



《离散数学》课程实验报告 6

Warshall 算法求传递闭包

院 系：交通运输工程学院

姓 名：唐宇瀚

学 号：1950152

授课教师：唐剑锋

2022 年 4 月



目录

1. 实验内容.....	2 -
2. 相关概念.....	2 -
3. 解题思路.....	2 -
4. Warshall 算法的具体流程	3 -
5. 数据结构.....	4 -
6. 运行结果.....	5 -
7. 心得体会.....	7 -
附：求解代码.....	7 -



1. 实验内容

本实验课程训练学生掌握用 Warshall 算法求传递闭包的算法，进一步能用它们来解决实际问题。通过实验提高学生编写实验报告、总结实验结果的能力；使学生具备程序设计的思想，能够独立完成简单的算法设计和分析。

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。

需要强调的是，在唐老师给出文档以前，我已经自己实现了根据关系矩阵，用 Warshall 算法求传递闭包。在此基础上，再给出已知集合中元素，求传递闭包的方法。故本报告最终会给出**两种代码实现方式**：

- (1) 输入集合中元素及其关系求传递闭包。（唐老师上课要求的方式）
- (2) 输入关系矩阵，求传递闭包。（自己额外实现的方式）

2. 相关概念

传递：如果对于任意的 x, y, z ，每当 xRy 和 yRz 时就有 xRz ，称关系 R 是传递的。

传递闭包：即在数学中，在集合 X 上的二元关系 R 的传递闭包是包含 R 的 X 上的最小的传递关系。

3. 解题思路

传递闭包的普通求解方法是将原矩阵与它的二次方、三次方矩阵一直到 n 次方矩阵做逻辑加。即程序申请了 n 个辅助数组来帮助实现这个函数。最外层的循环进行了 n 次，即找到从 i 通过长度为 0 到 $n-1$ 的路径可以到达的终点，才能将所有的可能路径都检验到。每一次的循环中都检查矩阵中只要 i 和 j 中有路，就扫描



j开头的那一行，即j的出边行，j和k中有路，那么i和k就必须连起来。时间复杂度为 $O(n^4)$ 。

而Walshall 算法的时间复杂度为 $O(n^3)$ 。在Warshall 算法中，若两个点确定，即可以判断，从一个点起到一个点为止，中间顶点序号不大于k，是否有路径，一旦发现有，就将这两个点直接连起来。此处用联结词或和与的思想实现。在代码中，当 $s[i][k]$ 和 $s[k][j]$ 同时不为0，同时为1时， $s[i][j]$ 的值就会被更新为1，如果他们有一个为0，但是上一步中 $s[i][j]$ 为1， $s[i][j]$ 仍为1。

4. Warshall 算法的具体流程

Warshall 算法的步骤为

- (1)置新矩阵 $A=M$;
- (2) $i=1$;
- (3)对所有 j 如果 $A[j, i]=1$ ，则对 $k=1, 2, \dots, n$, $A[j, k]=A[j, k] \vee A[i, k]$;
- (4)i 加 1; (i 是行, j 是列)
- (5)如果 $i \leq n$ ，则转到步骤 3)，否则停止。

例 已知关系R的关系矩阵为

$$M = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

利用 Warshall 算法求 $t(R)$ 的关系矩阵。

解

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

$i=1$ 时，第一列有 $A[3, 1]=1$ 、 $A[4, 1]=1$ ，将第三行、第四行分别和第一行各对应元素进行逻辑加，仍然分别记为第三行、第四行得

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

可举例理解这个算法的过程：



$i=2$ 时，第二列有 $A[1, 2]=1$ 、 $A[3, 2]=1$ 、 $A[4, 2]=1$ ，将第一行、第三行、第四行分别和第二行各对应元素进行逻辑加，仍然分别记为第一行、第三行、第四行得：

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$i=3$ 时，第三列有 $A[1, 3]=1$ 、 $A[2, 3]=1$ 、 $A[3, 3]=1$ 、 $A[4, 3]=1$ ，将第一行、第二行、第三行、第四行分别和第三行各对应元素进行逻辑加，仍然分别记为第一行、第二行、第三行、第四行得：

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$i=4$ 时，第四列有 $A[1, 4]=1$ 、 $A[2, 4]=1$ 、 $A[3, 4]=1$ 、 $A[4, 4]=1$ ，将第一行、第二行、第三行、第四行分别和第四行各对应元素进行逻辑加，仍然分别记为第一行、第二行、第三行、第四行得：

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

A 即为 $t(R)$ 的关系矩阵。

5. 数据结构

Warshall 算法仅仅才使用了一个 n 阶的数组保存关系矩阵。不像传统的算法需要另开两个同样的数组暂存结果，故空间复杂度和时间复杂度都比传统算法要小很多。时间复杂度更是因为算法本身从 $O(n^4)$ 降到了 $O(n^3)$ 。



6. 运行结果

(1) 输入集合中元素及其关系求传递闭包。(唐老师上课要求的方式)

程序编译后无警告，消除了原有程序中的警告。

```
#include <iostream>

using namespace std;

//二元关系
typedef struct
{
    char a;
    char b;
}BR;

int n, m; //n表示A集合中元素个数，m表示二元关系R中元素个数

//创建集合A并完成初始化
void init_aggregation(char* &A)
{
    tt:cout << "请输入A集合中的元素个数(正整数)，按回车键输入下一项：" << endl;
    cin >> n;
    if (cin.fail()) //若输入错误
        cout << "输入错误！" << endl;
    cin.clear(); //清除cin错误的记录
    cin.ignore(2048, '\n'); //释放内存
    goto tt;
}

A = new char[n];
cout << "请依次输入集合A中的：" << endl;
cout << n; //n是集合A中的元素个数
cout << "个元素(形如：a b c d .....这样的格式)，按回车键输入下一项：" << endl;
```

正确输入示例：

```
请输入A集合中的元素个数(正整数)，按回车键输入下一项：
5
请依次输入集合A中的5个元素(形如：a b c d .....这样的格式)，按回车键输入下一项：
a b c d e
请输入二元关系R中的元素个数(正整数)，按回车键输入下一项：
6
请依次输入R中的6个元素，一行是一个元素
(形如：
a b
b c
c d
.....
这样的格式)，按回车键输入下一项：
a b
b c
c c
c d
e a
e d
求得R的传递闭包为：
t(R) = { <a, b>, <a, c>, <a, d>, <b, c>, <b, d>, <c, c>, <c, d>, <e, a>, <e, b>, <e, c>, <e, d> } 请按任意键继续. . .
```

原程序中 R 的传递闭包作为一个集合的最后一个有序对的后面还有一个逗号，已经通过程序把它去掉。



程序已经变得更加健壮，输入错误示例后会有提示，并可继续输入，直到输入正确才会开始运算。

错误输入示例：

```
请输入A集合中的元素个数(正整数)，按回车键输入下一项：
b
输入错误！
请输入A集合中的元素个数(正整数)，按回车键输入下一项：
c
输入错误！
请输入A集合中的元素个数(正整数)，按回车键输入下一项：
ac
输入错误！
请输入A集合中的元素个数(正整数)，按回车键输入下一项：
5
请依次输入集合A中的5个元素(形如：a b c d .....这样的格式)，按回车键输入下一项：
a b c d e
请输入二元关系R中的元素个数(正整数)，按回车键输入下一项：
6
请依次输入R中的6个元素，一行是一个元素
(形如：
a b
b c
c d
.....
这样的格式)，按回车键输入下一项：
a b
b c
c c
c d
e a
e d
求得R的传递闭包为：
t(R) = { <a, b>, <a, c>, <a, d>, <b, c>, <b, d>, <c, c>, <c, d>, <e, a>, <e, b>, <e, c>, <e, d> } 请按任意键继续. . .
```

原程序在输入错误之后会自动退回，若输入正确仍可继续运算，程序不会崩溃。

至此，所有的三个问题都已解决。

(2) 输入关系矩阵，求传递闭包。(自己额外实现的方式)

正确输入示例：

```
请输入矩阵的行数:4 4
请输入矩阵的列数:请输入关系矩阵:

请输入矩阵的第0行元素(元素以空格分隔):0 1 0 0
请输入矩阵的第1行元素(元素以空格分隔):0 0 0 1
请输入矩阵的第2行元素(元素以空格分隔):0 0 0 0
请输入矩阵的第3行元素(元素以空格分隔):1 0 1 0
所求传递闭包为:
1111
1111
0000
1111
请按任意键继续. . .
```



错误输入示例:

```
请输入矩阵的行数:3
请输入矩阵的列数:3
请输入关系矩阵:

请输入矩阵的第0行元素(元素以空格分隔):1 2 1
输入错误
请按任意键继续. . .
```

7. 心得体会

我体会到了 Warshall 算法的妙处: 前一次传递关系合并产生的结果直接保存在原有数组上, 在这个算法中, 认为任意两个顶点都是两步到达的, 可能是 $1 \sim n$ 个顶点中的任一个是中间顶点, 故而循环有 n 次。这个算法的过程很像是我们人自己用关系图法寻找关系闭包的过程, 比较快速。另外, 该算法也检查了输入关系矩阵的合法性, 确保输入数组为布尔矩阵, 否则退出程序, 充分考虑了出错情况。

附: 求解代码

(1) 输入集合中元素及其关系求传递闭包。(唐老师上课要求的方式)

```
1. #include <iostream>
2.
3. using namespace std;
4.
5. //二元关系
6. typedef struct
7. {
8.     char a;
9.     char b;
10. }BR;
```




```
11.
12. int n, m; //n 表示 A 集合中元素个数, m 表示二元关系 R 中元素个数
13.
14. //创建集合 A 并完成初始化
15. void init_aggregation(char* &A)
16. {
17.     tt:cout << "请输入 A 集合中的元素个数(正整数), 按回车键输入下一项:
18.     " << endl;
19.     cin >> n;
20.     if(cin.fail()){//若输入错误
21.         cout<<"输入错误! "<<endl;
22.         cin.clear();//清除 cin 错误的记录
23.         cin.ignore(2048, '\n');//释放内存
24.         goto tt;
25.     }
26.     A = new char [n];
27.     cout << "请依次输入集合 A 中的";
28.     cout << n; //n 是集合 A 中的元素个数
29.     cout << "个元素(形如: a b c d .....这样的格式), 按回车键输入下一项:
30.     " << endl;
31.     for(int i = 0; i < n; i++) {
32.         cin >> A[i];
33.         getchar();
34.     }
35. }
36.
37. //创建 A 集合的二元关系 R 的集合并完成初始化
38. void init_BinaryRelation(BR* &R)
39. {
40.     cout << "请输入二元关系 R 中的元素个数(正整数), 按回车键输入下一项:
41.     " << endl;
42.     cin >> m;
43.     R = new BR [n];
44.     cout << "请依次输入 R 中的";
45.     cout << m; //m 是 R 中的元素个数
46.     cout << "个元素, 一行是一个元素" << endl;
47.     cout << "(形如: " <<endl << "a b" << endl;
48.     cout << "b c" << endl;
49.     cout << "c d" << endl;
50.     cout << "....." << endl;
51.     cout << "这样的格式), 按回车键输入下一项: " << endl;
```



```
52.     for(int i = 0; i < m; i++) {
53.         cin >> R[i].a;
54.         getchar();
55.         cin >> R[i].b;
56.     }
57. }
58.
59. int fun(char ch, char* &A) //辅助将二元关系 R 用关系矩阵表示
60. {
61.     for(int i = 0; i < n; i++) {
62.         if(ch == A[i]) {
63.             return i;
64.         }
65.     }
66.     return -1;
67. }
68.
69. //核心算法
70. void Warshall(char* &A, BR* &R, bool** &tR)
71. {
72.     int i, j, k;
73.     int x, y;
74.
75.     //将二元关系 R 用关系矩阵表示
76.     for(i = 0; i < m; i++) {
77.         x = fun(R[i].a, A);
78.         y = fun(R[i].b, A);
79.         tR[x][y] = 1;
80.     }
81.
82.     //传递包闭计算过程
83.     for(i = 0; i < n; i++) { //检索列
84.         for(j = 0; j < n; j++) { //检索行
85.             if(tR[j][i] == 1) {
86.                 for(k = 0; k < n; k++) {
87.                     tR[j][k] = (bool)( tR[j][k] + tR[i][k]); //Warshall 算法的
                        核心语句
88.                 }
89.             }
90.         }
91.     }
92. }
93.
94. //将传递包闭 t(R)的关系矩阵表示转为集合表示
```



```
95. void translation_output(char* &A, bool** &tR)
96. {
97.     cout << "求得 R 的传递闭包为: " << endl;
98.     cout << "t(R) = { ";
99.     int i, j;
100.    for(i = 0; i < n; i++) {
101.        for(j = 0; j < n; j++) {
102.            if(tR[i][j] == 1) {
103.                cout << "<" << A[i] << "," << A[j] << ">,";
104.            }
105.        }
106.    }
107.    printf("\b"); //使用退格符删去最后一个输出的逗号
108.
109.    system("pause");
110. }
111.
112. int main()
113. {
114.     char *A;
115.     init_aggregation(A); //初始化 A 集合
116.
117.     BR* R;
118.     init_BinaryRelation(R); //初始化二元关系
119.
120.     bool** tR; //传递闭包矩阵
121.
122.     //动态开辟 bool 类型的二维数组
123.     tR = new bool* [n];
124.     for(int i = 0; i < n; i++) {
125.         tR[i] = new bool [n*n];
126.     }
127.
128.     //初始化 (将 tR[i][j] 全赋值 0)
129.     for(int i = 0; i < n; i++) {
130.         for(int j = 0; j < n; j++)
131.         {
132.             tR[i][j] = 0;
133.         }
134.     }
135.
136.     Warshall(A, R, tR); //调用 Warshall 算法函数
137.
138.     translation_output(A, tR); //转译输出
```



```
139.  
140.     return 0;  
141. }
```

(2) 输入关系矩阵，求传递闭包。（自己额外实现的方式）

```
1. #include<iostream>  
2. #include<cstdio>  
3. using namespace std;  
4. int n, m;  
5. void output(int s[][100]){//输出所求关系矩阵  
6.  
7.     cout << "所求传递闭包为:" << endl;  
8.     for (int i = 0; i < n; i++)  
9.     {  
10.         for (int j = 0; j < m; j++)  
11.             cout << s[i][j];  
12.         cout << endl;  
13.     }  
14. }  
15. int main(){  
16.     int s[100][100];  
17.     cout << "请输入矩阵的行数:";  
18.     cin >> n;  
19.     cout << "请输入矩阵的列数:";  
20.     cin >> m;  
21.     cout << "请输入关系矩阵:" << endl;  
22.     for (int i = 0; i < n; i++){  
23.         cout << endl;
```



```
24.         cout << "请输入矩阵的第" << i << "行元素(元素以空格分隔).:";
25.         for (int j = 0; j < m; j++){
26.             cin >> s[i][j]; //输入矩阵信息, 用 s 数组储存
27.             if(s[i][j]!=0 && s[i][j]!=1){
28.                 cout<<"输入错误"<<endl;
29.                 system("pause");
30.                 return 0;
31.             }
32.         }
33.     }
34.
35.     for (int k = 0; k < n; k++) { //Warshall 算法求传递闭包
36.         for (int i = 0; i < n; i++) {
37.             for (int j = 0; j < m; j++) {
38.                 s[i][j] = (s[i][j] || s[i][k] & s[k][j]);
39.             }
40.         }
41.     }
42.     output(s); //输出 s
43.     system("pause");
44.     return 0;
45.
46. }
```