**《数据结构》**

**实验报告**

专 业 软件工程

年 级 2018级

姓 名 谢梓聪

学 号 201824100445

实 验 室 信工北414

使用日期 2019年12月19日

郑州大学信息工程学院

# 实验八 查找算法的实现

**一、实验目的及要求：**

熟练掌握顺序表和有序表的查找方法，掌握其时间复杂度的分析方法

**二、实验方法：**

1．使用windows平台。

2．使用DEV-C或Visual C软件。

**三、实验内容：**

（1）验证并设计顺序表的查找（顺序查找、折半查找）算法

（2）验证二叉排序树上的查找（创建、查找、插入）算法

（3）验证Hash表查找（Hash函数定义、建立，查找，插入）算法

**四、实验结果：**

#include<stdio.h>

#include<stdlib.h>

#define TRUE 1

#define FALSE 0

#define SUCCESS 1

#define UNSUCCESS 0

#define DUPLICATE -1

#define OK 1

#define ERROR 0

#define INFEASIBLE -1

#define OVERFLOW -2

#define HASH\_SIZE 100

#define LIST\_INIT\_SIZE 100

#define LISTINCREMENT 10

typedef int Status;

typedef int ElemType;

typedef struct{

ElemType \*elem; //顺序表的起始地址

int length;

int listSize;

}SqList;

typedef struct{

ElemType \*elem; //数据元素基址

int count; //当前数据元素个数

int SizeIndex;

}HashTable;

typedef int TElemType;

typedef struct BiTNode{ //二叉树数据表示

TElemType data;

struct BiTNode \*lChild,\*rChild; //左右孩子指针

}BiTNode,\*BiTree;

Status CreateBiTree(BiTree &T);

Status InOrderTraverse(BiTree T,Status(\* visit)(TElemType elem));

Status TElemOutput(TElemType elem);

Status InitList(SqList \* L);

Status ListInsert(SqList \* L,ElemType add);

Status ListDelete(SqList \* L,int num,ElemType \*delPt);

Status ListCreate(SqList \* L);

Status GetList(SqList \* L);

Status compare(ElemType elem1,ElemType elem2);

int SqSearch(SqList \*L,int key);

int BinarySearch(SqList \*L,int key);

//二叉排序树的查找和逐个插入

Status SearchBST(BiTree T,int key,BiTree f,BiTree &p);

Status InsertBST(BiTree &T,ElemType elem);

Status CreateBST(BiTree &T);

//hash表的创建、插入和查找以及开放地址法线性探测再散列去解决冲突

Status InitHash(HashTable &H);

int Hash(int key);

void collision(int &pos,int &cnt);

Status SearchHash(HashTable H,ElemType key,int &pos,int &cnt);

Status CreateHash(HashTable &H);

Status InsertHash(HashTable &H,ElemType elem);

/\*

二叉排序树样例输入：

45 24 53 45 12 24 90 q

\*/

/\*

哈希表样例输入：

105 106 107 205 q

5 6 7 8

\*/

int main(void)

{

int key,num,pos=0,cnt=0;

BiTree T=NULL,p=NULL;

HashTable H;

/\* 顺序表的顺序查找以及二分查找

SqList L;

InitList(&L);

ListCreate(&L);

getchar();

printf("请输入要查找的数字，输入q离开输入：");

while(scanf("%d",&key))

{

pos=SqSearch(&L,key);

cnt=BinarySearch(&L,key);

if(pos==-1)

printf("为查找到该数字!!\n");

else

printf("顺序查找结果%d，二分查找结果%d\n",pos+1,cnt+1);

} \*/

/\* 二叉排序树的创建、插入、查找

CreateBST(T);

printf("二叉排序树中序遍历序列为：\n");

InOrderTraverse(T,TElemOutput);

putchar('\n');

printf("请输入要插入的关键字，按q键离开输入：\n");

while(scanf("%d",&num))

InsertBST(T,num);

getchar();

printf("插入后二叉排序树中序遍历序列为：\n");

InOrderTraverse(T,TElemOutput);

putchar('\n');

printf("请输入要查找的关键字，按q键离开输入：");

while(scanf("%d",&key))

if(SearchBST(T,key,NULL,p))

printf("当前序列中存在该关键字\n");

else

printf("当前序列中不存在该关键字\n");

\*/

//哈希表的创建、插入以及查找

InitHash(H);

CreateHash(H);

printf("请输入要查找的关键字，按q键离开输入：");

while(scanf("%d",&key))

if(SearchHash(H,key,pos,cnt))

printf("当前序列中存在该关键字且位于地址%d\n",pos);

else

printf("当前序列中不存在该关键字\n");

return 0;

}

Status InitHash(HashTable &H)

{

int i;

H.elem=(ElemType\*)malloc(HASH\_SIZE\*sizeof(ElemType));

if(!H.elem) return ERROR;

H.count=0;

H.SizeIndex=HASH\_SIZE;

for(i=0;i<H.SizeIndex;i++)

{

H.elem[i]=0;

}

return OK;

}

Status SearchHash(HashTable H,ElemType key,int &pos,int &cnt)

{

pos=Hash(key);

while(H.elem[pos]!=0&&H.elem[pos]!=key)

collision(pos,++cnt);

if(key==H.elem[pos])

return SUCCESS;

else

return UNSUCCESS;

}

void collision(int &pos,int &cnt)

{

pos++;

}

int Hash(ElemType key)

{

return key%HASH\_SIZE;

}

Status InsertHash(HashTable &H,ElemType elem)

{

int pos=0,cnt=0;

if(SearchHash(H,elem,pos,cnt))

return DUPLICATE;

else

{

H.elem[pos]=elem;

H.count++;

return OK;

}

}

Status CreateHash(HashTable &H)

{

int num;

printf("请依次输入查找序列，q键离开输入来完成Hash表的创建：");

while(scanf("%d",&num))

InsertHash(H,num);

getchar();

return OK;

}

Status CreateBST(BiTree &T)

{

int num;

printf("请依次输入查找序列，q键离开输入来完成二叉排序树的创建：");

while(scanf("%d",&num))

InsertBST(T,num);

getchar();

return OK;

}

Status SearchBST(BiTree T,int key,BiTree f,BiTree &p)

{ //用T代表当前子树的指针，f代表当前子树指针的双亲指针，

//若查找成功则用p返回其结点的地址，否则用p返回查找路径上访问的最后一个结点，f初始值为空

if(!T) //该子树为空则必查找不成功

{

p=f;

return FALSE;

}

else if(key==T->data) //查找成功

{

p=T;

return TRUE;

}

else if(key<T->data) //小于，则继续访问左子树

return SearchBST(T->lChild,key,T,p);

else //大于，则继续访问右子树

return SearchBST(T->rChild,key,T,p);

}

Status InsertBST(BiTree &T,ElemType elem)

{

BiTNode \*p,\*f=NULL,\*ins=NULL;

if(!SearchBST(T,elem,f,p))

{

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

ins->data=elem;

ins->lChild=ins->rChild=NULL;

if(!p) //此时该子树为空

{

T=ins; //直接插入

}

else if(p->data>ins->data) //小于则在左子树上插

p->lChild=ins;

else

p->rChild=ins;

return TRUE;

}

else return FALSE;

}

int SqSearch(SqList \*L,int key)

{

int i;

for(i=0;i<L->length;i++)

{

if(L->elem[i]==key)

return i;

}

return -1;

}

int BinarySearch(SqList \*L,int key)

{

int left=0,right=L->length,mid;

while(left<=right)

{

mid=(left+right)/2;

if(L->elem[mid]>key)

right=mid-1;

else if(L->elem[mid]<key)

left=mid+1;

else

return mid;

}

return -1;

}

Status InitList(SqList \* L) //初始化顺序表，为顺序表分配内存空间及生成基地址

{

L->elem=(ElemType\*)malloc(LIST\_INIT\_SIZE\*sizeof(ElemType));

if(!L->elem)

return ERROR;

L->length=0;

L->listSize=LIST\_INIT\_SIZE;

return OK;

}

Status ListInsert(SqList \* L,ElemType add)

{

ElemType \*newBase;

if(L->length>L->listSize)

{

newBase=(ElemType\*)realloc(L->elem,(L->listSize+LISTINCREMENT)\*sizeof(ElemType));

if(!newBase) return OVERFLOW;

L->elem=newBase;

L->listSize+=LISTINCREMENT;

}

L->elem[L->length]=add;

L->length++;

return OK;

}

Status ListDelete(SqList \* L,int num,ElemType \*delPt)

{

int i;

if(L->length==0||num<1||num>L->length)

return ERROR;

\*delPt=L->elem[num-1];

for(i=num-1;i<L->length;i++)

{

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

}

L->length--;

return OK;

}

Status ListCreate(SqList \* L) //根据插入函数循环生成一个有序顺序表

{

ElemType elem;

printf("创建顺序表中,q离开输入：\n");

while(scanf("%d",&elem))

{

ListInsert(L,elem);

}

return OK;

}

Status compare(ElemType elem1,ElemType elem2)

{

return elem1>elem2;

}

Status GetList(SqList \* L)

{

int i;

if(L->length==0) return ERROR;

for(i=0;i<L->length;i++)

{

printf("%d\t",L->elem[i]);

}

printf("\n");

return OK;

}

Status TElemOutput(TElemType elem)

{

if(!printf("%d\t",elem)) return ERROR;

return OK;

}

Status CreateBiTree(BiTree &T)

{

char ch;

scanf("%c",&ch);

if(ch==' ')

{

T=NULL;

return 0;

}

else

{

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

T->data=ch;

CreateBiTree(T->lChild);

CreateBiTree(T->rChild);

}

return OK;

}

Status InOrderTraverse(BiTree T,Status(\* visit)(TElemType elem)) //二叉树中序递归遍历

{

if(!T) return 0;

else

{

InOrderTraverse(T->lChild,visit);

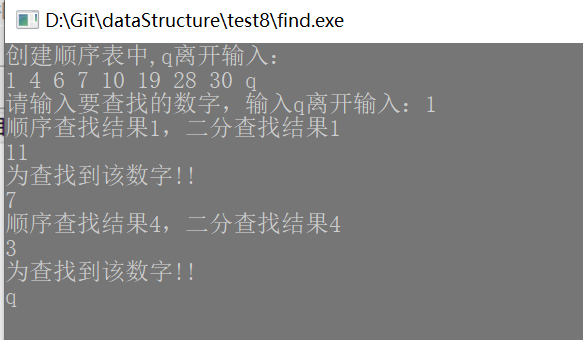
visit(T->data);

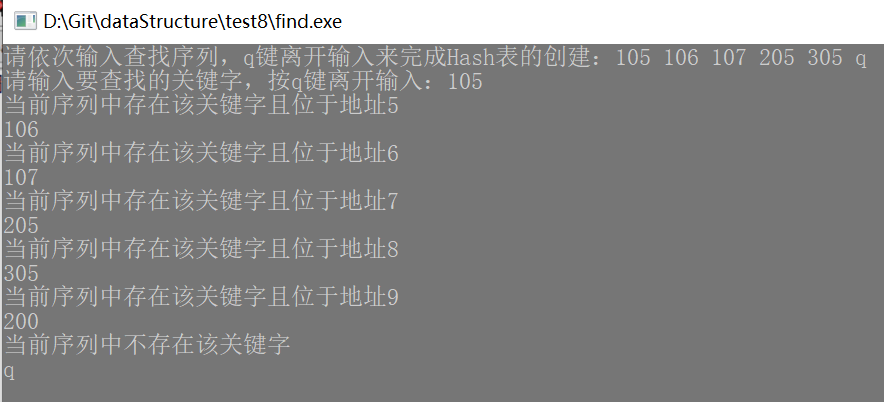
InOrderTraverse(T->rChild,visit);

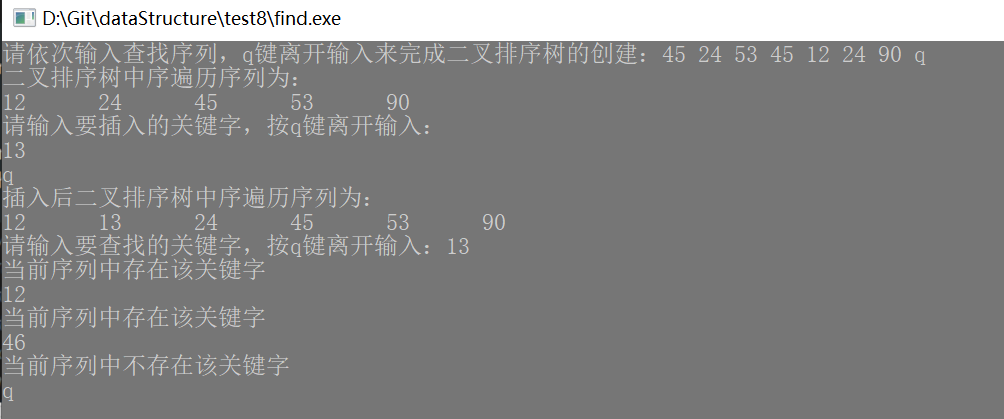
return OK;

}

}







**五、实验体会：**

对线性表的顺序查找即为遍历无序的线性序列比较相等输出，二分查找要求线性序列有序，同时利用有序的特性不断缩小查找范围从而提高查找效率，二叉排序树是一种动态查找表，即在查找的过程中能动态地插入和删除，根据结点的数据大小逐步构造二叉排序树，构造出来的二叉排序树中序遍历序列即为有序序列，然后利用二叉排序树特性大大提高了查找效率，平均查找长度为log2n。Hash表即为利用Hash函数直接把关键字映射为地址，从而消耗空间效率提高查找效率，对于映射冲突的情况该程序中使用的是开放地址法线性探测再散列去解决地址冲突。