第九章作业答案

//9.26 折半查找法的递归算法

int Search\_Bin\_Recursive(SSTable ST, KeyType key, int low, int high){

if (low > high)

return 0;

mid = (low + high) / 2;

if (ST.elem[mid].key == key)

return mid;

else if (ST.elem[mid].key > key)

return Search\_Bin\_Recursive(ST, key, low, mid - 1);

else

return Search\_Bin\_Recursive(ST, key, mid + 1, high);

}//Search\_Bin\_Recursive

//9.31 判断二叉树T是否二叉排序树,是则返回1,否则返回0

//利用二叉排序树性质：中序遍历二叉排序树可得关键字的有序序列

//我这里假设key都是非负数，或者一开始设置前驱为空，另外再加一个判断条件

//判一下空会更好一点

int last = 0, flag = 1;

int Is\_BSTree(Bitree T, int last){

if (T->lchild&& flag)

Is\_BSTree(T->lchild);

if (T->data < last)

flag = 0;

last = T->data;

if (T->rchild&& flag)

Is\_BSTree(T->rchild);

return flag;

}//Is\_BSTree

//9.33 从大到小输出二叉排序树T中所有不小于x的元素

//注：本题要求使用递归算法

void Print\_NLT(BiTree T, int x){

if (T->rchild) Print\_NLT(T->rchild, x);

if (T->data < x) exit(); //当遇到小于x的元素时立即结束运行

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

if (T->lchild) Print\_NLT(T->lchild, x); //先右后左的中序遍历

}//Print\_NLT

//9.37 二叉排序树以后继线索链表为存储结构，删除元素x

//很多同学没有做线索的维护

Status BSTree\_Delete\_key(BiThrTree& T, int x){

p = T;

last = NULL;

while (!p->Ltag)

p = p->lchild;

while (p){

pre = last;

ptr = p;

if (p->Rtag)

q = p->rchild;

else{

q = p->rchild;

while (!q->Ltag)

q = q->lchild;

}//q为节点p的直接后继

if (p->data == x) {

suc = q;

break;

}else {

last = p;

p = q;

}

}

if (!ptr) return ERROR;

Delete\_BSTree(ptr); //可以直接用书本上的，也可以用下面那个

if (pre && pre->Rtag)

pre->rchild = suc;

return OK;

}//BSTree\_Delete\_key

void Delete\_BSTree(BiThrTree &T)//课本上给出的删除二叉排序树的子树T的算法,按照线索二叉树的结构作了一些改动

{

q = T;

if (!T->Ltag&& T->Rtag) //结点无右子树,此时只需重接其左子树

T = T->lchild;

else if (T->Ltag&& !T->Rtag) //结点无左子树,此时只需重接其右子树

T = T->rchild;

else if (!T->Ltag&& !T->Rtag) //结点既有左子树又有右子树

{

p = T; r = T->lchild;

while (!r->Rtag)

{

s = r;

r = r->rchild; //找到结点的前驱r和r的双亲s

}

T->data = r->data; //用r代替T结点

if (s != T)

s->rchild = r->lchild;

else s->lchild = r->lchild; //重接r的左子树到其双亲结点上

q = r;

}//else

free(q); //删除结点

}//Delete\_BSTree

//分析:本算法采用了先求出x结点的前驱和后继,再删除x结点的办法,这样修改线索时会比较简单,直接让前驱的线索指向后继就行了.

//如果试图在删除x结点的同时修改线索,则问题反而复杂化了.

**9.19**

|  |
| --- |
| H(22) = 0;不冲突。查找次数为1次。 |
| H(41) = 2; 不冲突。查找次数为1次。 |
| H(53) = 5; 不冲突。查找次数为1次。 |
| H(46) = 6; 不冲突。查找次数为1次。 |
| H(30) = 2，冲突，重新分配；  d1=1\*（（7\*30）MOD 10）+ 1)=1，  H1(30) = (H(30) + d1)MOD11 = 3，不冲突。  此时30的查找次数为2次。 |
| H(13) = 6；冲突，重新匹配  H1(13) = 8；不冲突。  查找次数为2次。 |
| H(01)=3 冲突，重新分配；  d1=1\*((7\*k) MOD 10 +1)=1\*（(7\*1) mod 10 +1）=8 H1=(H(KEY)+d1) MOD 11=(3+8) mod 11=0 冲突，需要重新分配； d2=2\*((7\*k) MOD 10 +1)=2\*（(7\*1) mod 10 +1)=16 H2=(H(KEY)+d2) MOD 11=(3+16) mod 11=8冲突，需要重新分配； d3=3\*（(7\*k) MOD 10 +1）=3\*（(7\*1) MOD 10 +1）=24 H3=(H(KEY)+d3) MOD 11=(3+24)MOD11=5冲突，需要重新分配； d4=4\*（(7\*k) MOD 10 +1）=4\*（(7\*1) MOD 10 +1）=32 H4=(H(KEY)+d4) MOD 11=(3+32)MOD11=2冲突，需要重新分配； d5=5\*（(7\*k) MOD 10 +1）=5\*（(7\*1) MOD 10 +1）=40 H5=(H(KEY)+d5) MOD 11=(3+40) MOD 11=10，不冲突；  查找次数为6次。 |
| H(67) = 3；冲突，重新分配  H1(67) = 2；冲突，重新分配  H2(67) = 1；不冲突。  查找次数为3次。 |

所以平均查找次数ASL = (1+1+1+1+2+2+6+3)/8 = 17/8 = 2.125

**9.38**

//把二叉排序树S合并到T中

void BSTreeMerge(BiTree &T, BiTree& S)

{

if (S->lchild)

BSTreeMerge(T, S->lchild);

if (S->rchild)

BSTreeMerge(T, S->rchild);

InsertNode(T, S)

}

//把树结点S插入到T的合适位置上

void InsertNode(BiTree &T, BiTree &S)

{

if (S->data > T->data)

{

if (!T->rchild)

T->rchild = S;

else

InsertNode(T->rchild, S);

}

else if (S->data < T->data)

{

if (!T->lchild)

T->lchild = S;

else

InsertNode(T->lchild, S);

}

//将插入的新结点与原来的左右子树断绝关系

S->lchild = NULL;

S->rchild = NULL;

}

**9.40**

typedef struct {

int data;

int bf;

int lsize;

Node \*lchild, \*rchild;

}; \*Tree;

BTNode \*Locate\_Tree(Tree T, int k)

{

if (!T)

return NULL;

if (T->lsize == k)

return T;

else if (T->lsize > k)

return Locate\_Tree(T->lchild, k);

else

return Locate\_Tree(T->rchild, k - T->lsize);

//注意，如果所查找的结点是在当前结点的右子树，所查找的K值应改为(k - T->lsize)

}

**9.45**

typedef \*LNode[MAXSIZE] CHashTable;

bool BuildHash(CHashTable &T, int m)

{

if (m < 1)

return false;

//建立表头指针

T = malloc(m \* sizeof(WORD));

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

T[i] = NULL;

while ((key = Inputkey()) != NULL)

{

q = (LNode\*)malloc(sizeof(LNode));

q->data = key;

q->next = NULL;

n = H(key);

//如果当前链表表头没有存元素，则作为第一个结点存入

//如果已经有元素，则查到链表的尾部

if (!T[n])

T[n] = q;

else

{

for (p = T[n]; p->next; p = p->next);

p->next = q;

}

}

return true;

}