# 北京邮电大学软件学院

# 2023-2024学年第1学期实验报告

**课程名称： 数据结构**

**实验名称： 实验六 查找**

**实验完成人：**

**姓名：**\_\_\_\_禄禄鱼\_\_\_\_**学号：**\_\_\_\_\_\_\_\_**成绩：**\_\_\_\_\_\_\_\_

**指导教师：**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**（封面 5%）**

**日 期： 2023 年 12 月 11 日**

1. **实验目的**

（说明通过本实验希望达到的目的5%）

通过实验学习建立二叉排序树的基本原理和操作，包括插入、查找、遍历等操作，加深对二叉排序树的理解。

熟悉哈希表的设计与实现，了解哈希函数的构造方法、处理冲突的方式，以及在哈希表中进行查找的操作。

提高对数据结构与算法的应用能力，加深对二叉排序树和哈希表在实际问题中的应用理解。

1. **实验内容**

（说明本实验的内容5%）

1. **二叉排序树**

**[问题描述]**

　　从键盘读入一组数据，建立二叉排序树并对其进行查找、遍历、格式化打印等有关操作。

**[基本要求]**

　　建立二叉排序树并对其进行查找，包括成功和不成功两种情况，并给出查找长度。

**[测试数据]**

由学生依据软件工程的测试技术自己确定。注意测试边界数据。

1. **哈希表设计**

**[问题描述]**

　　针对某个集体中人名设计一个哈希表，使得平均查找长度不超过R，并完成相应的建表和查表程序。

**[基本要求]**

　　假设人名为中国人姓名的汉语拼音形式。待填入哈希表的人名共有30个，取平均查找长度的上限为2。哈希函数用除留余数法构造，用线性探测再散列法或链地址法处理冲突。

**[测试数据]**

　　取你周围较熟悉的30个人名。

1. **实验环境**

（说明本实验需要的环境5%）

Clion Windows10

1. **实验过程和实验结果**

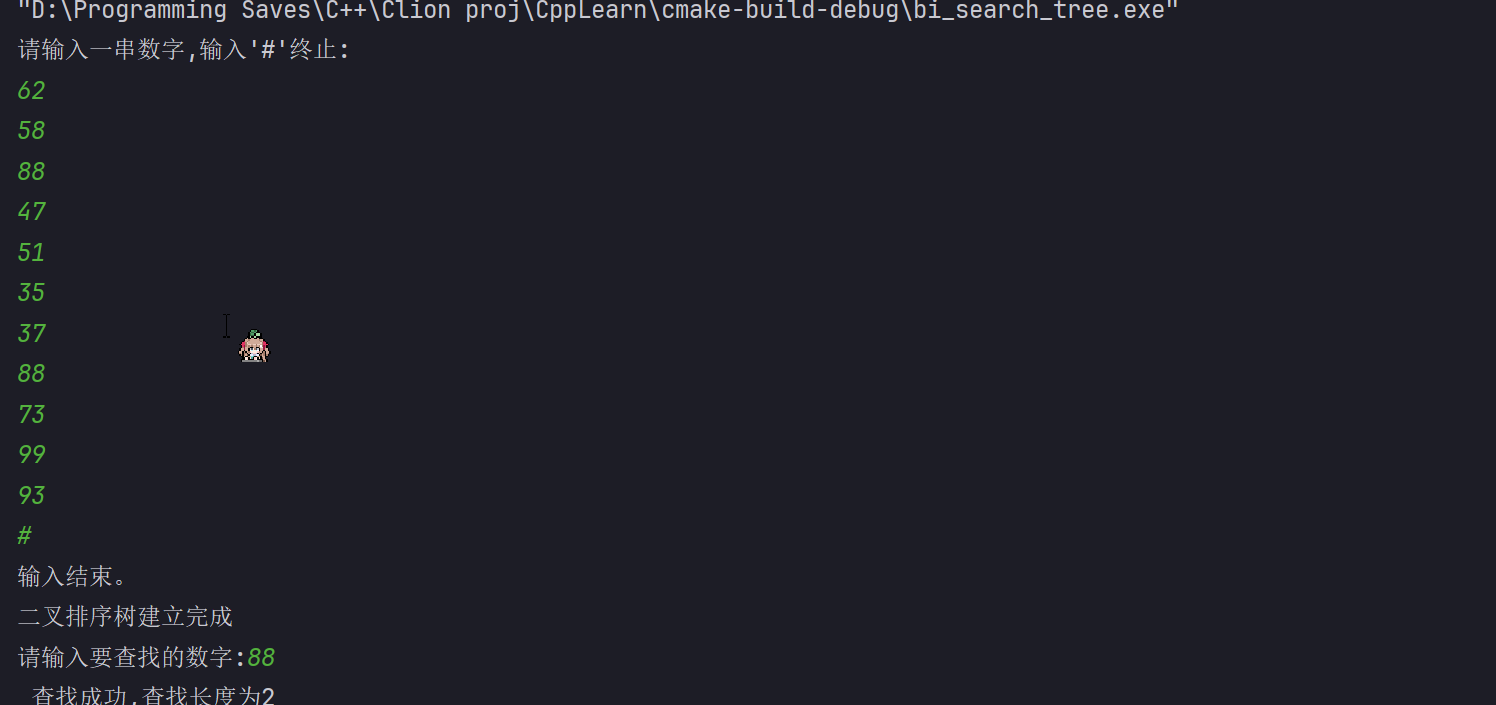
（问题分析，设计方案、算法、设计图、程序、调试过程截图、运行结果截图等70%）

1)

实验代码

#include <stdio.h>  
#include <stdlib.h>  
  
#define MAX\_NODES 30  
  
typedef struct BiTNode {  
 int data**;** struct BiTNode \*lchild**,** \*rchild**;**} BiTNode**,** \*BiTree**;**BiTree bi\_search\_tree\_build(int \*arr**,** int arr\_len)**;**void bi\_search\_from\_tree(BiTree pNode**,** int target)**;**int main() {  
 // 从键盘读取一串数字,每个数字之间用空格隔开,输入'#'终止  
 int num**,** index = 0**;** int nums[MAX\_NODES]**;** BiTree T = NULL**;** printf("请输入一串数字,输入'#'终止:\n")**;** while (1) {  
 if (index >= MAX\_NODES) {  
 printf("达到最大节点数限制，结束输入。\n")**;** break**;** }  
  
 if (scanf("%d"**,** &num) == 1) {  
 nums[index++] = num**;** } else {  
 if (getchar() == '#') {  
 printf("输入结束。\n")**;** break**;** }  
 printf("请输入有效数字。\n")**;** // 清除输入缓冲区  
 while (getchar() != '\n')**;** }  
 }  
  
 if (index <= 0) {  
 printf("未输入有效数字，无法建立二叉排序树。\n")**;** return 0**;** }  
 T = bi\_search\_tree\_build(nums**,** index)**;** //进行查找  
 int target**;** // 从键盘读取一个数字  
 printf("请输入要查找的数字: ")**;** scanf("%d"**,** &target)**;** bi\_search\_from\_tree(T**,** target)**;** return 0**;**}  
void bi\_search\_from\_tree(BiTree T**,** int target) {  
 int len = 0**;**//查找长度  
 if (T == NULL) {  
 printf("二叉排序树为空，查找失败。\n")**;** return**;** }  
 BiTree r = T**;**// r为当前进行比较的根节点  
 while (1) {  
 len ++**;** if (target < r->data) {  
 if (r->lchild == NULL) {  
 printf("查找失败,查找长度为%d\n"**,**++len)**;** break**;** } else {  
 r = r->lchild**;** }  
 } else if (target > r->data) {  
 if (r->rchild == NULL) {  
 printf("查找失败,查找长度为%d\n"**,**++len)**;** break**;** } else {  
 r = r->rchild**;** }  
 } else {  
 printf("查找成功,查找长度为%d\n"**,**len)**;** break**;** }  
 }  
}  
  
BiTree bi\_search\_tree\_build(int \*arr**,** int arr\_len) {  
 // 创建根节点  
 BiTree T = (BiTree)malloc(sizeof(BiTNode))**;** T->data = arr[0]**;** T->lchild = NULL**;** T->rchild = NULL**;** BiTree r = T**;** // r为当前进行比较的根节点  
  
 // 从第二个节点开始遍历  
 for (int i = 1**;** i < arr\_len**;** ++i) {  
 // 创建新节点  
 BiTNode \*p = (BiTNode \*)malloc(sizeof(BiTNode))**;** p->lchild = NULL**;** p->rchild = NULL**;** p->data = arr[i]**;** while (1) {  
 if (p->data < r->data) {  
 if (r->lchild == NULL) {  
 r->lchild = p**;** r = T**;** // 重置进行比较的根节点  
 break**;** } else {  
 r = r->lchild**;** }  
 } else if (p->data > r->data) {  
 if (r->rchild == NULL) {  
 r->rchild = p**;** r = T**;** // 重置进行比较的根节点  
 break**;** } else {  
 r = r->rchild**;** }  
 }  
 // 如果相等,则不插入  
 else {  
 break**;** }  
 }  
 }  
 printf("二叉排序树建立完成\n")**;** return T**;**}

实验结果



2)

实验代码

为什么选取53作为哈希表长度,哈希函数为何选择如此构造?

53是素数

因为中国人名字字数大多数是2-4个,如果有名字字数差别可以被区分开来(虽然大多数都是3个字,其实用处不大)

而且装填因子控制在0.5-0.7左右也是比较合适的

一般名字首字母组合起来重复的概率比较小,选作哈希函数还是比较合适的

//使用哈希表链地址法解决冲突  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#define MAX\_NAME\_LEN 20  
#define MAX\_NAME\_NUM 30  
//选择一个素数作为哈希表的长度  
#define HASH\_SIZE 53  
  
typedef struct HashNode  
{  
 char name[MAX\_NAME\_LEN]**;** struct HashNode \*next**;**} HashNode**,** \*pHashNode**;**typedef struct HashTable  
{  
 struct HashNode\* slots[HASH\_SIZE]**;**//指向第一个节点  
 short lens[HASH\_SIZE]**;**//记录链表长度  
} HashTable**,**\*pHashTable**;**int hash(char\* name) {  
 int sum = 0**;** while (\*name != '\0') {  
 if (\*name >= 'A' && \*name <= 'Z') {  
 sum += \*name**;** }  
 name++**;** }  
 return sum % HASH\_SIZE**;**}  
pHashTable hashTableBuild(char\*\* nameList**,**int nameNum)  
{  
 //构造哈希表  
 pHashTable pTable = (pHashTable)malloc(sizeof(HashTable))**;** //初始化哈希表  
 for (int i = 0**;** i < HASH\_SIZE**;** i++)  
 {  
 pTable->slots[i] = NULL**;** pTable->lens[i] = 0**;** }  
 //遍历名字列表,将名字插入哈希表  
 for (int i = 0**;** i < nameNum**;** i++)  
 {  
 //计算哈希值  
 int hashValue = hash(nameList[i])**;** //创建新节点  
 pHashNode pNewNode = (pHashNode)malloc(sizeof(HashNode))**;** strcpy(pNewNode->name**,** nameList[i])**;** pNewNode->next = NULL**;** //将新节点插入哈希表  
 if (pTable->slots[hashValue] == NULL)  
 {  
 pTable->slots[hashValue] = pNewNode**;** pTable->lens[hashValue] = 1**;** }  
 else  
 {  
 pHashNode pCurNode = pTable->slots[hashValue]**;** while (pCurNode->next != NULL)  
 {  
 pCurNode = pCurNode->next**;** }  
 pCurNode->next = pNewNode**;** pTable->lens[hashValue]++**;** }  
 }  
 return pTable**;**}  
int main()  
{  
  
 char\* nameList[MAX\_NAME\_NUM] = {  
 "ZhangXiaoHong"**,** "LiMing"**,** "WangLei"**,** "ChenWei"**,** "ZhaoYun"**,** "LiuXiao"**,** "SunXiaoyu"**,** "HuangQiang"**,** "XuJing"**,** "YangYang"**,** "HeYan"**,** "DengHui"**,** "LinFang"**,** "CaiLing"**,** "LuJun"**,** "GaoXin"**,** "FengMei"**,** "YuJie"**,** "XieCheng"**,** "HuJia"**,** "JiangWen"**,** "TangFei"**,** "ShiQingZheng"**,** "XiaoLan"**,** "SongHua"**,** "YuanLi"**,** "HanXue"**,** "MoYun"**,** "RenWei"**,**"OuYangXiaoLan"  
 }**;** int nameNum = 30**;** pHashTable pTable = hashTableBuild(nameList**,** nameNum)**;** //遍历哈希表  
 for (int i = 0**;** i < HASH\_SIZE**;** i++)  
 {  
 printf("第%d个槽位有%d个元素\n"**,** i**,** pTable->lens[i])**;** pHashNode pCurNode = pTable->slots[i]**;** while (pCurNode != NULL)  
 {  
 printf("%s "**,** pCurNode->name)**;** pCurNode = pCurNode->next**;** }  
 printf("\n")**;** }  
}

实验结果

第0个槽位有1个元素

GaoXin

第1个槽位有1个元素

HanXue

第2个槽位有2个元素

HeYan JiangWen

第3个槽位有1个元素

XuJing

第4个槽位有2个元素

WangLei YuJie

第5个槽位有2个元素

LiuXiao XiaoLan

第6个槽位有1个元素

YuanLi

第7个槽位有1个元素

MoYun

第8个槽位有0个元素

第9个槽位有0个元素

第10个槽位有1个元素

RenWei

第11个槽位有0个元素

第12个槽位有1个元素

SunXiaoyu

第13个槽位有0个元素

第14个槽位有1个元素

OuYangXiaoLan

第15个槽位有0个元素

第16个槽位有0个元素

第17个槽位有0个元素

第18个槽位有0个元素

第19个槽位有1个元素

YangYang

第20个槽位有1个元素

ZhaoYun

第21个槽位有0个元素

第22个槽位有0个元素

第23个槽位有0个元素

第24个槽位有0个元素

第25个槽位有0个元素

第26个槽位有0个元素

第27个槽位有0个元素

第28个槽位有0个元素

第29个槽位有0个元素

第30个槽位有0个元素

第31个槽位有0个元素

第32个槽位有0个元素

第33个槽位有0个元素

第34个槽位有1个元素

DengHui

第35个槽位有0个元素

第36个槽位有0个元素

第37个槽位有1个元素

CaiLing

第38个槽位有1个元素

ZhangXiaoHong

第39个槽位有0个元素

第40个槽位有2个元素

LinFang HuJia

第41个槽位有1个元素

FengMei

第42个槽位有1个元素

ShiQingZheng

第43个槽位有0个元素

第44个槽位有1个元素

LuJun

第45个槽位有0个元素

第46个槽位有0个元素

第47个槽位有2个元素

LiMing HuangQiang

第48个槽位有2个元素

ChenWei TangFei

第49个槽位有2个元素

XieCheng SongHua

第50个槽位有0个元素

第51个槽位有0个元素

第52个槽位有0个元素

1. **实验心得**

（10%）

通过本次实验，我深入理解了二叉排序树的建立和查找操作，以及哈希表的设计与实现。在实践中，我学到了如何构建一个合理的哈希函数，处理冲突的不同方式，以及如何通过实验数据验证平均查找长度是否在可接受范围内。

二叉排序树的建立过程中，我注意到了对数据的插入和查找操作，以及中序遍历的实现。这让我更好地理解了二叉排序树的特性和应用场景。

在哈希表的设计中，我学到了哈希函数的设计原则，以及解决冲突的两种基本方法。通过实验数据的反复测试，我进一步掌握了如何选择合适的哈希表长度和处理冲突的方式，以达到平均查找长度的控制目标。

总的来说，本次实验增强了我的数据结构和算法的实际应用能力，对二叉排序树和哈希表有了更深刻的认识