**数据结构实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 学号-姓名 | 22009200766-方子康 | 实验时间 | 2024 年 6 月 1日 |
| 诚信声明 | 所有实验均由本人亲自完成 | | |
| 实验题目 | 题目一 顺序查找、折半查找  题目二 二叉排序树的建立、查找、插入和删除运算  题目三 哈希表的设计和应用 | | |
| 实验过程中遇到的主要问题 | **题目一：**   1. **文件读取**：由于是通过字符串按行读取文件，所以还要将数据按空格分隔存入int数组 2. **序列有序**：二分查找要求序列有序，所有必须排序后进行查找操作   **题目二：**   1. **删除结点情况复杂**：在删除一个结点时，需要考虑三种情况：1.该结点是叶节点 2.该结点有左儿子 3.该结点有右儿子 每次删除时要同时找到被删结点的父节点。删除后重新拼接的时候也要考虑是拼接为左子树还是右子树。 2. **插入的时候要考虑空树**：在空树上做插入就没有左右儿子之分了，直接就是根节点，其他情况要比较大小来确定是左儿子还是右儿子   **题目三：**   1. **链地址法头结点问题**：一开时我将头结点不存数据，方便头插法插入冲突数据，但是出现无输出的情况。后来发现初始化时，所有指针为NULL，此时头结点没有next指针域，发生插入错误。将头结点存入数据后问题解决，其实换一种初始化方式，如record->phone = record->name = ‘\0’,data->next = NULL也行。 2. **线性探测法在表满时查找死循环**：查找元素的条件是ht[i]->data.key != key && ht[i] != NULL 当表满时并且不存在查找元素时，该条件就会一直成立，导致死循环。于是加入计数变量，当x = （x+1）% N, x第二次到达N – 1 时说明已经将表扫描了一遍，此时退出。 3. **表要初始化或者给出空元素标志**：表不初始化就会导致无法判断该位置有无元素的情况，空元素表示可以是NULL/-1等，不能与插入元素冲突。 | | |
| 实验小结 | **数据设计：**  题目一：1.随机数据输入  题目二：1.输入序列无序 2.输入序列有序 3.输入序列有重复值 4.输入序列有负值  题目三：1.随机数据输入  **实验小结：**  本次实验我最有感悟的是中间信息的输出很重要，在编写长代码时，需要习惯性地编写各种数据结构下数据打印的函数，便于在对数据插入或者删除的时候打印调试信息。将结果打印，便于分析各种操作是否有误。在做题目二时，我认识到严谨的重要性，在做结点删除的时候需要充分考虑删除的情况，以及子树拼接的情况，一点疏漏就会导致后续结果的错误，调试起来也比较费劲。本次实验也进一步加深了我对二叉排序树，以及排序查找过程的理解，同时也学习到哈希表在存储时的便利性，可以在近似o(1)时间复杂度查找任何字符串的位置，同时也可以将稀疏元素聚拢在一定范围类，非常有用。 | | |
| 数据结构  （自定义数据类型） | **题目一：**  //元素类型typedef int ElemType;typedef int KeyType;//顺序存储typedef struct{    ElemType \*elem;    int length;}SSTable;  题目二：  typedef int KeyType;typedef char OtherInfoType;typedef struct{    KeyType key;    OtherInfoType info;}ElemType;//二叉树存储结构typedef struct BiTNode {    ElemType data;    struct BiTNode \*lchild, \*rchild;} BiTNode, \*BiTree;  题目三：  //记录结构typedef struct{    string phone;    string name;    string address;}Record;  //链地址法  typedef Record ElemType;typedef struct HashNode{    ElemType data;  //数据域    HashNode\* next;  //指向下一个链表元素}HashNode;typedef HashNode\* ChainHashTable[N]; //创建链哈希数  //线性探测哈希表组typedef Record LinearHashTable[N]; | | |
| 主要算法  （或算法说明） | 题目一：比较简单  题目二：  伪代码如下：  **插入二叉搜索树**(T: 指向二叉树根节点的指针, e: 待插入元素)  p <- T  father <- NULL  当 p 不为空 且 p->data.key 不等于 e.key 时循环执行  father <- p  如果 e.key 大于 p->data.key 则  p <- p->rchild  否则  p <- p->lchild  如果 p 不为空 则返回失败  s <- 新建节点  s->data <- e  s->lchild <- NULL  s->rchild <- NULL  如果 father 为NULL 则  T <- s  否则如果 e.key 大于 father->data.key 则  father->rchild <- s  否则  father->lchild <- s  返回成功  **删除二叉搜索树节点**(T: 指向二叉树根节点的指针, key: 待删除的关键字)  如果 T 为空 则返回失败  如果 key 等于 T->data.key 则  调用删除节点过程(T)  返回成功  否则如果 key 小于 T->data.key 则  返回 删除二叉搜索树节点(T->lchild, key)  否则  返回 删除二叉搜索树节点(T->rchild, key)  **删除节点**(T: 指向待删除节点的指针)  q <- NULL  s <- NULL  如果 T->rchild 为空 则  q <- T  T <- T->lchild  释放内存(q)  否则如果 T->lchild 为空 则  q <- T  T <- T->rchild  释放内存(q)  否则  q <- T  s <- T->lchild  当 s->rchild 不为空循环执行  q <- s  s <- s->rchild  T->data <- s->data  如果 q 不等于 T 则  q->rchild <- s->lchild  否则  q->lchild <- s->lchild  释放内存(s) | | |