一、前言

在离散数学二元关系一章的某一节中，会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)，程序跑起来比较慢。

有没有一种时间复杂度更低的算法呢？有的。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)。下文就论述算法伪代码描述，例题分析，(C++)代码实现和数据测试四个方面来详述 Warshall 算法。

二、算法伪代码描述

我们借用如下的算法伪代码描述：

(1) 置新矩阵 **M : = AR** (AR 表示集合 A 的二元关系 R 的集合)

(2) 置 **j ： = 1**

(3) 对所有 **i** ，如果 **M [i , j] = 1**，则对 **k = 1，2，... ，n**，置

M [i , k] : = M [i , k] + M [j , k]

(4) j ： = j + 1

(5) 如果 **j <= n**，则转到步骤（3），否则停止

上述的算法伪代码描述可能难于理解，我们下面进行例题分析。

三、例题分析

为了把问题解释得更清楚，我们借用如下的例题：

(5)如果 $j \leq n$，则转到步骤(3)，否则停止。

例 2-33 $A=\{a_1,a_2,a_3,a_4,a_5\}$，$R=\{(a_1,a_2),(a_2,a_3),(a_3,a_3),(a_3,a_4),(a_5,a_1),(a_5,a_4)\}$，求 R 的传递闭包。

解 先写出 R 的关系矩阵

$$
\begin{array}{c}
 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5
\end{array}
M:=A_R=
\begin{array}{c}
\begin{array}{ccccc} a_1 & a_2 & a_3 & a_4 & a_5 \end{array} \\
\begin{bmatrix}
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0
\end{bmatrix}
\end{array}
$$

先考察第 1 列，其中仅有 $m_{51}=1$，于是应将第 1 行与第 5 行对应元素作布尔加，结果仍记在第 5 行上，也即将第 1 行元素加到第 5 行上去，得

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

考察第 2 列元素，现有 $m_{12}=1$ 和 $m_{52}=1$，于是应将第 2 行元素分别加到第 1 行和第 5 行上去，得

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

考察第 3 列元素，现有 $m_{13}=1, m_{23}=1, m_{33}=1, m_{53}=1$，于是应将第 3 行元素分别加到第 1 行，第 2 行，第 3 行，第 5 行上去，得

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

考察第 4 列元素，现有 $m_{14}=1, m_{24}=1, m_{34}=1, m_{54}=1$，于是应将第 4 行元素加到第 1 行，第 2 行，第 3 行，第 5 行上去，得

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

在第 5 列中没有元素为 1，所以上述矩阵即为 R 的传递闭包 $t(R)$ 的关系矩阵。

四、源代码(C++)实现

Warshall 算法的例题分析完成后，我们就可以来写源代码了，我们以 C++代码为例，可运行的代码如下(以下代码在 MS Visual Studio 2019 下调试、运行成功)：

```
#include <iostream>
using namespace std;

//用结构体来表示二元关系
typedef struct
{
    char a;
    char b;
}BR;

int n, m;      //n 表示集合 A 中的元素个数，m 表示二元关系 R 中的元素个数

//创建集合 A 并初始化
void init_aggregation(char*& A)
{
    cout << "请输入集合 A 中的元素个数(正整数)，按回车键输入下一项：" << endl;
    cin >> n;
    A = new char[n];
    cout << "请依次输入集合 A 中的";
    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;
    for (int i = 0; i < n; i++)
    {
        cin >> A[i];
    }
}

//创建集合 A 的二元关系 R 的集合并初始化
void init_BinaryRelation(BR*& R)
{
    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：" << endl;
    cin >> m;
    R = new BR[n];
    cout << "请依次输入 R 中的";
    cout << m;   //m 是 R 中的元素个数
    cout << "个元素，一行是一个元素" << endl;
    cout << "(形如：" <<endl << "a b" << endl;
    cout << "b c" << endl;
    cout << "c d" << endl;
    cout << "......" << endl;
    cout << "这样的格式)，按回车键输入下一项：" << endl;
```

```
	for(int i = 0; i < m; i++)
	{
		cin >> R[i].a;
		cin >> R[i].b;
	}
}

int fun(char ch, char*& A)
{
	for (int i = 0; i < n; i++)
	{
		if (ch == A[i])
		{
			return i;
		}
	}
	return -1;
}

//Warshall 算法的核心部分
void Warshall(char*& A, BR*& R, bool**& tR)
{
	int i, j, k;
	int x, y;

	//用关系矩阵表示二元关系 R
	for (i = 0; i < m; i++)
	{
		x = fun(R[i].a, A);
		y = fun(R[i].b, A);
		tR[x][y] = 1;
	}

	//计算传递闭包的过程
	for(i = 0; i < n; i++)
	{ //检索列
		for(j = 0; j < n; j++)
		{ //检索行
			if (tR[j][i] == 1)
			{
				for(k = 0; k < n; k++)
				{
					tR[j][k] = tR[j][k] + tR[i][k];
				}
```

```cpp
                }
            }
        }
    }

//将传递闭包 t(R)的关系矩阵表示转化为集合表示
void translation_output(char*& A, bool**& tR)
{
    cout << endl;
    cout << "R 的传递闭包(集合形式)为：" << endl;
    cout << "t(R) = {";
    int i, j;
    for(i = 0; i < n; i++)
    {
        for(j = 0; j < n; j++)
        {
            if(tR[i][j] == 1)
            {
                cout << "<" << A[i] << "," << A[j] << ">" << ",";
            }
        }
    }
    cout << "}" << endl;
}

//主函数
int main()
{
    char* A;
    init_aggregation(A); //初始化集合 A

    BR* R;
    init_BinaryRelation(R); //初始化二元关系

    bool** tR; //传递闭包矩阵

     //动态开辟 bool 类型的二维数组
    tR = new bool* [n];
    for(int i = 0; i < n; i++)
    {
        tR[i] = new bool[n * n];
    }

    //初始化二维数组(全部赋值为 0)
```

```
    for(int i = 0; i < n; i++)
    {
        for(int j = 0; j < n; j++)
        {
            tR[i][j] = 0;
        }
    }

    Warshall(A, R, tR);//调用 Warshall 算法函数

    translation_output(A, tR); //将传递闭包 t(R)的关系矩阵表示转化为集合表示

    return 0;
}
```

五、数据测试

数据测试见下面的截图：

```
请输入集合A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

R的传递闭包(集合形式)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>,}
```

至此，我们基本完成了实现 Warshall 算法的工作。

最后，上述程序还有三个问题：

(1)程序运行有一个警告：从"R"中读取的数据无效：可读大小为"n*2"个字节，但可能读取了"4"个字节。

(2)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，这不符合传递闭包的表现形式。

(3)程序还不够健壮，不按规则输入还有程序崩溃的现象。

请同学们至少解决问题(1)和(2)。谢谢！

同济大学软件学院 唐剑锋