作业 HW5 实验报告

姓名：何正潇 学号：1950095 日期：2022 年 1 月 7 日

1. 涉及数据结构和相关背景

不涉及特殊的数据结构，但是与数据结构中的搜索和排序息息相关。

2. 实验内容

5.1 折半查找

5.1.1 问题描述

描述

二分法将所有元素所在区间分成两个子区间，根据计算要求决定下一步计算是在左区间还是右区间进行；重复该过程，直到找到解为止。二分法的计算效率是O(logn),在很多算法中都采用了二分法，例如：折半查找，快速排序，归并排序等。
折半查找要求查找表是有序排列的，本题给定已排序的一组整数，包含重复元素，请改写折半查找算法，找出关键字key在有序表中出现的第一个位置或最后一个位置，保证时间代价是O(logn)。若查找不到，返回-1。

5.1.2 基本要求

输入

第1行输入一个正整数n,表示查找表的长度；
第2行输入n个有序排列的整数，以空格分割
后面若干行输入为查找的方式ope待查找的整数x
若ope为lower，则表示查找元素在数列中首次出现的位置
若ope为upper，则表示查找元素在数列中最后一次出现的位置
若ope为done，则表示查找结束

输出

查找元素在有序表中的位置，有序表从0开始存储。若查找不到，返回-1。

5.1.3 数据结构设计

本题无特殊的数据结构使用

5.1.4 功能说明（函数、类）

```
#define _CRT_SECURE_NO_WARNINGS
#include<string.h>
#include<map>
#include<string>
#include<stack>
#include<iostream>
#include<iomanip>
using namespace std;
```

```
void binary_search(int number, int sequence[],int temp,int signal)
{
    /*首次出现数字的位置*/
    if (signal == 0)
    {
        int low = 0, high = number - 1;
        while (low <= high)
        {
            int mid = (low + high) / 2;
            if (temp == sequence[mid])
            {
                while (mid-1>=0&&sequence[mid - 1] == temp)
                {
                    mid = mid - 1;
                }
                cout << mid << endl;
                break;
            }
            if (temp < sequence[mid])
                high = mid - 1;
            else
                low = mid + 1;


        }
        if (low > high)
            cout << -1 << endl;
    }
    /*最后一次出现数字的位置*/
    else
    {
        int low = 0, high = number - 1;
        while (low <= high)
        {
            int mid = (low + high) / 2;
            if (temp == sequence[mid])
            {
                while (sequence[mid + 1] == temp&&mid+1<=number-1)
                {
                    mid = mid + 1;
                }
                cout << mid << endl;
                break;
```

```
            }
            if (temp < sequence[mid])
                high = mid - 1;
            else
                low = mid + 1;

        }
        if (low > high)
            cout << -1 << endl;
    }
}
int main()
{
    int number;
    int* sequence = NULL;
    cin >> number;
    sequence = new int[number];
    for (int i = 0; i < number; i++)
    {
        cin >> sequence[i];
    }
    while (1)
    {
        char str[100];
        cin >> str;
        if (strcmp(str, "lower") == 0)
        {
            int temp;
            cin >> temp;
            binary_search(number, sequence, temp,0);
        }
        else if (strcmp(str, "upper") == 0)
        {
            int temp;
            cin >> temp;
            binary_search(number, sequence, temp, 1);
        }
        else
            break;
    }
    return 0;
}
```

5.1.5 调试分析（遇到的问题和解决方法）

没有碰到太多问题，比较简单的就通过了。

5.1.6 总结和体会

这道题让我更加熟悉了折半查找的思想和算法，以及应用，对未来进一步学习搜索和排序的其他方法有启发的作用。

5.2 题目二

5.2.1 问题描述

二叉排序树

描述

二叉排序树BST（二叉查找树）是一种动态查找表，或者是一棵空树，或者是具有下列性质的二叉树：
（1）每个结点都有一个作为查找依据的关键字(key)，所有关键字的键值互不相等。
（2）左子树(若非空)上所有结点的键值都小于它的根结点的键值。
（3）右子树(若非空)上所有结点的键值都大于它的根结点的键值。
（4）左子树和右子树也是二叉排序树。

二叉排序树的基本操作集包括：创建、查找，插入，删除，查找最大值，查找最小值等。
本题实现一个维护整数集合（允许有重复关键字）的BST，并具有以下功能：1. 插入一个整数 2.删除一个整数 3.查询某个整数有多少个 4.查询最小值 5. 查询某个数字的前驱

5.2.2 基本要求

输入

第1行一个整数n，表示操作的个数；
接下来n行，每行一个操作，第一个数字op表示操作种类：

- 若op=1，后面跟着一个整数x，表示插入数字x
- 若op=2，后面跟着一个整数x，表示删除数字x（若存在则删除，否则输出None，若有多个则只删除一个），
- 若op=3，后面跟着一个整数x，输出数字x在集合中有多少个（若x不在集合中则输出0）
- 若op=4，输出集合中的最小值（保证集合非空）
- 若op=5，后面跟着一个整数x，输出x的前驱（若不存在前驱则输出None，x不一定在集合中）

输出

一个操作输出1行（除了插入操作没有输出）

5.2.3 数据结构设计

```
typedef struct BiTNode {
    ElemType data;
    struct BiTNode* lchild, * rchild;
    int count;
}BiTNode, * BiTree;
```

5.2.4 功能说明（函数、类）

```
/*1950095 大数据 何正潇*/
#include <iostream>
#include <stack>
using namespace std;
typedef struct {
    int key;
}ElemType;
typedef struct BiTNode {
    ElemType data;
    struct BiTNode* lchild, * rchild;
    int count;
}BiTNode, * BiTree;

//插入一个整数 ，很朴素的想法
void InsertBST(BiTree& T, BiTree S)
{
    BiTree p, q = NULL;
    if (!T)
        T = S;
    else {
        p = T;
        while (p) {
            q = p;
            if (S->data.key == p->data.key) {
                p->count++;
                return;
            }
            else if (S->data.key < p->data.key)
                p = p->lchild;
            else
                p = p->rchild;
        }
        if (S->data.key < q->data.key)
            q->lchild = S;
        else
            q->rchild = S;
    }
}

//实现对 p 结点删除，并重接它的左右字数保持二叉树仍有序
bool Delete(BiTree& p)
{
```

```
    BiTree q, s;

    //删除时如果存在重复元素直接 count--否则删除结点
    if (p->count > 1) {
        p->count--;
        return true;
    }
    if (!p->rchild) //右子树为空则只需重接它的左子树
    {
        q = p;
        p = p->lchild;
        delete q;
    }
    else if (!p->lchild) //只需重接它的右子树
    {
        q = p;
        p = p->rchild;
        delete q;
    }
    else //左右子树都不为空
    {
        q = p;  //s 的前驱结点
        s = p->lchild;
        while (s->rchild) {
            q = s;
            s = s->rchild;
        } //转左，然后向右走到尽头
        p->data = s->data;
        p->count = s->count;
        if (q != p)
            q->rchild = s->lchild;
        else  //只是向左，并没有向右，因此 s 是 q 的左结点
            q->lchild = s->lchild;
        delete s;
    }
    return true;
}

bool DeleteBST(BiTree& T, int key)
{
    if (!T)   //不存在关键字等于 key 的数据元素
        return false;
    else {
        if (key == T->data.key)   //找到关键字等于 key 的数据元素
```

```
            return Delete(T);
        else if (key > T->data.key)
            return DeleteBST(T->rchild, key);
        else
            return DeleteBST(T->lchild, key);
    }
}
//查询最小值
int MinBST(BiTree T)
{
    BiTree p = T;
    while (p->lchild)
        p = p->lchild;
    return p->data.key;
}
//查询某个整数有多少个
int CountBST(BiTree T, int key)
{
    BiTree p = T;
    while (p) {
        if (key == p->data.key)
            return p->count;
        if (key < p->data.key)
            p = p->lchild;
        else
            p = p->rchild;
    }
    return 0;
}
//查询某个数字的前驱
void FindPreBST(BiTree T, int key, BiTree& pre)
{
    BiTree p = T;
    stack<BiTree> S;
    while (p || !S.empty()) {
        if (p) {
            S.push(p);
            p = p->lchild;
        }
        else {
            p = S.top();
            S.pop();
            if (p->data.key >= key)
                return;
```

```
            if (p->data.key < key)
                pre = p;
            p = p->rchild;
        }
    }
}

int main()
{
    BiTree T = NULL, p;
    int op, n, key;
    cin >> n;
    while (n--) {
        cin >> op;
        if (op == 1) {
            cin >> key;
            p = new BiTNode;
            p->data.key = key;
            p->lchild = NULL;
            p->rchild = NULL;
            p->count = 1;
            InsertBST(T, p);
        }
        else if (op == 2) {
            cin >> key;
            if (!DeleteBST(T, key))
                cout << "None" << endl;
        }
        else if (op == 3) {
            cin >> key;
            cout << CountBST(T, key) << endl;
        }
        else if (op == 4)
            cout << MinBST(T) << endl;
        else if (op == 5) {
            cin >> key;
            BiTree s = NULL;
            FindPreBST(T, key, s);
            if (!s)
                cout << "None" << endl;
            else
                cout << s->data.key << endl;
        }
    }
```

```
}
```

5.2.5 调试分析

这道题调试碰到的主要问题主要还是在于寻找前驱，有一些难度还有某些部分的递归逻辑没有理清楚。

5.2.6 总结和体会

这道题让我熟悉了线索二叉树的基本性质和算法运用，是比较有用的，同时为下面的 b 树以及红黑树打下了基础。

5.4 题目三

5.4.1 题目 哈希表

5.4.2

题目描述

描述

哈希表（hash table，散列表）是一种用于以常数平均时间执行插入、删除和查找的查找表，其基本思想是：找到一个从关键字到查找表的地址的映射*h*（称为散列函数），将关键字*key*的元素存到*h(key)*所指示的存储单元中。当两个不相等的关键字被散列到同一个值时称为冲突，产生冲突的两个（或多个）关键字称为同义词，冲突处理的方法主要有：开放定址法，再哈希法，链地址法。

本题针对字符串设计哈希函数。假定有一个班级的人名名单，用汉语拼音（英文字母）表示。

要求：

1. 首先把人名转换成整数，采用函数h(key)=((...(key[0] * 37+key[1]) * 37+...)*37+key[n-2])* 37+key[n-1]，其中key[i]表示人名从左往右的第i个字母的ascii码值(i从0计数,字符串长度为n，1<=n<=100)。
2. 采取除留余数法将整数映射到长度为P的散列表中，h(key)=h(key)%M，若P不是素数，则M是大于P的最小素数，并将表长P设置成M。
3. 采用平方探测法（二次探测再散列）解决冲突。（有可能找不到插入位置，当探测次数>表长时停止探测）

5.4.3 数据结构设计

没有特殊的数据结构使用

5.4.4 功能说明（函数、类）

```
#define _CRT_SECURE_NO_WARNINGS
#include<string.h>
#include<map>
#include<string>
#include<stack>
#include<iostream>
#include<iomanip>
#include<math.h>
```

```
using namespace std;
/*判断是不是质数*/
bool judge(int p)
{
    if (p == 1)
        return 1;
    if (p == 2)
        return 0;
    for (int i = 2; i <= sqrt(p); i++)
    {
        if (p % i == 0)
            return 1;
        else
            ;
    }
    return 0;

}
/*哈希散列程序*/
int hash_map(char *name,int p)
{
    long long sum = 0;
    for (int i = 0; i < (int)strlen(name)-1; i++)
    {
        sum += name[i];
        sum %= p;
        sum *= 37;
        sum %= p;
    }
    sum += name[strlen(name) - 1];
    sum %= p;
    return sum;
}
int main()
{
    int n,result;
    long p;
    char* temp;
    temp = new char[100000];
    cin >> n >> p;
    while (judge(p))
    {
        p++;
    }
```

```
    int* list = new(nothrow)int[p];
    memset(list, 0, p * sizeof(int));
    for (int i = 0; i < n; i++)
    {
        cin >> temp;
        result = hash_map(temp,p);
        if (list[result] == 0)
        {
            list[result] = 1;
            cout << result << " ";
        }
        else
        {
            bool signal = 0;
            int temp = result;
            for (int i = 1; i <= p/2; i++)
            {
                result = (temp + i * i) % p;
                if (list[result] == 0)
                {
                    list[result] = 1;
                    signal = 1;
                    cout << result << " ";
                    break;
                }
                result = (temp - i * i) % p;
                result = (result + p) % p;
                if (list[result] == 0)
                {
                    list[result] = 1;
                    signal = 1;
                    cout << result << " ";
                    break;
                }
            }
            if (signal == 0)
                cout << '-' << " ";
        }
    }
    return 0;
}
```

5.4.5 调试分析

最大的问题是我一开始把二次探测再散列当成了普通的二次探测，导致结果连续出错，后面进行算法

修改就完全正确。

5.4.6 总结和体会

这道题总体来说让我们明白了哈希表的基本工作原理。对后面的进一步学习提供帮助和启发。

5.5 题目求逆序对

5.5.1 题目描述

描述

对于一个长度为N的整数序列A，满足i < j 且 Ai > Aj的数对(i,j)称为整数序列A的一个逆序。

请求出整数序列A的所有逆序对个数

5.5.2 题目要求

输入

输入包含多组测试数据，每组测试数据有两行
第一行为整数N(1 <= N <= 20000)，当输入0时结束
第二行为N个整数，表示长为N的整数序列

输出

每组数据对应一行，输出逆序对的个数

5.5.3 主要数据结构

无特殊数据结构

5.5.4 功能说明（函数、类）

归并排序算法分治

```
.       #include<iostream>
.       using namespace std;
```

```
.
.           int n,nums[20005],tmp[20005],cnt;
.
.           void mergesort(int l,int r)
.           {
.           if(l>=r) return;
.           int m=(l+r)/2;
.           int i=l,j=m+1,k=0;
.           while(i<=m&&j<=r){
.           if(nums[i]>nums[j]){
.           tmp[k++]=nums[j++];
.           cnt+=m-i+1;
.           }
.           else{
.           tmp[k++]=nums[i++];
.           }
.           }
.           while(i<=m) tmp[k++]=nums[i++];
.           while(j<=r) tmp[k++]=nums[j++];
.           for(i=0;i<k;i++) nums[i+l]=tmp[i];
.           }
.
.           void mergearray(int l,int r){
.           if(l>=r) return;
.           int m=(l+r)/2;
.           mergearray(l,m);
.           mergearray(m+1,r);
.           mergesort(l,r);
.           }
.
.           int main()
.           {
.           while(scanf("%d",&n)&&n){
.           cnt=0;
.           for(int i=0;i<n;i++){
.           scanf("%d",&nums[i]);
.           }
.           mergearray(0,n-1);
.           printf("%d\n",cnt);
.           }
.           return 0;
.           }
```

5.5.5 调试分析

主要考察的是归并排序分治的思想，一开始本来采用的暴力做法，结果 30 行的程序果然超时了。

5.5.6 实验体会

熟悉了比较有用的一种算法，同时对于分治的思想有了一些了解。

2. 实验总结

本单元的实验主要编写的是一些有关基础排序搜索的程序，相对来说覆盖面比较全面，同时也比较有意思。折半查找，二叉排序树，哈希表以及归并排序都是非常重要的数据结构组成内容。这也是数据结构的最后一个报告了，谢谢老师和助教一学期的答疑和帮助。