



**数据结构设计性实验报告**

课程名称 **抽象数据类型的实现**

题 目 (10) B树

学 院 计算机学院

专 业 软件工程

班 别 4

学 号 3115005372

姓 名 杨宇杰

指导老师 杨劲涛

**2016 年 12 月 5日**

目录

[B树抽象数据类型实现 2](#_Toc468611220)

[（1）设计任务、要求及所用软件环境或工具 2](#_Toc468611221)

[设计要求: 2](#_Toc468611222)

[设计任务: 2](#_Toc468611223)

[所用的软件环境或工具： 2](#_Toc468611224)

[（2）抽象数据类型定义以及各基本操作的简要描述及实现和测试 3](#_Toc468611225)

[1. 题目 3](#_Toc468611226)

[2．存储结构定义 4](#_Toc468611227)

[3.关键算法设计流程图解释 6](#_Toc468611228)

[4. 算法设计 10](#_Toc468611229)

[5．测试 19](#_Toc468611230)

[6. 测试结果 20](#_Toc468611231)

[(3)思考和小结 26](#_Toc468611232)

[性能分析： 26](#_Toc468611233)

[小结： 26](#_Toc468611234)

[（4）实验总结和体会。 26](#_Toc468611235)

[实验总结： 26](#_Toc468611236)

[体会： 27](#_Toc468611237)

# B树抽象数据类型实现

## （1）设计任务、要求及所用软件环境或工具

### 设计要求:

1．首先了解设计的任务，然后根据自己的基础和能力从中选择一题。一般来说，选择题目应以在规定的时间内能完成，并能得到应有的锻炼为原则。 若学生对教材以外的相关题目较感兴趣，希望选作实验的题目时，应征得指导教师的认可，并写出明确的抽象数据类型定义及说明。

2. 实验前要作好充分准备，包括：理解实验要求，掌握辅助工具的使用，了解该抽象数据类型的定义及意义，以及其基本操作的算法并设计合理的存储结构。

3. 实验时严肃认真，要严格按照要求独立进行设计，不能随意更改。注意观察并记录各种错误现象，纠正错误，使程序满足预定的要求，实验记录应作为实验报告的一部分。

4. 实验后要及时总结，写出实验报告，并附所打印的问题解答、程序清单，所输入的数据及相应的运行结果。

### 设计任务:

本次实验我选择了B树，实现了B树的创建，查找，删除，插入，销毁，遍历这6种基本操作，并根据这个基本操作设计了友好的测试界面，操作简单易懂。

其中数据来源为一组在0~1000内的int型随机数，但数据由typedef int KeyType定义，若需要改变数据类型，只需要将int替换成所需的数据类型即可。

### 所用的软件环境或工具：

Microsoft Visual Studio 2015

## （2）抽象数据类型定义以及各基本操作的简要描述及实现和测试

### ****1. 题目****

采用KeyType类型(本次实验默认为int)为元素类型实现抽象数据类型BTree。

ADT BTree{

数据对象：T是具有相同特征的数据元素集合。

数据关系：

若D为空集，则称为空树；

（1）树中每个结点最多含有m棵子树；

（2）若根结点不是叶子结点，则至少有2个子树；

（3）除根结点之外的所有非终端结点至少有┌m/2┐棵子树；

（4）每个非终端结点中包含信息：（n，A0，K1，A1，K2，A2，…，Kn，An）。其中：

1）Ki（1<=i<=n）为关键字，且关键字按升序排序；

2）指针Ai（0<=i<=n）指向子树的根结点，Ai-1指向子树中所有结点的关键字均小于Ki，且大于Ki-1；

3）关键字的个数n必须满足：┌m/2┐-1<=n<=m-1。 （5）所有的叶子节点都在同一层，子叶结点不包含任何信息。

#### 基本操作：

void CreatBTree(BTree&T, int n, int m);

初始条件:初始化关键字个数n大于等于0，B树的阶数m大于3小于等于20

操作结果:构建一颗阶数为m,含有n个关键字的B树

void SearchBTree(BTree T, int k, result &r);

初始条件:树T存在

操作结果:在m阶B数T上查找关键字k，返回p{pt,i,tag}

void InsertBTree(BTree &T, int k, BTree q, int i,int m);

初始条件:树T存在

操作结果:在B树T上结点p->pt的key[i]和key[i+1]之间插入关键字k

void DeleteBTree(BTree p, int i, int m, BTree &T);

初始条件:B树上p结点存在

操作结果:删除B树T上结点p->pt的关键字k

void PrintBTree(BTree T);

初始条件:树T存在

操作结果:中序遍历B树

void DestroyBTree(BTree T);

初始条件:树T存在

操作结果:销毁B树

} ADT BTree

### ****2．存储结构定义****

公用头文件BTree.h:

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#define TRUE 1

#define FALSE 0

#define OVERFLOW -1

#define OK 1

#define ERROR 0

#define M 20//定义阶数最大值

typedef int KeyType;

typedef int Status;

typedef struct { //记录的结构定义

KeyType key;

char data;

}Record;

typedef struct BTNode{ //B树结点类型定义

int keynum; //结点中关键字个数，即结点的大小

KeyType key[M + 1]; //关键字，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

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

}BTNode,\*BTree; //B树结点和B树类型

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,\*resultPtr; //B树的查找结果类型

//接口定义

void CreatBTree(BTree&T, int n, int m);

/\*

初始条件:初始化关键字个数n大于等于0，B树的阶数m大于3小于等于20

操作结果:构建一颗阶数为m,含有n个关键字的B树

\*/

void SearchBTree(BTree T, int k, result &r);

/\*

初始条件:树T存在

操作结果:在m阶B数T上查找关键字k，返回p{pt,i,tag}

\*/

void InsertBTree(BTree &T, int k, BTree q, int i,int m);

/\*

初始条件:树T存在

操作结果:在B树T上结点p->pt的key[i]和key[i+1]之间插入关键字k

\*/

void DeleteBTree(BTree p, int i, int m, BTree &T);

/\*

初始条件:B树上p结点存在

操作结果:删除B树T上结点p->pt的关键字k

\*/

void PrintBTree(BTree T);

/\*

初始条件:树T存在

操作结果:中序遍历B树

\*/

void DestroyBTree(BTree T);

/\*

初始条件:树T存在

操作结果:销毁B树

\*/

int menu();

/\*

输出选择菜单

\*/

### 3.关键算法设计流程图解释

主函数：main();

功能：定义测试所需的变量，提供基本的菜单功能选择



函数名：void SearchBTree(BTree T, int k, result &r);

功能：在m阶B树t上查找关键码k，返回(pt,i,tag)。



函数名：void InsertBTree(BTree &T, int k, BTree q, int i,int m);

功能：在B树t上结点\*q的key[i]和key[i+1]之间插入关键字k。



函数名：void Restore(BTree &p, int i, int m, BTree &T);

功能：调整B树。



函数名：void DeleteBTree(BTree p, int i, int m, BTree &T);

功能：删除B树T上结点p的关键字k



### ****4. 算法设计****

操作函数源文件:OperationDefine.cpp

#include "BTree.h"

void CreatBTree(BTree &T, int n, int m) {//构建一颗阶数为m,含有n个关键字的B树(3<=m<=M,0<=n<=10000)

//创建B树

int i, j;

resultPtr p = NULL;

p = (result\*)malloc(sizeof(result));

srand((unsigned)time(NULL));

if (n == 0)

printf("已成功初始化一棵空树。\n");

else {

for (j = 0; j < n; j++) {

i = rand() % 1000; //生成随机数i

SearchBTree(T, i, \*p); //查找i插入位置

InsertBTree(T, i, p->pt, p->i, m); //进行插入

}

printf("创建B树成功！\n");

}

}

void PrintBTree(BTree T) {

//中序遍历B树

int i = 1;

if (NULL != T) {

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

PrintBTree(T->ptr[i - 1]);

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

}

PrintBTree(T->ptr[i - 1]);

}

}

int Search(BTree p, int k) {

int i = 1;

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

i++;

return i;

}

void SearchBTree(BTree T, int k, result &r) {

//在m阶B树T上查找关键字k，返回(pt,i,tag)

//若查找成功，则特征值tag=1,指针pt所致结点中第i个关键字等于k;否则

//特征值tag=0，等于k的关键字记录应插入在指针pt所指结点中第i-1个和第i个关键字间

int i = 0, found = 0;

BTree p = T, q = NULL;

while (p != NULL && 0 == found) {

i = Search(p, k);//在p->key[1..keynum]中查找p->key[i-1]<k<=p->p->key[i]

if (i > 0 && p->key[i] == k)

found = 1;//找到待查关键字

else{

q = p;

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

}

}

if (1 == found) {//查找成功

r.pt = p;

r.i = i;

r.tag = 1;

}

else {//查找不成功，返回key的插入位置i

r.pt = q;

r.i = i;

r.tag = 0;

}

}

void split(BTree &q, int s, BTree &ap) {

//将q结点分裂成两个结点，前一半保留，后一半移入新结点ap

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

ap = (BTNode\*)malloc(sizeof(BTNode));//生成新结点ap

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

for (i = s + 1, j = 1; i <= n; i++, j++) {//后一半移入ap结点

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

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

}

ap->keynum = n - s;

ap->parent = q->parent;

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

if (ap->ptr[i])

ap->ptr[i]->parent = ap;//将ap所有孩子结点指向ap

}

q->keynum = s - 1;//q结点的前一半保留，修改keynum

}

void newroot(BTree &T, BTree p, int x, BTree ap) {//生成新的根结点

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

T->keynum = 1;

T->ptr[0] = p;

T->ptr[1] = ap;

T->key[1] = x;

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

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

T->parent = NULL;//新根的双亲是空指针

}

void Insert(BTree &q, int i, int x, BTree ap) {//x和ap分别插到q->key[i]和q->ptr[i]

int j, n = q->keynum;

for (j = n; 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->key[i] = x;//赋值

q->ptr[i] = ap;

if (ap != NULL) ap->parent = q;

q->keynum++;//关键字数+1

}

void InsertBTree(BTree &T, int k, BTree q, int i, int m) {

//在B树T上q结点的key[i-1]和key[i]之间插入关键字k

//若引起结点过大,则沿双亲指针进行必要的结点分裂调整,使T仍是m阶的B树

int x, s, finished = 0, neednewroot = 0;

BTree ap;

if (NULL == q)//q为空，则新建根结点

newroot(T, NULL, k, NULL);

else {

x = k;

ap = NULL;

while (0 == neednewroot && 0 == finished) {

Insert(q, i, x, ap);//key和ap分别插到q->key[i]和q->ptr[i]

if (q->keynum < m) finished = 1;//插入完成

else {//分裂q结点

s = (m + 1) / 2;

split(q, s, ap);

x = q->key[s];

if (q->parent != NULL) {

q = q->parent;

i = Search(q, x);//在双亲结点中查找x的插入位置

}

else neednewroot = 1;

}

}//while

if (1 == neednewroot)//T是空树或者根结点已分裂为q和ap结点

newroot(T, q, x, ap);//生成含信息(q,x,ap)的新的根结点T

}

}

void Successor(BTree &p, int i) {//由后继最下层非终端结点的最小关键字代替结点中关键字key[i]。

BTNode \*temp;

temp = p->ptr[i];

for (; NULL != temp->ptr[0]; temp = temp->ptr[0]) ;//找出关键字的后继

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

p = temp;

}

void Remove(BTree &p, int i) { //从结点p中删除key[i]

int j;

int n = p->keynum;

for (j = i; j < n; j++) { //关键字左移

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

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

}

p->keynum--;

}

void Restore(BTree &p, int i, int m, BTree &T) {//调整B树

int j;

BTree ap = p->parent;

BTree lc, rc, pr;

int finished = 0, r = 0;

while (0 == finished) {

r = 0;

while (ap->ptr[r] != p)//确定p在ap子树的位置

r++;

if (r == 0) {

r++;

lc = NULL;

rc = ap->ptr[r];

}

else if (r == ap->keynum) {

rc = NULL;

lc = ap->ptr[r - 1];

}

else {

lc = ap->ptr[r - 1];

rc = ap->ptr[r + 1];

}

if (r > 0 && lc != NULL && (lc->keynum > (m - 1) / 2)) {//向左兄弟借关键字

p->keynum++;

for (j = p->keynum; j > 1; j--) {//结点关键字右移

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

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

}

p->key[1] = ap->key[r];//父亲插入到结点

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

p->ptr[0] = lc->ptr[lc->keynum];

if (NULL != p->ptr[0])//修改p中的子女的父结点为p

p->ptr[0]->parent = p;

ap->key[r] = lc->key[lc->keynum];//左兄弟上移到父亲位置

lc->keynum--;

finished = 1;

break;

}

else if (ap->keynum > r&&rc != NULL && (rc->keynum > (m - 1) / 2)) {

p->keynum++;

p->key[p->keynum] = ap->key[r];//父亲插入到结点

p->ptr[p->keynum] = rc->ptr[0];

if (NULL != p->ptr[p->keynum]) {//修改p中的子女的父结点为p

p->ptr[p->keynum] = rc->ptr[0];

}

ap->key[r] = rc->key[1];//右兄弟上移到父亲位置

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

for (j = 1; j < rc->keynum; j++) {//右兄弟结点关键字左移

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

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

}

rc->keynum--;

finished = 1;

break;

}

r = 0;

while (ap->ptr[r] != p) r++;//重新确定p在ap子树的位置

if (r > 0 && (ap->ptr[r - 1]->keynum <= (m - 1) / 2)) {//与左兄弟合并

lc = ap->ptr[r - 1];

p->keynum++;

for (j = p->keynum; j > 1; j--) {//将p结点关键字和指针右移1位

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

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

}

p->key[1] = ap->key[r];//父结点的关键字与p合并

p->ptr[1] = p->ptr[0];//从左兄弟右移一个指针

ap->ptr[r + 1] = lc;

for (j = 1; j <= lc->keynum + p->keynum; j++) {//将结点p中关键字移到p左兄弟中

lc->key[lc->keynum + j] = p->key[j];

lc->ptr[lc->keynum + j] = p->ptr[j];

}

if (p->ptr[0]) {//修改p中的子女的父结点为lc

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

p->ptr[p->keynum + j]->parent = lc;

}

}

lc->keynum = lc->keynum + p->keynum;//合并后的关键字个数

ap->keynum--;

pr = p;

free(pr);//释放p结点空间

pr = NULL;

p = lc;

}

else {//与右兄弟合并

rc = ap->ptr[r + 1];

if (r == 0) r++;

p->keynum++;

p->key[p->keynum] = ap->key[r];//父结点的关键字与p合并

p->ptr[p->keynum] = rc->ptr[0];//从右兄弟左移一个指针

rc->keynum = p->keynum + rc->keynum;//合并后关键字的个数

ap->ptr[r - 1] = rc;

for (j = 1; j <= (rc->keynum - p->keynum); j++) {//将p右兄弟的关键字和指针右移

rc->key[p->keynum + j] = rc->key[j];

rc->ptr[p->keynum + j] = rc->ptr[j];

}

for (j = 1; j <= p->keynum; j++) {//将结点p中关键字和指针移到p右兄弟中

rc->key[j] = p->key[j];

rc->ptr[j] = p->ptr[j];

}

rc->ptr[0] = p->ptr[0];

if (p->ptr[0]) {//修改p中的子女的父结点为rc

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

p->ptr[p->keynum + j]->parent = rc;

}

}

for (j = r; j < ap->keynum; j++) {//将父结点中关键字和指针左移

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

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

}

ap->keynum--;//父结点的关键字个数减1

pr = p;

free(pr);//释放p结点空间

pr = NULL;

p = rc;

}

ap = ap->parent;

if (p->parent->keynum >= (m - 1) / 2 || (NULL == ap&&p->parent->keynum > 0)) {

finished = 1;

}

else if (NULL == ap) {//若调整后出现空的根结点，则删除该根结点，树高减1

pr = T;

T = p;//根结点下移

free(pr);

pr = NULL;

finished = 1;

}

p = p->parent;

}

}

void DeleteBTree(BTree p, int i, int m, BTree &T) {

//删除B树上p结点第i个关键字

if (p->ptr[i - 1] != NULL) {

Successor(p, i); //若不是最下层非终端结点

DeleteBTree(p, 1, m, T); //由后继最下层非终端结点的最小关键字代替它

}

else {//若是最下层非终端结点

Remove(p, i); //从结点p中删除key[i]

if (p->keynum < (m - 1) / 2) //删除后关键字个数小于(m-1)/2

Restore(p, i, m, T); //调整B树

}

}

void DestroyBTree(BTree T) {

int i = 1;

if (NULL != T) {

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

DestroyBTree(T->ptr[i - 1]);

free(T->ptr[i - 1]);

}

DestroyBTree(T->ptr[i - 1]);

}

}

int menu() {//菜单

int choice;

printf("\t\t\t|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n");

printf("\t\t\t|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n");

printf("\t\t\t \_\_\_\_\_\_\_\_\_\_\_\_\_请先创建B树再进行操作！\_\_\_\_\_\_\_\_\_\_\n");

printf("\t\t\t| B树测试界面 |\n");

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

printf("\t\t\t| 1.创建B树 2.B树结点的查找 |\n");

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

printf("\t\t\t| 3.B树结点的插入 4.B树结点的删除 |\n");

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

printf("\t\t\t| 5.B树的遍历 6.B树的销毁 |\n");

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

printf("\t\t\t| 0.退出 |\n");

printf("\t\t\t|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

printf("\t\t\t|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n");

printf("\t\t\t|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n");

printf("\t\t\t| 15软件工程(4)班 |\n");

printf("\t\t\t| 3115005372 |\n");

printf("\t\t\t| 杨宇杰 |\n");

printf("\t\t\t|\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*S\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*|\n");

do {

printf("\t\t\t请选择功能（输入1-6任意一个数字）:");

scanf\_s("%d", &choice);

} while (choice<0||choice>6);//避免非法输入

return choice;

}

### 5．测试

BTree\_Test.cpp

#include<stdio.h>

#include"BTree.h"

int main() {

BTree T=NULL;

result r;

int choice, k, i, m, n;

do{

choice = menu();

if (choice >= 0 && choice <= 7) {

system("cls");//把菜单清除

switch (choice) {

case 1:

printf("请输入B树的阶数m:(3<=m<=20)\n");

scanf\_s("%d", &m);

printf("请输入B树的初始化关键字个数:(0<=n<=10000)\n");

scanf\_s("%d", &n);

CreatBTree(T, n, m);

break;

case 2:

printf("请输入要查找的关键字：\n");

scanf\_s("%d", &k);

SearchBTree(T, k, r);

if (r.tag) {

printf("该关键字的位置为该结点中第%d个关键字\n", r.i);

}

else {

printf("该关键字不存在！\n");

}

break;

case 3:

printf("请输入要插入的关键字k：\n");

scanf\_s("%d", &k);

SearchBTree(T, k, r);

InsertBTree(T, k, (&r)->pt, (&r)->i, m);

printf("插入成功！");

break;

case 4:

printf("请输入要删除B树T上的关键字：\n");

scanf\_s("%d", &i);

SearchBTree(T, i, r);

DeleteBTree(r.pt, r.i, m, T);

printf("删除成功！\n");

break;

case 5:

printf("此时的B树序列为：\n");

PrintBTree(T);

printf("\n");

break;

case 6:

DestroyBTree(T);

printf("销毁成功！\n");

break;

default:;

}

}

}while (choice > 0 && choice < 7);

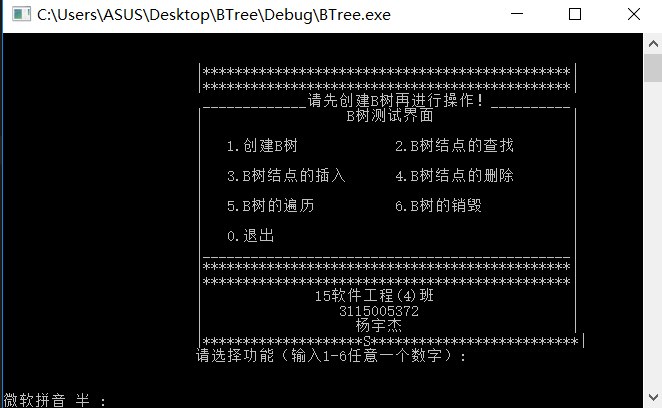
return 0;

}

### 6. 测试结果

/\*在VS 2015 的环境下进行测试\*/

1. 程序主界面：



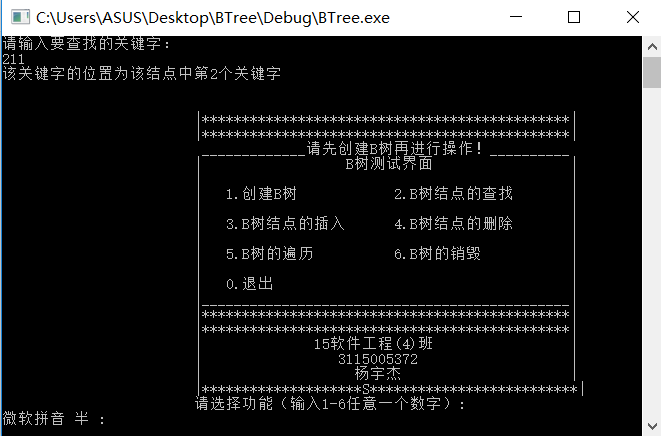
1. B树的创建：(随机创建一棵3阶B树，共有15个关键字)



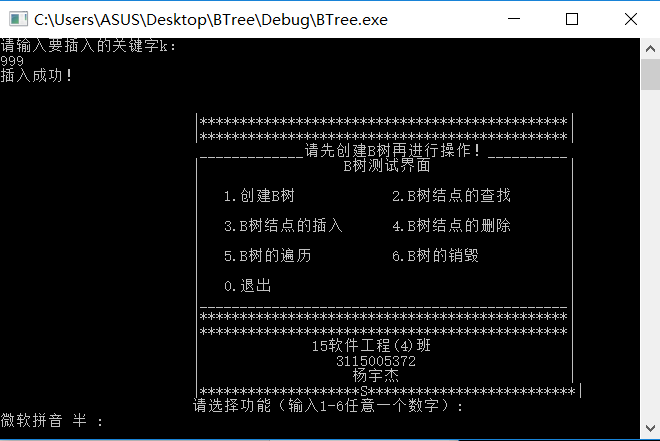
1. B树的遍历：



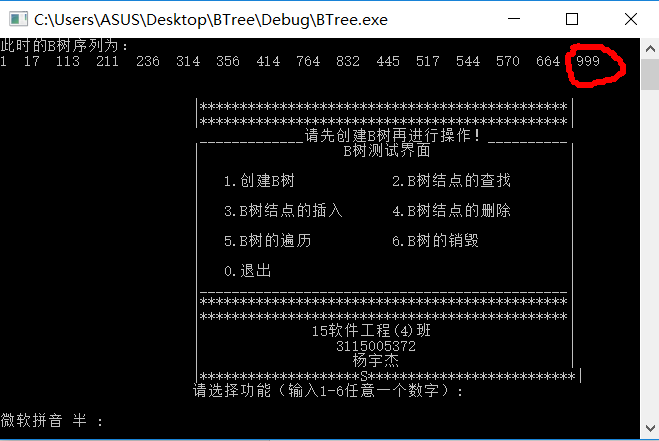
1. B树结点的查找：



1. B树结点的插入：

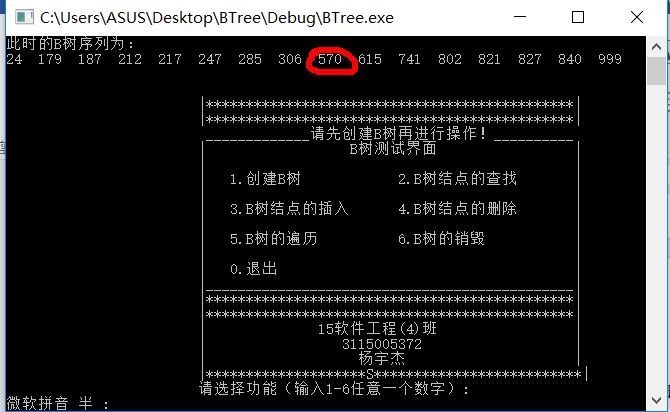


此时的B树中增加了999这个结点：

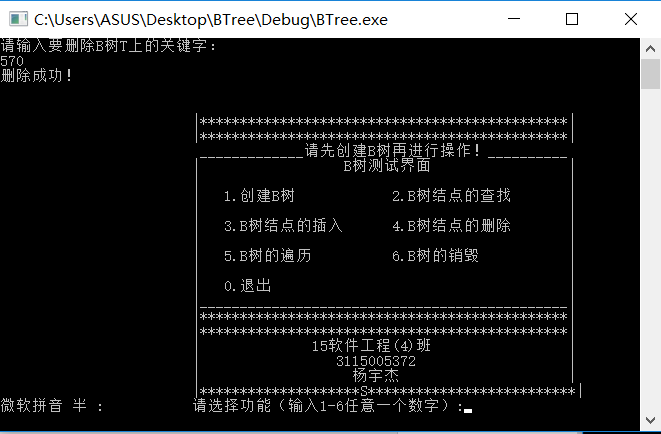


1. B树结点的删除(删除570)：

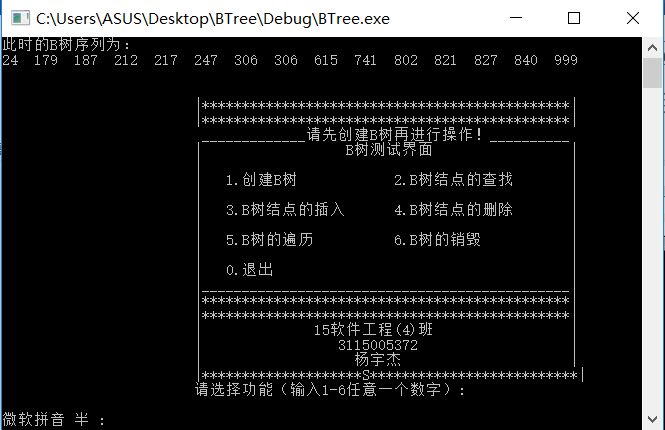
删除前：



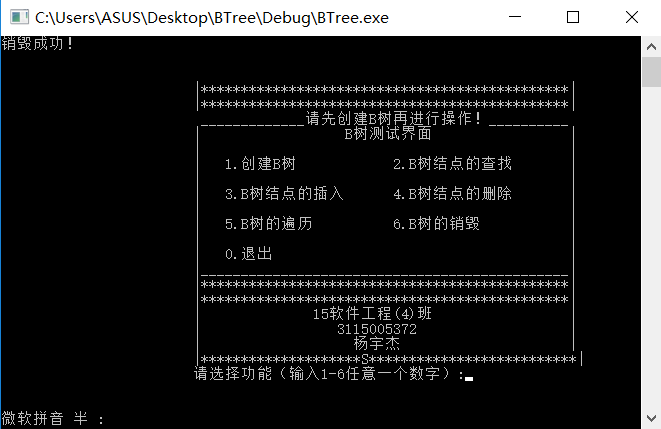
删除操作：



删除后已无570：



1. B树的销毁：



## (3)思考和小结

### 性能分析：

设B树包含N个关键字，因此有N+1个叶子结点，叶子都在第I层。因为根至少有两个孩子，因此第二层至少有两个结点。除根和叶子外，其它结点至少有┌m/2┐个孩子，因此在第三层至少有2\*┌m/2┐个结点，在第四层至少有2\*(┌m/2┐^2)个结点，．．．，在第I层至少有2\*(┌m/2┐^(l-2) )个结点，于是有：

N+1 ≥ 2\*┌m/2┐I-2

考虑第L层的结点个数为N+1，那么2\*(┌m/2┐^(l-2)）≤N+1，也就是L层的最少结点数刚好达到N+1个

即： I≤ log┌m/2┐((N+1)/2 )+2

所以，当B树包含N个关键关键字时，B树的最大高度为l-1（因为计算B树高度时，叶结点所在层不计算在内）

即：log┌m/2┐((N+1)/2 )+1。

这个公式保证了B树的查找效率是相当高的。

### 小结：

B树是一种查找效率很高的数据结构，具有很高的灵活性，一个结点可以有多个关键字，节省了空间，但实现难度较高。

## （4）实验总结和体会。

### 实验总结：

本次设计的难点在于B树的调整算法，B树在插入的过程中可能发生上溢，此时需要通过分裂结点来处理，在此过程中B树可能出现长高。而在删除的过程中则可能发生下溢，此时需要分：借左兄弟、借右兄弟、与左兄弟合并、与右兄弟这四种调整方式来处理，并要处理合并后根结点为空的情况，在此过程中B树可能出现高度降低。

### 体会：

这一次实验我选择的是B树，相比于其他数据结构，B树是算难度比较大的，所以这次也是对自己一个挺大的考验，总的来说还是花费了不少精力，有一些棘手的问题就通过查看技术大牛们的博客还有百度一下编程思想，感觉对自己是一种极大的提高。通过这次大作业，我更加深刻的了解到了课本的知识，对以往知识的疏忽也得到了补充。课程设计是一个重要的教学环节，通过课程设计让我们了解到一些实际与理论之间的差距。通过课程设计不仅可以巩固我的专业知识，还可以培养我熟练使用资料的能力，让我能把课本知识和实践结合起来，起到温故而知新的作用。但我认为我这次的课设还是可以有较大的提升空间的。

总而言之，在这次设计中我深入理解了B树的各项基本操作，也很好地锻炼了自己的动手能力，收获匪浅。