实验五　查找和排序

1、实验目的：

掌握排序和查找的概念；掌握邻接矩阵和邻接链表建立图的存储结构；排序和查找的基本算法

2、实验要求：

顺序结构学排序和查找的算法实现。

3、实验主要步骤：

每个人独立完成排序和查找的相关算法（不上交）。

试编写程序：从键盘随机输入（或随机产生）30 个整数，用顺序查找算法查找给定的某个数，对 30 个排序后进行折半查找，建立这 30 个数的二叉排序树，并进行查找。

4、注意事项：

实验报告要求：

（1）算法的完整代码；

（2）程序运行结果及分析；

（3）实验总结。

5、源程序：

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#define maxsize 100
#define NUM 30
int count_Binstree=0;
int count_Sqlist=0;
int count_maopao=0;
int count_zheban=0;
//二叉排序树定义类型
typedef int KeyType;
typedef struct
{
    KeyType key;
}DataType;
typedef struct BinSTreeNode
{
    DataType  elem;   //elem含有关键字域
    struct BinSTreeNode  *lchild;
    struct BinSTreeNode  *rchild;
}*BinSTree;
//顺序表定义类型
typedef struct
{
    KeyType key;
}SDataType;
typedef struct
{
    SDataType r[maxsize];  //数据元素存储空间
    int length;            //表的长度
}Sqlist;
Sqlist init_strlist(int num[],int len)
{//创建查找表
    Sqlist *s;
    s=(Sqlist *)malloc(sizeof(Sqlist));
    s->length=len;
    for(int i=0;i<s->length;i++)
        s->r[i].key=num[i];
    return *s;
}
//顺序查找
int SeqSearch(Sqlist s,KeyType k)
{
    int i;
    for(i=0;i<s.length;i++)
    {
        count_Sqlist++;
        if(s.r[i].key==k)
            return i;   //查找成功
    }
    return -1;   //查找失败
}
```

```
//冒泡排序
void maopao(Sqlist *s)
{
    int i,j,t;
    for(i=0;i<s->length-1;i++)
    {
        for(j=0;j<s->length-1-i;j++)
        {
            if(s->r[j].key>s->r[j+1].key)
            {
                t=s->r[j+1].key;
                s->r[j+1].key=s->r[j].key;
                s->r[j].key=t;
                count_maopao++;
            }
        }
    }
    printf("\n冒泡排序结果为：");
    for(i=0;i<s->length;i++)
        printf("%d ",s->r[i].key);
    printf("\n");
}
//折半查找
int BinSearch(Sqlist s , KeyType k)
{
    low = 0;
    high = s.length-1;
    while(low<=high)
    {
        count_zheban++;
        mid=(low+high)/2;       //取区间中点
        if (s.r[mid].key==k)
            return mid;         //查找成功
        else if (s.r[mid].key>k)
            high=mid-1;         //在左区间中查找
        else
            low=mid+1;          //在右区间中查找
     }
    return -10;    //查找失败
}
//二叉排序树查找算法
BinSTree BSTreeSearch(BinSTree t ,KeyType k)
{
    if (t==NULL)
    {
        printf("\n二叉排序树 %d 未查找到!",k);
        return NULL;
    }
    if (t->elem.key==k)
    {
        printf("\n二叉排序树 %d 查找成功!",k);
        return (t); //查找成功
    }
    if (t->elem.key>k)
    {
        count_Binstree++;
        BSTreeSearch(t->lchild ,k); //在左子树中查找
    }
    else
    {
        count_Binstree++;
        BSTreeSearch(t->rchild ,k); //在右子树中查找
    }
    return NULL;
}
//二叉排序树插入算法
BinSTree BSTreeInsert(BinSTree *t , KeyType k)
{
    BinSTree r;
    if (*t==NULL)
    {
        r=(BinSTree)malloc(sizeof(struct BinSTreeNode));
        r->elem.key=k;
        r->lchild=r->rchild=NULL;
        *t=r;      //若二叉排序树为空，被插结点作为树的根结点
        return *t;
    }
    else if(k<((*t)->elem.key))
        BSTreeInsert(&((*t)->lchild),k);  //插入到p的左子树中
    else
        BSTreeInsert(&((*t)->rchild),k);  //插入到p的右子树中
    return NULL;
```

```
}
//二叉排序树的结点删除算法
int BSTreeDelete (BinSTree *bt , KeyType k)
{
    BinSTree f,p,q,s;
    p=*bt;
    f=NULL;
    while (p&& p ->elem.key!=k)
    {
        f=p;  //f为指向结点*p的双亲结点的指针
        if(p->elem.key>k)
            p=p->lchild;   //搜索左子树
        else
            p=p->rchild;   //搜索右子树
    }
    if (p==NULL)
        return(0);  //找不到待删的结点时返回
    if (p->lchild==NULL) //待删结点的左子树为空
    {
        if(f==NULL)         //待删结点为根结点
            *bt=p->rchild;
        else if(f->lchild==p)  //待删结点是其父结点的左孩子
            f->lchild=p->rchild ;
        else     //待删结点是其父结点的右孩子
            f->rchild=p->rchild;
        free(p);
    }
    else     //待删结点有左子树
    {
        q=p;
        s=p->lchild;
        while(s->rchild)//在待删结点的左子树中查找最右下结点
        {
            q=s;
            s=s->rchild;
        }
        if (q==p)//将最右下结点的左子树链到待删结点上
            q->lchild=s->lchild;
        else
            q->rchild=s->lchild;
        p->elem.key=s->elem.key;
        free (s) ;
        return(1);
    }
    return(0);
}
void inorder1(BinSTree t)
{//中序遍历的递归算法
    if(t)
    {
        inorder1(t->lchild);
        printf("%d ",t->elem.key);
        inorder1(t->rchild);
    }
}
int main()
{
    BinSTree t=NULL,r=NULL;
    int i,a[NUM],b[NUM];
    srand((unsigned)time(NULL));
    //产生30个随机数
    printf("随机产生 %d 个10～99的随机数：
",NUM);
    for(i=0;i<NUM;i++)
    {
        a[i]=rand()%90+10;   //产生随机数
        b[i]=a[i];           //a数组用于冒泡排序，b数组用于建立二叉排序树
        printf("%d ",a[i]);
    }
    //要搜索的数组下标
    int randnum=rand()%NUM+1;  //从 1 到（9+1-1）的随机数，随机产生要搜索的数组下标
    printf("\n本次要查找的数据是第 %d 位：%d\n",randnum+1,a[randnum]);

    //建立顺序表
    Sqlist str=init_strlist(b, NUM);

    //顺序查找
    SeqSearch(str, a[randnum]);
    printf("\n   顺序查找 %d 次",count_Sqlist);

    //冒泡排序
    maopao(&str);
```

```
    printf("    冒泡排序交换 %d 次
\n",count_maopao);

    //折半查找
    int m;
    m=BinSearch(str, a[randnum]);
    printf("折半查找数据 %d 所在排序后的第 %d
位\n" ,a[randnum],m+1);
    printf("    折半查找 %d 次\n",count_zheban);

    //二叉树建立，中序遍历（顺序输出）
    for(i=0;i<NUM;i++)
    {
        if(i==0)  //只运行一次，记录根结点r
            r=BSTreeInsert(&t, a[i]);
        else
            t=BSTreeInsert(&r, a[i]);  //t为插
入时所在结点，没用处
    }
    printf("二叉排序树排序（中序）结果为：");
    inorder1(r);

    //查找结点
    BSTreeSearch(r, a[randnum]);
    printf("    二叉排序树查找 %d 次
\n",count_Binstree);

    //删除结点
    BSTreeDelete(&r, a[randnum]);
    printf("二叉排序树删除 %d 结果为：
",a[randnum]);
    inorder1(r);

    //再次查找结点
    BSTreeSearch(r, a[randnum]);
    printf("    二叉排序树查找 %d 次
\n",count_Binstree);
}
```

运行结果：

```
随机产生 30 个10~99的随机数: 30 29 77 27 83 87 55 14 95 68 93 18 88 11 39 16 80 65 89 14 99 73 50 80 32 44 78 70 63 65
本次要查找的数据是第 16 位: 16

   顺序查找 16 次
冒泡排序结果为: 11 14 14 16 18 27 29 30 32 39 44 50 55 63 65 65 68 70 73 77 78 80 80 83 87 88 89 93 95 99
   冒泡排序交换 207 次
折半查找数据 16 所在排序后的第 4 位
   折半查找 5 次
二叉排序树排序（中序）结果为: 11 14 14 16 18 27 29 30 32 39 44 50 55 63 65 65 68 70 73 77 78 80 80 83 87 88 89 93 95 99
二叉排序树 16 查找成功!    二叉排序树查找 5 次
二叉排序树删除 16 结果为: 11 14 14 18 27 29 30 32 39 44 50 55 63 65 65 68 70 73 77 78 80 80 83 87 88 89 93 95 99
二叉排序树 16 未查找到!    二叉排序树查找 11 次
Program ended with exit code: 0
```