烏 有 理 工 大 學

No Such University of Technology

课程设计报告书

科学文献管理系统

学　　院	计算机科学与艺术学院
专　　业	计算机科学与艺术
学生姓名	尼莫先生
学生学号	2019********
指导教师	Prof. Outis
课程编号	045114514
课程学分	1.0
起始日期	2021 年 3 月 1 日

教师评语	教师签名: 日　　期:
成绩评定	
备注	

科学文献管理系统

1 选题背景

文献管理软件是学者或者作者用于记录、组织、调阅引用文献的计算机程序. 一旦引用文献被记录, 就可以重复多次地生成文献引用目录. 例如, 在书籍、文章或者论文当中的参考文献目录. 科技文献的快速增长促进了文献管理软件的发展.

本设计期望实现一个基于朴素 K-V 数据库（不使用外部数据库）的科研文献管理系统. 该系统的功能应当有:

基本搜索功能 根据作者名或论文标题搜索, 能展示该论文的其他相关信息.

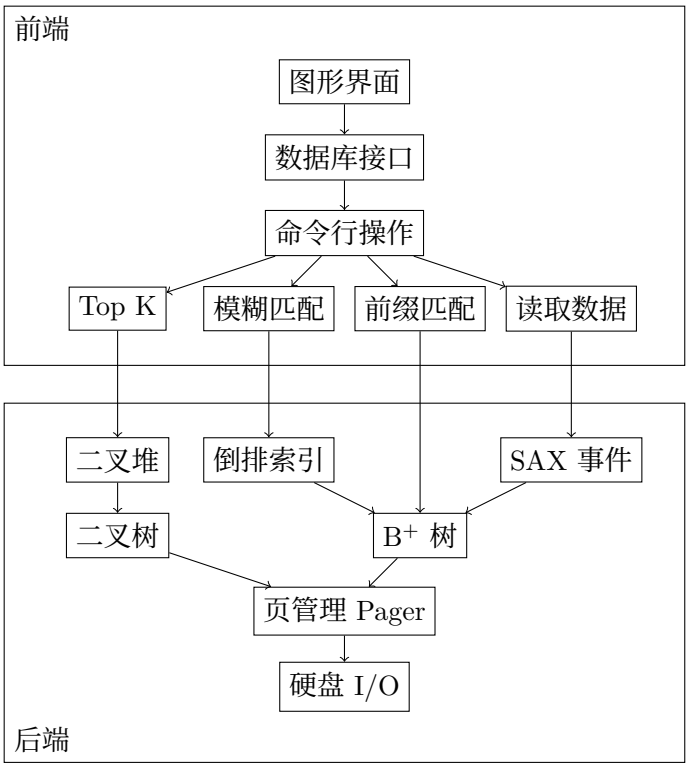
作者统计功能 输出写文献最多的前 K 名作者.

模糊搜索功能 给定若干个关键字, 能快速搜索到题目中包含该关键字的文献信息.

图形界面显示 通过图形界面展示作者之间合作关系及其相关文献信息.

2 方案论证

本选题的设计如下图所示:



2.1 对 XML 文件的处理

C++ 标准库中不提供 XML 解析器，所以必须选用另外的库．本设计选用 libxml2 库来完成与 XML 文件相关的一系列操作．

对 XML 文件的解析一般有两种方法，即 SAX 和 DOM．DOM 以一个分层的对象模型来映射 XML 文档，当数据比较复杂时，一般选用 DOM 解析．SAX 是事件驱动的，也即我们并不需要把整个文档读入内存．SAX 在读取 XML 流的时候同时处理它们，当数据量比较多时，一般选用 SAX 解析．

处理 DOM 的时候，我们需要读入整个 XML 文档并在内存中构建整棵 DOM 树，在内存中构建这样的树涉及大量的开销．本设计所需的 XML 文件大小超过 2GB，使用 DOM 一次性解析显然效率不佳．

本设计在构建索引的时候使用 SAX 解析，解析的结果是一篇文献在 XML 文件中的位置，我们将这个位置（`pos`，`len`）作为值存入 B^+ 树索引中．构建索引只需一次．

索引构建完成后，搜索一篇文献得到（`pos`，`len`）的二元组，然后直接在 XML 文件的 `pos` 位置读取长度为 `len` 的一部分，即一篇文献．我们默认一篇文献的 XML 数据量不会很大，所以本设计在后续查找解析的时候采用更加灵活的 DOM 解析，并根据需要将结果按照树形的方式显现在屏幕上．

2.2 Pager

构建内存中的 B^+ 树索引是容易的，但仅仅构建内存中的索引不能完成本设计的要求．内存用量过大是不可接受的，同时我们也希望在程序关闭后再次打开时无需重新读入大量数据．对于一个可使用的程序来说，在硬盘上持久地保存已构建的索引是重要的．本设计使用 Pager 来协助完成一系列硬盘读写操作．Pager 是 B^+ 树和硬盘读写的中间层，由此来完成对硬盘上某条数据的增删查操作．Pager 提供了 `save` 和 `recover` 方法来实现硬盘读写．

构建索引时，Pager 同时完成硬盘的写入，写入完成后，内存中的索引同时丢弃，这样可以保证同一时刻内存使用量是可以接受的．进行搜索时，程序借助 Pager 来实现从磁盘中提取数据．

2.3 B^+ 树

科研文献管理系统需要处理大量数据，所以必须建立适合的索引．本设计选择采用 B^+ 树索引．索引存储在硬盘上，方便后续读取．

B^+ 树是一种树数据结构，通常用于数据库和操作系统的文件系统中．B^+ 树的特点

是能够保持数据稳定有序，其插入与修改拥有较稳定的对数时间复杂度．B$^+$ 树元素自底向上插入，这与二叉树恰好相反．B$^+$ 树在节点访问时间远远超过节点内部访问时间的时候，比可作为替代的实现有着实在的优势．这通常在多数节点在次级存储比如硬盘中的时候出现．通过最大化在每个内部节点内的子节点的数目减少树的高度，平衡操作不经常发生，而且效率增加了．

一般来讲，一个容器应该具备插入、删除、查找、修改四种功能．但由于本设计是用于管理已有科研文献的，所以只需要着重完成插入和查找功能．虽然本设计也能实现修改，但修改一次会导致 XML 文件的索引偏移，需要重新花时间读入．虽然本设计也能实现删除，但所谓删除仅仅是打上一个「已删除」的标记，并没有真正在 B$^+$ 树中删除内容本身．

2.3.1 插入

本设计允许用户从 XML 文件向 B$^+$ 树索引中批量插入数据，修改源代码也可以手动插入单条数据（用于测试）．

首先查找待插入结点的位置，然后试图把值插入到这个结点中．如果这个结点有过多结点（`state == overflow`），则将其分裂成两个结点 `left_child` 和 `right_child`，每个结点都有最小数目的元素．然后递归向上继续处理，直到根结点．如果根结点被分裂，则在其之上创建一个新根结点．元素的最小和最大数目必须满足最小数不小于最大数的一半．

B$^+$ 树的插入算法如 Algorithm 1 所示，其时间复杂度为 $\mathcal{O}(N)$．

Algorithm 1: insert(b$^+$ tree)

```
   Input: 给定的值 value
1  state := insert_helper(value, root)
2  if state = 溢出状态 then
3  |  overflow := node 在 root_id 处的子结点
4  |  left_child := 新建结点
5  |  right_child := 新建结点
6  |  iter := 0
7  |  更新 left_child
8  |  在 overflow 内部插入数据
9  |  if overflow ∉ 叶结点 then
10 |  |  iter := iter + 1
11 |  end
12 |  更新 right_child
13 |  overflow 的元素数 := 1
14 end
```

其中，insert_helper 用于完成插入的递归，算法如 Algorithm 2 所示．

Algorithm 2: insert_helper(b$^+$ tree)

Input: 给定的值 $value$，B$^+$ 树中的一个结点 $node$
Output: 当前状态 $state$

$pos := 0$
while $pos <$ $node$ 的元素数 ***and*** $node$ 在 pos 处的元素值 $< value$ **do**
  $pos := pos + 1$
end
if $node$ 在 pos 处存在子结点 **then**
  $child :=$ $node$ 在 pos 处的子结点
  $state :=$ insert_helper$(value, child)$
  if $state =$ 溢出状态 **then**
    $overflow :=$ $node$ 在 pos 处的子结点
    $left\_child := overflow$
    $left\_child$ 的元素数 $:= 0$
    $right\_child :=$ 新建结点
    $iter := 0$
    更新 $left\_child$
    在 $node$ 内部插入数据
    if $overflow \notin$ 叶结点 **then**
      $iter := iter + 1$
    else
      $right\_child$ 的最右子结点 $:= left\_child$ 的最右子结点
      $left\_child$ 的最右子结点 $:= right\_child$ 的 key
      $node$ 在 $pos + 1$ 处的子结点 $:= right\_child$ 的 key
    end
    更新 $right\_child$
  end
else
  在 $node$ 内部插入数据
end
if $node$ 结点溢出 **then**
  return 溢出状态
else
  return 完成状态
end

2.3.2 查找

查找以典型的方式进行，类似于二叉查找树．起始于根节点，自顶向下遍历树，选择其分离值在要查找值的任意一边的子指针．在节点内部典型的使用是二分查找来确定这个位置．查找的结果是一个指向查询结果的 B$^+$ 树迭代器，如果查询不到则返回 `end()`.

本设计允许用户输入一个关键词，查找以该关键词为前缀的所有键指向的值．具体到科研文献管理系统，这个关键词可以是文献标题或作者姓名，修改代码亦可建立其他键-值索引.

B$^+$ 树的查找算法如 Algorithm 3 所示，其时间复杂度为 $\mathcal{O}(\log n)$.

Algorithm 3: find(b$^+$ tree)

Input: 给定的值 *value*
Output: 迭代器 *it*

```
1 it := find_helper(value, root)
2 if *it = value then
3 |   return it
4 else
5 |   return end()
6 end
```

其中，find_helper 用于完查找的递归，算法如 Algorithm 4 所示.

Algorithm 4: find_helper(b$^+$ tree)

Input: 给定的值 *value*，B$^+$ 树中的一个结点 *node*
Output: 迭代器 *it*

```
1  pos := 0
2  if node ∉ 叶结点 then
3  |   while pos < node 的元素数 and node 在 pos 处的元素值 ⩽ value do
4  |   |   pos := pos + 1
5  |   end
6  |   child := node 在 pos 处的子结点
7  |   return find_helper(value, child)
8  else
9  |   while pos < node 的元素数 and node 在 pos 处的元素值 < value do
10 |   |   pos := pos + 1
11 |   end
12 |   it.current_pos := node
13 |   it.index := pos
14 |   if pos = node 的元素数 then
15 |   |   it 自增
16 |   end
17 |   return it
18 end
```

2.4 倒排索引

倒排索引用来存储在全文搜索下某个单词在一个文档或者一组文档中的存储位置的映射．利用倒排索引，我们可以实现全文关键词模糊查询．本设计的倒排索引是文中一个单词的水平反向索引，包含每个单词在 XML 文档中的位置．索引存在另一棵 B$^+$ 树内.

2.4.1 插入

为了节省空间，本设计选用将关键词的哈希值作为键，(`pos`，`len`) 二元组作为值插入一棵新的 B $^+$ 树中.

倒排索引的插入算法如 Algorithm 5 所示.

Algorithm 5: insert(inverted index)

Input: 给定的键 key，给定的值 $value$
把键分解为 $word$ 序列
while 读入 $word$ **do**
    $hashed\_key$:= hash_fn($word$)
    向 B^+ 树中插入 $(hashed\_key, value)$
end

2.4.2 查找

本设计的倒排索引只支持完整的关键词查询．将关键词的哈希值作为键查询，结果返回符合关键词的所有迭代器的集合（C++ 标准库中的 `set`）．如果查询不存在，返回空的 `set`，即 `result_set.empty() == true`.

倒排索引的查找算法如 Algorithm 6 所示.

Algorithm 6: find(inverted index)

Input: 给定的待查找序列 $word\_list$
Input: 查找到的记录集合 $result$
$R := \emptyset$
foreach $x \in word\_list$ **do**
    $r :=$ 在 B^+ 树中搜索 x 的结果（r 是不同搜索结果的集合）
    $R := R \cup r$（R 是 r 的集合）
end
$result := \cap R$（即符合所有关键词的结果的集合）
return $result$

2.5 Top K 问题

本设计需要实现作者统计功能，即需要能够输出写文献最多的前 K 名作者．本设计先建立作者名-写文献数目的 B^+ 树，然后遍历树中的元素，对于这个问题，本设计使用 Top K 问题的典型解法.

Top K 问题即在一组数据中找到偏序前 K 的数据．假设有 N 个数，希望找出其中前 K 大的数据，自然的想法是采用小顶堆．首先读入前 K 个数来创建大小为 K 的最小堆，建堆的时间复杂度为 $\mathcal{O}(K \log K)$，然后遍历后续的数字，并于堆顶数字进行比较．如果比最小的数小，则继续读取后续数字；如果比堆顶数字大，则替换堆顶元素并重新调整堆为最小堆．整个过程直至 N 个数全部遍历完为止．最后按照中序遍历的方式输出当前堆中的所有 K 个数字．该算法的时间复杂度为 $\mathcal{O}(N \times K \log K)$.

3 过程论述

完成本设计的各组员并没有明确的分工，我主要完成后端代码的编写．本章中我将着重论述 XML 文件解析、B^+ 树、倒排索引和 Pager 的实现．

本设计中使用的外部库有 libxml2 和 fmt．本设计代码使用 g++ 编译．

3.1 XML 文件

3.1.1 SAX 解析——构建索引时

XML 文件的 SAX 解析主要在 read_xml.hh 中实现．

read_xml.hh 中提供了一个状态机 `ParserState`，这个状态机的状态有 `AUTHOR` 和 `TITLE`，它决定了 SAX 解析需要往哪个索引里添加键．

SAX 解析由：`on_start_element`、`on_end_element` 和 `on_characters` 三个 handler 组成．这三个 handler 是 libxml2 提供的，需要使用者来完善它们．本设计中，`on_start_element` 在 SAX 解析起始时调用．在其内部为状态机赋值，方便接下来的处理．`on_end_element` 读取状态机的值，然后将读到的内容作为键，XML 文件中的位置作为值插入到状态机的值决定的 B^+ 树中．为了代码简便，我们还建立了状态机值-B^+ 树指针的 `map`．`on_characters` 仅仅是用来读取数据的．

SAX 解析的其他部分直接调用 libxml2 写好的接口即可，这些可以在 libxml2 的手册查到，所以不再赘述．

3.1.2 DOM 解析——读取记录时

XML 文件的 DOM 解析的主体是 `print_dom_tree(*)` 函数．该函数解析 XML 文件中的一部分并打印整棵 DOM 树．打印 DOM 树的方式如同一般树的遍历．

使用 libxml2 解析名为 `file_name` 的 XML 文件从 `pos` 开始长度为 `len` 的数据．libxml2 返回一棵 DOM 树的根结点，本函数递归遍历该树并将 DOM 树包含的 XML 信息打印在屏幕上．每棵 DOM 树表明了一条文献记录．

3.2 B^+ 树的实现

B^+ 树主要在 bptree.hh 中实现．

bptree.hh 中提供了 B^+ 树结点 `Node` 类的实现、B^+ 树迭代器 `Iterator` 类的实现和 B^+ 树正体 `BplusTree` 类的实现．

3.2.1 `Node` 类

Node
−page_id : int64_t −count : int64_t −right : int64_t −data : array<T, ORDER + 1> −children : array<int64_t, ORDER + 2>
#insert_in_node(pos, value) #delete_in_node(pos) #is_overflow() : bool #is_underflow() : bool #is_leaf() : bool

`Node` 类表明了 B^+ 树的结点. 它是一个类模板, 模板参数有 `class T` 和 `int16_t ORDER`. 其中, `T` 是结点数据类型, `ORDER` 是该结点的度.

`Node` 类的各成员变量和成员函数的功能都如其名.

3.2.2 `Iterator` 类

Iterator
−index : int64_t −current_pos : pointer<node> −pager : pointer<pager>
+operator->() : pointer<T> +operator*() : T +operator++() : Iterator& +operator=(that) : Iterator& +operator!=(that) : bool

正如大多数 STL 容器设计的那样, 我们的 B^+ 树需要一个迭代器 `Iterator` 类. `Iterator` 类也是一个类模板, 其模板参数和 `Node` 类一致.

`Iterator` 类的成员变量有 `index` 和 `current_pos`, 其中, `index` 是该迭代器指向的结点中的位置, `current_pos` 是迭代器的内部指针. 例如, 某迭代器指向结点 A 的第 2 条数据, 那么有 `current_pos == &A, index == 1`.

`Iterator` 类的成员函数都是基本的运算符重载.

3.2.3 `BplusTree` 类

BplusTree
−print_count : int64_t
−pager : pointer<Pager>
−header : pointer<Header>
+begin() : iterator
+find(value) : iterator
+find_geq(value) : iterator
+end() : iterator
+insert(value)
+print()
−write_node(id, n_ptr)
−read_node(id) : nodeptr
−new_node() : nodeptr
−find_helper(value, root) : iterator
−print_helper(n_ptr)
−insert_helper(n_ptr, value) : State
−reset_children(parent, child, p, iter)
−write_these_nodes(n1, n2, n3, pos)

`BplusTree` 类是 B^+ 树的正体．`BplusTree` 类也是一个类模板，其模板参数和 `Node` 类一致，其中 `ORDER` 在不指定的情况下默认为 3.

`BplusTree` 类的成员变量有 `print_count`，`pager`，`header`．其中，`print_count` 是为了保证在打印的数据过多时，最多打印 64 条数据．`pager` 是指向该 B^+ 树专属 Pager 的指针，`header` 指明了该 B^+ 树的根结点的 ID 等参数．

模仿 STL 容器的实现，`BplusTree` 类的公有成员函数有 `begin()`、`end()`、`find(*)` 等，见上图．其中，`find(*)` 返回的迭代器指向欲查找的值．如未查找到，返回 `end()`．而 `find_geq(*)` 返回指向偏序大于等于 `value` 的第一个值的迭代器．

`BplusTree` 类的私有成员函数众多，见上图．其中，`*_node(*)` 是对 B^+ 树结点的硬盘读写操作，`*_helper(*)` 是辅助函数，算法见「方案论证」.

3.3 倒排索引的实现

倒排索引在 inverted_index.hh 中实现.

InvertedIndex
−page_manager : pointer<Pager> −record_manager : pointer<Pager> −bt : pointer<BplusTree> −hash_fn : hash<string>
+init_ii(iiname, new_file) +build(source, pos, len) +find(value_list) : vector<...> −insert(key, pos, len) −intersection(result_list) : result_set −find_single_value(v) : result_set

`InvertedIndex` 类的成员变量有 `id`、`*_manager`、`bt`、`hash_fn`. 其中，`id` 是倒排索引在记录中的 ID，`*_manager` 用于管理记录，`hash_fn` 是类型为 `hash<string>` 的 STL 哈希函数.

`InvertedIndex` 类的公有成员函数有 `build(*)`、`find(*)` 等, 见上图. 其中, `build()` 用于从包含若干单词的一条记录建立倒排索引，`find()` 查询索引，取这些单词索引指向的位置的交集.

3.4 Pager 的实现

Pager 在 bptree.hh 中实现.

Pager
+empty : bool
+get_id(reg) : int64_t +save(n, reg) +recover(n, reg) : bool +erase(n)

Pager 作为 B^+ 树和硬盘读写的中间层，提供了包装了的二进制文件读写功能. `save(*)` 的作用是索引文件中插入一条键-ID 的对, 然后在记录文件的 ID×`reg.size()` 处插入一条 `(len, pos)` 对. `recover(*)` 与之相反，它从 `n`×`reg.size()` 处读取一条数据，存在 `reg` 里.

4 结果分析

4.1 功能分析

本设计中我完成的部分不涉及图形界面，所以功能分析使用控制台界面的方式．由于本设计原本需要的 dblp.xml 过于庞大，在功能分析阶段采用 dblp.xml 中的一小部分（sample.xml）作为源 XML 文档．对于海量数据处理的性能分析见「性能分析」章节．

4.1.1 基础设计

进入程序后，使用 `create` 创建数据库，使用 `open` 打开数据库，使用 `read` 读取 XML 文档，使用 `select` 选择数据（测试用），使用 `find` 进行精确前缀搜索，使用 `search` 进行倒排索引的模糊搜索，使用 `top` 选择写文献最多的 K 位作者，使用 `whoami` 获得当前数据库名，使用 `close` 关闭数据库，使用 `exit` 结束程序．

测试数据库的一系列基本功能

```
tssndb version 1.4.0
i.e. too simple sometimes naive database
MDB >>> open sample
Database does not exist.
Create it now? (y/n) y
Database sample is open.
MDB >>> read
READ OK
MDB >>> open another
Database sample is open.
Please close it first.
MDB >>> close
Database sample is closed.
MDB >>> open sample
Database sample is open.
MDB >>> whoami
Who am I? Database sample!
MDB >>> help
create a database: create [database_name]
open a database: open [database_name]
read from xml file: read
select from table: select [title|author]
find (prefix) in table: find [title|author] [keyword]
search (fuzzy) in table: search [keyword]
get authors with top article counts: top [number]
get the name of current opening database: whoami
close a database: close
end the program: exit
MDB >>> unknown
Command not found: unknown
```

4.1.2 前缀匹配

命令格式：`find author|title [something]`

例如，查找作者前缀为 'Mark F. Hornick' 的文献，查询到作者为 **Mark F. Hornick** 的文献.

查找作者一例

```
MDB >>> find author "Mark F. Hornick"
 [1] ----------------------------------------------------------
     <article>   mdate = 2019-10-25
                 key = tr/gte/TM-0332-11-90-165
                 publtype = informal
     <author>    Frank Manola
                 Mark F. Hornick
                 Alejandro P. Buchmann
     <title>     Object Data Model Facilities for Multimedia Data Types.
     <journal>   GTE Laboratories Incorporated
     <volume>    TM-0332-11-90-165
     <month>     December
     <year>      1990
     <url>       db/journals/gtelab/index.html#TM-0332-11-90-165
-------------------------------------------------------------
 [2] ----------------------------------------------------------
     <article>   mdate = 2019-10-25
                 key = tr/gte/TR-0174-12-91-165
                 publtype = informal
     <author>    Mark F. Hornick
                 Joe D. Morrison
                 Farshad Nayeri
     <title>     Integrating Heterogeneous, Autonomous, Distributed Applications Using the DOM Prototy
     <journal>   GTE Laboratories Incorporated
     <volume>    TR-0174-12-91-165
     <month>     December
     <year>      1991
     <url>       db/journals/gtelab/index.html#TR-0174-12-91-165
-------------------------------------------------------------
FIND OK (0.00043s)
```

例如，查找标题前缀为 'Integrating Obj' 的文献. 查询到标题为 **Integrating Obj**ect-Oriented Applications and Middleware with Relational Databases. 的文献.

查找标题一例

```
MDB >>> find title "Integrating Obj"
 [1] ----------------------------------------------------------
     <article>   mdate = 2019-10-25
                 key = tr/gte/TR-0310-11-95-165
                 publtype = informal
     <author>    Frank Manola
     <title>     Integrating Object-Oriented Applications and Middleware with Relational Databases.
     <journal>   GTE Laboratories Incorporated
     <volume>    TR-0310-11-95-165
     <month>     November
     <year>      1995
     <url>       db/journals/gtelab/index.html#TR-0310-11-95-165
-------------------------------------------------------------
FIND OK (0.000179s)
```

查询输入只有一个单词时，不必使用引号包围输入.

例如，查找标题前缀为 'In' 的文献. 查询到标题为 **In**heritance for ADTs (revised)、标题为 **In**tegrating Object-Oriented Applications... 和标题为 **In**tegrating Heterogeneous, Autonomous, Distributed... 的文献.

查找标题一例

```
MDB >>> find title In
 [1] ---------------------------------------------------------
     <article>   mdate = 2019-10-25
                 key = tr/sql/X3H2-91-133rev1
                 publtype = informal
     <author>    Krishna G. Kulkarni
                 Jim Melton
                 Jonathan Bauer
                 Mike Kelley
     <title>     Inheritance for ADTs (revised)
     <journal>   ANSI X2H2
     <volume>    DBL:KAW-006 X3H2-91-133rev1
     <month>     July
     <year>      1991
     <url>       db/conf/x3h2/index.html#X3H2-91-133rev1
     <cdrom>     SQL/x3h2-91-133rev1.pdf
-------------------------------------------------------------
 [2] ---------------------------------------------------------
     <article>   mdate = 2019-10-25
                 key = tr/gte/TR-0174-12-91-165
                 publtype = informal
     <author>    Mark F. Hornick
                 Joe D. Morrison
                 Farshad Nayeri
     <title>     Integrating Heterogeneous, Autonomous, Distributed Applications Using the DOM Prototy
     <journal>   GTE Laboratories Incorporated
     <volume>    TR-0174-12-91-165
     <month>     December
     <year>      1991
     <url>       db/journals/gtelab/index.html#TR-0174-12-91-165
-------------------------------------------------------------
 [3] ---------------------------------------------------------
     <article>   mdate = 2019-10-25
                 key = tr/gte/TR-0310-11-95-165
                 publtype = informal
     <author>    Frank Manola
     <title>     Integrating Object-Oriented Applications and Middleware with Relational Databases.
     <journal>   GTE Laboratories Incorporated
     <volume>    TR-0310-11-95-165
     <month>     November
     <year>      1995
     <url>       db/journals/gtelab/index.html#TR-0310-11-95-165
-------------------------------------------------------------
FIND OK (0.000589s)
```

4.1.3 模糊匹配

命令格式：`search [something]...`

模糊搜索的参数可以不止一个，本设计取搜索结果的交集输出.

例如，模糊搜索 'Data' 和 'Model'，搜索到标题为 Object **Data Model** Facilities for Multimedia Data Types. 的一篇文献.

模糊搜索 'Language',搜索到标题为 Object Data **Language** Facilities for Multimedia Data Types. 和标题为 Implementation Aspects of a Natural **Language** Understanding System in a Prolog/DB Environment 的文献.

模糊搜索 'Database' 和 'Thomas'，搜索到标题为 LILOG-DB: **Database** Support for Knowledge-Based Systems、作者为 **Thomas** Ludwig 0001 的一篇文献.

模糊搜索两例

```
MDB >>> search Data Model
Search for Data + Model:
 [1] ------------------------------------------------------------
     <article>   mdate = 2019-10-25
                 key = tr/gte/TM-0332-11-90-165
                 publtype = informal
     <author>    Frank Manola
                 Mark F. Hornick
                 Alejandro P. Buchmann
     <title>     Object Data Model Facilities for Multimedia Data Types.
     <journal>   GTE Laboratories Incorporated
     <volume>    TM-0332-11-90-165
     <month>     December
     <year>      1990
     <url>       db/journals/gtelab/index.html#TM-0332-11-90-165
------------------------------------------------------------
SEARCH OK (0.000224s)
MDB >>> search Language
Search for Language:
 [1] ------------------------------------------------------------
     <article>   mdate = 2019-10-25
                 key = tr/gte/TR-0169-12-91-165
                 publtype = informal
     <author>    Frank Manola
     <title>     Object Data Language Facilities for Multimedia Data Types.
     <journal>   GTE Laboratories Incorporated
     <volume>    TR-0169-12-91-165
     <month>     December
     <year>      1991
     <url>       db/journals/gtelab/index.html#TR-0169-12-91-165
------------------------------------------------------------
 [2] ------------------------------------------------------------
     <article>   mdate = 2017-06-08
                 key = tr/ibm/LILOG3
                 publtype = informal
     <author>    Rudi Studer
                 Bernd Walter
     <title>     Implementation Aspects of a Natural Language Understanding System in a Prolog/DB Env
     <journal>   LILOG-Report
     <volume>    3
     <year>      1986
     <publisher> IBM Deutschland GmbH
------------------------------------------------------------
SEARCH OK (0.000225s)
```

模糊搜索一例

```
MDB >>> search Database Thomas
Search for Database + Thomas:
 [1] ------------------------------------------------------------
     <article>   mdate = 2017-06-08
                 key = tr/ibm/LILOG56
                 publtype = informal
     <author>    Thomas Ludwig 0001
                 Bernd Walter
                 Michael Ley
                 Albert Maier
                 Erich Gehlen
     <title>     LILOG-DB: Database Support for Knowledge-Based Systems
     <journal>   LILOG-Report
     <volume>    56
     <year>      1988
     <publisher> IBM Deutschland GmbH
------------------------------------------------------------
SEARCH OK (0.000371s)
```

利用模糊搜索可以完成其他功能，比如查询若干作者合作完成的文献.

例如，模糊搜索 'Thomas'、'Michael' 和 'Daniel'，搜索到由 **Thomas** Prescher 0002、**Michael** Schwarz 0001、**Daniel** Genkin 和 **Daniel** Gruss 合作的文献.

模糊搜索一例

```
MDB >>> search Thomas Michael Daniel
Search for Thomas + Michael + Daniel:
 [1] ------------------------------------------------------------
     <article>   mdate = 2020-06-25
                 key = tr/meltdown/s18
                 publtype = informal
     <author>    Paul Kocher
                 Daniel Genkin
                 Daniel Gruss
                 Werner Haas 0004
                 Mike Hamburg
                 Moritz Lipp
                 Stefan Mangard
                 Thomas Prescher 0002
                 Michael Schwarz 0001
                 Yuval Yarom
     <title>     Spectre Attacks: Exploiting Speculative Execution.
     <journal>   meltdownattack.com
     <year>      2018
     <ee>        https://spectreattack.com/spectre.pdf
                 type = oa
------------------------------------------------------------
 [2] ------------------------------------------------------------
     <article>   mdate = 2020-06-25
                 key = tr/meltdown/m18
                 publtype = informal
     <author>    Moritz Lipp
                 Michael Schwarz 0001
                 Daniel Gruss
                 Thomas Prescher 0002
                 Werner Haas 0004
                 Stefan Mangard
                 Paul Kocher
                 Daniel Genkin
                 Yuval Yarom
                 Mike Hamburg
     <title>     Meltdown
     <journal>   meltdownattack.com
     <ee>        https://meltdownattack.com/meltdown.pdf
                 type = oa
     <year>      2018
------------------------------------------------------------
SEARCH OK (0.000396s)
```

4.1.4 Top K 问题

命令格式：`top [k]`

使用其查询论文数量前 K 的作者. 若存在论文数量并列第 N 的作者，XML 文件中靠前读入的作者优先.

例如，查询论文数量前 20、前 5、以及第 1 的作者.

Top K 查询一例

```
MDB >>> top 20
 [1] Frank Manola (8)
 [2] Erich Gehlen (3)
 [3] Thomas Ludwig 0001 (3)
 [4] Jonathan Bauer (2)
 [5] Krishna G. Kulkarni (2)
 [6] Michael Stonebraker (2)
 [7] Jim Melton (2)
 [8] Daniel Gruss (2)
 [9] Paul Kocher (2)
[10] Michael Ley (2)
[11] David Beech (2)
[12] Stefan Mangard (2)
[13] Yuval Yarom (2)
[14] Mike Hamburg (2)
[15] Ulrike Schwall (2)
[16] Michael Schwarz 0001 (2)
[17] Farshad Nayeri (2)
[18] Werner Haas 0004 (2)
[19] Alejandro P. Buchmann (2)
[20] Thomas Prescher 0002 (2)
MDB >>> top 5
 [1] Frank Manola (8)
 [2] Erich Gehlen (3)
 [3] Thomas Ludwig 0001 (3)
 [4] Jonathan Bauer (2)
 [5] Krishna G. Kulkarni (2)
MDB >>> top 1
 [1] Frank Manola (8)
```

4.2 性能分析

4.2.1 完整读入 XML 文件

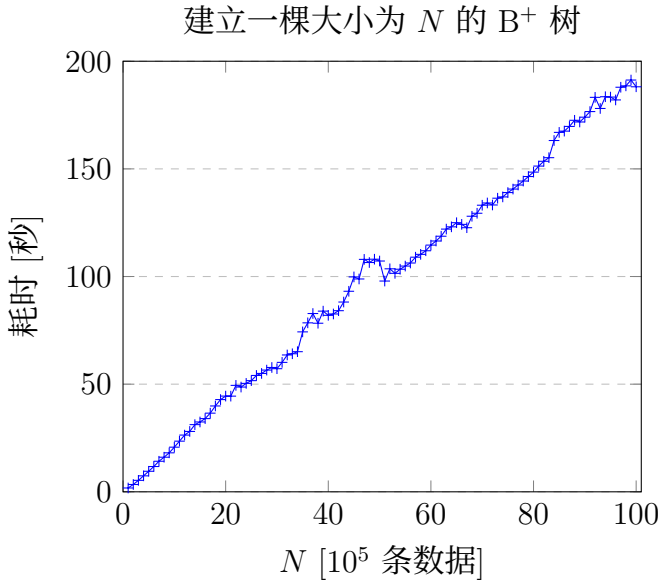
完整地读入 dblp.xml 的时间复杂度是 $\mathcal{O}(N)$，由于需要用大量索引来组织数据，读入耗时超过一小时（6356 秒）. 在读入过程中，系统内存占用始终不超过 2MB，读入完成后，对数据的查找均能在极短的时间内完成.

4.2.2 B$^+$ 树的性能

本设计使用 gtest 完成仅对 B$^+$ 树性能的测试，测试结果如下所示：

建立 B$^+$ 树的测试结果

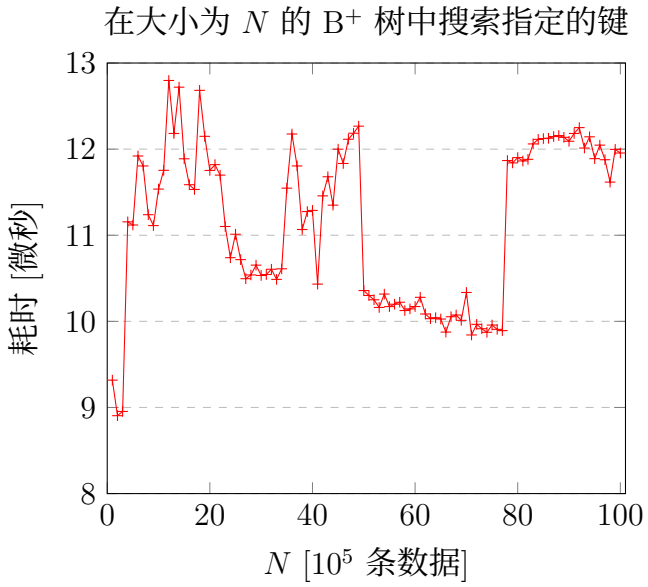
N(10^5)	t(ms)	N(10^5)	t(ms)	N(10^5)	t(ms)	N(10^5)	t(ms)
1	1.781425	26	54.046547	51	97.900864	76	140.626513
2	3.400530	27	54.986994	52	103.585874	77	142.511979
3	5.325446	28	56.543819	53	101.479068	78	144.348970
4	7.417500	29	57.751350	54	103.390573	79	146.446114
5	9.541544	30	57.188077	55	104.853369	80	148.496183
6	11.743532	31	60.118893	56	106.279980	81	151.306340
7	14.194829	32	63.567985	57	108.984568	82	153.529497
8	16.073469	33	64.126489	58	110.299808	83	155.202801
9	18.107131	34	65.089374	59	111.971625	84	163.176417
10	20.734623	35	74.290350	60	114.599729	85	166.951478
11	23.530393	36	78.485969	61	116.503794	86	167.543810
12	26.355439	37	82.813723	62	118.718296	87	169.722865
13	28.043771	38	78.304505	63	122.030007	88	172.569438
14	31.206282	39	83.925942	64	123.081670	89	171.786546
15	32.446850	40	81.978329	65	124.997648	90	174.010937
16	34.004407	41	82.606239	66	124.224232	91	176.621359
17	36.538088	42	84.134092	67	122.676468	92	183.223403
18	39.876663	43	88.149992	68	127.998831	93	178.063253
19	42.843113	44	93.151503	69	129.388945	94	183.516562
20	44.458483	45	99.819907	70	133.123668	95	183.350263
21	44.439210	46	98.865290	71	134.166681	96	182.027158
22	49.440304	47	107.962198	72	133.237427	97	187.920420
23	48.658051	48	106.509132	73	136.380405	98	188.563105
24	50.365754	49	108.046396	74	136.980158	99	191.266885
25	51.639247	50	107.199910	75	138.998958	100	188.136308



如图所示，B$^+$ 树写入耗时和待写入的数据大小大致成正比，时间复杂度为 $\mathcal{O}(N)$.

搜索键的测试结果

N(10^5)	t(μs)	N(10^5)	t(μs)	N(10^5)	t(μs)	N(10^5)	t(μs)
1	9.317200	26	10.716891	51	10.300429	76	9.907360
2	8.904675	27	10.495568	52	10.250812	77	9.892381
3	8.954030	28	10.539163	53	10.161531	78	11.867849
4	11.155672	29	10.652806	54	10.316825	79	11.839947
5	11.119076	30	10.533234	55	10.171272	80	11.901834
6	11.920372	31	10.543786	56	10.199096	81	11.862768
7	11.804633	32	10.605229	57	10.221979	82	11.881705
8	11.238504	33	10.487440	58	10.125632	83	12.060473
9	11.112163	34	10.610217	59	10.144990	84	12.113433
10	11.536063	35	11.546722	60	10.172248	85	12.122694
11	11.754138	36	12.175621	61	10.279177	86	12.126278
12	12.797368	37	11.804746	62	10.085675	87	12.146236
13	12.182272	38	11.066874	63	10.031137	88	12.157525
14	12.719468	39	11.274978	64	10.042149	89	12.137174
15	11.887903	40	11.288641	65	10.027188	90	12.091672
16	11.586831	41	10.432551	66	9.875522	91	12.182070
17	11.531985	42	11.457331	67	10.053508	92	12.249926
18	12.682813	43	11.679011	68	10.073686	93	12.012617
19	12.148816	44	11.349523	69	10.012039	94	12.143251
20	11.752805	45	11.999329	70	10.335453	95	11.889455
21	11.819900	46	11.833693	71	9.842020	96	12.046322
22	11.698372	47	12.114702	72	9.965494	97	11.876889
23	11.101660	48	12.184724	73	9.914585	98	11.617167
24	10.739266	49	12.266666	74	9.873176	99	11.997059
25	11.011732	50	10.357777	75	9.956987	100	11.955136



B$^+$ 树搜索的时间复杂度应当为 $\mathcal{O}(\log N)$. 测试结果有较大的随机性, 但基本上符合本设计的效率预期.

5 课程设计总结

5.1 现存的问题

5.1.1 XML 文件读入过慢

本设计中唯一直观的效率问题就是 XML 文件读入过慢，读取 3GB 大小的 XML 文件需要耗时 1.5 小时左右．由于准备不充分加上不太了解 libxml2 库，目前读取的方法可能过于粗暴，可能存在大量重复运算．

目前的设计采用一次读取一条文献记录，读完即把这条记录丢弃．或许可以适当改变读取的粒度，将内存占用稳定在 100MB 左右（目前的内存占用在 1.5MB 左右），换取读入效率的提升．

5.1.2 软件工程问题

作者没有系统学过软件工程，所以程序写得低内聚、高耦合，难以进行修改．今后学习了更多工程知识后可以适当重构本设计．

5.1.3 是否可以将本设计扩展为一般 K-V 数据库?

本设计基于朴素的 K-V 数据库，为科学文献管理系统服务．或许可以适当修改本设计，把它完善成一个正规的、可以实现用户定义的各种功能（当然也包括科学文献管理功能）的 K-V 数据库．

5.2 收获和体会

通过本次课设，我对常用数据结构的理解更加深入，学习了 libxml2 和 fmt 的使用．在编写程序过程中，我学习了一些基本的数据库实现方法，对 C++ 本身也有了更深的理解．