同济大学

离散数学说明文档

--利用Warshall算法求解传递闭包

姓　　名：胡正华

学　　号：2353741

任课教师：李　冰

1. 题目简介

根据用户给出的矩阵，利用Warshall算法(简便)求解传递闭包。

2. 解题思路

本实验的目的是通过 Warshall 算法计算一个有向图的传递闭包。传递闭包可以用来表示图中任意两个顶点之间是否存在路径，是图论中的重要概念。以下是解题的具体思路：

2.1 问题分析

给定一个有向图，用邻接矩阵 G 表示，其中 G[i][j]为 1 表示从顶点 i 到顶点 j 有一条边，否则为 0。

传递闭包要求，对于每对顶点 i 和 j，如果 i 能通过若干条边到达 j，那么在传递闭包矩阵中 G[i][j]应为 1。即，传递闭包矩阵可以表示图中任意两个顶点之间的可达性。

2.2算法设计

Warshall 算法是一种基于动态规划的方法，用来计算图的传递闭包。它通过不断迭代，逐步检查顶点之间的路径可达性，最终构造出传递闭包矩阵。

2.2.1 初始化邻接矩阵：

输入图的邻接矩阵 G，初始时 G[i][j]表示直接从顶 i 到顶点 j 否有边。

2.2.2 动态规划更新传递闭包矩阵：

算法通过一个三重循环逐步更新传递闭包矩阵：

第一个循环遍历每个中间顶点 k；

第二个和第三个循环分别遍历起点 i 和终点 ；

若通过顶点 k，从 i 到 j 是可达的（即 G[i][k]=1 且 G[k][j]=1），则将 G[i][j]更新为 1。

2.2.3 输出传递闭包矩阵：

更新完成后，传递闭包矩阵 G 中的每个元素 G[i][j]表示从顶点 i 到顶点 j 是否可达。

3. 代码实现

```cpp
#include <iostream>
#include <vector>
using namespace std;
void warshall(vector<vector<int>>& graph) {
    int n = graph.size();
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {

                if (graph[i][k] && graph[k][j])
{                   graph[i][j] = 1;
                }
            }
        }
    }
}

int main() {
    int n;
    cout << "请输入图的顶点数: ";
    cin >> n;
    vector<vector<int>> graph(n, vector<int>
(n));cout << "请输入邻接矩阵(用0和1表示): " << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> graph[i][j];
        }
    }
    warshall(graph);
    cout << "传递闭包矩阵为: " << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cout << graph[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

4. 心得体会

通过本次实验，我深入学习并实践了 Warshall 算法，从理论到实际实现，对动态规划在图论中的应用有了更深刻的理解。这不仅让我对图的传递闭包有了全面的认识，还体会到了算法设计和实现中的许多关键点。

4.1 动态规划思想的直观性

Warshall 算法通过逐步增加中间节点的方式，动态更新顶点之间的可达性。这种递进式的计算方法让我体会到动态规划的思想精髓——通过解决子问题来逐步构建全局问题的解。在传递闭包的计算中，每一步都会在前一步的基础上进一步扩展顶点间的可达性，最终得到完整的传递闭包矩阵。

4.2 矩阵操作的直观性与实现

邻接矩阵是图的基本表示方式之一，其结构简单直观，非常适合用于 Warshall 算法。在本实验中，通过邻接矩阵的三重嵌套循环更新传递闭包矩阵，我更深入地理解了矩阵操作在图算法中的重要性。同时，矩阵中每个元素的更新逻辑（通过某个中间节点的可达性）也非常清晰，让我对图的路径概念有了更深的理解。

4.3 时间与空间复杂度的权衡

Warshall 算法的时间复杂度为 O(n^3)，对于小规模的图问题可以很好地满足需求，但对于大规模图来说，效率可能会成为瓶颈。在实验中，我思考了可能的优化方向，例如使用稀疏矩阵存储图，或者结合并查集等数据结构处理传递闭包问题。这让我意识到，设计高效算法时，时间复杂度和空间复杂度的权衡是不可忽视的。