



**课 程 设 计**

**课程名称 数据结构**

**题目名称 图书管理系统**

**学生学院 计算机学院**

**专业班级 计算机科学与技术（5）班**

**学 号 3219004950**

**学生姓名 吴泳诗**

**指导教师 曾孜**

**2020 年 12 月 28 日**

**报告：**

报告内容：□详细 □完整 □基本完整 □不完整

设计方案：□非常合理 □合理 □基本合理 □较差

算法实现：□全部实现 □基本实现 □部分实现 □实现较差

测试样例：□完备 □基本完备 □基本完备 □不完备

文档格式：□规范 □比较规范 □基本规范 □不规范

**答辩：**

□理解题目透彻，回答问题流利

□理解题目较透彻，回答问题基本正确

□部分理解题目，部分问题回答正确

□未能完全理解题目，答辩情况较差

**总评成绩：**

□优秀 □良好 □中等 □及格 □不及格

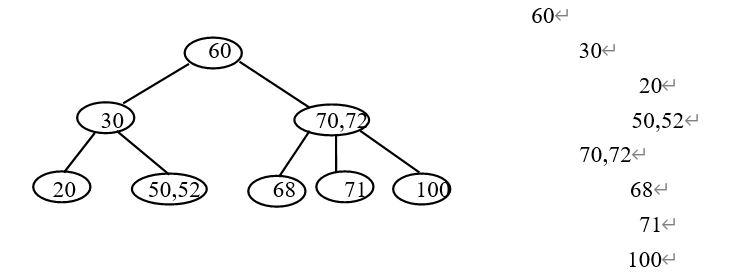
1. **题目：图书管理系统**

[问题描述]

图书管理基本业务活动包括：对一本书的采编入库、清除库存、借阅和归还等等。试设计一个图书管理系统，将上述业务活动借助于计算机系统完成。

[基本要求]

1. 每种书的登记内容至少包括书号、书名、著者、现存量和总库存量等五项。
2. 作为演示系统，不必使用文件，全部数据可以都在内存存放。但是由于上述四项基本业务活动都是通过书号(即关键字)进行的，所以要用B树(2-3树)对书号建立索引，以获得高效率。
3. 系统应实现的操作及其功能定义如下：
   1. 采编入库：新购入一种书，经分类和确定书号之后登记到图书账目中去。如果这种书在帐中已有，则只将总库存量增加。
   2. 清除库存：某种书已无保留价值，将它从图书账目中注销。
   3. 借阅：如果一种书的现存量大于零，则借出一本，登记借阅者的图书证号和归还期限。
   4. 归还：注销对借阅者的登记，改变该书的现存量。
   5. 显示：以凹入表的形式显示B树。这个操作是为了调试和维护的目的而设置的。下列B树的打印格式如下所示：



[实现提示]

1. 2-3树的查找算法是基础，入库和清除操作都要调用。难点在于删除关键字的算法，因而只要算法对2-3树适用就可以了，暂时不必追求高阶B树也适用的删除算法。
2. 每种书的记录可以用动(或静)态链式结构。
3. 借阅登记信息可以链接在相应的那种书的记录之后。

[基本操作]

1. B树基本操作接口
2. **查找关键字**

**int** Search(**BTree** p,**int** k)

操作结果：在关键字所在结点p中的关键字数组中寻找关键字k。

**result** Search\_BTree(**BTree** T,**KeyType** k)

操作结果：在m阶B树T上查找关键字k，返回B树查找结果类型result（其中包含信息为关键字所在/应插入的结点、关键字在结点中的位序、查找是否成功标志），若查找成功，则标记tag=1，指针pt所指结点中第i个关键字等于k；否则tag=0，若要插入关键字为k的记录，应位于pt结点中第i-1个和第i个关键字之间 。

1. **插入关键字**

**void** New\_Root(**BTree** &T, **BTree** p, **int** k, **BTree** ap, **Record** \*rec)

操作结果：当根节点要进行分裂时，要调用该方法生成新的根节点。

**void** Split\_Node(**BTree** &q, **int** s, **BTree** &ap)

操作结果：当该结点的关键字个数达到B树的阶数m时，要对该结点进行分裂。将q结点分裂成两个结点，前一半保留在原结点，后一般移入ap所指新结点。

**void** Insert(**BTree** &q, **int** i, **int** k, **BTree** &ap, **Record** \*rec)

操作结果：关键字x和新结点指针ap分别插入到q->key[i]和q->ptr[i]。

**Status** Insert\_BTree(**BTree** &T, **KeyType** k, **BTree** q, **int** i, **Record** \*rec)

操作结果：在B树中q结点的key[i-1]和key[i]之间插入关键字k，若插入后结点关键字个数等于B树的阶，则沿双亲指针链进行结点分裂，保证T仍是m阶B树。

1. **删除关键字**

**void** Successor(**BTree** &q, **int** i)

操作结果：在子树中找出最下层非终端结点的最小关键字并替代。（把删除最下层非终端结点的操作转换为删除最下层非终端结点中的最小关键字）

**void** Remove(**BTree** &q, int i)

操作结果：从结点q中删除key[i]。

**Status** Borrow(**BTree** &q)

操作结果：若q的兄弟结点关键字大于(m-1)/2,则从兄弟结点上移最小（或最大）的关键字到双亲结点，而将双亲结点中小于(或大于）且紧靠该关键字的关键字下移至q中,并返回OK，否则返回EREOR。

**void** Merge(**BTree** &q)

操作结果：合并结点。

**Status** Delete\_BTree(**BTree** &T, **KeyType** k)

初始条件：T不为空。

操作结果：在m阶B树上删除关键字k及其对应记录，并返回OK。如T上不存在关键字k，则返回ERROR。

1. **凹入表形式打印B树**

**void** Print\_BTree(**BTree** T,**int** tab)

操作结果：递归以凹入表形式显示B树T,每层的缩进量为tab

**void** Print(**BTree** T)

操作结果：打印展示B树前后的分界线，并调用Print\_BTree方法以凹入表形式打印B树。（传入第一层缩进量tab为4）

1. **B树的遍历**

**void** BTree\_Traverse(**BTree** T, **void**(\*visit)(**BTree**))

操作结果：遍历B树并对遍历到的结点调用visit方法。

1. 图书馆基本操作接口
2. **新增书籍**

**void** Insert\_Book(**BTree** &L**, BookType** B**, result** res)

操作结果：向树L中新增书籍B

1. **删除书籍**

**Status** Delete\_Book(**BTree** &L, **BookType** B)

初始条件：树L不为空，否则返回ERROR

操作结果：删除树L中的书本B，如果B不存在于L中，则删除失败并返回ERROR，否则返回OK。

1. **判断书籍是否可以出借**

**Status** Can\_Borrow(**BTree** L, **BookType** B, **ReaderType** R)

初始条件：书库中存在此书

操作结果：如果书库中该书现存量大于0则返回TRUE，否则返回FALSE

1. **出借书籍**

**void** BorrowBook**(BTree** L, **BookType** B, **ReaderType** R)

初始条件：Can\_Borrow方法返回TRUE表示可出借

操作结果：添加借阅者借阅书本的时间信息，并更新该书的借阅者信息，现存量-1。

1. **还书**

**Status** ReturnBook(**BTree** L, **int** ID, **int** readerID**, BookType** &B, **ReaderType** &R)

初始条件：该书存在

操作结果：把借阅者的还书时间设置为借书时间作为已归还的标志，并令该书的现存量+1。

1. **表格形式打印书本内容**

**void** Print\_Head()

操作结果：打印表格头

**void** Print\_Body(**BookType** B)

操作结果：打印表格内容

**void** Print\_Tail()

操作结果：打印表格尾

**void** Print\_BookNode(**BTree** p)

操作结果：显示B树每个结点（书本）的记录信息

**void** Print\_All(**BTree** L)

操作结果：显示书库L所有图书信息

1. **以表格形式显示一本书的基本信息**

**void** Print\_Book(**BookType** B)

操作结果：以表格形式显示一本书的基本信息

1. **以表格形式显示一本书的借阅者信息**

**void** Print\_Borrower(**BookType** B)

操作结果：打印B书的借阅者信息

1. **输出指定某本书的信息**

**Status** Show\_Book\_info(**BTree** L, **int** ID)

初始条件：关键字ID存在于树L中，不存在会返回ERROR

操作结果：打印B树中关键字为ID的对应的书本的信息并返回OK

1. **输出选择菜单**

**int** menu()

操作结果：打印选择菜单

1. 时间操作接口

**void** GetDate(**tm** &tim)

操作结果：获取系统时间，赋予tm结构体变量tim

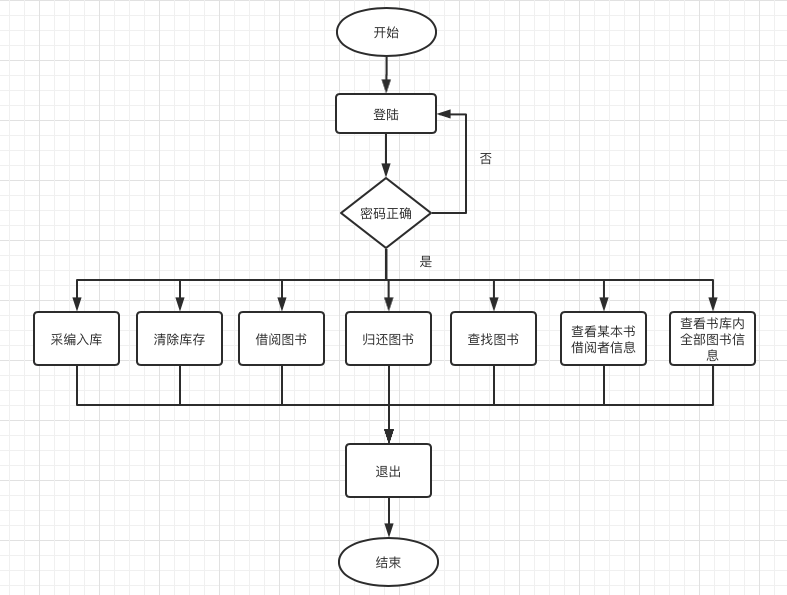
**void** AddDate(**tm** &date2, **tm** date1, **int** day)

操作结果：把date1的日期加day天后赋给date2

**int** Earlier(**tm** borDate, **tm** retDate)

操作结果：首先比较borDate和retDate是否一致（因为还书时会把这两个时间设为一致作为已归还的标志）。如果一致则返回RETURNED表示已归还；如果不一致则获取系统当前时间today并与retDate比较，如果today早于/等于retDate则返回NORMAL，否则返回OVERDUE。

[需求分析]



|  |  |  |  |
| --- | --- | --- | --- |
| 系统功能 | 方法入口名称 | 需要信息 | 功能描述 |
| 登陆 | login | 管理员密码 | 用户输入管理员密码（4950）后进入系统 |
| 采编入库 | Insert\_Book | 书本信息：ID号、书名、作者、入库数、出版年份、定价 | 图书采编入库 |
| 清除库存 | Delete\_Book | ID号 | 把指定书籍从书库中移除 |
| 借阅图书 | Borrow\_Book | 借阅者信息：ID号、姓名，图书ID | 读者借阅书本 |
| 归还图书 | Return\_Book | 借阅者信息：ID号、姓名，图书ID | 读者归还书本 |
| 查找图书 | Show\_Book\_Info | 图书ID | 查找某本书的信息 |
| 查看某本书借阅者信息 | Print\_Borrower | 图书ID | 查看借阅过本书的读者信息（ID、姓名、归还时间、状态：正常/逾期/已归还） |
| 查看书库内全部图书信息 | Print\_All | 无 | 表格形式打印图书馆内所有图书的信息 |

**2．基本定义**

1. 公用头文件:

#**include** <windows.h>

#**include** <stdio.h>

#**include** <stdlib.h>

#**include** <time.h>

1. 返回类型常量

**#define** m 3 //B树的阶，设为3

**#define** TRUE 1

**#define** FALSE 0

**#define** OK 1

**#define** ERROR 0

#**define** SUCCESS 1

#**define** FOUND 1

#**define** NOTFOUND 0

#**define** OVERFLOW -1

**typedef** **int** KeyType;

**typedef** **int** Status;

1. 图书馆操作相关常量

**#define** Manager\_Code 4950 //管理员密码

**#define** NAME\_LEN 20 //姓名长度

**#define** BKNAME\_LEN 100 //书名长度

**#define** KEEP\_DAYS 90 //借阅天数

**#define** OUT\_DATE\_PRICE 0.1 //超期每天一毛钱

#**define** RETURNED 2 //已归还

#**define** NORMAL 1 //正常

#**define** OVERDUE 0 //逾期

1. B树存储结构

**typedef** **struct** BTNode{

**int** keynum; //结点当前的关键字个数

**KeyType** key[m+1]; //关键字数组，key[0]未用

**struct** BTNode \* parent; //双亲结点指针

**struct** BTNode \* ptr[m+1]; //孩子结点指针数组，0号单元未用

**Record** \* recptr[m+1]; //记录指针向量，0号单元未用

} **BTNode**, \***BTree**; //B树的结点及指针类型

**typedef** **struct** {

**BTree** pt; //指向找到的结点

**int** i; //1≤i≤m，在结点中的关键字位序

**int** tag; //1：查找成功，0：查找失败

} **result**; //B树的查找结果类型

1. 书本和借阅者存储结构

**typedef** **BookNode** Record; //图书馆（B树）的记录为书本

**typedef struct** ReaderNode {

**int** ID; //ID

**char** name[NAME\_LEN]; //姓名

struct {

**tm** borDate; // 借书日期

**tm** retDate; // 还书日期

};

**ReaderNode** \*rnext; //指向下一个借阅者的指针

}**ReaderNode**, \***ReaderType**; //借阅者的指针及结点类型

**typedef** **struct** BookNode {

**int** ID; //图书编号

**char** bookname[BKNAME\_LEN]; //书名

**char** author[NAME\_LEN]; //作者名字

**int** current; //现存量

**int** total; //总库存

**int** publishedYear; //出版年份

**float** price; //价格

**ReaderNode** \* reader; //借阅者链表指针

}**BookNode**, \***BookType**; //图书类型

1. **算法设计**
2. **B树相关接口实现**
3. **搜索结点**

//在p->key[1..p->keynum]找k

**int** Search(**BTree** p,**int** k)

{

**int** i = 1;

while(i <= p->keynum && k > p->key[i]) i++;

return i;

}

**result** Search\_BTree(**BTree** T,**KeyType** k)

{

/\*\*

\* 在m阶B树T上查找关键字k，用r返回(pt,i,tag)

\* 若查找成功，则标记tag=1，指针pt所指结点中第i个关键字等于k

\* 否则tag=0，若要插入关键字为k的记录，应位于pt结点中第i-1个和第i个关键字之间

\*/

**result** r;

**int** i = 0,found = **NOTFOUND**;

//初始，p指向根节点；p将用于指向待查结点，q指向其双亲

**BTree** p = T,q = **NULL**;

while(p && found == **NOTFOUND**) {

//在p->key[1..p->keynum]中查找p->key[i-1]<k<=p->key[i]

i = Search(p,k);

//找到待查关键字

if(i <= p->keynum && p->key[i] == k) found = 1;

else {

//没找到，指针下移

q = p;

p = p->ptr[i-1];

}

}

if(found == FOUND) {

//查找成功，返回k的位置p及i

r.pt = p;

r.i = i;

r.tag = **FOUND**;

} else {

//查找不成功，返回k的插入位置q及i

r.pt = q;

r.i = i;

r.tag = **NOTFOUND**;

}

return r;

}

1. **生成新的根结点**

**void** New\_Root(**BTree** &T, **BTree** p, **int** k, **BTree** ap, **Record** \*rec) {

T = (**BTNode**\*)malloc(sizeof(**BTNode**));

T->keynum = 1;

T->ptr[0] = p;

T->ptr[1] = ap;

T->key[1] = k;

T->recptr[1] = rec;

if (p != **NULL**) p->parent = T;

if (ap != **NULL**) ap->parent = T;

T->parent = **NULL**; //新根的双亲是空指针

}

1. **分裂结点**

**void** Split\_Node(**BTree** &q,**int** s,**BTree** &ap) {

//将q结点分裂成两个结点，前一半保留在原结点，后一般移入ap所指新结点

**int** i,j,n = q->keynum;

//生成新结点

ap = (**BTNode** \*)malloc(sizeof(**BTNode**));

ap->ptr[0] = q->ptr[s];

//后一半移入ap结点

for(i = s+1,j= 1;i <= n;i++,j++) {

ap->key[j] = q->key[i];

ap->ptr[j] = q->ptr[i];

ap->recptr[i-s] = q->recptr[i];

}

ap->keynum = n-s;

ap->parent = q->parent;

//修改新结点的子节点的parent域

for(i = 0;i <= n-s;i++) {

//将ap所有孩子结点指向ap

if(ap->ptr[i])

ap->ptr[i]->parent = ap;

//q结点的前一半保留，修改keynum

q->keynum = s-1;

}

}

1. **插入结点**

/\*\*

\* k和ap分别插到q->key[i+1]和q->ptr[i+1]

\* 并插入关键字为k的记录recprt

\*/

**void** Insert(**BTree** &q, **int** i, **int** k, **BTree** &ap, **Record** \*rec) {

int j = q->keynum;

for (j; j >= i; j--) {

q->key[j + 1] = q->key[j]; //关键字指针向后移一位

q->ptr[j + 1] = q->ptr[j]; //孩子结点指针向后移一位

q->recptr[j+1] = q->recptr[j];

}

q->key[i] = k; //赋值

q->ptr[i] = ap;

q->recptr[i] = rec;

if (ap != NULL)

ap->parent = q;

q->keynum++; //插入关键字后该结点的关键字总数+1

}

/\*\*

\* 在m阶B树T上结点\*q的key[i]与key[i+1]之间插入关键字K和记录rec。

\* 若引起结点过大，则沿双亲链进行必要的结点分裂调整，使T仍是m阶B树。

\*/

**Status** Insert\_BTree(**BTree** &T, **KeyType** k, **BTree** q, **int** i, **Record** \*rec)

{

**BTree** ap = **NULL**;

**int** finished = **FALSE**;

if (!q) New\_Root(T, NULL, k, NULL, rec); // T是空树，生成仅含关键字K的根结点\*T

else {

while (!finished)

{

Insert(q, i, k, ap, rec); // 将k和ap分别插入到q->key[i+1]和q->ptr[i+1]

if (q->keynum < m) finished = **TRUE**; // 插入完成

else {

Split\_Node(q, (m + 1) / 2, ap); // 分裂结点Q

k = q->key[(m + 1) / 2];

rec = q->recptr[(m + 1) / 2];

if (q->parent)

{ // 在双亲结点\*q中查找k的插入位置

q = q->parent;

i = Search(q, k);

}

else finished = **OVERFLOW**; // 根节点已分裂为\*q和\*ap两个结点

}

}

if (finished == **OVERFLOW**) // 根结点已分裂为结点\*q和\*ap

New\_Root(T, q, k, ap, rec); // 需生成新根结点\*T,q和ap为子树指针

}

return **OK**;

}

1. **寻找后继结点**

**void** Successor(**BTree** &q, **int** i) {

//\*q结点的第i个关键字为k，用q的后继关键字代替q，且令q指向后继所在结点

**BTree** p = q;

q = q->ptr[i];

while (q->ptr[0]) q = q->ptr[0]; //查找p的后继

p->key[i] = q->key[1]; //记录代替

p->recptr[i] = q->recptr[1];

i = 1; //代替后应该删除q所指结点的第1个关键字

}

1. **删除结点内的关键字**

**void** Remove(**BTree** &q, **int** i) {

//删除q所指结点第i个关键字及其记录

for (; i < q->keynum; i++) { //关键字和记录指针前移

q->key[i] = q->key[i + 1];

q->recptr[i] = q->recptr[i + 1];

}

q->keynum--; //关键字数目减1

}

1. **向兄弟借关键字**

/\*\*

\* 若q的兄弟结点关键字大于(m-1)/2,则从兄弟结点上移最小（或最大）的关键字到双亲结点，

\* 而将双亲结点中小于(或大于）且紧靠该关键字的关键字下移至q中,并返回OK，否则返回EREOR。

\*/

**Status** Borrow(**BTree** &q)

{

**int** i;

**BTree** p = q->parent, b = **NULL**; // p指向q的双亲结点

for (i = 0; p->ptr[i] != q; i++); // 查找q在双亲p的子树位置

if (i >= 0 && i + 1 <= p->keynum && p->ptr[i + 1]->keynum > (m - 1) / 2)

{ // 若q的右兄弟关键字个数大于(m-1)/2

b = p->ptr[i + 1]; // b指向右兄弟结点

q->ptr[1] = b->ptr[0]; // 子树指针也要同步移动

q->key[1] = p->key[i + 1]; // 从父节点借第i+1个关键字

q->recptr[1] = p->recptr[i + 1];

p->key[i + 1] = b->key[1]; // b第一个关键字上移到父节点

p->recptr[i + 1] = b->recptr[1];

for (i = 1; i <= b->keynum; i++) // b第一个关键字上移，需把剩余记录前移一位

{

b->key[i] = b->key[i + 1];

b->recptr[i] = b->recptr[i + 1];

b->ptr[i - 1] = b->ptr[i];

}

}

else if (i > 0 && p->ptr[i - 1]->keynum > (m - 1) / 2)

{ // 若q的左兄弟关键字个数大约(m-1)/2

b = p->ptr[i - 1]; // b指向左兄弟结点

q->ptr[1] = q->ptr[0];

q->ptr[0] = b->ptr[b->keynum];

q->key[1] = p->key[i]; // 从父节点借第i个关键字

q->recptr[1] = p->recptr[i];

p->key[i] = b->key[b->keynum]; // 将b最后一个关键字上移到父节点

p->recptr[i] = b->recptr[b->keynum];

}

else return **ERROR**; // 无关键字大于(m-1)/2的兄弟

q->keynum++;

b->keynum--;

for (i = 0; i <= q->keynum; i++)

if (q->ptr[i]) q->ptr[i]->parent = q; // 刷新q的子结点的双亲指针

return **OK**;

}

1. **合并结点**

**void** Merge(**BTree** &q) {

**int** i, j;

**BTree** p = q->parent, b = **NULL**; //p指向q的父亲

for (i = 0; p->ptr[i] != q; i++); //查找q在父亲p中的子树位置

if (i == 0) { //若为0，则需合并为兄弟的第一个关键字

b = p->ptr[i + 1];

for (j = b->keynum; j >= 0; j--) { //将b的关键字和记录后移一位

b->key[j + 1] = b->key[j];

b->recptr[j + 1] = b->recptr[j];

b->ptr[j + 1] = b->ptr[j];

}

b->ptr[0] = q->ptr[0]; //合并

b->key[1] = p->key[1];

b->recptr[1] = p->recptr[1];

}

else if (i > 0) {//若q在父亲的子树位置大于0，需合并为兄弟b的最后一个关键字

b = p->ptr[i - 1];

b->key[b->keynum + 1] = p->key[i]; //合并

b->recptr[b->keynum + 1] = p->recptr[i];

b->ptr[b->keynum + 1] = q->ptr[0];

}

if (i == 0 || i == 1)//若i为0或1，需将父结点p关键字前移一位

for (i; i < p->keynum; i++) {

p->key[i] = p->key[i + 1];

p->ptr[i] = p->ptr[i + 1];

p->recptr[i] = p->recptr[i + 1];

}

p->keynum--;

b->keynum++;

free(q);

q = b; //q指向修改的兄弟结点

for (i = 0; i <= b->keynum; i++)

if (b->ptr[i]) b->ptr[i]->parent = b; //刷新b的子结点的双亲指针

}

1. **删除结点**

/\*\*

\* 在m阶B树T上删除关键字k及其对应记录，并返回OK。

\* 如T上不存在关键字k，则返回ERROR。

\*/

**Status** Delete\_BTree(**BTree** &T, **KeyType** k)

{

**KeyType** x = k;

**BTree** q, b = **NULL**;

int finished = **FALSE**, i = 1;

**result** res = Search\_BTree(T, k); // 在T中查找关键字k

if (res.tag == 0) return **ERROR**; // 未搜索到

else

{

q = res.pt; // q指向待删结点

i = res.i;

if (q->ptr[0]) Successor(q, i); // 若q的子树不空，(非底层结点)

// 则以其后继代之，且令q指向后继所在结点

Remove(q, i); // 删除q所指向结点中第i个关键字及记录

if (q->keynum >= (m - 1) / 2 || !q->parent) // 若删除后关键字个数不小于(m-1)/2或q是根节点

{

finished = TRUE; // 删除完成

if (q->keynum == 0) T = **NULL**; // 若q的关键字个数为0 ，则为空树

}

while (!finished)

{

if (Borrow(q)) finished = **TRUE**; // 若q的相邻兄弟结点关键字大于(m-1)/2,则从该

// 兄弟结点上移一个最大（或最小）关键字到

// 父节点，从父节点借一关键字到q

else { // 若q相邻兄弟关键字个数均等于┌m /2┑-1

Merge(q); // 将q中的剩余部分和双亲中的相关关键字合并至q的一个兄弟中

q = q->parent; // 检查双亲

if (q == T && T->keynum == 0) // 若被删结点的父节点是根T且T的关键字个数为0

{

T = T->ptr[0]; // 新根

T->parent = **NULL**;

free(q); // 删除原双亲结点

finished = **TRUE**;

}

else if (q->keynum >= m / 2) finished = **TRUE**;

} // 合并后双亲关键字个数不少于(m-1)/2，完成

}

}

return **OK**;

}

1. **遍历B树**

//遍历B树T，对每个结点调用visit函数

**void** BTree\_Traverse(**BTree** T, **void**(\*visit)(**BTree**)) {

if (!T) return;

//打印T结点内对应的所有关键字信息

visit(T);

for (int i = 0; i <= T->keynum; i++) {

if (T->ptr[i]) BTree\_Traverse(T->ptr[i], visit);

}

}

1. **凹入表形式打印B树**

// 递归以凹入表形式显示B树T,每层的缩进量为tab

**void** Print\_BTree(**BTree** T,**int** tab)

{

if (!T) return;

**int** i;

for (i = 0; i <= tab; i++) putchar(' '); // 缩进x

for (i = 1; i <= T->keynum; i++)

{

printf("%d ", T->key[i]);

}

printf("\n");

for (i = 0; i <= T->keynum; i++) // 递归显示子树结点关键字

Print\_BTree(T->ptr[i], tab + 3);

}

**void** Print(**BTree** T)

{

printf("========================================\n");

printf("== 凹入表形式显示B树 ==\n");

Print\_BTree(T,4);

printf("========================================\n");

}

1. **图书馆相关接口实现**
2. **采编入库**

/\*\*

\* 书库L已存在，res包含B书在书库L中的位置或应该插入的位置

\* 如果书库中已存在B书，则只将B书的库存量增加，否则插入B书到书库L中

\*/

**void** Insert\_Book(**BTree** &L, **BookType** B, **result** res) {

**BTree** pt;

**BookType** book;

//书库中不存在该书，则插入

if (res.tag == **NOTFOUND**) {

if(OK == Insert\_BTree(L, B->ID, res.pt, res.i, B)) printf("采编成功！\n");

else printf("采编失败！\n");

}

else {

book = (res.pt)->recptr[res.i];

book->current += B->total; //现存量增加

book->total += B->total; //总库存增加

printf("采编成功！\n");

}

}

1. **清空库存**

//如果书库中存在B书，则从书库中删除B书的信息，并返回OK，否则返回ERROR

**Status** Delete\_Book(**BTree** &L, **BookType** B) {

if (Delete\_BTree(L, B->ID)) return **OK**; //删除成功

else return **ERROR**; //删除失败

}

1. **借阅书籍**

//如果书库中存在B书，若B书现存量大于0，则可出借，返回TRUE，否则返回FALSE

**Status** Can\_Borrow(**BTree** L, **BookType** B, **ReaderType** R) {

if (B->current > 0) return **TRUE**; //现存量大于零

else return **FALSE**; //其他情况均不允许出借

}

/\*\*

\* 书库L存在，B书是书库中的书并且可被读者R借阅（已通过AvaiBorrow()判断）

\* 借出一本B书，登记借阅者R的信息，改变现存量。

\*/

**void** Borrow\_Book(**BTree** L, **BookType** B, **ReaderType** R)

{

**ReaderType** r;

GetDate(R->borDate); // 获取系统当前时间为借书时间日期

AddDate(R->retDate, R->borDate, KEEP\_DAYS); //当前时间加上90天为还书日期

if (!B->reader) B->reader = R; // 无其他借阅者，则直接登记

else {

for (r = B->reader; r->rnext; r = r->rnext);

r->rnext = R; // 否则到借阅者链表尾，登记

}

B->current--; // 现存量减1

}

1. **归还书籍**

/\*\*

\* booknum为还书图书编号，IDnum是借阅者的ID号

\* 若书库中不存在图书编号为booknum的书，则搜索出错，返回-1

\* 若有借阅记录，则注销该记录，并用B和R返回图书信息和借阅者信息并返回1

\* 若没有r借阅的记录，则用B返回图书信息，并返回0

\*/

**Status** Return\_Book(**BTree** L, **int** ID, **int** readerID, **BookType** &B, **ReaderType** &R) {

**result** res = Search\_BTree(L, ID); //在B树上按图书编号搜索

if (res.tag == **NOTFOUND**) return **OVERFLOW**;

B = (res.pt)->recptr[res.i];

**ReaderType** reader = (res.pt)->recptr[res.i]->reader;

**ReaderType** pre = reader;

while (reader) {

if (reader->ID == readerID) {

R = reader;

B->current++; //现存量加一

reader->retDate = reader->borDate; //把归还了书的借书者的还书时间设为借书时间，便于之后查阅借阅者信息时判断

return **OK**;

}

pre = reader;

reader = reader->rnext;

}

return **ERROR**;

}

1. **查看某图书信息**

//表格头部格式

**void** Print\_Head() {

printf("\n");

printf("|---------------------------------图书基本信息----------------------------------|\n");

printf("| 编号 | 书名 | 作者 | 现存 | 总库存 | 出版年份 | 定价 |\n");

}

//显示B书基本信息

**void** Print\_Body(**BookType** B) {

printf("|--------|--------------------|---------------|------|--------|----------|------|\n");

printf("| %-4d |%-20s|%-15s|", B->ID, B->bookname, B->author);

printf(" %-5d| %-4d | %-6d |%-6.1f|\n", B->current, B->total, B->publishedYear, B->price);

}

//表格底部格式

**void** Print\_Tail() {

printf("|--------|--------------------|---------------|------|--------|----------|------|\n");

}

//显示B树每个结点的记录信息

**void** Print\_BookNode(**BTree** p) {

**int** i;

for (i = 1; i <= p->keynum; i++) {

Print\_Body(p->recptr[i]);

}

}

//以表格形式显示一本书的基本信息

**void** Print\_Book(**BookType** B) {

Print\_Head();

Print\_Body(B);

Print\_Tail();

printf("\n");

}

/\*\*

\* 若书库L中存在图书编号为ID的书，则现在该书的所有信息并返回OK

\* 书库L不存在则返回ERROR

\*/

**Status** Show\_Book\_Info(**BTree** L, **int** ID) {

**BookType** B;

**result** res = Search\_BTree(L, ID);

if (res.tag == **NOTFOUND**) {

printf("书库中不存在此书，查找失败。\n");

return **ERROR**;

}

B = (res.pt)->recptr[res.i];

Print\_Book(B);

return **OK**;

}

1. **查看某书籍借阅者信息**

//以表格形式显示一本书的借阅者信息

**void** Print\_Borrower(**BookType** B) {

**ReaderType** p;

**ReaderType** r = B->reader;

**char** rpos[3][7] = { "超期","正常","已归还" };

**char** \* status;

printf("|-------------------------图书借阅信息------------------------|\n");

printf("\n");

printf("| ID号| 姓名 | 出借日期 | 归还日期 | 状态 |\n");

printf("|-----|--------|----------------|----------------|------|\n");

for (p = B->reader; p; p = p->rnext)

{

status = rpos[Earlier(p->borDate, p->retDate)];

printf("|%-5d|%-8s| %-4d年%2d月%2d日 | %-4d年%2d月%2d日 |%-6s|\n",

p->ID, p->name, p->borDate.tm\_year, p->borDate.tm\_mon, p->borDate.tm\_mday,

p->retDate.tm\_year, p->retDate.tm\_mon, p->retDate.tm\_mday, status);

}

}

1. **查看书库中所有书籍信息**

//显示书库L所有图书信息

**void** Print\_All(**BTree** L) {

Print\_Head();

BTree\_Traverse(L, Print\_BookNode);

Print\_Tail();

}

1. **时间操作相关接口实现**
2. **获取系统时间**

// 获取系统时间，赋予tm结构体变量tim

**void** GetDate(**tm** &tim)

{

time\_t curtime = time(0);

tim = \*localtime(&curtime);

tim.tm\_year += 1900; // tm 年份少1900年

tim.tm\_mon++; // tm month 从0-11，故加1

}

1. **时间相加**

// 把date1的日期加day天后赋给date2

**void** AddDate(**tm** &date2, **tm** date1, **int** day)

{

date2.tm\_year = date1.tm\_year + (day / 30 + date1.tm\_mon) / 12;

date2.tm\_mon = (date1.tm\_mon + (day / 30)) % 12;

date2.tm\_mday = (date1.tm\_mday + day) % 30;

}

1. **判断时间先后**

/\*\*

\* 首先比较borDate和retDate是否一致（因为还书时会把这两个时间设为一致作为已归还的标志）

\* 如果一致则返回RETURNED表示已归还

\* 如果不一致则获取系统当前时间today并与retDate比较

\* 如果today早于/等于retDate则返回NORMAL，否则返回OVERDUE

\*/

**int** Earlier(**tm** borDate, **tm** retDate)

{

if((borDate.tm\_year == retDate.tm\_year) && (borDate.tm\_mon == retDate.tm\_mon) && (borDate.tm\_mday == retDate.tm\_mday))

return **RETURNED**;

else {

**tm** today;

GetDate(today);

if (today.tm\_year < retDate.tm\_year) return **NORMAL**;

if (today.tm\_year > retDate.tm\_year) return **OVERDUE**;

if (today.tm\_mon < retDate.tm\_mon) return **NORMAL**;

if (today.tm\_mon > retDate.tm\_mon) return **OVERDUE**;

if (today.tm\_mday < retDate.tm\_mday) return **NORMAL**;

return **OVERDUE**;

}

}

1. **界面相关接口实现**

**int** menu()

{ //菜单

**int** choice;

printf("------------------------------------------------------\n");

printf("-- --\n");

printf("-- 图书馆管理系统 --\n");

printf("-- --\n");

printf("-- 1.采编入库 --\n");

printf("-- 2.清除库存 --\n");

printf("-- 3.借阅图书 --\n");

printf("-- 4.归还图书 --\n");

printf("-- 5.查看某图书信息 --\n");

printf("-- 6.查看某书借阅者信息 --\n");

printf("-- 7.查看图书馆全部图书 --\n");

printf("-- 0.退出 --\n");

printf("-- --\n");

printf("------------------------------------------------------\n");

do {

printf("请选择指定操作功能（输入0-7任意一个数字）:");

scanf("%d", &choice);

} while (choice < 0 || choice > 7); //避免非法输入

return choice;

}

//登陆界面

**Status** login() {

**int** password, i;

printf("---------------------------------\n");

printf("-- --\n");

printf("-- 图书管理系统 --\n");

printf("-- Welcome to Library System --\n");

printf("-- --\n");

printf("-- --\n");

printf("-- 数据结构课设 --\n");

printf("-- 3219004950 --\n");

printf("-- 吴泳诗 --\n");

printf("-- --\n");

printf("---------------------------------\n");

printf("请输入图书管理员密码:"); //密码为4950

scanf("%d", &password);

fflush(stdin);

Sleep(100);

system("cls"); //清屏

if (password == Manager\_Code)//判断是不是管理员

return **OK**;

else

printf("密码错误，请重新输入！\n");

login();

}

1. **程序主方法**

**int** main() {

**BTree** L = **NULL**;

**BookType** B;

**result** res;

**ReaderType** R;

**int** booknum, IDnum, book\_num;

**Status** k;

**char** flag;

printf("----------------------------\n");

printf("-- 数据结构课设 --\n");

printf("-- 图书管理系统 --\n");

printf("-- 3219004950 --\n");

printf("-- 吴泳诗 --\n");

printf("----------------------------\n");

if (login()) {

while (1) {

switch (menu())

{

//采编图书

case 1:

{

B = (BookType)malloc(sizeof(BookNode));

B->reader = **NULL**;

printf("请输入要入库的图书编号：\n");

fflush(stdin);

**int** ID;

scanf("%d", &ID);

B->ID = ID;

res = Search\_BTree(L, B->ID);

if (res.tag == **NOTFOUND**) {

fflush(stdin);

printf("请输入该书书名：\n");

fflush(stdin);

scanf("%s",B->bookname);

printf("请输入该书作者：\n");

fflush(stdin);

scanf("%s",B->author);

printf("请输入入库数：\n");

fflush(stdin);

scanf("%d", &B->current);

printf("请输入出版年份：\n");

fflush(stdin);

scanf("%d", &B->publishedYear);

printf("请输入书本价格：\n");

fflush(stdin);

scanf("%f", &B->price);

B->total = B->current;

Insert\_Book(L, B, res);

//如果是新书入库则把整棵树打印出来

Print(L);

break;

}

else { //图书已存在，只增加库存

printf("书库已有该书记录，请输入增加的册数：\n");

fflush(stdin);

scanf("%d", &B->total);

Insert\_Book(L, B, res);

B = (res.pt)->recptr[res.i];

//如果是旧书增加库存则只打印该树最新信息

Print\_Book(B);

system("pause");

break;

}

}

//清空库存

case 2:

{

printf("请输入要删除的图书图书编号：\n");

fflush(stdin);

scanf("%d", &booknum);

res = Search\_BTree(L, booknum);

if (res.tag) {

B = (res.pt)->recptr[res.i];

Print\_Book(B);

printf("\n您确认删除上面的图书<Y确认，其余按键返回主菜单>?");

fflush(stdin);

flag = getchar();

if (flag == 'Y' || flag == 'y') {

Delete\_Book(L, B);

printf("\n清除完毕。\n");

Print(L);

}

}

else {

printf("\n书库不存在此书，删除失败。\n");

}

printf("\t");

system("pause");

break;

}

//借阅图书

case 3:

{

**ReaderType** reader = (ReaderType)malloc(sizeof(ReaderNode));

reader->rnext = **NULL**;

printf("请输入借阅图书图书编号：\n");

fflush(stdin);

scanf("%d", &booknum);

getchar();

res = Search\_BTree(L, booknum);

if (res.tag) {

B = (res.pt)->recptr[res.i];

printf("请输入您的姓名：\n");

fflush(stdin);

scanf("%s",reader->name);

printf("请输入您的ID号：\n");

fflush(stdin);

scanf("%d", &reader->ID);

if (Can\_Borrow(L, B, reader)) {

Borrow\_Book(L, B, reader);

printf("\n借书完成,请注意借阅最长时间为90天.");

}

else {

printf("\n库存不足，借阅失败。");

free(R);

}

}

else {

printf("\n库存无此书，借阅失败。");

free(R);

}

printf("\t");

system("pause");

break;

}

//归还图书

case 4:

**int** temp;

printf("\n请输入归还图书编号：\n");

scanf("%d", &booknum);

printf("请输入您的ID号：\n");

scanf("%d", &IDnum);

k = Return\_Book(L, booknum, IDnum, B, R);

if (k == **TRUE**) {

**tm** today;

GetDate(today);

if (!Earlier(today, R->retDate))

{

for (temp = -1; Earlier(R->retDate, today); temp++)

AddDate(R->retDate, R->retDate, 1);

double price = (float)temp \* OUT\_DATE\_PRICE;

printf("\n您借的书已经逾期%d天，需扣除逾期费用%0.1f元!", temp, price);

}

printf("\n借阅者%d号%s还书完成。\n",R->ID,R->name);

R = **NULL**;

}

else if (k == **FALSE**) {

printf("\n还书失败，系统没有您借该书记录。");

}

else {

printf("\n书库中不存在此书!");

}

printf("\t");

system("pause");

break;

//查找某种图书

case 5:

{

**int** ID;

printf("请输入要查找的图书的编号：\n");

scanf("%d", &ID);

Show\_Book\_Info(L, ID);

break;

}

//输出某图书的借阅者信息

case 6:

printf("请输入要查找的图书的编号：\n");

scanf("%d", &book\_num);

res = Search\_BTree(L, book\_num);

if (res.tag) {

B = (res.pt)->recptr[res.i];

printf("借阅者信息如下：\n");

Print\_Borrower(B);

}

else {

printf("\n书库不存在此书，查找失败。\n");

}

printf("\t");

system("pause");

break;

//输出总库存

case 7:

printf("图书馆总库存信息如下：\n");

Print\_All(L);

break;

}

}

}

else {

printf("密码输入错误。请重新登录\n");

system("pause");

}

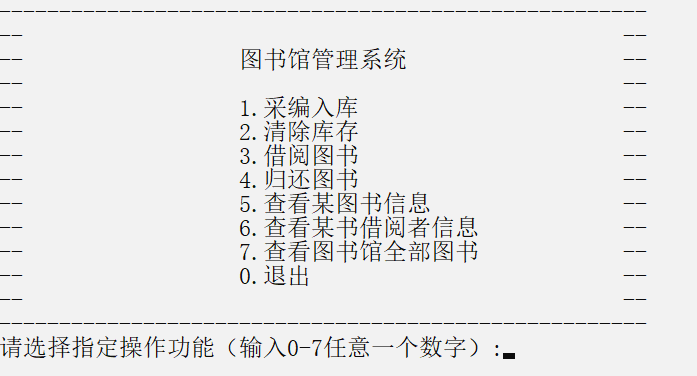
return 0;

}

1. **测试**
2. 登陆界面（此处输入管理员密码为4950，学号后四位）



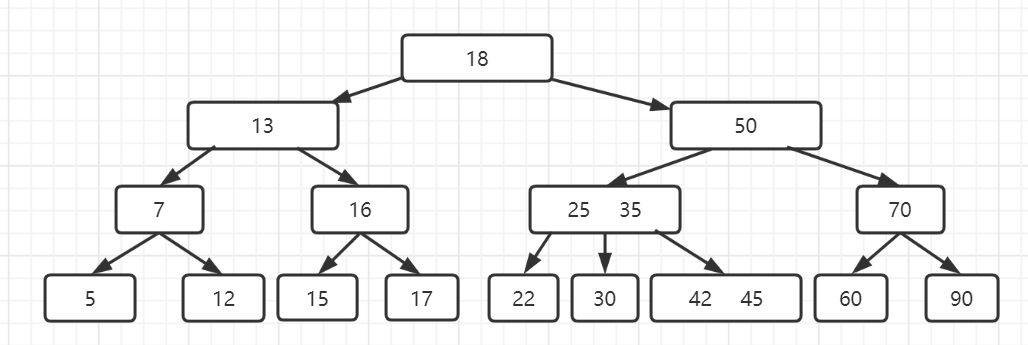
1. 选择操作界面



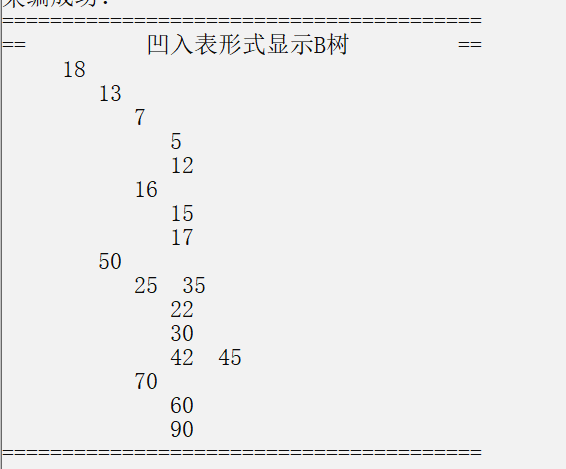
1. 采编入库

测试数据：按顺序入库35, 16, 18, 70, 5, 50, 22, 60, 13, 17, 12 , 45, 25, 42, 15, 90, 30, 7

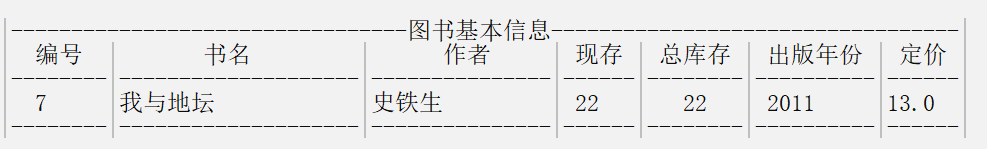
预期结果：



测试结果：（插入操作过程忽略）



特殊功能展示：当某个编号的书已经采编过时，不再提示输入书本等相关信息，而是提示输入增加册数。入库完成后打印该书的最新信息。

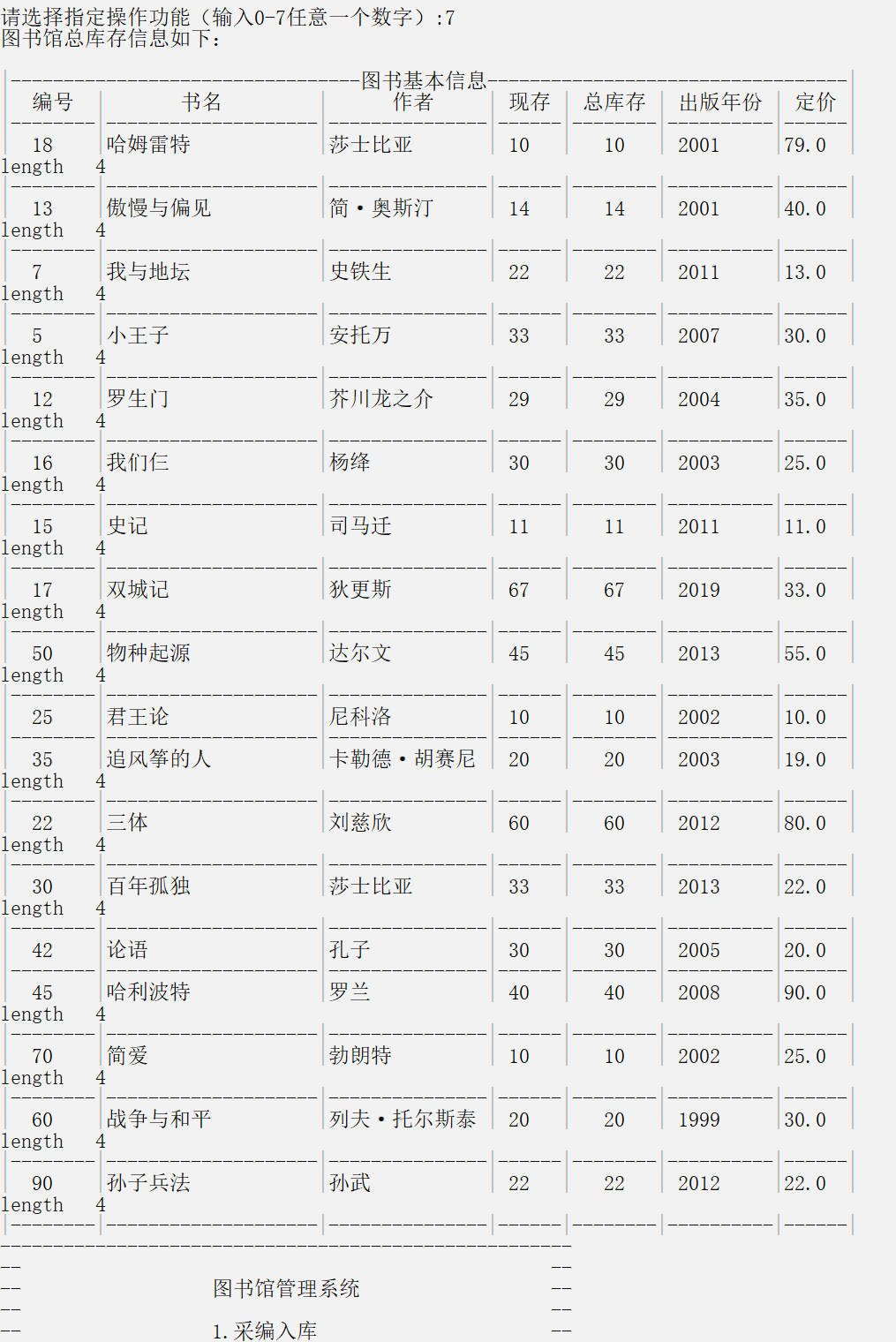


1. 查看图书馆全部图书信息

测试数据

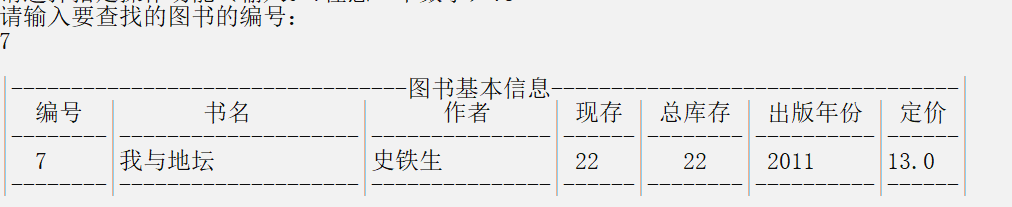
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 书号 | 书名 | 作者 | 现存量 | 总库存 | 出版年份 | 定价 |
| 35 | 追风筝的人 | 卡勒德·胡赛尼 | 20 | 20 | 2003 | 19 |
| 16 | 我们仨 | 杨绛 | 30 | 30 | 2003 | 25 |
| 18 | 哈姆雷特 | 莎士比亚 | 10 | 10 | 2001 | 79 |
| 70 | 简爱 | 勃朗特 | 10 | 10 | 2002 | 25 |
| 5 | 小王子 | 安托万 | 33 | 33 | 2007 | 30 |
| 50 | 物种起源 | 达尔文 | 45 | 45 | 2013 | 55 |
| 22 | 三体 | 刘慈欣 | 60 | 60 | 2012 | 80 |
| 60 | 战争与和平 | 列夫·托尔斯泰 | 20 | 20 | 1999 | 30 |
| 13 | 傲慢与偏见 | 简·奥斯汀 | 14 | 14 | 2001 | 40 |
| 17 | 双城记 | 狄更斯 | 67 | 67 | 2019 | 33 |
| 12 | 罗生门 | 芥川龙之介 | 29 | 29 | 2004 | 35 |
| 45 | 哈利波特 | 罗兰 | 40 | 40 | 2008 | 90 |
| 25 | 君王论 | 尼科洛 | 10 | 10 | 2002 | 10 |
| 42 | 论语 | 孔子 | 30 | 30 | 2005 | 20 |
| 15 | 史记 | 司马迁 | 11 | 11 | 2011 | 11 |
| 90 | 孙子兵法 | 孙武 | 22 | 22 | 2012 | 22 |
| 30 | 百年孤独 | 莎士比亚 | 33 | 33 | 2013 | 22 |
| 7 | 我与地坛 | 史铁生 | 22 | 22 | 2011 | 13 |

测试结果：

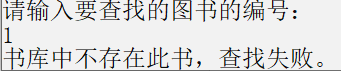


1. 查看某图书信息

查找成功：



查找失败：

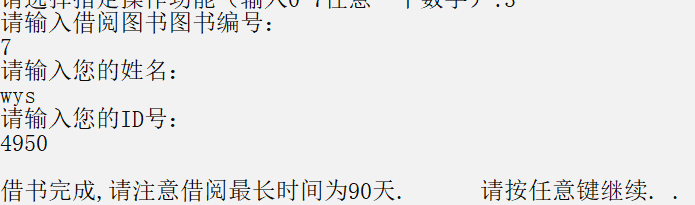


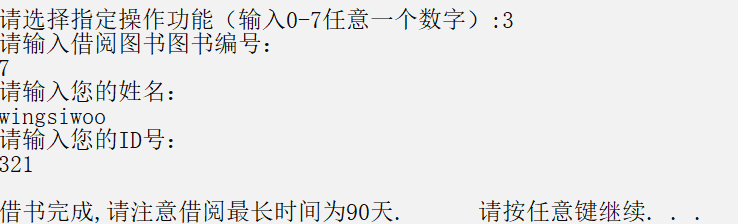
1. 借阅图书

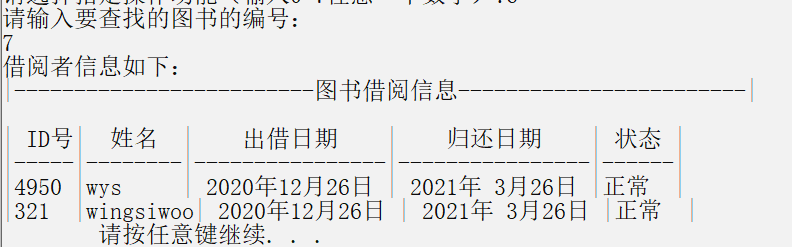
测试数据：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 借阅者ID | 借阅者姓名 | 借阅书本ID | 借书时间 | 预计还书时间 |
| 4950 | wys | 7 | 2020年12月26日 | 2021年 3月26日 |
| 321 | wingsiwoo | 7 | 2020年 12月26日 | 2021年 3月26日 |

测试结果：



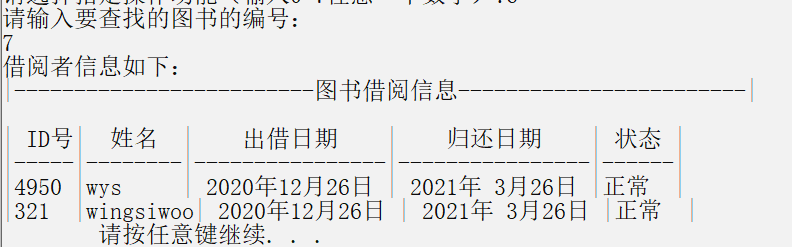




1. 查看某书借阅者信息

测试数据：7号书本借阅者有4950、321

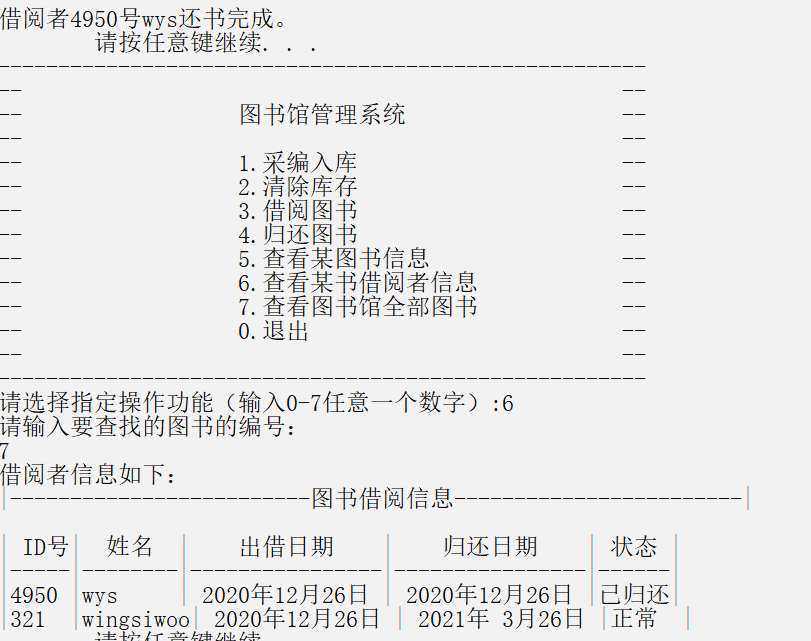
测试结果：



1. 归还图书

测试数据：4950号借阅者归还图书7

测试结果：

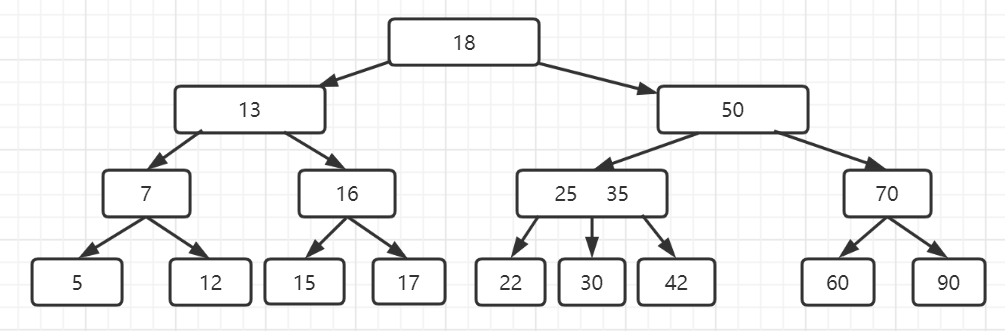


1. 清除库存

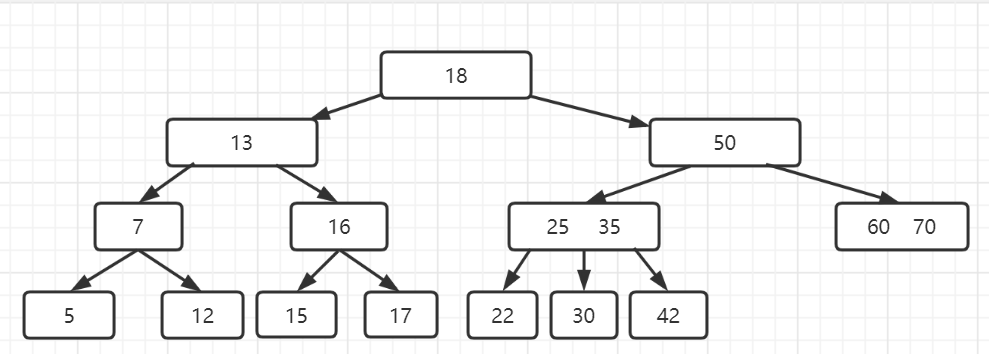
测试数据：在（1）基础上按顺序删除45, 90, 50, 22, 42

预期结果：

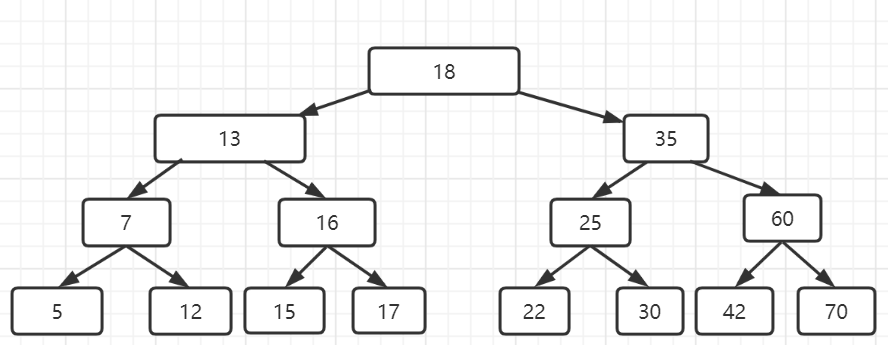
**删除45**



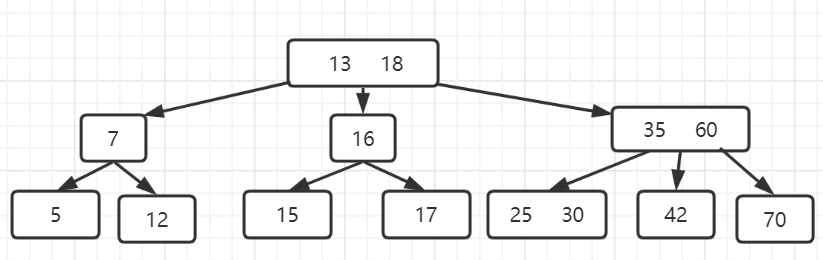
**删除90**



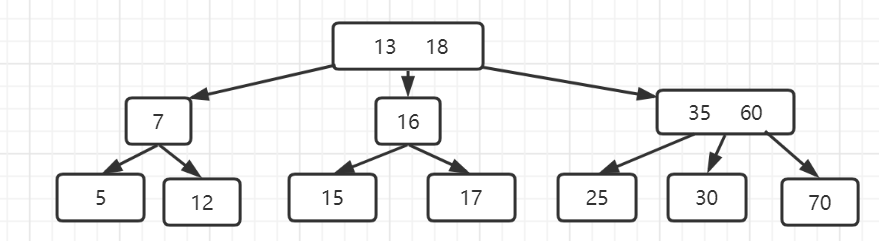
**删除50**



**删除22**

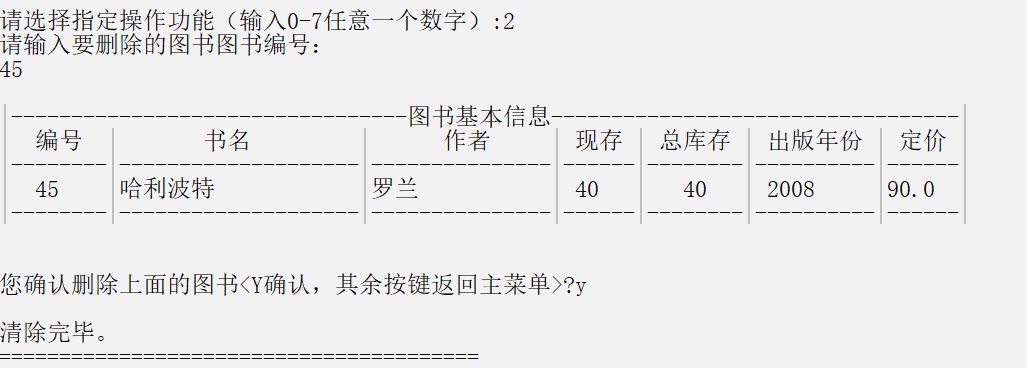


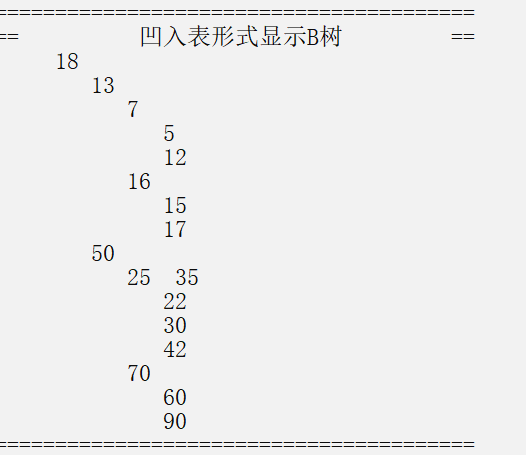
**删除42（最终结果）**



测试结果：

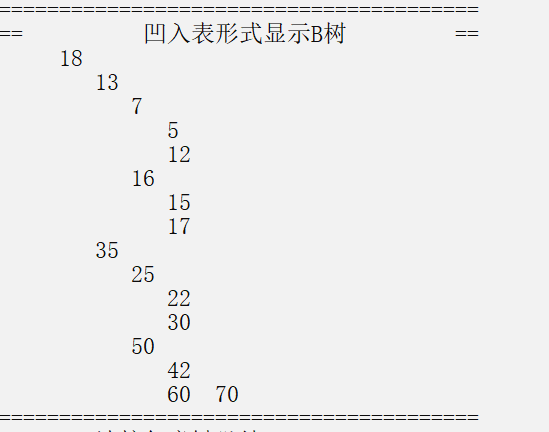
**删除45**



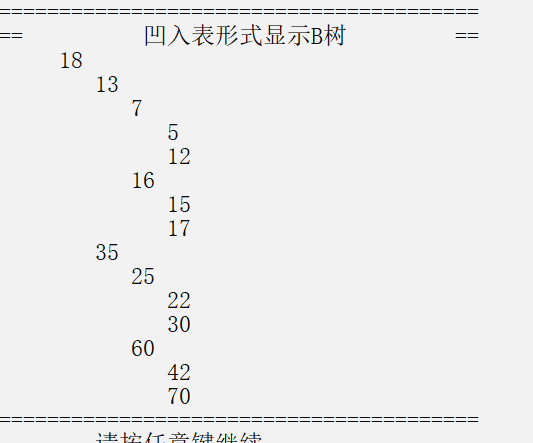


其余数据的删除操作演示省略，只展示每个结点删除后的树。

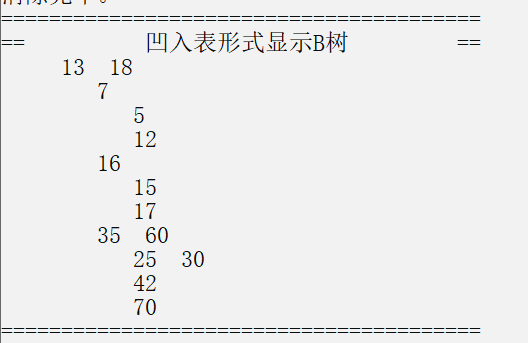
**删除90**



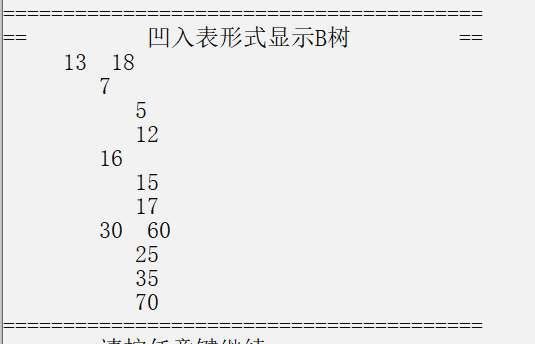
**删除50**



**删除22**



**删除42**



1. **思考与小结**
2. 代码健壮性：

图书管理系统面向用户，要有一定的兼容性和容错性。比如选择操作菜单中如果用户不慎输入了错误的操作代码，应该提示重新输入，如果不做处理会导致程序无响应，导致用户体验感降低。清空、删除、借阅、归还等操作前都要先判断书库是否为空，以及该书是否真实存在。

1. 时间复杂度：

B树的查找操作时间复杂度为O(logn)。 每一次查找操作都是一次磁盘IO，而由于磁盘IO的效率远远低于程序运行的效率，因此程序查找操作的效率主要取决于磁盘IO的次数。因此我们设计算法的时候应该尽可能降低磁盘IO次数以提升查询效率。而B树、B+树是十分有效的解决方法，因此许多数据库（比如MySQL）都是以B树或B+树作为索引的存储结构，目的是以空间换时间，大大提高查询效率。本次系统演示中B树的阶数为3，而实际应用中B树的阶往往可以达到100多。

1. 空间复杂度：

B树是一种以空间换时间的策略，需要树形结构，相比顺序存储需要占用更多的空间，但也有链接型数据结构灵活可拓展的优点。

1. 用户友好度

操作界面应该尽可能简洁明了，清晰。