《离散数学》课程实验报告

6-Warshall 算法求解传递闭包

一、题目背景简介

定义：传递闭包 $t(R)=R\cup R^2\cup R^3\cup\cdots$

传统求传递闭包：考虑 n+1 个矩阵的序列 M^0，M^1，…，M^n，将矩阵 M^k 第 i 行第 j 列的元素记作 $M^k[i,j]$。对于 k=0，1，…，n，$M^k[i,j]=1$ 当且仅当在 R 的关系图中存在一条从 x_i 到 x_j 的路径，并且这条路径除端点外中间只经过$\{x_1, x_2, \cdots, x_k\}$中的顶点。不难证明 M^0 就是 R 的关系矩阵，而 M^n 就对应了 R 的传递闭包。

二、核心算法及依据

传统的求传递闭包的算法的时间复杂度是 $O(n^4)$，当 n 足够大时，程序跑起来比较慢。

有没有一种时间复杂度更低的算法呢？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 $O(n^3)$。

Warshall 算法的依据是：从 M^0 开始，顺序计算 M^1，M^2，…，直到 M_n 为止。从 $M^k[i,j]$ 计算 $M^{k+1}[i,j]$：i，j∈V 的方法为顶点集 $V_1=\{1, 2, \cdots, k\}$，$V_2=\{k+2, \cdots, n\}$，$V=V_1\cup\{k+1\}\cup V_2$, 那么 $M^{k+1}[i,j]=1$ 当且仅当存在从 i 到 j 中间只经过 $V1\cup\{k+1\}$中点的路径。

这些路径分为两类：第 1 类是只经过 V1 中点；第 2 类是经过 k+1 点。若存在第 1 类路径，则 $M^k[i,j]=1$；若存在第 2 类路径，则 $M^k[i,k+1]=1$ $M^k[k+1,j]=1$。

用伪代码表示 Warshall 算法为：

（1）置新矩阵 A=M；

（2）置 i=1；

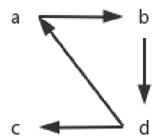
（3）对于所有 j，如果 A[j,i]=1，则对 k=1，2，…，n，$A[j,k]\vee A[i,k]$；

（4）i++；

（5）如果 i≤n，则跳转到步骤（3），否则停止。

三、解题思路

这里我想用一个例题来进行解题思路的讲解。



上图对应的 R={<a,b><b,d><d,a><d,c>}，根据二元关系易得 R 的关系矩阵：

$$M^0=\begin{bmatrix}0&1&0&0\\0&0&0&1\\0&0&0&0\\1&0&1&0\end{bmatrix}$$

先考察第 1 列元素，其中只有 $M^0[4,1]$等于 1，于是应将第 1 行与第 4 行对应元素做布尔加，结果仍记在第 4 行上，得到：

$$M^1=\begin{bmatrix}0&1&0&0\\0&0&0&1\\0&0&0&0\\1&1&1&0\end{bmatrix}$$

再考察第 2 列元素，有 $M^1[1,2]$和 $M^1[4,2]$等于 1，于是将第 2 行元素分别加到第 1 行和第 4 行，得到：

$$M^2=\begin{bmatrix}0&1&0&1\\0&0&0&1\\0&0&0&0\\1&1&1&1\end{bmatrix}$$

再考察第 3 列元素，仅有 $M^2[4,3]$等于 1，于是将第 3 行元素加到第 4 行，得到的 $M^3=M^2$。

最后考察第 4 列元素，有 $M^3[1,4]$、$M^3[2,4]$和 $M^3[4,4]$等于 1，按照要求分别进行布尔加，得到最终矩阵即为 R 的传递闭包 t(R)的关系矩阵：

$$M^4=\begin{bmatrix}1&1&1&1\\1&1&1&1\\0&0&0&0\\1&1&1&1\end{bmatrix}$$

根据上述操作可以总结出状态转移方程为 $M^k[i,j]=M^{k-1}[i,j]$ or（$M^{k-1}[i,k]$ and $M^{k-1}[k,j]$）。

于是，进行代码的框架构造。我打算用一个 get_matrix 函数来从键盘输入获得 R 的关系矩阵、用一个 output_matrix 函数对形参传递的二维数组进行元素的输出显示、用一个 Warshall 函数重点进行传递闭包的求解。

四、代码与运行结果

```
#include <iostream>
#include<cstring>
using namespace std;
#define R_num 4//宏定义 R 中元素个数为 4
bool get_matrix(int M[R_num][R_num])
{
    int i = 0, j = 0;
    for (i = 0; i < R_num; i++) {
        for (j = 0; j < R_num; j++) {
            cin >> M[i][j];
            if (M[i][j] != 0 && M[i][j] != 1)
                return false;//若输入非 0 或 1 的数，则输入错误，返回 false
        }
    }
    return true;
}
void output_matrix(int M[R_num][R_num])
{
    int i = 0, j = 0;
    for (i = 0; i < R_num; i++) {
        for (j = 0; j < R_num; j++) {
            cout << M[i][j] << ' ';
        }
        cout << endl;
    }
}
void Warshall(int R[][4]) {
    /*(1)i=1;
     (2)对所有 j 如果 a[j, i]=1，则对 k=0, 1, …, n-1, a[j, k]=a[j, k]∨a[i, k];
       (3)i 加 1;
       (4)如果 i<n，则转到步骤 2，否则停止*/
        int i = 0, j = 0, k = 0;
        for (k = 0; k < R_num; k++) {//进行 R_num-1 次 R 矩阵的变换，得到 R 的 R_num 次方
            for (i = 0; i < R_num; i++) {
                for (j = 0; j < R_num; j++) {
                    R[i][j] = R[i][j] || (R[i][k] && R[k][j]);
                }
            }
        }
      output_matrix(R);//输出 t(R)的关系矩阵
}
int main()
{
    int M[R_num][R_num] = { 0 };
PART:
    cout << "请输入一个" << R_num << "×" << R_num << "的关系矩阵 M" << endl;
    if (!get_matrix(M)) {
        cout << "输入错误，矩阵元素只能为 0 或 1！" << endl << "请重新输入：" << endl;
        goto PART;
    }
    cout << "利用 Warshall 算法得到的 t(R)关系矩阵为：" << endl;
    Warshall(M);
    return 0;
}
```

运行结果：

```
请输入一个4×4的关系矩阵M
0 1 0 0
0 0 0 1
0 0 0 0
1 0 1 0
利用Warshall算法得到的t(R)关系矩阵为：
1 1 1 1
1 1 1 1
0 0 0 0
1 1 1 1
```

五、体会与心得

由于先前在大作业中看到需要实现 Warshall 算法，于是在老师讲课的时候仔细听讲并揣摩了一番。不同于考察每一列后进行布尔加，老师讲到的另一种理解思路是：M^{k-1} 矩阵的第 k 行第 k 列的元素不动，对于其余的元素 $M^{k-1}[i,j]$（$i\neq k$ 且 $j\neq k$），若为 1，则依然保持为 1,；若为 0，则 $M^{k-1}[i,j]$ 由其在第 k 行第 k 列的元素的与值决定。我跟着老师对照例题过了一遍，自以为深刻明白了 Warshall 算法的要义。

然而本次在做大作业，根据网上相关文档的解释，我发现与先前的了解又有些无法联系。但是经过一番阅读之后，我发现了：无论是我课上弄明白的思路，还是今天在项目中讨论的思路，都围绕着一个要义，就是我在前文提到的总结状态转移方程：$M^{k}[i,j]=M^{k-1}[i,j]$ or（$M^{k-1}[i,k]$ and $M^{k-1}[k,j]$）。

基于对 Warshall 算法双重理解的基础再进行代码的编写就变得轻松许多。

当然，在网上浏览了一些其他人的代码之后，我发现我的代码还有可修改或完善的地方：可以将输入关系矩阵变为输入 R 中的所有二元关系；关系矩阵的输入可以不固定矩阵的大小，从而减少了输入的限制，增强代码的应用全面性。然而，我觉得如果要更改为输入 R 中的所有二元关系，那么输入的工作量就更大，也更容易从输入端产生错误，所以还是 pass 了这种想法。