# 文献管理与分析系统实验讲解

## 阶段一：基础数据管理与表示

### 任务 1.1：定义论文数据结构

**功能简介：**此任务是系统的基础，它定义了单个论文的结构体（封装了论文的各项属性，如ID、标题、作者、关键词等），以及用于存储所有论文的顺序表（一个固定大小的论文结构体数组）。这是后续所有数据操作的基石。

**知识点介绍：**

1. C++结构体定义：学习如何使用`struct`关键字将多个相关联的不同类型数据项封装成一个自定义类型。
2. 基本数据类型：掌握如何选择合适的C++基本数据类型（如`int`、`char[]`）来存储论文的各种属性。特别是固定大小字符数组和二维字符数组在C语言风格字符串处理中的应用。
3. 顺序表的相关定义：理解顺序表（基于固定大小数组）的基本概念，包括其数据存储方式（连续内存）、容量限制以及如何通过一个整数变量来追踪当前存储的元素数量。

**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。

**示例代码：**

// 定义一些常量，用于固定大小数组的长度

#define MAX\_TITLE\_LENGTH 256

#define MAX\_AUTHORS\_LENGTH 512

#define MAX\_KEYWORDS\_PER\_PAPER 6 // 固定数量的关键词

#define MAX\_KEYWORD\_LENGTH 64 // 每个关键词的最大长度

#define MAX\_ABSTRACT\_LENGTH 1024

#define MAX\_PAPERS 1000 // 全局顺序表的最大论文数量

/\*\*

\* @brief 定义单个学术论文的数据结构。

\*/

struct Paper {

int id; // 论文ID

char title[MAX\_TITLE\_LENGTH]; // 论文标题

char authorsString[MAX\_AUTHORS\_LENGTH]; // 合并后的作者字符串，作者之间用 ';' 分隔

char keywords[MAX\_KEYWORDS\_PER\_PAPER][MAX\_KEYWORD\_LENGTH]; // 固定数量的关键词二维字符数组

int actualKeywordCount; // 实际存储的关键词数量

int year; // 发表年份

int citations; // 引用次数

char abstractSnippet[MAX\_ABSTRACT\_LENGTH]; // 摘要片段

};

/\*\*

\* @brief 定义存储论文的顺序表结构。

\*/

struct PaperList {

Paper papers[MAX\_PAPERS]; // 固定大小的论文结构体数组

int count; // 当前存储的论文数量

};

// 在程序中声明一个全局的顺序表实例

PaperList global\_papers;

### 任务 1.2：从文件加载论文数据

**功能简介：**实现将预定义的`papers.txt`文件中的学术论文数据读取到内存中，并存储到全局顺序表`global\_papers`中。这是系统启动后获取初始数据的主要方式，确保系统能够处理外部数据源。

**主函数功能对应：** `1.加载论文数据`

**知识点介绍：**

1. 文件I/O：学习如何使用C++文件流（`ifstream`）打开、读取文本文件，包括逐行读取文件内容。
2. 字符串解析：掌握基本的C++字符串处理技术，例如使用`string`类的查找（`find`）和截取（`substr`）方法，或者C风格字符串函数（如`strtok`），根据分隔符（`|`）从每行中提取不同的字段。
3. 数据类型转换：学习如何将从文件中读取的字符串数据转换为相应的数值类型。
4. 错误处理：理解在文件操作中进行错误检查的重要性，例如检查文件是否成功打开，处理文件读取过程中可能出现的错误，以及数据转换失败的情况。
5. 容量检查：学习在向固定大小数据结构中添加元素时，如何检查是否超出其最大容量，防止缓冲区溢出。

**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。

**期望输入：**

1. 用户操作：在主菜单选择`1`。
2. 隐式输入：`papers.txt`文件（需存在且格式正确）。

**期望输出：**

1. 成功：显示“数据加载成功，共加载X篇论文”的提示信息。
2. 失败：显示“文件打开失败”、“数据解析错误”或“顺序表容量不足”等错误信息。
3. 内部：全局顺序表`global\_papers`被填充。

### 任务 1.3：论文的添加、删除与保存

**功能简介：**提供对论文数据的基本管理功能。允许用户查看所有论文的简要信息，手动输入并添加新论文到系统中，根据论文ID从系统中删除现有论文，并将当前内存中的所有数据保存回文件，实现数据的持久化。

**主函数功能对应：**

2.显示所有论文(简要信息)

3.添加新论文

4.删除论文

5.保存当前数据到文件

**知识点介绍：**

1. 顺序表的基本操作：掌握在顺序表（数组）末尾添加元素（通过增加长度和赋值），以及通过元素移动（将删除点之后的元素前移）来删除指定位置元素的操作。
2. 文件I/O：学习如何使用C++文件流（`ofstream`）打开、写入文本文件，将内存中的结构化数据按照特定格式写入到文件中。
3. 用户交互：学习如何从控制台获取用户输入（使用`cin`或`getline`），并进行基本的输入处理。
4. 字符串处理：学习如何将内存中的数据（例如结构体字段）格式化为文件所需的字符串格式，并使用指定的分隔符连接。

**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。

**期望输入：**

1. 显示所有论文：用户在主菜单选择`2`。
2. 添加新论文：用户在主菜单选择`3`，然后根据提示依次输入新论文的ID、标题、作者字符串、关键词（多个）、年份、引用次数、摘要片段。
3. 删除论文：用户在主菜单选择`4`，然后输入要删除的论文ID。
4. 保存数据：用户在主菜单选择`5`。

**期望输出：**

1. 显示所有论文：打印出所有论文的ID、标题、年份和引用次数的列表。
2. 添加新论文：成功则显示“论文添加成功”，失败则显示“顺序表已满”或“输入格式错误”。
3. 删除论文：成功则显示“论文IDXXXX已删除”，失败则显示“未找到论文IDXXXX”。
4. 保存数据：成功则显示“数据已成功保存到papers.txt”，失败则显示“文件写入失败”。

## 阶段二：高效查找

### 任务 2.1：构建论文索引——按标题的二叉搜索树(BST)

**功能简介：** 构建一个二叉搜索树（BST）作为论文标题的索引，实现根据标题进行高效查找。树的节点仅存储论文标题（作为键）和其在全局顺序表中的索引，避免了数据的冗余存储，提高了内存利用率和灵活性。

**主函数功能对应：**

1.加载论文数据`(内部调用`buildBSTFromPapers`构建BST)

3.添加新论文`(内部调用`insertPaperIntoBST`或重建BST)

6.按标题查找论文(BST)

**知识点介绍：**

1. 二叉搜索树（BST）的定义与性质：理解BST的核心特性，即左子树节点键值小于根节点，右子树节点键值大于根节点，以及这种有序性如何支持高效的查找、插入操作（理想情况下O(logN)）。
2. `strcmp(str1,str2)`函数的使用：掌握C风格字符串比较函数`strcmp`的用法，及其返回值（<0,==0,>0）在决定BST遍历方向或插入位置时的应用。
3. 职责分离原则：理解将原始论文数据（存储在`global\_papers`顺序表）与索引数据（存储在`BST\_NODES`数组中，仅包含标题和指向`global\_papers`的索引）分离存储的好处，包括避免数据冗余、提高灵活性和维护效率。
4. 数组模拟链表/树：学习如何使用固定大小的数组来模拟树的节点，并通过数组索引而非指针来表示节点间的父子关系。

**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。

**期望输入：**

1. 构建/重建BST：隐式触发（数据加载或添加新论文后）。
2. 按标题查找论文：用户在主菜单选择`6`，然后输入要查找的论文标题。

**期望输出：**

1. 构建/重建BST：无（内部数据结构`BST\_NODES`和`bst\_root\_index`被填充/更新）。
2. 按标题查找论文：
3. 成功：显示匹配论文的详细信息（ID、标题、作者、关键词、年份、引用次数、摘要）。
4. 失败：显示“未找到标题为'XXX'的论文”。

### 任务 2.2：实现快速查找——哈希表按ID

**功能简介：** 构建一个哈希表，以论文ID为键，实现对论文的极速精确查找。采用链地址法解决哈希冲突，确保在大量数据下仍能保持高效的查找、插入和删除性能。

**主函数功能对应：** `7.按ID查找论文(哈希表)`

**知识点介绍：**

1. 哈希函数：理解哈希函数的概念，学习如何设计一个简单的哈希函数，将论文ID（或其他键）映射到哈希表中的一个数组索引。
2. 冲突解决：链地址法：掌握当多个不同的键映射到同一个哈希索引时，如何使用链表（在这里是数组模拟的链表）来存储这些冲突的元素。
3. 哈希表节点结构：定义哈希表节点，包含论文ID、论文在全局顺序表中的索引，以及指向冲突链中下一个节点的索引。
4. 哈希表插入与查找操作：学习哈希表的基本操作流程：计算哈希值，定位哈希桶，然后根据链地址法在链表中插入或查找元素。
5. 数组模拟链表：再次实践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，避免动态内存分配的复杂性。

**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。

**期望输入：**

按ID查找论文：用户在主菜单选择`7`，然后输入要查找的论文ID。

**期望输出：**

1. 成功：显示匹配论文的详细信息。
2. 失败：显示“未找到ID为'XXX'的论文”。

### 任务 2.3：高级字符串查找——KMP 算法

**功能简介：** 利用KMP(Knuth-Morris-Pratt)算法实现高效的模糊查找功能。用户输入一个搜索关键词或短语（模式串），系统在所有论文的标题和/或摘要片段（主串）中查找该模式串的所有匹配项，并显示匹配的论文信息。KMP算法通过避免不必要的回溯，显著提高了字符串匹配的效率。

**主函数功能对应：** `8.按内容模糊查找论文(KMP)`

**知识点介绍：**

1. KMP算法原理：理解KMP算法的核心思想，即在模式串与主串不匹配时，如何利用模式串自身的特性（通过`next`或`LPS`数组）来确定模式串下一次应该从哪里开始比较，从而避免主串指针的回溯。
2. `next`数组构建：学习如何根据给定的模式串计算其`next`数组，该数组存储了模式串的每个前缀的最长公共前后缀的长度。
3. KMP匹配过程：掌握如何使用`next`数组在主串中高效地查找模式串的匹配过程。
4. 字符串操作：实践C风格字符串（`char[]`）的比较、遍历和长度计算等基本操作。

**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。

**期望输入：**

用户在主菜单选择`8`，然后输入要搜索的关键词/短语。

**期望输出：**

1. 显示所有标题或摘要中包含该关键词/短语的论文简要信息列表。
2. 如果未找到匹配项，显示“未找到包含'XXX'的论文”。

## 阶段三：排序与有序列表操作

### 任务 3.1：论文排序 - 按年份、引用次数或标题

**功能简介：** 实现对全局论文列表进行排序的功能。用户可以选择按年份降序、引用次数降序或标题字母顺序升序排列论文，并显示排序后的简要信息。这有助于用户以不同维度浏览和分析论文数据。

**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

**知识点介绍：**

1. 排序算法：学习并实现一到两种高效的排序算法，例如快速排序(QuickSort)。理解其基本原理（分治、枢轴选择、分区操作）和平均时间复杂度O(NlogN)。
2. 原地排序：理解原地排序的概念，即排序过程直接在原始数据结构（全局顺序表）上进行，不需要额外的存储空间（除了递归栈空间）。
3. 比较函数：学习如何根据不同的排序标准（整数比较用于年份和引用次数，`strcmp`用于标题字符串）实现不同的比较逻辑，并将其集成到排序算法中。

**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。

**期望输入：**

用户在主菜单选择`9`、`10`或`11`。

**期望输出：**

显示按指定标准排序后的论文简要信息列表。

### 任务 3.2：合并两个有序论文列表

**功能简介：** 演示如何高效地合并两个已经按相同标准排序的论文列表，生成一个新的、同样有序的合并列表。这模拟了将新导入的数据整合到现有数据中，同时保持数据有序性的常见场景。

**主函数功能对应：**12.合并两个有序论文列表

**知识点介绍：**

1. 有序列表合并算法：理解合并两个有序列表的原理，即如何利用输入列表已有的有序性，通过一次线性扫描高效地构建合并后的有序列表，避免重新排序整个数据集。
2. 双指针技术：学习并应用双指针技术，即同时维护两个指针（或索引）分别指向两个输入列表的当前元素，通过比较它们的大小来决定哪个元素应该先加入到合并后的列表中。
3. 新的顺序表创建与填充：学习如何创建一个新的顺序表来存储合并结果，并逐步将元素从两个输入列表添加到新列表中。

**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。

**期望输入：**

1. 用户在主菜单选择`12`。
2. 隐式输入：当前的`global\_papers`列表（需预先按某个标准排序）和一个模拟的“新导入论文列表”（需预先按相同标准排序）。

**期望输出：**

显示合并后（并保持有序）的论文简要信息列表。

## 阶段四：关联分析与推荐

### 任务 4.1：构建论文关联图 - 基于关键词

**功能简介：** 根据论文之间共享关键词的情况，构建一个无向图。图中的节点是论文，边表示两篇论文至少共享一个关键词。这个图结构是进行后续相关论文查找和更复杂推荐功能的基础。

**主函数功能对应：** 无（内部功能，通常在数据加载后或数据修改后需要调用以更新图结构）。

**知识点介绍：**

1. 图的概念：理解图的基本组成部分——节点（或顶点，这里是论文）和边（表示节点之间的关系，这里是关键词共享）。
2. 邻接表表示法：学习图的常用存储方式之一——邻接表。理解如何使用固定大小的数组作为头部数组（每个索引对应一篇论文），每个头部指向一个由固定大小数组索引模拟的链表，该链表存储了与当前论文相邻的所有论文。
3. 图的构建：掌握如何遍历所有论文，并对于任意两篇论文，检查它们是否满足建立边的条件（至少共享一个关键词），然后向图中添加无向边（即在两个方向上都添加边）。

**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。

**期望输入：**

1. 隐式触发：数据加载或修改后。
2. 隐式输入：`global\_papers`列表中的论文关键词数据。

**期望输出：**无（内部图的邻接表结构被填充）。

### 任务 4.2：查找相关论文 - BFS

**功能简介：** 利用广度优先搜索(BFS)算法，从用户指定的起始论文开始，在论文关联图中查找并显示所有通过关键词关联的“相关”论文。BFS算法能够以“层”的形式发现与起始论文距离递增的相关论文。

**主函数功能对应：**13.查找相关论文(BFS)

**知识点介绍：**

1. 图遍历算法：广度优先搜索(BFS)：理解BFS的基本原理，即从起始节点开始，先访问所有直接邻居，再访问邻居的邻居，以此类推，逐层向外扩展。
2. 队列：理解队列(Queue)作为一种先进先出(FIFO)数据结构在BFS中的核心作用，它用于存储待访问的节点。学习如何使用固定大小数组模拟队列结构，并实现入队(enqueue)、出队(dequeue)、判断空/满等基本操作。
3. 访问标记：学习使用一个布尔数组（`visited`数组）来记录每个论文节点是否已被访问过，以避免在图遍历中陷入循环（对于有环图）和重复处理节点。

**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。

**期望输入：**

用户在主菜单选择`13`，然后输入起始论文的ID。

**期望输出：**

1. 显示从起始论文开始，通过BFS遍历到的所有相关论文的简要信息列表。
2. 如果起始论文未找到，显示“未找到起始论文IDXXXX”。

### 任务 4.3：简单的推荐功能 - 基于关键词匹配（Top-N）

**功能简介：** 为用户指定的起始论文推荐Top-N篇最相似的论文。相似度通过统计两篇论文之间共享关键词的数量来计算。为了高效地找出相似度最高的N篇论文，系统会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。

**主函数功能对应：** `14.获取关键词推荐(Top-N)`

**知识点介绍：**

1. 优先队列/堆：理解优先队列（PriorityQueue）的概念，以及如何使用堆（Heap，这里是最小堆）作为其底层数据结构来高效地解决Top-N问题。学习固定大小数组模拟最小堆的实现，包括插入元素、向上调整(heapify-up)和向下调整(heapify-down)堆等核心操作。
2. 相似度计算：学习如何定义和实现两组关键词之间的相似度计算方法。最简单的例子是统计它们之间共享关键词的数量（集合交集的大小）。
3. Top-N算法思想：掌握使用最小堆来维护“当前最大的N个元素”的通用算法思想：当堆未满时直接插入；当堆已满且新元素比堆顶元素（当前N个中最小的）更大时，替换堆顶并调整堆。

**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。

**期望输入：**

1. 用户在主菜单选择`14`，然后输入起始论文的ID。
2. 用户输入推荐数量N。

**期望输出：**

1. 显示一个按相似度降序排列的Top-N推荐论文列表，每篇论文包含简要信息和相似度分数。
2. 如果起始论文未找到，显示“未找到起始论文IDXXXX”。