****

**离散数学课程实验报告**

**Warshall算法求传递闭包**

2351041

刘浩田

完成日期:2024.11.3

1. **索引**

### 实验目的

### 实验环境

### 实验要求

### 实验原理

### 实验过程

### 实验结果

### 实验小结

1. **实验目的**

本实验旨在通过实践操作加深对离散数学中关系属性理论的理解，特别是传递性和传递闭包的概念。通过实现 Warshall 算法，我们希望达成以下几个核心目标：

首先，实验的主要目的是帮助学生深入理解传递闭包的概念。传递闭包是关系理论中的重要组成部分，其定义是：如果关系 R 中的元素 a 与 b 相关联，且 b 与 c 相关联，那么 a 与 c 也应当相关联。通过实现 Warshall 算法，学生将能够在实际编程中识别和应用这一理论，增强对传递性在各种数学及计算问题中的重要性的认识。

其次，实验将帮助学生掌握 Warshall 算法的原理和实际应用。Warshall 算法是一个经典的计算传递闭包的算法，基于动态规划的思想，能够有效地处理有向图的传递关系。通过对该算法的研究，学生不仅能够理解其基本逻辑与结构，还能在具体编程实践中体会如何将其应用于计算给定关系的传递闭包，从而提高解决实际问题的能力。

此外，实验还旨在提升学生的编程技能和逻辑思维能力。通过 C++ 编程实现 Warshall 算法，学生将面对逻辑运算、循环控制、条件判断等多种编程技术的挑战。这一过程将促使学生在解决复杂问题时，培养更强的逻辑思维和分析能力，使其能够在今后的学习和工作中更有效地应用所学知识。

同时，实验强调数据结构在算法实现中的重要性。学生将学习如何使用矩阵这一基本的数据结构来表示和处理关系。在实现过程中，学生需要考虑如何选择合适的数据结构以优化算法性能，理解数据结构选择对算法效率的影响，从而提升其在实际编程中的综合能力。

通过本实验，学生不仅能够巩固离散数学的基本概念，还能在编程实践中提升解决问题的能力。这将为日后在更复杂的计算问题中应用这些知识打下坚实的基础。

最后，实验还旨在提高学生对算法的验证和测试能力。通过一系列的测试案例，学生将验证所实现算法的正确性与效率。这一过程将使学生意识到在软件开发中，严格的测试和验证是不可或缺的环节，从而培养其良好的工程意识和质量意识。

综上所述，本实验的目的不仅在于实现 Warshall 算法计算传递闭包的功能，更在于通过这一过程提升学生的理论理解、编程能力、逻辑思维、数据结构应用能力，以及对算法测试和验证的重视。这些目标的实现将为学生未来的学术和职业发展奠定坚实的基础。

1. **实验环境**

## 系统：Windows11

## 编译器：VisualStudio2022x86

## 语言：C++

1. **实验要求**

本实验的核心要求是理解和应用关系的自反、对称和传递闭包的计算方法，通过编程实践来解决具体的数学问题。首先，需要掌握这三种关系属性的理论基础。其次，应能够使用矩阵作为数据结构来表示关系，这是将抽象数学概念具体化的关键步骤。

在实验中，需要编写一个C++程序，该程序能够实现关系的自反闭包、对称闭包和传递闭包的计算过程。程序的核心功能是遍历所有可能的关系组合，这是一个涉及逻辑运算和条件判断的复杂过程。程序必须能够准确地评估每一种组合，输出正确结果。

此外，实验还要求设计的程序输出结果必须清晰明确，明确指出计算结果。这意味着程序不仅要正确执行数学计算，还要具有良好的用户交互界面，能够以易于理解的方式呈现计算结果。

总之，本次实验要求不仅要在理论上掌握关系的自反、对称和传递闭包的计算，还要在实践中应用这些知识，通过编程来解决实际问题。这不仅考验了的编程能力，还锻炼了数学思维和问题解决技巧。通过这样的实验，能够将理论与实践相结合，从而更深入地理解和掌握关系的自反、对称和传递闭包的计算知识。

1. **实验原理**

本实验的核心目的是通过实现 Warshall 算法来计算传递闭包。为了达到这一目的，以下几个关键概念和原理是实验的基础。

## 5.1 关系的传递性

关系的传递性是离散数学中关系理论的重要属性之一。一个关系 R 是传递的，如果对于任意三个元素 a、b 和 c，只要 (a, b) 和 (b, c) 都在 R 中，那么 (a, c) 也必须在 R 中。传递性在许多数学和计算领域中都非常重要，尤其是在定义序关系和等价关系时。例如，在排序问题中，如果我们知道 A 大于 B，B 大于 C，那么根据传递性，我们可以得出 A 也大于 C。

在计算关系的传递闭包时，目标是确保如果存在一个通过中间元素可以到达的路径，则直接路径也应该存在。这一过程通常涉及复杂的计算，特别是在处理更大规模的关系时，如何高效地实现传递闭包的计算就显得尤为重要。

## 5.2 Warshall 算法

Warshall 算法由 Stephen Warshall 在 1962 年提出，是一种高效的算法，用于计算有向图的传递闭包。它是理解和实现计算机科学中的关系闭包概念的关键方法，尤其在数据分析、图论和算法设计领域具有重要应用。

Warshall 算法的基本思想是使用动态规划的方法，通过逐步更新关系矩阵来构建传递闭包。算法的核心在于检查所有顶点对，判断是否存在通过中间顶点连接的路径。

### 5.2.1 算法步骤

Warshall 算法的具体步骤如下：

**初始化矩阵**：首先，创建一个与输入关系矩阵相同的副本，称为 warshall 矩阵。这个矩阵用于存储计算结果，以便在不改变原始数据的情况下进行处理。

**三重循环迭代**：算法使用三层嵌套循环来遍历每一个可能的起点、终点和中间顶点。外层循环遍历所有可能的中间顶点 k，中层循环遍历所有起始顶点 i，内层循环遍历所有终止顶点 j。

**路径判断与更新**：在每次迭代中，检查是否存在从 i 到 k 的路径（即 warshall[i][k] 为 1）以及从 k 到 j 的路径（即 warshall[k][j] 为 1）。如果这两个条件都满足，则说明可以通过 k 连接 i 和 j，因此将 warshall[i][j] 设置为 1。

**返回结果**：经过所有迭代后，warshall 矩阵即为输入关系的传递闭包。

### 5.2.2 算法复杂度

Warshall 算法的时间复杂度为 O(n³)，其中 n 是图中顶点的数量。这是因为算法包含三个嵌套循环，每个循环都遍历 n 个元素。虽然时间复杂度较高，但在实际应用中，对于中小规模的关系计算，Warshall 算法通常能在可接受的时间内完成任务。

空间复杂度为 O(n²)，这是由于需要额外的空间来存储传递闭包矩阵。因此，算法的空间复杂度主要取决于输入矩阵的大小。

。

1. **实验过程**

## 6.1实验思路

本实验的主要思路是设计并实现一个程序，用于计算任意给定关系的传递闭包。实验将结合离散数学的理论知识与实际编程技能，帮助学生深入理解传递闭包的概念及其应用。首先，利用矩阵这一数据结构来表示和存储关系矩阵，确保在输入、处理和输出过程中数据的有效管理。

在程序设计中，我们将实现 Warshall 算法，该算法以其高效性和简洁性著称，适用于计算有向图的传递闭包。具体步骤包括：首先接收用户输入的矩阵大小和元素值，然后通过三层循环实现算法逻辑，逐步更新关系矩阵，以计算出传递闭包。

此外，程序将提供用户友好的界面，确保用户能够轻松输入数据，并清晰地展示计算结果。在实现过程中，特别注重输入的合法性检查和输出格式的美观性，力求提升用户体验。通过这一系列的设计与实现，学生将能够巩固理论知识，提升编程能力，并为后续更复杂的计算问题做好准备。

## 6.2 实验设计

本实验的设计包括多个方面，以确保程序的功能完整性和用户体验的友好性。以下是实验设计的主要内容。

### 6.2.1 数据结构设计

程序主要使用vector<vector<int>>来表示关系矩阵。这是一个二维整数向量，提供了灵活的内存管理和方便的操作接口。选择vector而不是传统的二维数组，是因为vector可以动态调整大小，更适合处理不同规模的关系矩阵。

### 6.2.3 程序主体架构设计

程序主体架构设计为模块化结构，主要包括以下几个部分：

**初始化和界面提示**：在程序启动时，使用 system("cls") 清屏，为用户提供一个清晰的输入界面，并显示程序的标题和说明。

**输入模块**：设计 CheckCin 函数用于输入矩阵大小，确保用户输入在合法范围内。InputMatrix 函数负责接收用户输入的关系矩阵元素，确保输入的准确性。

**闭包计算模块**：实现 Warshall 函数计算传递闭包。该函数将根据输入的关系矩阵进行三层循环操作，逐步更新矩阵，以完成传递闭包的计算。

**输出模块**：设计 PrintMatrix 函数负责格式化输出矩阵结果，确保输出清晰、易读，便于用户理解。

**交互控制模块：**

- 主循环控制程序的重复执行。

- InputChoice函数处理用户是否继续的选择。

这种模块化设计使得程序结构清晰，易于理解和维护。各个模块之间职责明确，降低了代码的耦合度。

## 6.3 程序功能实现

### 6.3.1 逻辑值输入功能的实现

InputValue函数实现了逻辑值的输入：

|  |
| --- |
| 附件、InputValue函数 |
| void InputValue(int& value)  {  char invalue;  while (1) {  invalue = \_getch();  if (invalue == '0' || invalue == '1') {  cout << invalue;  break;  }  }  value = (int(invalue) - 48);  return;  } |

这个函数使用\_getch()函数从键盘获取单个字符，而不是使用标准的cin。

它可以实现即时响应，无需按回车键。同时限制输入只能是'0'或'1'，提高了输入的准确性。也通过while循环，确保只有在输入有效字符时才退出。

函数将字符转换为整数值（0或1）并存储。这种设计既保证了输入的正确性，又提高了体验。

### 6.3.2 输入关系矩阵大小功能的实现

CheckCin函数用于获取用户输入的整数，并确保输入在指定范围内：

|  |
| --- |
| 附件、CheckCin函数 |
| void CheckCin(int& value, const int lower\_limit = 0, const int higher\_limit = 10000)  {  while (1) {  cin >> value;  if (cin.fail()) {  cout << "输入错误，请重输\n";  cin.clear();  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  continue;  }  else if (value < lower\_limit || value>higher\_limit) {  cout << "输入错误，请重输\n";  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  continue;  }  else {  cin.ignore(numeric\_limits<streamsize>::max(), '\n');  break;  }  }  return;  } |

这个函数的设计考虑了多种错误情况。使用cin.fail()检查输入是否失败（例如，输入非数字字符）。检查输入值是否在指定范围内。对于任何错误情况，都会清除错误状态并要求用户重新输入。使用cin.ignore()清除输入缓冲区，防止后续输入受到影响。

它设计确保了程序能够获得有效的矩阵大小，增强了程序的健壮性。

### 6.3.3 输入关系矩阵功能的实现

InputMatrix函数用于输入整个关系矩阵：

|  |
| --- |
| 附件、InputMatrix函数 |
| void InputMatrix(vector<vector<int>>& matrix) {  int size = int(matrix.size());  for (int i = 0; i < size; i++) {  for (int j = 0; j < size; j++) {  int value;  InputValue(value);  matrix[i][j] = value;  if (j != size - 1) {  cout << " ";  }  }  cout << endl;  }  } |

它使用嵌套循环遍历矩阵的每个元素。调用InputValue函数获取每个元素的值，确保输入的正确性。在输出时添加适当的空格和换行，使输入过程更加直观。使得可以清晰地看到矩阵的结构，减少输入错误的可能性。

### 6.3.4 输出关系矩阵功能的实现

PrintMatrix函数用于格式化输出关系矩阵：

|  |
| --- |
| 附件、PrintMatrix函数 |
| void PrintMatrix(const vector<vector<int>> matrix) {  int size = int(matrix.size());  for (int i = 0; i < size; i++) {  for (int j = 0; j < size; j++) {  cout << matrix[i][j];  if (j != size - 1) {  cout << " ";  }  }  cout << endl;  }  return;  } |

它使用嵌套循环遍历并打印矩阵的每个元素。在元素之间添加空格，提高可读性。每行结束时换行，保持矩阵的结构。这种输出方式使得矩阵结构清晰可见，便于理解和验证结果。

### 6.3.5 退出程序功能的实现

程序使用while循环来反复执行操作，并通过InputChoice函数来决定是否继续：

|  |
| --- |
| 附件、InputChoice函数 |
| void InputChoice(int& value)  {  char invalue;  while (1) {  invalue = \_getch();  if (invalue == 'y') {  cout << invalue << endl;  value = 1;  break;  }  else if (invalue == 'n') {  cout << invalue << endl;  value = 0;  break;  }  }  return;  } |

它使用\_getch()实现即时响应。只接受'y'或'n'作为有效输入。立即显示用户的选择并返回相应的值。

这种设计提供了一种简单而直接的方式来控制程序的继续或退出，增强了交互性。

### 6.3.6主函数实现

|  |
| --- |
| 附件、main函数 |
| int main()  {  while (1) {  system("cls");  cout << "+--------------------------------------------------------------+\n";  cout << "| Warshall方法求传递闭包 |\n";  cout << "+--------------------------------------------------------------+\n";  cout << "请输入矩阵大小(1~100)\n";  int size;  CheckCin(size, 0, 100);  vector<vector<int>> matrix(size, vector<int>(size));  cout << "请依次输入矩阵元素值\n";  InputMatrix(matrix);  cout << "Warshall算法求传递闭包为：\n";  PrintMatrix(Warshall(matrix));  int continue\_value;  cout << "是否继续运算？ （y/n）: ";  InputChoice(continue\_value);  if (!continue\_value) {  break;  }  }  return 0;  } |

主函数清晰调用了各个功能函数，简洁易懂，实现了程序的正确循环，提升了可操作性。

## 6.4 核心算法实现

传递闭包的计算通过 Warshall 算法实现，具体的实现步骤如下：

### 6.4.1 初始化和拷贝原始矩阵

在算法开始时，首先创建一个与输入关系矩阵相同的副本，称为 warshall 矩阵。这样做的目的是为了在不修改原始矩阵的情况下进行操作，确保原始数据的完整性。副本矩阵将用于存储计算后的传递闭包结果。

### 6.4.2 获取矩阵大小

获取关系矩阵的大小，即顶点的数量，用于控制后续循环迭代的次数。该大小将用于设置循环的边界条件，确保所有元素都能被正确访问。

### 6.4.3 三层循环迭代更新

Warshall 算法的核心在于三重循环的迭代更新。具体步骤如下：

1. **外层循环（变量 k）**：遍历所有可能的中间顶点 k。对于每一个中间顶点，检查是否能通过该顶点连接起始顶点和终止顶点。
2. **中层循环（变量 i）**：遍历所有可能的起始顶点 i。对每一个起始顶点，检查其与其他顶点之间的连接关系。
3. **内层循环（变量 j）**：遍历所有可能的终止顶点 j。通过检查 warshall[i][k] 和 warshall[k][j] 的值，判断是否存在从 i 到 j 的路径。如果 warshall[i][k] 和 warshall[k][j] 均为 1，则说明可以通过中间顶点 k 连接起始顶点 i 和终止顶点 j，此时将 warshall[i][j] 设置为 1。

通过上述三层循环，算法能够有效地检查并更新所有顶点对之间的关系，从而实现传递闭包的计算。

### 6.4.4 返回计算结果

经过所有迭代后，warshall 矩阵即为输入关系的传递闭包。算法将返回这一矩阵，供后续的输出和验证使用。

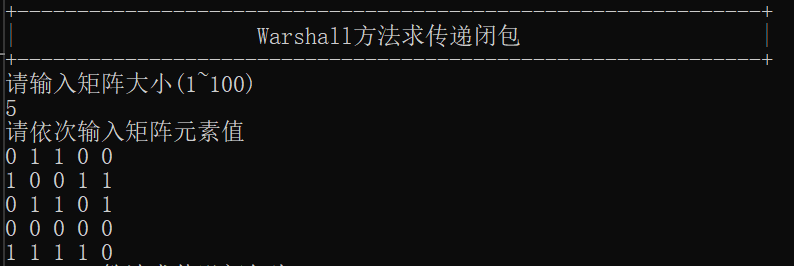
|  |
| --- |
| 附件、Warshall函数 |
| vector<vector<int>> Warshall(const vector<vector<int>> matrix)  {  int size = int(matrix.size());  vector<vector<int>> warshall = matrix;  for (int k = 0; k < size; k++) {  // 对每个起点i  for (int i = 0; i < size; i++) {  // 对每个终点j  for (int j = 0; j < size; j++) {  // 如果i->k和k->j都存在路径,则i->j存在路径  if (warshall[i][k] && warshall[k][j]) {  warshall[i][j] = 1;  }  }  }  }  return warshall;  } |

1. **实验结果**

在本实验中，我们实现了关系的自反闭包、对称闭包和传递闭包的计算算法，并对这些算法进行了测试。以下是实验的主要结果：

## 7.1 输入关系矩阵

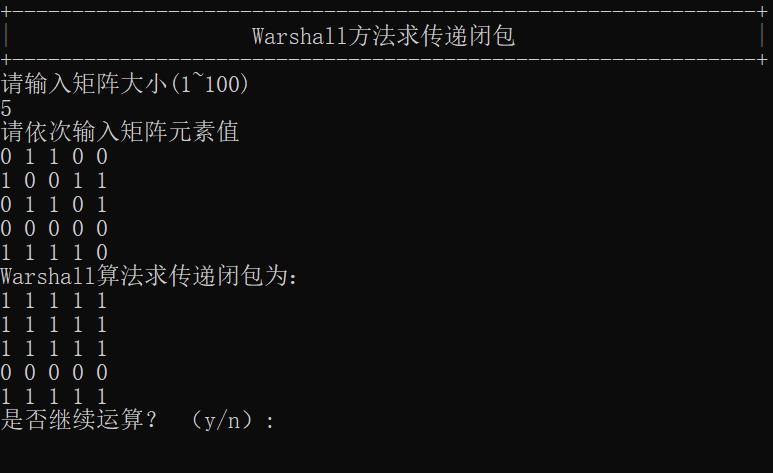
我们使用以下4x4的关系矩阵作为测试输入：



这个矩阵代表了一个包含5个元素的关系。

## 7.2 传递闭包结果

应用Warshall传递闭包算法后，得到的结果矩阵如下：



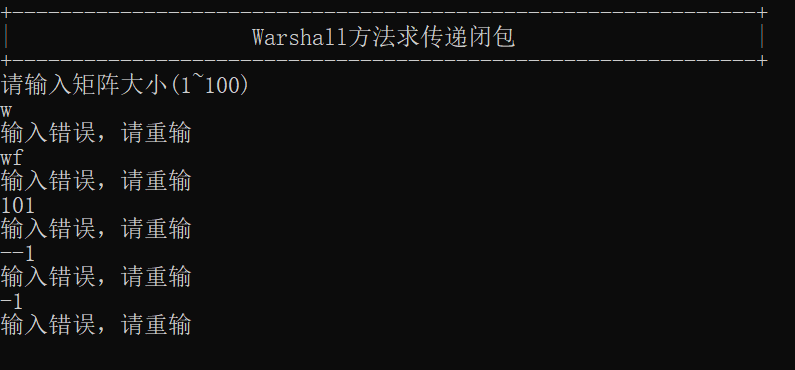
这个结果表明，经过传递闭包计算后，矩阵中的所有相对应元素都变为了1。这意味着在传递关系下，每个除全0行的元素都与其他所有元素（包括自身）存在关系。这是因为原始关系中存在的路径使得除全0行的元素之间都可以通过某种方式相连。

## 7.3 结果分析

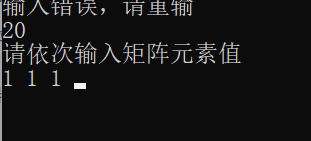
传递闭包：算法捕捉到了所有可能的直接和间接关系。在这个特定的例子中，由于原始关系的性质，传递闭包导致除全0行的元素形成了完全连接的关系网络。

这些结果验证了我们实现的算法的正确性和有效性。此算法可以应用于更大规模的关系分析中，帮助理解和处理复杂的关系结构。

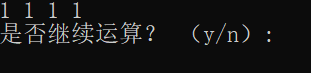
## 7.6错误数据分析



输入错误矩阵大小时，程序做出了正确处理，禁止了错误输入。



当键入非0、1的数字时，程序光标会闪烁，但不会输入结果并显示，只有输入0、1才会显示。



当键入非y、n的选项时，程序光标会闪烁，但不会输入结果并显示，只有输入y/n才会成功再次输入矩阵或结束程序。

1. **实验小结**

通过本次《离散数学》课程的实验，我对关系的传递闭包及其计算算法有了更深入的理解，并在实践中提升了编程能力和逻辑思维能力。这次实验主要围绕 Warshall 算法的实现，通过编写 C++ 程序来计算给定关系的传递闭包，结合理论与实践，使我对离散数学的相关概念有了更加深刻的认识。

首先，实验使我明确了传递闭包的定义及其重要性。在离散数学中，传递性是关系的一个基本属性。通过对传递闭包的计算，我们能够确保如果存在通过中间元素的关系，那么直接的关系也应成立。这一概念在许多实际问题中都有应用，比如在网络路由、社交网络分析和数据库管理等领域。通过本次实验的实现，我体会到了传递闭包的实际意义，并理解了它在数据结构和算法中的应用。

其次，Warshall 算法的实现让我深入了解了动态规划的思想。该算法通过逐步更新关系矩阵，能够高效地计算出传递闭包。通过三重循环的迭代，我掌握了如何在编程中实现复杂的逻辑判断和数据更新。这一过程不仅锻炼了我的编程能力，也增强了我在实际问题中应用理论知识的能力。

在编程过程中，我遇到了一些挑战。例如，在处理用户输入时，确保输入的合法性和准确性是一个重要环节。我实现了输入验证机制，确保用户只能输入合法的逻辑值（0或1），并在输入错误时给予明确的提示。这一部分的实现让我意识到，良好的用户体验对于软件开发是至关重要的。一个优秀的程序不仅要在功能上满足需求，还要考虑到用户的操作习惯和反馈。

此外，实验还让我体会到了数据结构的重要性。在本次实验中，我主要使用了矩阵这一基本的数据结构来存储和处理关系。这使我理解到，选择合适的数据结构对算法的性能有着直接影响。通过对矩阵的操作，我能够更高效地实现算法，提升了程序的执行效率。

在结果验证方面，我设计了一系列测试用例来确保算法的正确性和效率。通过对不同规模和类型的关系矩阵进行测试，我能够观察到算法在处理各种输入时的表现。这一过程不仅提高了我对算法性能的分析能力，也让我理解到系统测试的重要性。确保程序在各种情况下都能稳定运行，是软件开发中不可或缺的一部分。

总结来说，本次实验为我在离散数学领域的学习提供了宝贵的实践经验。通过实际编程，我不仅巩固了理论知识，还提升了编程技能和问题解决能力。这些经验无疑对我未来在更复杂的计算问题中应用所学知识打下了坚实的基础。我期待在今后的学习和研究中，继续探索离散数学及其在计算机科学中的应用，努力将理论与实践相结合，以应对更具挑战性的课题。