作业 HW5 实验报告

姓名：刘博洋 学号：2153538 日期：2022年12月25日

1. 涉及数据结构和相关背景

查找表是由同一类型的数据元素构成的集合，是一种非常灵活方便，在生活工程中都运用广泛的数据结构，查找表可以分为动态查找表和静态查找表，查找的数据元素中某个数据值成为关键字，用来标识某种数据元素或者记录。静态表查找主要分为顺序表查找、折半查找、静态树表查找等等，动态查找表则有排序二叉树、平衡二叉树，B树和hash表等等，都是十分常用的数据结构和算法，需要我们熟练掌握。

2. 实验内容

2.1 和有限的最长子序列

2.1.1 问题描述

给你一个长度为n的整数数组nums和一个长度为m的整数数组queries，返回一个长度为m的数组answer，其中answer[i]是nums中元素之和小于等于queries[i]的子序列的最大长度。

子序列是由一个数组删除某些元素（也可以不删除）但不改变元素顺序得到的一个数组。

2.1.2 基本要求

输入

第一行包括两个整数n和m，分别表示数组nums和queries的长度

第二行包括n个整数，为数组nums中元素

第三行包含m个整数，为数组queries中元素

对于20%的数据，有1<=n,m<=10

对于40%的数据，有1<=n,m<=100

对于100%的数据，有1<=n,m<=1000

对于所有数据，1<=nums[i],queries[i]<=106

下载编译并运行p126_data.cpp以生成随机测试数据

输出

输出一行，包括m个整数，为answer中元素

2.1.3 数据结构设计

```
int nums[1000] = { 0 };
int queries[1000] = { 0 };
int answer[1000] = { 0 };
```

由于是求子序列小于某个数的最大长度，我们不妨采用贪心的思想，将nums数组先从小到大进行排序，然后遍历queries数组，对每一个元素找到有序的nums中和不超过其值的子序列长度

2.1.4**功能说明（函数、类）**

```
int main()
{
    int n, m;
    cin >> n >> m;
    for (int i = 0;i < n;i++)
        cin >> nums[i];
    for (int j = 0;j < m;j++)
        cin >> queries[j];
    for (int i = 0;i < n;i++)
    {
        for (int j = 0;j < n - i - 1;j++)
        {
            if (nums[j] > nums[j+1])
            {
                int temp = nums[j];
                nums[j] = nums[j + 1];
                nums[j + 1] = temp;
            }//将nums排序
        }
    }
    for (int j = 0;j < m;j++)
    {
        int count = 0;
        int sum = 0;//遍历queries数组
        for (int i = 0;;i++)
        {
        if (sum <= queries[j])//找到最长子序列
        {
            sum += nums[i];
            count++;
            if (count > n)
            {
                answer[j] = n;
                break;
            }
        }
        else
        {
            answer[j] = count-1;
            break;
        }
    }
}
for (int i = 0;i < m;i++)
{
    cout << answer[i] << " ";
}
return 0;
```

}

2.1.5 调试分析（遇到的问题和解决方法）

本题较为简单，没有太多的调试与分析，主要还是看到题目后建立思路和确定采用贪心思想的过程比较重要。

2.1.6 总结和体会

本题不难，主要的易错点应该还是在于没有想到要采用贪心的方法先对nums数组进行排序，如果直接进行比较和计算的话还是比较复杂的。本题的主要收获是在做题之前需要提前想好相应的思路和步骤，有条不紊。

2.2 二叉排序树

2.2.1问题描述

二叉排序树BST（二叉查找树）是一种动态查找表，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（集合中比该数字小的最大值）。

2.2.2 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）

2.2.3 数据结构设计

```
typedef struct BSTNode {
    BSTNode *lchild, *rchild;
    int data;
    int coun;//coun表示此元素在集合中出现的次数，为0时需要删除
}BSTNode,*BSTree;
BSTree T;
```

构建二叉排序树，加入一个数据域coun记录此数字在集合中出现了多少次，然后根据要求构造各种函数，主要比较复杂的函数是删除的部分，删除的时候要分几种情况，首先找到删除的节点，第一如果这个节点是叶子节点，那么直接删除，如果有左孩子或者有右孩子，那么将其左右孩子接入父亲节点上，主要是当同时有左右孩子的时候，先在左子树找到一个最小的节点将其复制到此节点上，然后在找到这个节点将其删除。

对于找到最小值的操作，只需要根据二叉排序树的性质，找到最左边的节点即可

2.2.4 功能说明（函数类）

```
BSTree search(BSTree &T, int target,BSTree &front)
{
    if (!T)
        return NULL;
    BSTree p = T;
    while (p)
    {
        if (p->data == target)
            return p;
        else {
            if (p->data > target)
            {
                front = p;
                p = p->lchild;
            }

            else
            {
                front = p;
                p = p->rchild;
            }

        }
    }
    return NULL;
}
void insertBST(BSTree& T,int x)
{
    if (!T)
    {
        T = (BSTree)malloc(sizeof(BSTNode));
        T->data = x;
        T->coun = 1;
        T->lchild = NULL;
        T->rchild = NULL;
    }
    else
    {
        BSTree q=new BSTNode;
        BSTree p=search(T, x,q);
        if (p!= NULL)
        {
            p->coun++;
        }
        else
        {
            p = (BSTree)malloc(sizeof(BSTNode));
            p->data = x;
            p->lchild = NULL;
            p->rchild = NULL;
            p->coun = 1;
            BSTree s = T;
            int flag = 0;
            while (1)
            {
```

```
                if (p->data < s->data)
                {
                    if (s->lchild != NULL)
                        s = s->lchild;
                    else
                    {
                        flag = 1;
                        break;
                    }
                }

                else if (p->data > s->data)
                {
                    if (s->rchild != NULL)
                        s = s->rchild;
                    else
                    {
                        flag = 2;
                        break;
                    }
                }
            }

            if (flag == 1)
            {
                s->lchild = p;
            }
            else if (flag == 2)
            {
                s->rchild = p;
            }
        }

    }
}
void deletenode(BSTree& T, int x)
{
    BSTree q= (BSTree)malloc(sizeof(BSTNode));
    BSTree p = search(T, x,q);
    if (p == NULL)
    {
        cout << "None" << endl;
        return;
    }
    else {
        p->coun--;
        if (p->coun == 0)
        {

            if (p->lchild == NULL && p->rchild == NULL)
            {
                if (q->lchild == p)
                {
                    q->lchild = NULL;
                }
```

```
                else if (q->rchild == p)
                {
                    q->rchild = NULL;
                }
            }
            else if (p->lchild == NULL )
            {
                q = p;
                p = p->rchild;
                /*free(q);*/
            }
            else if (p->rchild == NULL)
            {
                q = p;
                p = p->lchild;
                /*free(q);*/
            }
            else {
                BSTree s=p->lchild;
                q = p;
                while (s->rchild)
                {
                    q = s;
                    s = s->rchild;
                }
                p->data = s->data;
                p->coun = s->coun;
                if (q != p)
                {
                    q->rchild = s->lchild;
                }
                else q->lchild = s->lchild;
                delete s;
            }
        }
    }
}
void getmin(BSTree& T)
{
    BSTree p = T;
    if (!T)
    {
        cout << -1 << endl;
        return;
    }
    while (p->lchild)
    {
        p = p->lchild;
    }
    cout << p->data << endl;
}
void getprior(BSTree &T,int x)
{
    BSTree q=new BSTNode;
    BSTree p = search(T, x, q);
```

```
    BSTree minx = T;
    if (T)
    {
        while (minx->lchild)
        {
            minx = minx->lchild;
        }
        if (p == minx)
        {
            cout << "None" << endl;
            return;
        }
    }
     if (p == NULL)
    {
        BSTree min = T;
        while (min->lchild)
        {
            min = min->lchild;
        }
        if (x < min->data)
            cout << "None" << endl;
        else
        {
            BSTree m = T;
            while (1)
            {
                if (m->data < x)
                {
                    if (m->rchild != NULL)
                        m = m->rchild;
                    else
                    {
                        cout << m->data << endl;
                        break;
                    }
                }

                else if (m->data > x)
                {
                    if (m->lchild != NULL)
                        m = m->lchild;
                    else
                    {
                        getprior(T, m->data);
                        break;
                    }
                }
            }
        }
        return;
    }
    else {
        if (p->lchild!= NULL)//如果有左子树
        {
```

```
            BSTree s = p->lchild;
            while (s->rchild)
            {
                s = s->rchild;
            }
            cout << s->data << endl;
        }
        else {
            if (p == q->rchild)
            {
                cout << q->data << endl;
            }
            else if (p == q->lchild)
            {
                while (1)
                {
                    BSTree front=new BSTNode;
                    BSTree ret=search(T, q->data, front);
                    if (front == T&&front->lchild==q||front==NULL)
                    {
                        cout << "None" << endl;
                        break;
                    }
                    if (front->rchild == q)
                    {
                        cout << front->data << endl;
                        break;
                    }
                    else
                        q = front;

                }
            }
        }
    }
}
void getcoun(BSTree& T, int x)
{
    BSTree q;
    BSTree p = search(T, x, q);
    if (p == NULL)
    {
        cout << 0 << endl;
    }
    else cout << p->coun<<endl;
}
```

2.2.5调试分析

本题调试的主要内容在于删除节点的操作上，特别是对于要删除的节点同时有左右孩子的时候，此时应当要么在其父节点的左子树找到最大的节点与之交换，要么在父节点的右子树找到最小的节点与之交换，交换后再删除找到的节点即可。还有就是在找前驱的问题上也经历了一些调试和修改，花费了一些时间。

2.2.6总结与收获

应该说这是一个比较基础的问题，但是中间几个函数的调试还是花费了不少时间，说明我的熟练度和准确性还需要提高，仅仅理解算法还不够，要自己手动实现的比较熟练才是硬道理。

2.3 哈希表

2.3.1问题描述

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）

2.3.2基本要求

输入

第1行输入2个整数N、P，分别为待插入关键字总数、散列表的长度。若P不是素数，则取大于P的最小素数作为表长。
第2行给出N个字符串，每一个字符串表示一个人名

输出

在1行内输出每个字符串插入到散列表中的位置，以空格分割，若探测后始终找不到插入位置，输出一个'-'。

2.3.3数据结构设计

```
int N, P,M;//N为关键字个数，P为表长
    long long int hkey=0;
    int* hash = (int*)malloc(sizeof(int) * 10000);
```

本题主要是根据题目描述的方法一步一步进行操作，首先在把名字转化为整数的时候要注意由于数据过大，很容易造成溢出，所以我们采用一边转化一边取模的方法求得最后的位置坐标，在处理冲突时，我们利用平方探测法进行处理，把散列表看成循环的，每次加1，4，9...当探测次数>表长时停止探测。最后输出每个字符串的存放位置即可。

2.3.4功能说明（函数、类）

```
bool is_su(int x)
{
    for (int i = 2;i <= x - 1;i++)
    {
        if (x % i == 0)
            return false;
    }
    return true;
}
void hashtable(char key[], long long int hkey, int M, int* hash)
{
    hkey = key[0];
    int length = 0;
```

```
    int location;
    while (1)
    {
        if (key[length] != '\0')
            length++;
        else {
            break;
        }
    }
    for (int i = 1;i<=length-1;i++)
    {
        hkey = ((((hkey%M) *(37%M))%M)%M + key[i]%M)%M;//先取模再转化
    }
    if (length == 1)
        location = hkey % M;
    else location = hkey;
    if (hash[location]==-1)
    {
        hash[location] = hkey;
        cout << location << " ";
    }
    else
    {
        int flag = 0;
        for (int i = 1;i<=M-1;i++)
        {
            location += i * i;
            if (location > M-1)//看成循环的
            {
                location = (location)% M;
            }
            if (hash[location] == -1)
            {
                hash[location] = hkey;
                cout << location << " ";
                flag = 1;
                break;
            }
        }
        if (flag == 0)
        {
            cout << "-" << " ";
        }
    }
}
```

2.3.5调试分析

本题比较重要的是边界条件，对于每一个过程要分析清楚起始条件和终止条件，对于哈希表和平方探测处理矛盾的方法要比较熟悉，我当时再调试处理循环的次数和先求模再转化这两个部分出现了一些问题，花费了不少时间调试。

2.3.6总结与收获

本题是对哈希表知识的总结与练习，基本上包含了哈希表的大部分内容，并且提示我们在不知道数据是否会越界溢出时，最好要采取一定的措施进行补救和修改，比如本题先取模再求和。

2.4 换座位

2.4.1问题描述

期末考试，监考老师粗心拿错了座位表，学生已经落座，现在需要按正确的座位表给学生重新排座。假设一次交换你可以选择两个学生并让他们交换位置，给你原来错误的座位表和正确的座位表，问给学生重新排座需要最少的交换次数。

2.4.2基本要求

输入描述

- 两个n*m的字符串数组，表示错误和正确的座位表old_chart和new_chart，old_chart[i][j]为原来坐在第i行第j列的学生名字

  对于100%的数据，1<=n,m<=200；

  人名为仅由小写英文字母组成的字符串，长度不大于5

  下载编译并运行p138_data.cpp生成随机测试数据

输出描述

- 一个整数，表示最少交换次数

2.4.3数据结构设计

```
int row=0,col=0;
map<string, string> map_fun;
map<string, string>::iterator iterator_fun;
```

可以利用map容器来进行操作，主要是对容器中不同的元素进行交换，最后看能否形成一个环，即检查交换后是否交换的双方都正确。

2.4.4功能说明（函数类）

```
class Solution {
public:
    int solve(std::vector<vector<std::string>> &old_chart,
std::vector<std::vector<std::string>> &new_chart) {
        row = old_chart.size();
        col = old_chart[0].size();
        for(int i = 0;i < row;i++)
            for (int j = 0; j < col; ++j) {
                if(new_chart[i][j] == old_chart[i][j]) continue;
                else map_fun[new_chart[i][j]] = old_chart[i][j];  //创建键值对
            }
        int ans = 0;
        while (1){
            int flag = 0;
            if(map_fun.size() == 1) break;  //如果此时只剩一个元素了，那么相当于已
经排完了。
            for(auto it:map_fun){  //遍历整个map，消除相同的键值对
                if(it.second == it.first)
map_fun.erase(map_fun.find(it.first));
            }
            for(auto it = map_fun.begin(); it != map_fun.end();it++){    //
遍历整个容器
```

```
                if(map_fun.empty()) break;    //如果容器为空，说明已经完成任务
                auto a = map_fun.find(it->second);    //find 函数，返回一个迭代器常量
                if(a == map_fun.end()){ cout << "0" << endl;return 0;}  //此种情况说明，新旧座位表有名字对不上，直接返回0
                if(a->first == it->first) {map_fun.erase(a);}  //说明此时新旧的表是对应的，将其擦除
                else{                                          //交换键值对，完成一次操作
                    string tmp = map_fun[it->first];
                    map_fun[it->first] = a->second;
                    a->second = tmp;
                    ans++;
                    if(ans > 30000 && ans %10 == 0){
                        flag = 1;
                    }
                    map_fun.erase(a); //此时对于a所对应的键值对相同，将其擦除；
                }
                if (flag == 1) break;
                if(map_fun.size() == 1) break;
            }
            if(map_fun.empty()) break;    //如果容器为空，说明已经完成任务
        }
        return ans;
    }
};
```

2.4.5调试与分析

本题调试了很多遍，主要是出现了TLE的问题，由于数据集较大时，map迭代器很容易造成越界访问，所以再一定规模操作之上，每完成10次操作，都要重新开始遍历，这样可以在增加由有限性能开销的同时，大幅降低越界访问的可能。

2.4.6总结与收获

虽然map容器并不是我们课程学习内容，但是由于之前的代码基础不好，这些好用的库函数也用的不多，本题让我比较熟悉的使用和了解了这些容器的方便之处，也是对自身综合能力的一个提升

3. 实验总结

本次实验是本学期数据结构课的最后一个实验，也是查找的有关内容，作业期间也是因为一些备考、身体的原因出现了一些困难，但是最后还是比较好的克服了，感谢助教和老师们对这门课程付出的心血和努力，我也对我自己这一学期产生的进步感到欣慰和高兴，新年伊始，希望2023年我能不断进步，继续加油!