****

**Tongji University**

**离散数学实验报告**

**006-WarShall算法求传递闭包**

专 业： 软 件 工 程

指导教师： 唐 剑 锋

学 号： 2 1 5 3 0 6 1

姓 名： 谢 嘉 麒

1. **题目概览**
   1. **实验背景**

在离散数学中求关系的闭包计算是十分重要的操作，其中自反闭包及对称闭包可以通过r(R)=R∪R0和s(R)=R∪R−1两个公式简便求出，但求传递闭包的公式t(R)=R∪R2∪R3∪…在进行手动运算和程序实现时都较为繁琐复杂。这一点在程序中主要体现于较多的时间开销和空间开销。

传统的求传递闭包的算法的时间复杂度是 O(n4)，程序的时间开销极高。为了解决这一问题，提高程序的运行效率，Warshall 在 1962 年提出了一种求传递闭包的复杂度更低的算法，即 Warshall 算法。Warshall 算法的时间复杂度从传统的求传递闭包的算法的 O(n4)降到了 O(n3)。很大程度上减小了时间开销。

* 1. **实验要求**

本次试验要求利用WarShall实现求关系的传递闭包。

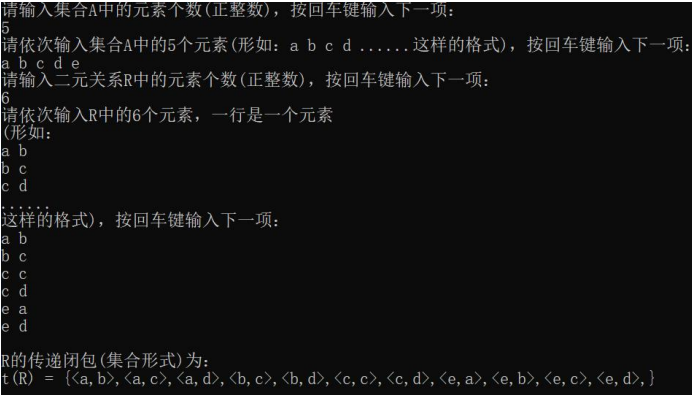
* + 1. **输入要求**

实验要求客户分别输入集合A中元素个数，关系R中元素个数以及各个关系的具体形式。输入时程序会给出详细输入形式。

* + 1. **输出要求**

将关系的传递闭包以集合形式输出，具体为r(R) = {< , >,< , >…}

* + 1. **程序样例**

****

* 1. **实验思路与分析**

根据实验背景可知，本次实验重点在于完成WarShall算法，并对键入的关系集合进行求传递闭包，最后打印出来。考虑到题目强调对二元关系进行运算，因此可以定义结构体整体存储二元关系的两个元素，并且可以通过使用动态内存申请的方法进行数据存储。

* + 1. **完善项目功能**

考虑到用户的使用体验和需求优化，故需要添加输入错误处理，保证用户输入正确。具体可以用输入输出流结合while循环实现，方便用户使用。

* + 1. **保证执行效率**

保证用户体验的重要环节之一就是保证代码的执行效率。本次项目为了保证程序的运行效率以及避免不必要的空间浪费，摒弃了传统的求传递闭包方法，选择使用WarShall算法，减小时间复杂度和空间复杂度。

* + 1. **注重代码健壮性**

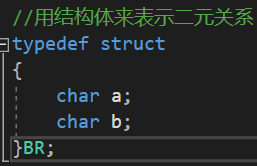
程序需要在用户输入非法字符或者不合理数据的时候进行报错提示，确保正确性，避免程序崩溃，体现代码健壮性。

* + 1. **代码复用的处理**

通过增加可复用的函数来提高代码的复用能力，使整个程序简洁美观，减少重复率，降低风险，便于修改。

1. **数据结构的设计与使用**
   1. **二元关系结构体的定义**

二元关系由两个元素构成，考虑到调用的便利性和存储的整体性，故选用结构体的方法整体处理和存储二元关系。具体结构体定义如下：



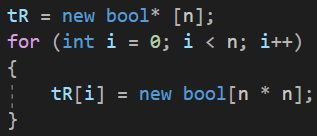
通过在结构体中定义两个字符型变量来存储二元关系的两个元素，将结构体整体typedef为BR，方便后续的使用。

* 1. **动态申请内存处理数据**

考虑到优化存储和防止占用过多用户内存的问题，本次实验广泛使用动态内存申请的方法为实验变量申请内存。包括：

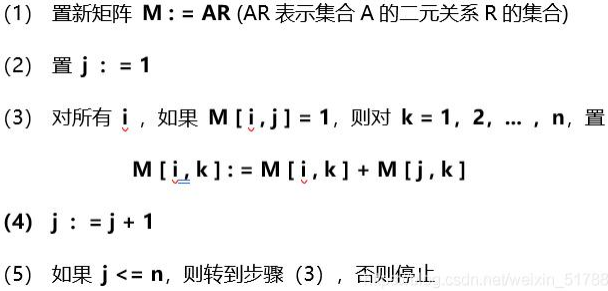
1. 存储集合A中元素的字符型指针char\* A；
2. 存储二元关系的结构体指针BR\* R；
3. 存储传递闭包矩阵的布尔指针型指针bool\*\* tR

具体动态申请操作分别为：

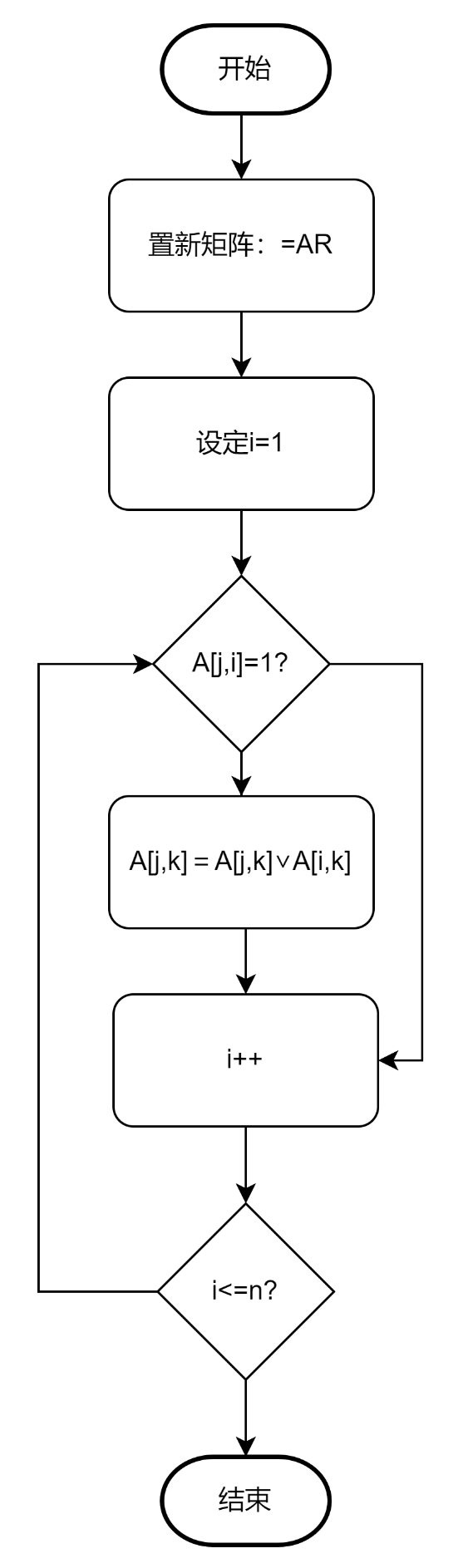
1. 
2. 
3. 
4. **核心算法的设计与使用**
   1. **WarShall算法的设计与使用**

传递闭包的一种有效算法—Warshall 算法，这种算法也便于计算机实现，其优势主要体现在算法实现契合计算机语言习惯，同时算法时间复杂度比较传统方法大大降低。具体算法实现思路如下：

1. 置新矩阵 A＝M；
2. 设定整型变量i＝1；
3. 对所有整型变量j 如果 A[j，i]＝1，则对 k＝1，2，…，n，A[j，k]＝A[j，k]∨A[i，k]（逻辑加）；
4. 令i + 1；（i 是行，j 是列）
5. 如果 i≤n，则转到步骤 （3），否则停止。
   * 1. **WarShall算法的伪代码实现**



* + 1. **WarShall算法流程图解**



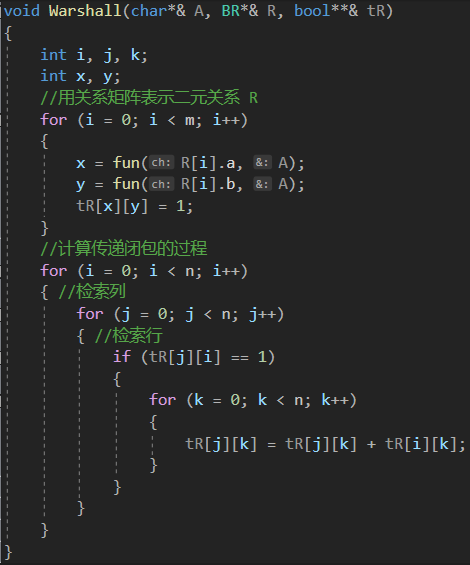
否

是

否

是

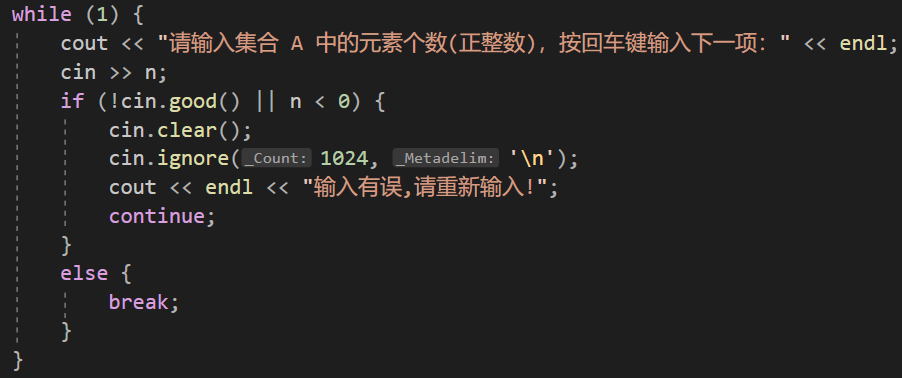
* + 1. **WarShall算法代码实现**

****

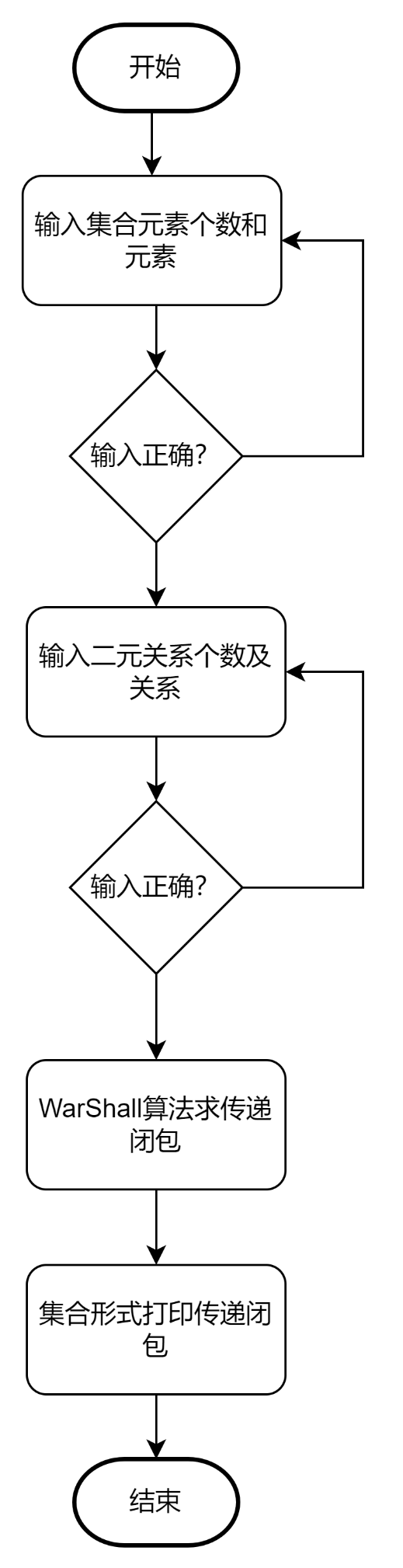
* 1. **用户输入错误处理和避免程序崩溃的设计**

本次试验需要对于用户输入是否正确进行判断，其中涉及到C++语言输入流特点以及输入缓冲区的处理，尝试后笔者选择运用while循环结合cin.clear()和cin.ignore()两个函数进行处理。将不为0/1的输入判误，清理缓冲区后提示客户重新输入，使程序更加健壮。

* + 1. **算法代码实现**



1. **程序整体流程图解**

****

否

是

否

是

1. **心得体会**
   1. **掌握并理解Warshall算法**

本次实验中，通过学习并实现Warshall算法，笔者进一步体会到算法的原理思想和实现过程，并且意识到算法优化中提高程序运行效率，降低时间开销和空间开销等的重要性。

* 1. **对于warning的处理**

本次实验原始代码在Visual Studio 2022中编译运行会导致warning“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”。这一问题是由于内存访问可能访问到非法内存而导致的，通过修改动态内存申请语句，将内存扩大1，可以解决此问题。

* 1. **对于输出格式的处理**

本次实验原始代码在Visual Studio 2022中编译运行时输出会有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的问题，这不符合传递 闭包的表现形式

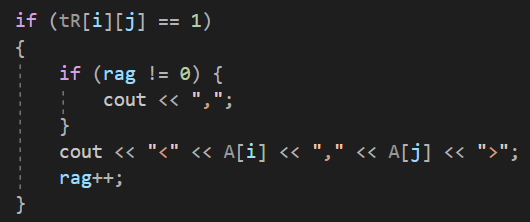
通过添加标识计数变量rag可以解决此问题：



在输出循环前设定rag变量，作为计数指针指示已输出元素的个数。逗号处理的本质是在逗号输出前添加限制条件，将逗号移至二元关系之前输出，并且第一个二元关系前不输出。

则只需要用rag变量记录，rag=0时代表此处为第一个二元关系，其前方不输出逗号；当rag大于0时，表示此时二元关系前可以输出逗号。

具体代码处理如下：



1. **程序测试**

测试数据：5

abcde

6

a b

b c

c c

c d

e a

e d

结果：t(R） = {<a,b>,<a,c>,<a,d>,<b,c>，<b,d>,<c,c>,<c,d>,<e,a>,<e,b>,<e,c>,<e,d>}

测试截图：

