>
3 #include <cmath>
4 #include <string>
5 #include <sstream>
6 #include <vector>
7 #include <map>
8
9 using namespace std;
```

```
10
11  Matrix<bool> readMatrix()
12  {
13      int n;
14
15      while (true)
16      {
17          cout << " 请输入矩阵的阶数 : ";
18          cin >> n;
19          if (cin.fail() || n <= 0)
20          {
21              cin.clear();
22              cin.ignore(numeric_limits<streamsize>::max(), '\n');
23              cout << " 请输入正整数! " << endl;
24          }
25          else { break; }
26      }
27
28      int rows = n, cols = n;
29      cout << " 请输入关系矩阵 : " << endl;
30
31      Matrix<bool> m(rows, cols);
32      for (int i = 0; i < rows; i++)
33      {
34          cout << " 请输入矩阵的第 " << i + 1 << " 行元素 (元素以空格分割) : ";
35
36          for (int j = 0; j < cols; j++)
37          {
38              while (true)
39              {
40                  int value;
41                  cin >> value;
42                  if (cin.fail() || (value != 0 && value != 1))
43                  {
44                      cin.clear();
45                      cin.ignore(numeric_limits<streamsize>::max(), '\n');
46                      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;
47                      j = 0;
48                  }
49                  else
50                  {
51                      m.set(i, j, value);
52                      break;
53                  }
54              }
55          }
56      }
57      return m;
58  }
59
60  bool askToContinue()
61  {
62      char ch;
63      while (true)
64      {
65          cout << " 是否继续运算 (Y/N)？" << endl;
66          cin >> ch;
67          if (ch == 'Y' || ch == 'y')
```

```
68          {
69              cin.clear();
70              cin.ignore(numeric_limits<streamsize>::max(), '\n');
71              return true;
72          }
73          else if (ch == 'N' || ch == 'n')
74          {
75              return false;
76          }
77          else
78          {
79              cout << " 输入错误，请重新输入" << endl;
80              cin.clear();
81              cin.ignore(numeric_limits<streamsize>::max(), '\n');
82          }
83      }
84  }
85
86  int main()
87  {
88      while (true)
89      {
90          cout << " 请选择求解传递闭包的操作 : (1. 集合形式; 2. 关系矩阵形式)" << endl;
91          int choice;
92          while (true)
93          {
94              cin >> choice;
95              if (cin.fail() || choice != 1 && choice != 2)
96              {
97                  cin.clear();
98                  cin.ignore(numeric_limits<streamsize>::max(), '\n');
99                  cout << " 输入错误，请重新输入" << endl;
100             }
101             else
102             {
103                 break;
104             }
105         }
106         if (choice == 1)
107         {
108             int numElements;
109             cout << " 请输入集合 A 中元素个数: ";
110             while (true)
111             {
112                 cin >> numElements;
113                 if (cin.fail() || numElements <= 0)
114                 {
115                     cin.clear();
116                     cin.ignore(numeric_limits<streamsize>::max(), '\n');
117                     cout << " 请输入正整数! " << endl;
118                 }
119                 else
120                 {
121                     break;
122                 }
123             }
124
125             vector<char> elements(numElements);
126             map<char, int> elementIndex;
```

```
127             cout << " 请输入集合 A 中的各个元素: ";
128             for (int i = 0; i < numElements; ++i)
129             {
130                 cin >> elements[i];
131                 elementIndex[elements[i]] = i;
132             }
133 
134             int numRelations;
135             cout << " 请输入二元关系 R 中元素个数: ";
136             while (true)
137             {
138                 cin >> numRelations;
139                 if (cin.fail() || numRelations <= 0)
140                 {
141                     cin.clear();
142                     cin.ignore(numeric_limits<streamsize>::max(), '\n');
143                     cout << " 请输入正整数! " << endl;
144                 }
145                 else
146                 {
147                     break;
148                 }
149             }
150 
151             Matrix<bool> relationshipMatrix(numElements, numElements);
152             cout << " 请依次输入二元关系 R 中的各个元素 (例如 a b 表示 <a, b>):" << endl;
153             for (int i = 0; i < numRelations; ++i)
154             {
155                 char a, b;
156                 cin >> a >> b;
157                 relationshipMatrix.set(elementIndex[a], elementIndex[b], true);
158             }
159 
160             Matrix<bool> transitiveClosureWarshall = transitiveWarshall(relationshipMatrix);
161 
162             cout << " 传递闭包 t(R) = { ";
163             bool first = true;
164             for (int i = 0; i < numElements; ++i)
165             {
166                 for (int j = 0; j < numElements; ++j)
167                 {
168                     if (transitiveClosureWarshall.get(i, j))
169                     {
170                         if (!first)
171                         {
172                             cout << ", ";
173                         }
174                         cout << "<" << elements[i] << ", " << elements[j] << ">";
175                         first = false;
176                     }
177                 }
178             }
179             cout << " }" << endl;
180 
181             if (!askToContinue())
182             {
183                 break;
184             }
185         }
```

```
186         if (choice == 2)
187         {
188 
189             Matrix<bool> relationshipMatrix = readMatrix();
190             cout << " 输入的关系矩阵为 : " << endl
191                  << relationshipMatrix << endl;
192 
193             cout << " 传递闭包 (Warshall 算法) : " << endl;
194             Matrix<bool> transitiveClosureWarshall = transitiveWarshall(relationshipMatrix);
195             cout << transitiveClosureWarshall << endl;
196 
197             if (!askToContinue())
198             {
199                 break;
200             }
201         }
202     }
203     return 0;
204 }
```

3 项目测试

3.1 正常测试

3.1.1 集合形式

```
请选择求解传递闭包的操作 ：(1.集合形式；2.关系矩阵形式)
1
请输入集合 A 中元素个数：5
请输入集合 A 中的各个元素： a b c d e
请输入二元关系 R 中元素个数：6
请依次输入二元关系 R 中的各个元素 (例如 a b 表示 <a, b>):
a b
b c
c c
c d
e a
e d
传递闭包 t(R) = { <a, b>, <a, c>, <a, d>, <b, c>, <b, d>, <c, c>, <c, d>, <e, a>,
 <e, b>, <e, c>, <e, d> }
是否继续运算(Y/N)?
```

3.1.2 矩阵形式

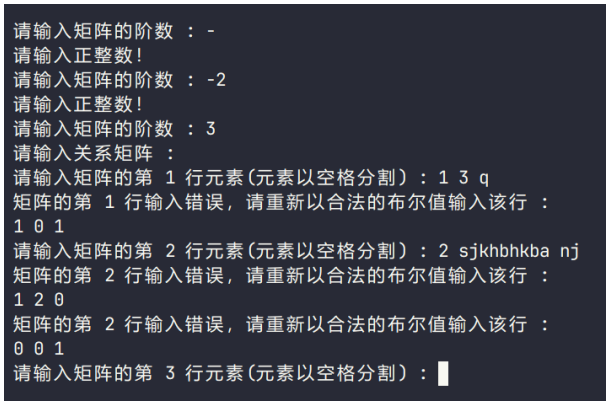
```
请选择求解传递闭包的操作 ：(1.集合形式；2.关系矩阵形式)
2
请输入矩阵的阶数 ：4
请输入关系矩阵 ：
请输入矩阵的第 1 行元素(元素以空格分割)：0 1 0 0
请输入矩阵的第 2 行元素(元素以空格分割)：0 0 0 1
请输入矩阵的第 3 行元素(元素以空格分割)：0 0 0 0
请输入矩阵的第 4 行元素(元素以空格分割)：1 0 1 0
输入的关系矩阵为 ：
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0

传递闭包 (Warshall 算法)：
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1

是否继续运算(Y/N)?
```

3.2 健壮性测试

程序具有良好的健壮性，对于输入的不合法数据，程序能够给出合理的提示，如下图所示：



4 心得体会

在性能提升方面，Warshall 算法完成了从 $O(n^4)$ 到 $O(n^3)$ 的显著优化，这不只是计算速率的重大突破，也构建了理论与实践的连接。算法避免了使用额外内存来暂存中间数据，这种在空间复杂度上的改进，反映了算法设计中的巧妙与简洁性。它通过较低的实现复杂度达到了高效的计算效果，是全方位优化的杰出代表。

真正将 Warshall 算法应用于现实问题的核心在于对其算法逻辑的深入掌握。该算法巧妙地将传递闭包的问题转化为构建可达性矩阵的问题，这是一种从细节到全局的思考模式。基于最简单的可达性逻辑——如果 a 能到达 b，b 能到达 c，那么 a 就能到达 c——通过这种逻辑步骤的连续迭代，最终能够涵盖所有潜在路径，形成完整的传递闭包视图。这种算法不仅简化了计算流程，而且深入揭示了问题的核心。