**文献管理与分析系统**

1. **实验目标**

通过分阶段完成一系列任务，构建一个简化的学术文献管理与分析系统。该系统能够加载、存储、查询、排序学术论文，并能进行简单的关联分析和推荐。通过实践，学生将深入理解各种数据结构（如数组、链表、树、哈希表、图、堆）和算法（如查找、排序、图遍历）的原理及其在实际问题中的应用。

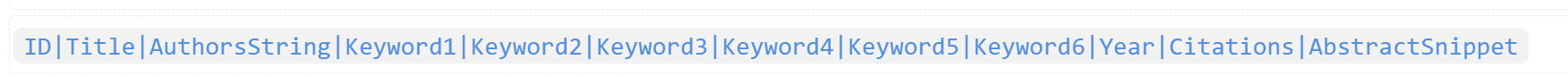
掌握多种数据结构（数组、链表、树、图、堆）的实现与应用。

理解并实现多种算法（线性查找、二叉搜索树查找、哈希查找、排序、BFS、堆排序等）。

提升模块化编程和问题解决能力。

体验从数据加载到功能实现的系统构建过程。

1. **实验材料**

为了简化数据获取，我们将提供一个预定义的文本文件papers.txt。每行代表一篇论文，字段之间用 | 分隔。作者列表作为一个单独的字符串，作者之间用 ; 分隔。关键词列表是固定数量的独立字段，不足时用空字段 | 填充。示例行格式：

详见附件papers.txt。

我们还提供了一个简单的用户主界面，包含以下基本功能：



用户界面 (UI) 搭建：提供清晰的菜单选项，引导用户操作。

输入处理：接收用户选择，并进行基本的输入验证，防止程序因无效输入而崩溃。

模块化设计：主函数作为调度中心，将不同的功能逻辑（如加载、显示、查找、排序等）抽象为独立的“待办事项”（ToDo），为后续阶段的功能实现预留接口。每个功能都应被视为一个独立的模块，最终将实现在不同的源文件和头文件中，并通过 include 语句引入。

持续交互：完成一个功能后，程序应返回主菜单，允许用户继续选择其他操作，直到用户明确选择“退出”。

详见main.cpp。

1. **阶段一：基础数据管理与表示**

**【目标】**定义论文数据结构，并能从文件加载数据到内存。掌握结构体/类、顺序表（固定大小数组实现）、文件I/O、基本字符串处理。

**【何时使用顺序表】**当需要存储固定数量（或最大数量可预估）的同类型数据，并且主要操作是按索引访问、在末尾添加/删除时，顺序表是合适的选择。在本实验中，主论文数据（所有加载的论文）将使用顺序表存储。

**任务1.1：定义论文数据结构**

**【知识点】**

1. C++结构体定义：如何封装多个相关联的数据项。
2. 基本数据类型：整数（ID、年份、引用次数、关键词数量）、固定大小字符数组（标题、合并后的作者字符串、摘要片段）、固定大小二维字符数组（用于存储多个关键词。）
3. 顺序表的相关定义

**【任务内容】**

1. 定义一个论文结构体，包含论文ID、标题、一个用于存储所有作者的合并字符串、一个用于存储固定数量关键词的二维字符数组、记录实际关键词数量的整数、年份、引用次数和摘要片段。
2. 定义一个顺序表结构，包含一个固定大小的论文结构体数组和一个整数来表示当前存储的论文数量。
3. 在程序中声明一个全局的顺序表实例，用于存储所有论文数据。

**任务1.2:从文件加载论文数据**

**【知识点】**

1. 文件I/O：使用文件流打开、读取文本文件。
2. 字符串解析：逐行读取文件内容，使用字符串查找和截取操作，按照|分隔符提取每个字段。
3. 数据类型转换：将字符串转换为整数。
4. 错误处理：检查文件是否成功打开，处理文件读取错误，以及数据转换错误。
5. 容量检查：确保加载的论文数量不超过顺序表的最大容量。

**【主函数功能】**

1. 加载论文数据：从paper.txt加载论文的所有数据到顺序表中

**【任务内容】**

1. 打开papers.txt文件。
2. 跳过文件头行。
3. 逐行读取文件内容。对于每一行：
   1. 使用字符串查找和截取操作，按|分隔符依次提取论文ID、标题、合并后的作者字符串、固定数量的关键词字段、年份、引用次数和摘要片段。
   2. 将提取的字符串数据转换为相应的类型（例如，ID、年份、引用次数转换为整数）。
   3. 将解析出的数据填充到一个新的论文结构体实例中。
   4. 将该论文实例添加到全局顺序表的末尾，并更新其长度。
   5. 如果遇到解析错误或超出容量，则跳过当前行或停止加载。

**任务1.3：论文的添加、删除与保存**

**【知识点】**

1. 顺序表的基本操作：在末尾添加元素，通过元素移动删除指定位置元素。
2. 文件I/O：使用文件流打开、写入文本文件。
3. 用户交互：从控制台获取用户输入。
4. 字符串处理：将用户输入或内存中的数据格式化为文件所需的字符串格式。

**【主函数功能对应】**

2. 显示所有论文(简要信息)：用户选择功能2后，打印出所有论文的信息

3. 添加新论文：从输入中获取用户输入的信息，并添加到顺序表中

4. 删除论文：从顺序表中删除相关论文的所有信息

5. 保存当前数据到文件：将当前顺序表中的所有论文信息保存到文件中

**【任务内容】**

1. 实现添加新论文的逻辑：
   1. 检查全局顺序表是否已满。
   2. 提示用户输入新论文的各项信息（ID、标题、合并后的作者字符串、每个关键词、年份、引用次数、摘要片段）。
   3. 将用户输入的数据填充到新的论文结构体中。
   4. 将新论文添加到全局顺序表的末尾，并更新其长度。
2. 实现删除论文的逻辑：
   1. 提示用户输入要删除的论文ID。
   2. 在全局顺序表中查找具有该ID的论文。
   3. 如果找到，将该论文之后的所有论文向前移动一位，覆盖被删除的论文。
   4. 更新全局顺序表的长度。
3. 实现保存当前数据到文件的逻辑：
   1. 打开一个文件（例如papers.txt）。
   2. 遍历全局顺序表中的所有论文，将每篇论文的字段按照指定的分隔符格式写入文件。
4. 实现显示所有论文简要信息的逻辑：
   1. 遍历全局顺序表，逐一显示每篇论文的ID、标题、年份和引用次数。

**额外任务1.4优化删除操作**

思考在固定大小顺序表中，除了通过移动元素来删除外，是否还有其他策略（例如，标记删除）。分析不同删除策略的优缺点，尤其是在频繁删除操作下的性能影响和空间利用率。

**额外任务1.5健壮性增强**

改进文件加载和用户输入时的错误处理。例如，当文件中的某一行数据格式不正确时，如何跳过该行并记录错误，而不是直接终止加载。对于用户输入，如何确保输入的年份、引用次数是有效的数字。

1. **阶段二：高效查找**

**【目标】**利用树、哈希表和字符串匹配算法实现高效。掌握二叉搜索树(BST)、哈希表（链地址法）、KMP字符串匹配算法。

**【何时使用二叉搜索树(BST)】**当需要根据某个键（如标题）进行高效的插入、删除和查找，并且需要保持数据有序时，BST是合适的选择。

**【何时使用哈希表】**当需要根据某个键（如ID）进行极速的精确查找、插入和删除时，哈希表是合适的选择。

**【何时使用KMP算法】**当需要在长文本（如标题、摘要）中查找特定模式串时，KMP算法比朴素匹配算法更高效。

**任务2.1：构建论文索引——按标题的二叉搜索树(BST)**

**【知识点】**

1. 二叉搜索树是一种特殊的二叉树，它具有以下核心性质：（1）有序性：对于树中的任意一个节点，其左子树中所有节点的“键值”都小于该节点的“键值”，而其右子树中所有节点的“键值”都大于该节点的“键值”。在这个实验中，我们的“键值”就是论文的标题字符串。“小于”和“大于”指的是字符串的字母顺序（或称字典序）。例如，“Apple”小于“Banana”，“Banana”小于“Cat”。可以通过strcmp(str1,str2)函数实现。（2）快速查找：正是由于这种有序性，BST能够实现高效的查找。（在理想情况下）平均时间复杂度为O(logN)，N是树中节点的数量。
2. strcmp(str1,str2)。在C/C++中，我们使用strcmp函数来比较两个C风格的字符串（即以空字符\0结尾的字符数组）。它位于<string.h>头文件中。返回值：strcmp(str1,str2)<0：表示str1在字母顺序上小于str2。strcmp(str1,str2)==0：表示str1和str2完全相同。strcmp(str1,str2)>0：表示str1在字母顺序上大于str2。在构建BST时，我们将利用strcmp的返回值来判断新标题应该插入到左子树还是右子树，以及在查找时判断搜索方向。
3. 职责分离。在这个实验任务中，我们有两类数据：**原始论文数据：**存储在全局的Paper结构体顺序表（例如global\_papers）中，每项包含论文的完整信息（ID、标题、作者、年份、引用次数等）。这个顺序表是我们的“数据仓库”。**二叉搜索树的节点数据：**存储在独立的BSTNode结构体数组（例如BST\_NODES）中。这种分离的好处包括：**（1）避免数据冗余：**BSTNode只需存储标题（作为键）和指向global\_papers中实际论文的索引，而不需要复制完整的论文数据。**（2）灵活性：**你可以为同一份global\_papers数据创建多个不同的索引结构（例如，一个BST按标题查找，一个哈希表按ID查找）。如果BSTNode直接存储完整论文数据，那么创建多个索引就会导致大量的数据重复存储和内存浪费。**（3）结构纯净：**Paper结构体保持纯粹，只描述论文本身的属性，不包含任何与特定索引结构相关的字段（如left\_child\_idx）。**（4）维护效率：**当论文数据发生变化时，通常只需要更新global\_papers顺序表，然后（如果需要）重建或更新索引结构，而不需要修改所有地方的数据。

**【主函数功能对应】**

1. 加载论文数据：在论文数据成功加载到global\_papers顺序表并更新当前论文数量后，应调用 buildBSTFromPapers 函数来构建BST。

3. 添加新论文：在用户添加新论文并将其存储到 global\_papers 顺序表后，应考虑重新调用 buildBSTFromPapers 函数来重建 BST，以确保新论文被正确索引。

6. 按标题查找论文(BST)：在主菜单中选择此项时，应提示用户输入标题，然后调用 searchBST 函数进行查找，并根据返回的索引显示 global\_papers 数组中对应的论文详细信息。

**【任务内容】**

1. 定义BST节点结构(BSTNode)：
   1. 定义一个结构体，包含一个固定大小的字符数组来存储论文标题（作为键）。
   2. 包含一个整数，用于存储该论文在全局Paper结构体数组（global\_papers）中的索引。
   3. 包含两个整数left\_child\_idx和right\_child\_idx，分别存储其左子节点和右子节点在BST\_NODES数组中的索引。使用-1表示空子节点。
2. 设置全局变量：
   1. 声明一个固定大小的BSTNode结构体数组（例如BST\_NODES），作为所有BST节点的存储池。
   2. 声明一个整数变量（例如bst\_root\_index）来存储BST根节点在BST\_NODES数组中的索引，初始值为-1（表示空树）。
   3. 声明一个整数变量（例如next\_free\_bst\_node\_index）来记录BST\_NODES数组中下一个可用的空闲位置。
3. 实现节点分配逻辑(getNewBSTNodeIndex)：
   1. 编写一个函数，负责从BST\_NODES数组中“分配”一个未使用的位置作为新节点。
   2. 该函数应返回新节点的数组索引，并更新next\_free\_bst\_node\_index。
   3. 需要包含容量检查，如果BST\_NODES数组已满，则返回一个错误指示（例如-1）。
   4. 新分配的节点的left\_child\_idx和right\_child\_idx应初始化为-1。
4. 实现插入逻辑(insertPaperIntoBST)：
   1. 编写一个函数，接收待插入论文的标题和其在global\_papers数组中的索引。
   2. 该函数应从bst\_root\_index开始，递归地或迭代地遍历BST。
   3. 在每一步，使用strcmp函数比较待插入标题与当前节点的标题。
   4. 根据比较结果，决定是向左子树还是向右子树继续。
   5. 当找到一个空位置（即当前节点的left\_child\_idx或right\_child\_idx为-1）时，通过调用节点分配逻辑，创建一个新节点，将论文标题和数据索引存储进去，并将新节点的索引连接到父节点的相应子节点索引上。
   6. 考虑如何处理标题重复的情况（例如，简单地忽略或输出警告）。
5. 实现查找逻辑(searchBST)：
   1. 编写一个函数，接收要查找的目标论文标题。
   2. 该函数应从bst\_root\_index开始，递归地或迭代地遍历BST。
   3. 在每一步，使用strcmp函数比较目标标题与当前节点的标题。
   4. 根据比较结果，决定是向左子树还是向右子树继续。
   5. 如果找到匹配的标题，返回该节点存储的论文在global\_papers数组中的索引。
   6. 如果遍历到空子节点（索引为-1）仍未找到，则返回一个特殊值（例如-1）表示未找到。
6. 实现构建/重建逻辑(buildBSTFromPapers)：
   1. 编写一个函数，用于在数据加载或修改后，构建或重建整个BST。
   2. 该函数首先应调用一个重置BST的辅助函数（将bst\_root\_index设为-1，next\_free\_bst\_node\_index设为0）。
   3. 然后，遍历global\_papers数组中的所有论文，逐一调用插入逻辑(insertPaperIntoBST)将每篇论文的标题及其数据索引添加到BST中。
   4. 建议在数据加载后或大量数据修改后完整重建BST，以简化删除和修改操作对树结构的维护复杂性。

**任务 2.2：实现快速查找——哈希表按ID**

**【知识点】**

1. 哈希函数：将论文ID映射到哈希表中的一个索引。
2. 冲突解决：链地址法，使用数组模拟链表来处理多个键映射到同一索引的情况。
3. 哈希表节点结构：包含论文ID、论文在全局顺序表中的索引，以及冲突链中下一个节点在预分配节点池中的索引。
4. 哈希表插入操作：计算哈希值，将新节点插入到对应哈希桶的链表头部。
5. 哈希表查找操作：计算哈希值，遍历对应哈希桶的链表查找目标ID。
6. 数组模拟链表：使用固定大小的数组作为节点池，通过数组索引来模拟链表连接。

**【主函数功能对应】**

7. 按ID查找论文 (哈希表)：提示用户输入ID查找论文

**【任务内容】**

1. 定义一个哈希表节点结构，包含论文ID、论文在全局顺序表中的索引，以及表示冲突链中下一个节点索引的整数。
2. 实现从预分配节点池中分配新哈希表节点的逻辑。
3. 实现一个哈希函数，将论文ID映射到哈希表的索引。
4. 实现将论文ID及其在全局顺序表中的索引插入哈希表的逻辑（使用链地址法处理冲突）。
5. 实现根据论文ID在哈希表中查找论文的逻辑，返回其在全局顺序表中的索引。

**任务 2.3：高级字符串查找——KMP 算法**

**【知识点】**

1. KMP 算法原理：理解 next (或 LPS) 数组的作用，以及如何利用它避免不必要的回溯。
2. next 数组构建：实现根据模式串计算 next 数组的逻辑。
3. KMP 匹配过程：实现使用 next 数组在主串中查找模式串的逻辑。
4. 字符串操作：C风格字符串的比较和遍历。

**【主函数功能对应】**

8. 按内容模糊查找论文 (KMP)：提示用户输入关键词/短语，使用模糊查找匹配相应的论文

**【任务内容】**

1. 实现一个buildNextArray函数，输入一个模式串pattern，输出其对应的next数组。
2. 实现一个kmpSearch函数，输入主串 text、模式串 pattern 和 next 数组，返回模式串在主串中第一次出现的位置索引。如果模式串多次出现，可以扩展为返回所有出现的位置。
3. 实现一个功能函数，提示用户输入一个搜索关键词/短语。
4. 遍历全局顺序表中的所有论文，使用kmpSearch函数在每篇论文的Title字段和/或AbstractSnippet字段中查找该关键词/短语。
5. 显示所有匹配的论文简要信息。

**额外任务2.4：BST 性能分析与可视化：**

分析：记录BST插入和查找操作的平均比较次数，并与线性查找进行对比。讨论在最坏情况下（例如，按标题字母顺序插入所有论文）BST 性能退化为O(N)的原因。

可视化：尝试通过简单的文本输出（例如，使用缩进表示树的层次）来打印BST的结构，帮助理解树的平衡性。

**额外任务2.5：哈希函数优化与冲突分析：**

优化：尝试设计并实现至少两种不同的哈希函数（例如，除法哈希法、乘法哈希法）用于论文ID，并比较它们在实际数据上的冲突率。

分析：统计哈希表中每个哈希桶的链表长度，计算平均链表长度和最大链表长度，以此评估哈希表的性能。

**额外任务2.6：KMP 算法的扩展应用：**

查找所有匹配：修改KMP查找函数，使其能够找出模式串在主串中出现的所有起始位置，而不仅仅是第一次出现。

多模式匹配思考：查阅资料，思考如果要在文本中同时查找多个模式串（例如，多个关键词）时，KMP算法的局限性，并了解是否存在更高效的算法。

1. **阶段三：排序与有序列表操作**

**【目标】**理解并实现多种排序算法，并掌握有序列表的合并技术。

**任务 3.1: 论文排序 - 按年份、引用次数或标题**

**【知识点】**

1. 排序算法：选择一到两种高效的排序算法，例如快速排序。
2. 原地排序：直接修改全局顺序表中的元素顺序。
3. 比较函数：根据不同的排序标准（年份、引用次数、标题）实现不同的比较逻辑。

**【主函数功能对应】**

9. 按年份降序排序并显示

10. 按引用次数降序排序并显示

11. 按标题字母顺序排序并显示

**【****任务内容】**

1. 任选一种排序算法实现按年份降序排序的逻辑。
2. 任选一种排序算法实现按引用次数降序排序的逻辑。
3. 任选一种排序算法实现按标题字母顺序升序排序的逻辑。
4. 排序完成后，调用显示简要论文信息的逻辑来展示结果。

**任务 3.2: 合并两个有序论文列表**

**【知识点】**

1. 有序列表合并算法：理解如何利用两个输入列表的有序性，通过双指针（或类似机制）高效地构建一个合并后的有序列表。
2. 双指针技术：在两个有序列表中同时移动指针进行比较和选择。
3. 新的顺序表创建与填充：将合并结果存储到一个新的顺序表。

**【主函数功能对应】**

12. 合并两个有序论文列表：读取有序的论文数据与当前的顺序表进行合并

**【任务内容】**

1. 先将当前的全局论文列表按某个标准（例如，论文年份）进行排序。
2. 手动创建或从一个简化的文件中加载一个小的、已按相同标准排序的“新导入论文列表”（例如，包含几篇新论文）。
3. 实现一个mergeSortedPaperLists函数，接收两个已按相同标准（如论文年份）排序的论文顺序表作为输入。
4. 该函数应返回一个新的顺序表，其中包含两个输入列表的所有论文，并且仍然保持按相同标准排序。
5. 显示合并后的论文列表的简要信息。

**额外任务3.3 比较不同排序算法：**

实现： 除了快速排序，再实现一种不同的排序算法（例如，插入排序或冒泡排序）。

分析： 比较快速排序和新实现的排序算法在不同数据规模（例如，10篇、100篇、1000篇论文）下的性能差异（通过统计比较次数或运行时间）。讨论它们各自的适用场景。

**额外任务3.4 有序列表合并的变体：**

去重合并： 修改合并函数，使其在合并过程中自动去除重复的论文（例如，ID 相同的论文只保留一篇）。

多列表合并思考： 思考如何将 K 个已排序的论文列表合并成一个大的有序列表，而不是仅仅两个。简要描述你的思路（例如，使用优先队列）。

1. **阶段四：关联分析与推荐**

**【目标】**利用图结构表示论文间的关联，并实现简单的推荐逻辑。

**【何时使用图】**当数据之间存在复杂的多对多关系，需要进行路径查找、连通性分析等时，图是合适的选择。

**【何时使用队列】**实现广度优先搜索 (BFS) 等需要先进先出 (FIFO) 访问顺序的算法。

**【何时使用堆/优先队列】**实现 Top-N 推荐等需要高效获取最大/最小元素的场景。

**任务 4.1: 构建论文关联图 - 基于关键词**

**【知识点】**

1. 图的概念：节点（论文）、边（关联）。
2. 邻接表表示法：使用固定大小的数组作为头部数组，每个头部指向一个由固定大小数组索引模拟的链表，表示邻居。
3. 图的构建：遍历所有论文，根据关键词共享情况添加边。

**【主函数功能】**

该任务是后续图遍历和推荐功能的基础，不直接对应主菜单选项，但在加载数据后或数据修改后需要调用以更新图结构。

**【任务内容】**

1. 定义一个边结构，包含目标论文在全局顺序表中的索引，以及邻接链表中下一个边在预分配边池中的索引。
2. 实现从预分配边池中分配新边节点的逻辑。
3. 实现向图中添加一条无向边的逻辑（需要添加双向边）。
4. 实现构建论文关联图的逻辑：
   1. 清空图的邻接表结构。
   2. 遍历全局顺序表中的每篇论文。
   3. 对于任意两篇不同的论文，检查它们是否至少共享一个关键词。
   4. 如果共享关键词，则在它们之间添加一条无向边。
   5. 为了避免重复添加边，可以使用一个辅助的布尔数组来标记已经添加过的边对。

**任务 4.2: 查找相关论文 - BFS**

**【知识点】**

1. 图遍历算法：广度优先搜索 (BFS)。
2. 队列：BFS 使用队列 (FIFO) 来管理待访问的节点。需要使用固定大小数组模拟队列结构，并实现入队、出队、判断空等基本操作。
3. 访问标记：使用一个布尔数组来记录每个论文是否已被访问，避免循环和重复处理。

**【主函数功能】**

13. 查找相关论文 (BFS)：用户输入论文标题或者ID，查找与当前论文相关的论文（基于关键词）

**【****任务内容】**

1. 定义一个队列结构，包含一个固定大小的整数数组和表示队头、队尾、当前大小的整数。实现队列的初始化、入队、出队、判断空/满等操作。
2. 实现查找相关论文（BFS）的逻辑：
   1. 提示用户输入一个起始论文ID。
   2. 根据ID找到该论文在全局顺序表中的索引。
   3. 初始化一个布尔数组，标记所有论文为未访问。
   4. 创建一个队列，将起始论文的索引入队，并标记为已访问。
   5. 当队列不为空时：
      1. 从队列中取出一个论文索引。
      2. 显示该论文的简要信息。
      3. 遍历该论文的所有邻居（通过图的邻接表）。
      4. 如果邻居未被访问，则将其入队并标记为已访问。

**任务 4.3: 简单的推荐功能 - 基于关键词匹配（Top-N）**

**【知识点】**

1. 优先队列/堆：实现 Top-N 问题的核心数据结构。需要使用固定大小数组模拟最小堆，并实现插入、向上/向下调整堆等操作。
2. 相似度计算：定义两组关键词之间的相似度计算方法（例如，共享关键词的数量）。

**【主函数功能】**

14. 获取关键词推荐 (Top-N)：用户指定一篇“起始论文”，然后遍历所有其他论文，计算它们与起始论文之间的关键词相似度（例如，通过统计共享关键词的数量）。为了高效地找出相似度最高的 N 篇论文，需要维护一个固定大小的最小堆。最终，堆中保留的 N 篇论文将作为推荐结果，它们是与起始论文关键词匹配度最高的 N 篇论文。

**【****任务内容】**

1. 定义一个推荐项结构，包含论文在全局顺序表中的索引和该论文的相似度分数。
2. 定义一个最小堆结构，包含一个固定大小的推荐项数组和堆的当前大小。实现堆的初始化、插入元素、向上/向下调整堆等操作。
3. 实现获取关键词推荐（Top-N）的逻辑：
   1. 提示用户输入一个起始论文ID。
   2. 根据ID找到目标论文在全局顺序表中的索引。
   3. 初始化一个大小为 N 的最小堆。
   4. 遍历全局顺序表中的所有其他论文：
      1. 计算当前论文与目标论文之间的关键词相似度（例如，统计共享关键词的数量）。
      2. 如果相似度大于0：
      3. 如果堆未满，将当前论文及其相似度作为一个推荐项插入堆中。
      4. 如果堆已满，且当前论文的相似度大于堆顶元素（最小相似度），则替换堆顶元素并调整堆。
      5. 最终，堆中剩下的 N 篇论文就是推荐结果。将堆中元素取出到一个临时数组，然后按相似度降序排序并显示这些推荐论文的简要信息和相似度分数。

**额外任务4.4：图的深度优先搜索 (DFS)：**

实现： 实现图的深度优先搜索 (DFS) 算法，用于查找相关论文。

对比： 比较 BFS 和 DFS 在查找相关论文时的行为差异（例如，搜索路径、发现顺序），并讨论它们各自的优缺点和适用场景。

**额外任务4.5：关键词相似度加权：**

改进： 改进关键词相似度的计算方法。例如，可以为不同的关键词赋予不同的权重（遍历全局顺序表中的所有论文。对于每个唯一的关键词，统计它在多少篇论文的关键词列表中出现过。根据出现频率为不同的关键词赋予权重），或者考虑关键词的语义相似度（仅需概念性讨论）。

实现：修改推荐功能，使其使用加权后的相似度来计算 Top-N 推荐。

**额外任务4.6：堆排序的实现：**

实现：利用已实现的最小堆（或改造为最大堆），实现一个完整的堆排序算法，用于对论文列表进行排序。

分析：比较堆排序与快速排序在性能上的差异和各自的特点。