

**课 程 报 告**

**（2023/2024 学年 第2学期）**

课程名称： 算法设计与分析

班 级： 计科221

姓 名： 董自经

学 号： 19222126

指导老师： 袁培森 副教授

南京农业大学 人工智能学院

**实 验 报 告1**

|  |  |  |  |
| --- | --- | --- | --- |
| **实验名称** | Divide and Conquer Approach | **实验时间** | 2024.5.28 |
| 1. **实验目的和要求**   **实验目的**：实现数组的逆序对计数，依次指示数组排序的距离，如果数组已经排序，则逆序对计数为0，但是如果数组以相反的顺序排序，则逆序对计数为最大值  如果且，则两个元素和构成一个倒置。  **实验要求：**   * 对算法的基本思想进行描述； * 对算法的时间复杂度和空间复杂度进行分析； * 给出工程代码并运行以及分析结果； | | | |
| 二、**实验环境(实验设备)**  硬件：  设备名称 MateBook 14  处理器 Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz  机带 RAM 16.0 GB (15.8 GB 可用)  软件：  Windows版本 Windows 11 家庭中文版：版本 23H2  操作系统版本 22631.3593  Microsoft Visual Studio2022   1. **实验原理及内容**   **实验原理：**  数组反转计数实验采用了分治法，分治法的基本思想是从全局考虑问题，将一个比较大的、复杂的问题分解为K个相似的、可解的子问题，对这K个子问题分别进行求解，之后将K个子问题的解进行合并，求出原较为复杂的问题的解。  适合使用分治法求解的问题都必须具有两个基本的特征：最优子结构性质和子问题独立性质。  1．最优子结构性质：原问题的最优解包含着其子问题的最优解,这种性质称为最优子结构性质。利用该性质，自底向上的递归地从子问题的最优解中构造出原问题的最优解。最优子结构性质是问题可以使用分治法求解的基本前提。  2．子问题独立性质：子问题的不同划分不会影响最终的解，每个子问题都是独立的，即子问题之间不包含公共的子问题，每个子问题独立的进行求解，每个解之间也不存在依赖关系。最后可以通过合并子问题的解求解出原问题的解。  分治法的基本步骤可以概括为”分、治、合”三个过程：  分：自顶向下地将原问题分解为K个较小的、相似的子问题，这些子问题和原问题属于相同类型，而且彼此之间相互独立，互不影响。  治：求解子问题，分解出的子问题应该是可以直接进行求解的，对分解出来的子问题进行求解，但如果子问题规模仍然较大，无法直接进行求解，则继续递归分解子问题为子子问题，直至子问题的规模小到足够可以直接求出结果。  合：合并子问题的解，进而求出原问题的解，并检验解是否符合条件[1]。  **算法应用：**  分治法的核心思想是分解问题，之后递归求解子问题，并将子问题的解合并为原问题的解，因为分治法可以处理较大规模的问题，在计算中有着广泛的应用。分治法的典型应用有：矩阵乘法、排序算法、最大子数组问题等等[2][3]。  正因为分治法的高效性，分治法也经常和其他算法相结合进而提高算法的效率，提高解决问题的能力。如将分治法和贪心算法相结合进行联合断站目标关联定位，将分治法与动态规划方法结合进行序列对齐等等[4][5]。  **实验内容**：  算法名称：使用分治法对数组进行反转计数  实验输入：长度为n的数组。  实验输出：数组的逆序对计数，其中 。  1．要将数组分解为两个子数组，。   |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 原数组 |  |  | 6 | 4 | 16 | 15 | 9 | 3 | 8 | 12 |  |  | |  |  |  | |  | | --- | |  | |  |  |  |  |  |  |  |  |  | | 子数组 |  | 6 | 4 | 16 | 15 |  |  | 9 | 3 | 8 | 12 |  | |  | |  | | --- | |  | |  |  |  |  |  |  | |  | | --- | |  | |  |  |  |  | | 子子数组 | 6 | 4 |  |  | 16 | 15 |  | 9 | 3 |  | 8 | 12 |   表1：分割原数组  2．递归求解子问题：  求解：求出子数组中的逆序对数目；  求解：求出子数组中的逆序对数目。  3．合并两个子数组的解和，同时求解：跨越子数组的逆序对数目。  4．得到最终解。  算法的关键在于对进行求解，在将两个子数组进行合并时，如果直接对进行求解，则需要遍历，两个数组，时间复杂度为，结合前面的分治法，总的时间复杂度公式为，由主定理可得，与传统的蛮力法[3]遍历数组求解逆序对数目的时间复杂度并没有提升[6]。  因此，可以对已知逆序对数目的子数组，进行排序，具体操作为：  依次扫描已经排序的子数组，，如果，i+1，如果则逆序对数目加一，j+1。在此过程中归并排序[7]可以有效的保证合并后数组的顺序的正确性，同时使得求解的时间复杂度下降至。   |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 目标数组 |  | 3 | 4 | 6 | .. | .. | .. |  |  |  | |  |  |  |  |  |  |  |  |  |  |  | | 子数组 | 4 | 6 | 15 | 16 |  |  | 3 | 8 | 9 | 12 | |  |  |  | i=2 |  |  |  |  | j=5 |  |  | |  |  |  |  |  | 反转计数:7 | |  |  |  |  |   表2：合并子数组  **复杂性分析：**  时间复杂度：  在递归部分，分割原数组为子数组的操作的时间复杂度为，由主定理得，该部分的时间复杂度为。  在合并部分，最坏的情况是子数组中的每一个元素都要与子数组中的每一个元素进行比较已确定是否交换位置，因此，合并部分的时间复杂度为。  由于递归部分将数组分为两部分，所以一共有个子数组，合并部分是在此基础上进行合并，所以合并操作的总时间复杂度为  因此，逆序对计数的总时间复杂度就是  空间复杂度：  因为算法需要使用一个临时的数组temp来存储数据，大小与原数组相同为n，所以空间复杂度为  **实验结果：**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 实验序号 | 数组大小 | 数组内容 | 逆序对个数 | 排序距离 | | 1 | 4 | {3,1,2,4} | 2 | 3 | | 2 | 8 | {2,3,6,7,8,1,5,4} | 12 | 4 | | 3 | 16 | {7,16,9,1,15,13,6,11,12,5,3,4,,8,14,10,2} | 72 | 64 | | 4 | 64 | {24,45,2,50,8,11,34,42,47,51,1,55,35,40,43,28,12,37,14,44,21,13,25,48,58,19,6,27,10,38,39,56,62,17,5,46,7,26,53,30,20,61,57,18,15,3,36,63,60,33,29,16,64,52,9,23,22,54,49,59,41,31,32,4} | 933 | 892 | | 5 | 128 | {82,96,87,61,20,40,21,127,32,76,9,66,58,23,33,78,91,41,95,27,73,30,84,125,80,5,2,116,17,121,53,24,93,55,94,118,83,123,69,117,39,75,98,64,92,108,56,48,45,1,60,111,19,6,115,28,97,4,57,52,46,3,71,11,103,43,114,8,65,85,128,10,77,90,34,101,113,13,44,47,120,63,35,102,38,104,107,122,59,72,81,42,16,88,54,50,106,119,68,36,7,124,89,100,29,62,37,70,86,15,14,105,49,67,126,110,74,31,79,109,99,25,22,51,112,26,12,18} | 4032 | 4110 |   表3：不同数组的反转计数和排序距离  **结果分析：**  由于数据是由系统的随机函数生成，并且设置了不重复的筛查，所以在随机生成的数据中，逆序对个数和排序距离都会随着数组中元素数目的增加而增加。  **对比分析：**  对于数组反转计数一般有三种经典解法：暴力求解法、树状数组法/线段树法已经本报告使用的分治法。  暴力求解法是指直接遍历数组中的每一对元素并检查他们是否会构成逆序对，如果是则进行计数。此种方法实现较为简单，只需要两层循环即可，但也正因为两层循环，使得该算法的时间复杂度为，时间复杂度较差，已经淘汰使用。  树状数组法在进行逆序对计数之前要先将数组进行离散化，将数组元素映射到连续的整数上，以便于使用树状数组来维护每个元素之前比他大的元素的个数。之后遍历数组，查询每个元素之前有多少比他大的元素，并对计数进行修改。该算法的时间复杂度同本报告使用的分治法一致，为，也是一种较为便捷快速的算法。  **参考文献：**  [1]胡昆鹏,仵子俊.分治递归算法的解题能力[J].电脑编程技巧与维护,2023(01):44-47.DOI:10.16184/j.cnki.comprg.2023.01.045.  [2]张忠诚,鲁法明.基于递归与分治的排序算法教学探究[J].计算机与数字工程,2019,47(09):2109-2114.  [3]陈艳,文晓棠.蛮力法、分治法和动态规划法求解最大子数组问题的思考[J].现代计算机,2023,29(18):24-29. [4]Shubham ,Surya P ,Pramod G .An Algorithm for the Sequence Alignment with Gap Penalty Problem using Multiway Divide-and-Conquer and Matrix Transposition[J].Information Processing Letters,2021,(prepublish):106166-.  [5]王冠群,张春华,张舒然.基于分治贪心思想的联合多站目标关联定位[J].兵工学报,2021,42(12):2700-2709.  [6]俞露.基于分治法逆序计数的一个实际应用[J].电脑编程技巧与维护,2013(08):38-39.DOI:10.16184/j.cnki.comprg.2013.08.043.  [7]Sirilak K ,Apisit R .Parallel Multi-Deque Partition Dual-Deque Merge sorting algorithm using OpenMP[J].Scientific Reports,2023,13(1):6408-6408.  **四、实验小结**（包括问题和解决方法、心得体会、意见与建议等）  在进行本次实验时，要对分治法”分而治之”的思想有一定的理解，对”分、治、合”三个过程都要有清楚的认识，在”分”阶段将数组分割成子数组，没有太大的问题；在”治”阶段对逆序数对进行计数时，采用何种方法求解A3的结果直接影响了算法的效率和功能强弱与否，在”合”阶段，将子数组的解合并为原问题的解时要注意检查解是否符合原问题的约束。  一开始采用的遍历两个子数组的方法虽然简介明了、易于实现，但复杂度达到了O(n2)级别，效率较低，不适合求解大规模问题，后采用了对子数组进行排序之后二分查找的方法，虽然将时间复杂度降低到O(nlog2n)数量级，有一定的改善，但数量级仍然较大，求解大规模问题时效率不尽人意。最后采取了归并排序的同时对逆序数对进行计数的方法，将数量级降低到了O(nlogn)级别。  分治法作为一种高效的算法思想，通过适当分解、递归求解、子问题合并的过程，能有效解决较大规模的问题，提高算法的效率和解决问题的能力。算法的空间复杂度分析和时间复杂度计算也为算法实际应用提供了重要参考。将分治法与其他算法相结合，应该能够为解决大规模问题实际问题提供新的、更加快速、适用性更加广泛的方法。 | | | |

**实 验 报 告2**

|  |  |  |  |
| --- | --- | --- | --- |
| **实验名称** | **最短通用超序列** | **实验时间** | 2024.6.3 |
| 1. **实验目的和要求**   **实验目的**：实现对两个字符串 和的最短通用超序列的查找。其中，通过对进行删除若干字符（不同位置，顺序不定）可以分别得到str1和str2，最后将其长度作为结果输出。  **实验要求：**   * 对算法的基本思想进行描述； * 对算法的时间复杂度和空间复杂度进行分析； * 给出工程代码并运行以及分析结果； | | | |
| 二、**实验环境(实验设备)**  硬件：  设备名称 MateBook 14  处理器 Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz  机带 RAM 16.0 GB (15.8 GB 可用)  软件：  Windows版本 Windows 11 家庭中文版：版本 23H2  操作系统版本 22631.3593  Microsoft Visual Studio2022   1. **实验原理及内容**   **实验原理：**  最短通用超序列实验采用了动态规划[1]的方法，动态规划的基本思想是将一个复杂的问题分解为多个较为简单的子问题，经过分解之后的子问题往往不是彼此独立的，可以使用一个表来记录所有已经解决的子问题的答案，无论该子问题是否会被用到，均将计算得到的结果填入表中，并在需要的时候重新使用这些解，从而避免了大量的重复计算依次提高效率。适合使用动态规划方法求解的问题都必须具有两个基本的特征：最优子结构性质和子问题重叠性质[2]。  1．最优子结构性质：原问题的最优解包含着其子问题的最优解,这种性质称为最优子结构性质。利用该性质，自底向上的递归地从子问题的最优解中构造出原问题的最优解。最优子结构性质是问题可以使用动态规划方法求解的基本前提。  2．子问题重叠性质：在动态规划中，递归求解子问题时，每次产生的子问题并不都是新的问题，有的子问题会被重复计算多次。在动态规划中，每个子问题只求解一次，将解保存在一个表格中，后续在需要解该问题时，只需要使用较短时间查看结果即可。子问题重叠性质是动态规划算法用空间换取时间的重要因素。  动态规划的步骤一般有以下四个步骤[3]：  1．划分：按照问题本身的空间或者时间的特征，先将问题分解为若干个可以排序的子阶段  2．确定状态：将问题发展到各个阶段时所处的各种客观情况使用不同的状态表示  3．确定决策和状态转移方程：状态转移是根据上一阶段的状态和做出决策来导出本阶段的状态，所以一般确定了决策，也就确定了状态转移方程。  4．寻找边界条件：状态转移方程是一个递推式，需要确定一个终止条件来设置递推的终止条件和边界条件。  **算法应用：**  因为动态规划算法的实际运用价值极高，所以被广泛应用于算法问题中，也会与其他算法相结合解决如背包问题、旅行商问题等[4]，同时也被应用于实际生活中，如路径优化[6]、能源系统分层控制[6]、船舶自动避免碰撞[7]等等。  **实验内容**：  算法名称：使用动态规划求解最短通用超序列  实验输入：两个字符串，  实验输出：两个字符串，的一个最短通用超序列以及其长度   1. 建立动态规划数组用于存储解 2. 确定状态转移方程   2.1  两个字符一样，直接加入超序列，写入动态规划数组：     |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | str1 | b | t | x | r | a | m |  |  | str1 | b | t | x | r | a | m | | str2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |  | str2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | | u | 1 | 2 | 3 | 4 | 5 | 6 | 6 |  | u | 1 | 2 | 3 | 4 | 5 | 6 | 6 | | l | 2 | 3 | 4 | 5 | 6 | 7 | 7 |  | l | 2 | 3 | 4 | 5 | 6 | 7 | 7 | | t | 3 | 4 | 4 |  |  |  |  |  | t | 3 | 4 | 4 | 5 | 6 | 7 | 8 | | r | 4 |  |  |  |  |  |  |  | r | 4 | 5 | 5 | 5 |  |  |  | | a | 5 |  |  |  |  |  |  |  | a | 5 |  |  |  |  |  |  | | k | 6 |  |  |  |  |  |  |  | k | 6 |  |  |  |  |  |  | | b | 7 |  |  |  |  |  |  |  | b | 7 |  |  |  |  |  |  |   表4：两字符一样时对dp的写入 表5：两个字符不一样时对dp的写入  2.1  两个字符不一样，可能是 加入超序列中，也可能是加入超序列中，选择二者较小的加入，写入动态规划数组：    2.1 边界条件：当或为0时，相应的字符串为空，最短通用超序列就等于另一个字符串的长度，写入动态规划数组：     1. 利用动态规划求最短公共超序列的长度，并将解写入数组中 2. 利用上一步得到的数组反推最短通用超序列，从两个字符串，的尾部开始遍历，设   4.1  及说明此字符为两字符串共有字符，加入到最终解 中  4.2 ，则根据来判断上一个字符来自于还是   |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | str1 | b | t | x | r | a | m |  |  | str1 | b | t | x | r | a | m | | str2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 |  | str2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | | u | 1 |  |  |  |  |  |  |  | u | 1 | 2 | 3 | 4 | 5 | 6 | 6 | | l | 2 |  |  |  |  |  |  |  | l | 2 | 3 | 4 | 5 | 6 | 7 | 7 | | t | 3 |  |  |  |  |  |  |  | t | 3 | 4 | 4 | 5 | 6 | 7 | 8 | | r | 4 |  |  |  |  |  |  |  | r | 4 | 5 | 5 | 5 | 6 | 7 | 8 | | a | 5 |  |  |  |  |  |  |  | a | 5 | 6 | 6 | 6 | 7 | 7 | 8 | | k | 6 |  |  |  |  |  |  |  | k | 6 | 7 | 7 | 7 | 8 | 8 | 9 | | b | 7 |  |  |  |  |  |  |  | b | 7 | 8 | 8 | 8 | 9 | 9 | 10 |   表6：边界条件 表7：完整的dp数组  **复杂性分析：**  时间复杂度：  动态规划部分：在动态规划过程中，需要对两个字符串进行叠加遍历，用到了两层循环，时间复杂度为。  求解部分：在最坏情况下，需要遍历整个和的剩余部分才能得到最终的解，所以构建最终结果字符串的过程的时间复杂度为。  因此，总体时间复杂度取决于动态规划部分，为。  空间复杂度：  动态规划部分：生成了一个大小为的二维数组dp，其空间复杂度为。  求解部分：构建最终结果字符串时，需要额外的个空间来存储最短公共超序列的字符串，空间复杂度为。  因此，总体空间复杂度也取决于动态规划部分，为。  **实验结果：**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 实验序号 | str1 | str2 | str.size() | str | | 1 | “39yiW9” | "QkR08J7Z" | 14 | “39yiW9QkR08J7Z” | | 2 | "rDyrZ41j0AX7" | "7fTC7GFOh7rUvJD274L2" | 29 | "7fTC7GFOh7rUvJDyrZ41j0AX274L2" | | 3 | "xf45vHV2S17C59PG7RK4fHCbb" | "2ML7JnKMDQ5X6709Jqe114y7Z0CEQI" | 49 | "xf45vHV2S1ML7CJnKMDQ59PGX67RK09Jqe114fHy7Z0CbbEQI" | | 4 | "aR7U72Bobt7J8PlluL8K0r7ayH27kdm47107LWY7Imwo0oESygoHzj1902v3Fe7k6wdAZY7SkYY2yEIYMgVol8VkQp8hcwuoMfHN" | "u7a5h641i1cph2b7n0KLsefF2PJUya5QdwIzJxdHZ4cfGvm7P90H3e9U1MMNaK00epH2wvne5rEIVX7KP0LnNPoHDFtoQg5WyMd0" | 178 | "aR7U72Bobt7J8PlluL8K0r7ayH27kdm5h64710i1cph2b7n0KLWY7sefF2PJUya5QdwIzJxdHZ4cfGvmwo7P90oESygoHzj13e9U1MMNaK00epH2wv3Fne7k6wdAZY7SkYY2y5rEIYMgVX7KP0LnNPol8VkQp8hcwuHDFtoQg5WyMfHNd0" |   表8：不同字符串的最短通用超序列及其长度  **对比分析：**  现阶段，最短通用超序列的求解只有动态规划这一个较为可行的方法，即使是使用回溯法也需要结合动态规划方法进行求解。但可以通过一些方法对动态规划方法进行优化。如只存储当前行和前一行的数组值来降低时间复杂度。同时还可以提前终止对的计数，即若已经确定一个子序列是最短通用超序列的一部分时，可以提前终止进一步的计算。  **参考文献：**  [1]秦丹.动态规划法之最优性原理教学[J].电脑知识与技术,2011,7(31):7819-7820.  [2]宁静雁.动态规划算法研究[J].电子世界,2014(10):452-453.  [3]林秀娣.动态规划算法解题思路[J].福建电脑,2021,37(01):178-180.DOI:10.16707/j.cnki.fjpc.2021.01.071.  [4]李胜华.Fibonacci数列在递归与动态规划算法教学中的应用[J].电脑知识与技术,2023,19(01):157-159.DOI:10.14004/j.cnki.ckt.2023.0031.  [5]李佳.动态规划在优化运输路径中的应用[J].物流工程与管理,2024,46(04):32-35.  [6]Zhao L ,Yin L .Knowledge-shareable adaptive deep dynamic programming for hierarchical generation control of distributed high-percentage renewable energy systems[J].Renewable Energy,2024,228120627-.  [7]Zaccone R .A Dynamic Programming Approach to the Collision Avoidance of Autonomous Ships[J].Mathematics,2024,12(10):  **四、实验小结**（包括问题和解决方法、心得体会、意见与建议等）  本次实验为最短通用超序列（LCS）问题的求解，通过应用动态规划方法，实现了对两个给定字符串的最短通用超序列及其长度的计算。  在实验中，首先构建了动态规划数组，用于存储子问题的解。随后，根据问题的特性，确定了状态转移方程，并妥善处理了边界条件。通过两层循环遍历两个字符串，逐步填充了动态规划数组，并得出了最短通用超序列的长度。最后，利用该数组反向推导出最短通用超序列本身。  实验过程中，动态规划算法的高效性得到了充分体现。通过避免重复计算子问题，显著提高了求解效率。同时，实验也验证了动态规划算法在解决具有最优子结构性质和子问题重叠性质问题时的适用性。  从算法设计的角度来看，动态规划算法的成功应用依赖于准确的状态定义、状态转移方程的确立以及边界条件的处理。这些步骤的正确执行，是确保算法正确性和高效性的关键。  本次实验成功地应用了动态规划算法求解最短通用超序列问题，不仅验证了算法的有效性，也提高了对动态规划算法的理解和应用能力。同时，实验结果对于理解动态规划算法在算法设计和优化中的应用具有一定的指导意义。 | | | |

**实 验 报 告3**

|  |  |  |  |
| --- | --- | --- | --- |
| **实验名称** | **MST Kruskal（3）** | **实验时间** | 2024.6.10 |
| 1. **实验目的和要求**   **实验目的**：通过Union\_find优化Kruskal算法以实现对给定的图的最小生成树的输出（输出最小生成树的相应的边即可）  **实验要求：**   * 对算法的基本思想进行描述； * 对算法的时间复杂度和空间复杂度进行分析； * 给出工程代码并运行以及分析结果； | | | |
| 1. **实验环境(实验设备)**   硬件：  设备名称 MateBook 14  处理器 Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz  机带 RAM 16.0 GB (15.8 GB 可用)  软件：  Windows版本 Windows 11 家庭中文版：版本 23H2  操作系统版本 22631.3593  Microsoft Visual Studio2022   1. **实验原理及内容**   **实验原理：**  求取图的最小生成树有多种试验方法[1]，本实验采用了Kruskal算法，其基本思想是先将图中所有的边按照权值的大小进行排序，形成一个有序的边列表，之后从权值最小的边开始，逐一考虑每一条边，如果将其加入最小生成树中且没有构成环[2]，则符合条件，否则考虑下一条边，重复此过程直至最小生成树中含有条边，其中为图的顶点数。由于着重点在于图的边Kruskal算法更加适合于求解稀疏图的最小生成树。  Kruskal算法的基本步骤有以下几步[3]：  1．初始化：创建一个空的图（只包含顶点，不包含边），创建一个空的集合来存储最小生成树的边，初始化所有顶点为不同的集合（每个顶点都是它自己的集合）。  2．排序：将图中所有的边按照权重从小到大进行排序。  3．选择边：从排序后的边列表中，选择权重最小的边，如果将其加入最小生成树中且没有构成环，那么将它加入到最小生成树中，并将这两个顶点所在的集合合并成一个新的集合，否则继续选择下一条权重最小的边。  4．重复：重复步骤3，直到选择了条边（是顶点的数量），或者边列表中的所有边都已经被考虑过。  5．结果：选择的条边和所有的顶点构成了一个最小生成树。  Kruskal算法中使用了贪心策略来选取当前情况下符合条件的权值最小的边。贪心算法（Greedy Algorithm）是一种在每一步选择中都采取在当前状态下最好或最优（即最有利）的选择，从而希望导致结果是全局最好或最优的算法。因为每一步都是当前状态的最优解，所以贪心算法并不保证所得到的解一定是最优解，但是它能提供一种较好的解或者能在一个较短时间内求得问题的解。  使用贪心算法的问题必须要满足最优子结构性质和贪心选择性质  1．最优子结构性质：原问题的最优解包含着其子问题的最优解,这种性质称为最优子结构性质。利用该性质，自底向上的递归地从子问题的最优解中构造出原问题的最优解。最优子结构性质是问题可以使用动态规划方法求解的基本前提。  2．贪心选择性质：问题的全局最优解可以通过一系列局部最优解的选择以实现，并且每次选择可以依赖于前面做出的选择，而不依赖于后面所要做出的选择。这是贪心算法的核心思想，确保了贪心算法的无后效性以及简单高效的特点。  实验中还采取了Union\_find（并查集）[4]的方法来优化Kruskal算法，并查集是一种用于管理分组的数据结构，它的主要操作包括合并两个集合（Union）和查找一个元素所属的集合（Find）。并查集的基本思想是通过树形结构来表示集合，每个集合对应一棵树，同一棵树上的所有节点都属于同一个集合。在并查集中，每个元素都会存储其父节点的信息，以便快速进行查找和合并操作。  在Kruskal算法中，Union Find主要用于判断两条边是否会形成环，在算法开始时，每个顶点都初始化为一个独立的集合（即每棵树只有一个节点）。然后，在每次选择边时，都使用Union Find的Find操作来判断两个端点是否在同一个集合中，并使用Union操作来合并两个集合。这样，通过更新一个Union Find数组，Kruskal算法可以高效地避免形成环，从而正确地构建出最小生成树。  **算法应用：**  最小生成树对于路径的选择具有无与伦比的高效性，在可以与其他算法相结合解决某些问题，如使用最小生成树计算供应链中最短的运输路径[6]，将最小生成树与MILP结合以进行城市规划[7]等等。  **实验内容**：  算法名称：使用Union\_find优化Kruskal算法求解最小生成树  实验输入：第一行包含两个整数V和E，分别表示顶点数和边数，接下来E行，每行包含三个整数u，v，w，表示顶点u和顶点v之间存在一条权值为w的边。  实验输出：最小生成树所含有的所有边   1. 以右边的无向无环图为例建立下面的Union\_find集合以及边集合  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | vertex[i] | 1 | 2 | 3 | 4 | 5 | 6 | | subsets[i].parent | 1 | 2 | 3 | 4 | 5 | 6 |     表9：建立的Union\_find集合  图1：无向无环图   1. 对已经建立的边集合使用函数进行升序排序。 2. 从排好序的边集合中选取最小值，通过Union\_find集合进行检验，如果没有形成环，则加入到最小生成树中，否则选取下一条边。   3.1寻找父集  3.1.1 ，则函数返回  3.1.2 ，则直至    3.2函数合并集合  3.2.1，则将的父节点置为，即    3.2.2，则将的父节点置为，即      3.2.3，则将的父节点置为，同时将的 加一，即   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | vertex[i] | 1 | 2 | 3 | 4 | 5 | 6 | | subsets[i].parent | 1 | 2 | 1 | 4 | 5 | 6 |   表10：联合过程中的Union\_find集合    图2：选择最小边建立最小生成树 图3：最小生成树  4．重复上述步骤直至所有顶点都已经加入同一联合。   |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | vertex[i] | 1 | 2 | 3 | 4 | 5 | 6 | | subsets[i].parent | 1 | 1 | 1 | 1 | 1 | 1 |   表11：最终的Union\_find集合   1. 输出最小生成树的边   **复杂性分析：**  时间复杂度：  算法中对边进行排序使用了C++中已经建立的函数进行快速排序，时间复杂度为[5]，其中E为边的数量。在使用Union\_find方法进行查找合并的操作的时间复杂度为，集中是顶点数量V相关的某个函数，该函数表示在V趋向于无穷大时，的增长速率要慢于V的任意正幂次，所以通常取。  而在一般图中，因此，总体的时间复杂度主要取决于排序的时间复杂度，为。  空间复杂度：  算法使用了Union-Find数据结构，空间复杂度主要取决于Union-Find数据结构本身的空间复杂度。对于Union-Find数据结构，因为每个顶点都需要一个指向其父节点的指针，而在最坏情况下，每个顶点都可能成为一个独立的集合，因此需要个指针，所以其空间复杂度是，其中是顶点的数量。此外，还需要的空间来存储边的信息，其中是边的数量。  因此，Kruskal算法的总空间复杂度是。  **实验结果：**   |  |  |  |  | | --- | --- | --- | --- | | 实验序 号 | 无向无环图  {vertex1，vertex2，value} | MST所含边  {vertex1，vertex2，value} | MST边总权值 | | 1 | {1,2,2}{1,3,3}{1,4,5}{2,4,6}{3,4,7} | {1,2,2}{1,3,3}{1,4,5} | 10 | | 2 | {1,2,4}{1,3,6}{1,4,7}{1,5,8}{1,6,9}{2,4,12}{2,5,9}{2,6,34}{3,4,5}{3,6,12} | {1,2,4}{3,4,5}{1,3,6}{1,5,8}{1,6,9} | 32 | | 3 | {1,3,4}{1,5,6}{1,6,8}{2,4,6}{2,7,9}{3,5,8}{3,6,8}{4,6,10}{4,7,8}{6,7,1} | {6,7,1}{1,3,4}{2,4,6}{1,5,6}{4,7,8}{3,6,8} | 33 | | 4 | {1,7,9}{2,7,5}{3,23,4}{3,24,8}{4,23,2}{5,10,2}{5,13,2}{6,9,5}{6,21,1}{7,12,3}{8,9,2}{8,10,4}{8,12,6}{9,115,}{10,14,1}{11,13,2}{11,15,8}{12,15,2}{13,17,3}{14,18,2}{15,18,7}{16,19,8}{17,20,2}{19,20,6}{21,22,6}{22,23,3} | {6,21,1}{10,14,1}{17,20,2}{4,23,2}{5,10,2}{11,13,2}{12,15,2}{14,18,2}{8,9,2}{7,12,3}{13,17,3}{22,23,3}{3,23,4}{8,10,4}{5,13,5}{2,7,5}{6,9,5}{8,12,6}{19,20,6}{22,22,6}{3,24,8}{16,19,8}{1,7,9} | 91 |   表12：最小生成树求解  **对比分析：**  最小生成树有三种经典的求解方法：Kruskal算法、Prim算法以及破圈法。  Prim算法与Kruskal算法相似，但着重点在顶点，每次寻找一个距离当前最小生成树距离最小的顶点将其加入树中，同时将该顶点的便加入候选判断集合中，在效率上更加适合求解稠密图的最小生成树。  经典Prim算法的时间复杂度为(为顶点数)，因为在最坏的情况下，需要检查所有未加入树的结点的边。使用邻接表优化后的Prim算法，通过维护一个预先队列来寻找当前距离书最小的节点，可以将时间复杂度降低到。  破圈法是指若看到图中有圈则将此圈中最大的权边去掉，重复此步骤直至途中没有圈，由于本报告使用的图均为无向无环图，所以破圈法并不适用于本报告中的图的最小生成树的求解，在此也不过多阐述。  **参考文献：**  [1]李龙霞,陈燕,于晓倩.最小生成树三种求解方法的分析与实现[J].电脑知识与技术,2021,17(33):44-46.DOI:10.14004/j.cnki.ckt.2021.3316.  [2]宋慧敏,孙薇,吴建良.求最小树的Kruskal算法中无圈判断的进一步思考[J].数学学习与研究,2021(13):151-153.  [3]袁威威.应用Kruskal的改进算法求最小生成树[J].江苏第二师范学院学报,2017,33(06):12-13.  [4]张海波.UNION-FIND算法中数据结构的应用[J].濮阳职业技术学院学报,2006(01):21+25.  [5]李家宏,孙庆英.两种交换排序算法的分析比较[J].电脑知识与技术,2023,19(16):35-37.DOI:10.14004/j.cnki.ckt.2023.0839.  [6]Shahin A, Jaferi F.The shortest route for transportation in supply chain by minimum spanning tree[J].Int. J. of Logistics Systems and Management,2015,22(1):43-54.  [7]Pavon W, Torres M, Inga E.Integrating Minimum Spanning Tree and MILP in Urban Planning: A Novel Algorithmic Perspective. Buildings. 2024; 14(1):213.  **四、实验小结**（包括问题和解决方法、心得体会、意见与建议等）  本次实验研究了Kruskal算法在求解图的最小生成树问题中的应用。Kruskal算法的核心思想是基于贪心策略，它总是优先选取当前权值最小的边，并判断其是否会导致环路的形成，这种策略确保了最终构建的树是权值和最小的。实验结果验证了无论是对于稠密图还是稀疏图，Kruskal算法在寻找最小生成树方面都具有准确性和高效性，都能得到正确的结果。  在实验过程中，详细记录了算法的执行步骤，包括边的排序、边的选择以及Union-Find数据结构的运用。这些步骤不仅展示了Kruskal算法的具体实现细节，还体现了贪心算法在解决优化问题时的优势。  本次实验加深了我对Kruskal算法的理解和掌握。通过扩展实验范围、优化算法实现以及探索新的应用场景，可以进一步挖掘Kruskal算法的潜力和价值，为解决更多实际问题提供有力的工具和方法。 | | | |