数据结构实验报告

（2017-2018第一学期）

**实验题目：** 实验七 检索

**学 号：**  20151305212

**姓 名：**  崔健骅

**专业年级：** 信息安全16-1

**指导教师：** 崔青

**实验日期：**  2017-11-17

实验七 检索

**实验目的：**

1. 掌握查找的含义。
2. 掌握基本查找操作的算法与实现。
3. 掌握二叉排序树查找的算法与实现。
4. 深入理解各种检索算法的结构特点及各算法之间的区别。
5. 能通过所学的检索算法解决一些实际问题。

**实验内容：**

1. 建立一个线性表，对表中数据元素存放的先后次序没有任何要求。输入待查数据元素的关键字进行查找。（为了简化算法，数据元素只含一个整型量关键字字段，数据元素的其余数据部分忽略不考虑。）
2. 查找表的存储结构为有序表进行二分检索，即表中记录按关键字大小排序存放。输入待查数据元素的关键字进行查找。（为了简化算法，记录只含一个整型量关键字字段，记录的其余数据部分略不考虑。程序要求对整型量关键字数据的输入按从小到大排序输入。）
3. 编程实现二叉排序树的创建、查找、插入、删除、输出等算法。设计主函数测试。

程序源代码：

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define ERROR 0

#define OK 1

#define MAXSIZE 100

typedef int Status;

typedef struct {

int data[MAXSIZE];

int length;

}SeqList; //下标0处不存放数据

typedef struct BiTNode{

int data;

struct BiTNode \*left;

struct BiTNode \*right;

}BiTNode, \*BiTree;

void InsertSort(SeqList \*L);

int SeqSearch(SeqList L, int key);

int BinarySearch(SeqList L, int key);

Status SearchBST(BiTree T, int key, BiTree Par, BiTree \*Pos);

Status InsertBST(BiTree \*T, int key);

void CreateBST(BiTree \*T, int \*Num, int n);

Status Delete(BiTree \*T);

Status DeleteBST(BiTree \*T, int key);

void PrintBST(BiTree T);

void Menu();

void Menu()

{

SeqList L, SL;

BiTree T = NULL;

BiTree Pos;

int BNum[MAXSIZE];

int key, n, i;

char cmd,tmp;

const char \*legal="eEsSbBcCfFiIdDpPqQ";

do

{

putchar('\n');

printf("e,E 线性表创建\n");

printf("s,S 顺序查找\n");

printf("b,B 二分查找\n");

printf("c,C 二叉排序树创建\n");

printf("f,F 二叉排序树查找\n");

printf("i,I 二叉排序插入\n");

printf("d,D 二叉排序删除\n");

printf("p,P 二叉排序输出\n");

printf("q,Q 退出\n");

putchar('\n');

do

{

printf("cmd: ");

while((tmp=getchar()) == '\n')

;

cmd = tmp;

}

while( strchr(legal, cmd)==NULL );

putchar('\n');

switch(cmd)

{

case 'e':

case 'E':

printf("Input the Num Of Data: ");

scanf("%d", &(L.length));

if(L.length>MAXSIZE || L.length<=0)

printf("The Data Is Too Long or Too Short!Retry!\n");

else

{

printf("Input the Data: ");

for(i=1; i<=L.length; i++)

scanf("%d", &(L.data[i]));

}

break;

case 's':

case 'S':

printf("Input the Searched Data: ");

scanf("%d", &key);

if(i=SeqSearch(L, key))

printf("Loc: %d\n", i);

else

printf("Not Found!\n");

break;

case 'b':

case 'B':

printf("Input the Searched Data: ");

scanf("%d", &key);

SL.length = L.length;

memcpy(SL.data, L.data, (sizeof(int)\*(L.length+1)));

InsertSort(&SL);

printf("Sorted Data: ");

for(i=1; i<=SL.length; i++)

printf("%d ", SL.data[i]);

putchar('\n');

if(i=BinarySearch(SL, key))

printf("Loc: %d\n", i);

else

printf("Not Found!\n");

break;

case 'c':

case 'C':

printf("Input the Num Of Data: ");

scanf("%d", &n);

if(n>MAXSIZE || n<=0)

printf("The Data Is Too Long or Too Short!Retry!\n");

else

{

printf("Input the Data: ");

for(i=0; i<n; i++)

scanf("%d", &BNum[i]);

CreateBST(&T, BNum, n);

putchar('\n');

PrintBST(T);

}

break;

case 'f':

case 'F':

printf("Input the Searched Data: ");

scanf("%d", &key);

if(SearchBST(T, key, NULL, &Pos))

printf("Found!");

else

printf("Not Found!");

break;

case 'i':

case 'I':

printf("Input the Data: ");

scanf("%d", &key);

if(InsertBST(&T, key))

{

putchar('\n');

PrintBST(T);

}

else

printf("The Data exists!\n");

break;

case 'd':

case 'D':

printf("Input the Data: ");

scanf("%d", &key);

DeleteBST(&T, key);

putchar('\n');

PrintBST(T);

break;

case 'p':

case 'P':

putchar('\n');

PrintBST(T);

break;

}

}

while((cmd!='q') && (cmd!='Q'));

}

int main(void)

{

Menu();

return 0;

}

void InsertSort(SeqList \*L)

{

int i, j;

int key;

for(j=2; j<=L->length; j++)

{

key = L->data[j];

i = j-1;

while(i>=1 && key<L->data[i])

{

L->data[i+1] = L->data[i];

i--;

}

L->data[i+1] = key;

}

}

int SeqSearch(SeqList L, int key)

{

int i;

L.data[0] = key;

i = L.length;

while(L.data[i] != key)

i--;

return i;

}

int BinarySearch(SeqList L, int key)

{

int Low, Mid, High;

Low = 1;

High = L.length;

while(Low<=High)

{

Mid = (Low + High) / 2;

if(L.data[Mid] < key)

Low = Mid + 1;

else if(L.data[Mid] > key)

High = Mid - 1;

else

return Mid;

}

return 0;

}

/\*

\*@T: Current Node

\*@Par: Current Node's Parent

\*@Pos: Successful ? Final Node : Former Node

\*/

Status SearchBST(BiTree T, int key, BiTree Par, BiTree \*Pos)

{

if(!T)

{

\*Pos = Par;

return ERROR;

}

else if(T->data == key)

{

\*Pos = T;

return OK;

}

else if(key < T->data)

SearchBST(T->left, key, T, Pos);

else if(key > T->data)

SearchBST(T->right, key, T, Pos);

}

Status InsertBST(BiTree \*T, int key)

{

BiTree Pos, tmp;

if( !SearchBST(\*T, key, NULL, &Pos) )

{

tmp = (BiTNode \*)malloc(sizeof(BiTNode));

tmp->data = key;

tmp->left = tmp->right = NULL;

if(!Pos) //T为空树

\*T = tmp;

else if(key < Pos->data)

Pos->left = tmp;

else

Pos->right = tmp;

return OK;

}

else

return ERROR;

}

void CreateBST(BiTree \*T, int \*Num, int n)

{

int i;

for(i=0; i<n; i++)

InsertBST(T, Num[i]);

}

Status Delete(BiTree \*T)

{

BiTree tmp, Prev;

if((\*T)->left == NULL)

{

tmp = \*T;

\*T = (\*T)->right;

free(tmp);

}

else if((\*T)->right == NULL)

{

tmp = \*T;

\*T = (\*T)->left;

free(tmp);

}

else

{

tmp = \*T;

Prev = (\*T)->left;

while(Prev->right)

{

tmp = Prev;

Prev = Prev->right;

}

(\*T)->data = Prev->data;

if(tmp!=(\*T))

tmp->right = Prev->left;

else

(\*T)->left = Prev->left;

free(Prev);

}

return ERROR;

}

Status DeleteBST(BiTree \*T, int key)

{

if(!\*T) //未找到

return ERROR;

else

{

if((\*T)->data == key) //找到

return Delete(T);

else if(key < (\*T)->data)

return DeleteBST(&((\*T)->left), key);

else

return DeleteBST(&((\*T)->right), key);

}

}

void PrintBST(BiTree T)

{

if(T)

{

PrintBST(T->left);

printf("%d ", T->data);

PrintBST(T->right);

}

}

调试分析：

运行结果1:

e,E 线性表创建

s,S 顺序查找

b,B 二分查找

c,C 二叉排序树创建

f,F 二叉排序树查找

i,I 二叉排序插入

d,D 二叉排序删除

p,P 二叉排序输出

q,Q 退出

cmd: e

Input the Num Of Data: 5

Input the Data: 3 -3 0 1 2

cmd: s

Input the Searched Data: 2

Loc: 5

cmd: b

Input the Searched Data: 2

Sorted Data: -3 0 1 2 3

Loc: 4

cmd: c

Input the Num Of Data: 7

Input the Data: 3 -3 0 1 2 9 5

-3 0 1 2 3 5 9

cmd: f

Input the Searched Data: 9

Found!

cmd: i

Input the Data: 7

-3 0 1 2 3 5 7 9

cmd: d

Input the Data: 2

-3 0 1 3 5 7 9

cmd: p

-3 0 1 3 5 7 9

cmd: q

运行结果2:

e,E 线性表创建

s,S 顺序查找

b,B 二分查找

c,C 二叉排序树创建

f,F 二叉排序树查找

i,I 二叉排序插入

d,D 二叉排序删除

p,P 二叉排序输出

q,Q 退出

cmd: e

Input the Num Of Data: 7

Input the Data: 7 6 -1 4 5 1 9

cmd: s

Input the Searched Data: 1

Loc: 6

cmd: b

Input the Searched Data: 6

Sorted Data: -1 1 4 5 6 7 9

Loc: 5

cmd: c

Input the Num Of Data: 7

Input the Data: 7 6 -1 4 5 1 9

-1 1 4 5 6 7 9

cmd: f

Input the Searched Data: 5

Found!

cmd: f

Input the Searched Data: 10

Not Found!

cmd: i

Input the Data: 10

-1 1 4 5 6 7 9 10

cmd: d

Input the Data: -1

1 4 5 6 7 9 10

cmd: p

1 4 5 6 7 9 10

cmd: q

实验心得体会：

1. 二叉排序树的中序遍历结果即为排序结果。利用二叉排序树可维护一个有序数 列。
2. 二叉排序树中不允许出现重复元素，故在插入操作时要确定要插入元素是否已 在二叉排序树中，若已存在，则返回错误信息。