**《数据结构与算法》作业**

**22920212204392 黄勖**

# 习题5 查找

**5-1 设顺序表的长度为30，平均分成5块，每块6个元素。如果采用分块查找，则其平均查找长度为(   C )。**

**(A) 5**

**(B) 5.7**

**(C) 6.5**

**(D) 8.2**

解：

分块查找会分两部分进行,第一步先进行索引表查找判断其在那个字表中,第二步然后进行在字表中的查找

索引表有5个元素 所以平均查找长度为:(1+5)/2=3

字表中有6个元素,所以平均查找长度为:(1+6)/2=3.5

所以总的平均查找长度为3+3.5=6.5

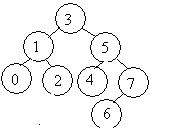
**5-2 将关键字2，4，6，8，10，12，14，16依次存放于一维数组A[0...7]中，如果采用折半查找方法查找关键字，在等概率情况下查找成功时的平均查找长度为( A )。**

**(A) 21/8**

**(B) 7/2**

**(C) 4**

**(D) 9/2**



查找成功平均长度=1/8(1+2\*2+3\*4+4\*1)=21/8 ，故选A

**5-3 简单描述静态查找和动态查找的区别。**

答：动态查找表在查找过程中插入元素或者从查找表中删除元素；静态查找表只是查找特定元素或者检索特定元素的属性。

**1、静态查找**

首先无论是静态查找还是动态查找，都要有查找的对象，也就是包含很多同类型数据的“表”，这个“表”可以理解为一个由同类型数据元素组成的一个“集合”，该集合可以用各种容器来存储，例如数组、链表、树等，我们统称这些存储数据的数据结构为——查找表。可见，查找表有时是我们传统意义的表，有时候是很复杂的一种结构。

静态查找就是我们平时概念中的查找，是“真正的查找”。之所以说静态查找是真正的查找，因为在静态查找过程中仅仅是执行“查找”的操作，即：（1）查看某特定的关键字是否在表中（判断性查找）；（2）检索某特定关键字数据元素的各种属性（检索性查找）。这两种操作都只是获取已经存在的一个表中的数据信息，不对表的数据元素和结构进行任何改变，这就是所谓的静态查找。

**常见的静态查找（表）**：顺序查找、二分法查找、索引顺序查找（分块查找）、斐波那契查找等

**2、动态查找**

看到上面静态查找的概念，动态查找就很好理解了，个人总觉得动态查找不像是“查找”，动态查找它更像是一个对表进行“创建、扩充、修改、删除”的过程。动态查找的过程中对表的操作会多两个动作：（1）首先也有一个“判断性查找”的过程，如果某特定的关键字在表中不存在，则按照一定的规则将其插入表中；（2）如果已经存在，则可以对其执行删除操作。动态查找的过程虽然只是多了“插入”和“删除”的操作，但是在对具体的表执行这两种操作时，往往并不是那么简单。

**5-4 设数组A中只存放正数和负数。试设计算法，将A中的负数调整到前半区间，正数调整到后半区间。分析算法的时间复杂度。**

答：

可以直接采用冒泡排序，按升序排列就好。

public void bubbleSort(int arr[]) {

boolean didSwap;

for(int i = 0, len = arr.length; i < len - 1; i++) {

didSwap = false;

for(int j = 0; j < len - i - 1; j++) {

if(arr[j + 1] < arr[j]) {

int temp;

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

didSwap = true;

}

}

if(didSwap == false)

return;

}

}

最佳情况为O（n），最坏的情况为O（n2)

**5-5 按照“逐点插入方法”建立一个二叉排序树，树的形状取决于( B )。**

**(A) 数据序列的存储结构**

**(B) 数据元素的输入次序**

**(C) 序列中的数据元素的取值范围**

**(D) 使用的计算机的软、硬件条件**

**5-6 用利用逐点插入法建立序列(50, 72, 43, 85, 75, 20, 35, 45, 65, 30)对应的二叉排序树以后，查找元素35要在元素间进行( B )次比较。**

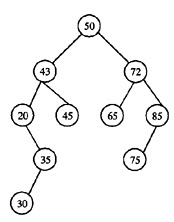
**(A) 3**

**(B) 4**

**(C) 5**

**(D) 8**

按上述次序创建的二叉排序树如图所示。查找元素35需要比较4次。



**5-7 给定n个整数，设计算法实现：**

**(1) 构造一棵二叉排序树；**

**(2) 从小到大输出这n个数。**

解：

BST的中序遍历即为严格单调的遍历，故求中序遍历即可，程序如下：

1. #include <iostream>
2. using namespace std;
4. *// BST的结点*
5. typedef struct node
6. {
7. int key;
8. struct node \*lChild, \*rChild;
9. }Node, \*BST;
11. *// 在给定的BST插入element, 使之称为新的BST*
12. bool BSTInsert(Node \* &p, int element)
13. {
14. if(NULL == p) *// 空树*
15. {
16. p = new Node;
17. p->key = element;
18. p->lChild = p->rChild = NULL;
19. return true;
20. }
22. if(element == p->key) *// BST中不能有相等的值*
23. return false;
25. if(element < p->key) *// 递归*
26. return BSTInsert(p->lChild, element);
28. return BSTInsert(p->rChild, element); *// 递归*
29. }
31. *// 创建BST*
32. void createBST(Node \* &T, int a[], int n)
33. {
34. T = NULL;
35. int i;
36. for(i = 0; i < n; i++)
37. {
38. BSTInsert(T, a[i]);
39. }
40. }
42. void inOrderTraverse(BST T)
43. {
44. if(NULL != T)
45. {
46. inOrderTraverse(T->lChild);
47. cout << T->key << endl;
48. inOrderTraverse(T->rChild);
49. }
50. }
52. int main()
53. {
54. int a[10] = {4, 5, 2, 1, 0, 9, 3, 7, 6, 8};
55. int n = 10;
57. BST T = NULL;
59. *// 并非所有的a[]都能构造出BST,所以，最好对createBST的返回值进行判断*
60. createBST(T, a, n);
62. inOrderTraverse(T);
64. return 0;
65. }

**5-8 在平衡二叉树中，插入关键字46后得到一颗新的平衡二叉树。在新的平衡二叉树中，关键字37所在结点的左、右孩子结点中保存的关键字是( C )。**

**(A) 18，46**

**(B) 25，46**

**(C) 25，53**

**(D) 25，69**

解释:插入46以后，该二叉树根结点的平衡因子由-1变为-2，在最小不平衡子树根结点的右子树（R）的左子树（L）中插入新结点引起的不平衡属于RL型平衡旋转，需要做两次旋转操作（先右旋后左旋）。

18

**5-9 用依次插入关键字的方法，为序列{ 5, 4, 2, 8, 6, 9 }构造一棵平衡二叉树(要求分别画出构造过程中的各棵不平衡二叉树)。**

答：给出一种可能的构造过程：

将5插入二叉树中，作为根节点，此时二叉树为：

5

将4插入二叉树中，作为5的左子节点，此时二叉树为：

5

/

4

将2插入二叉树中，作为4的左子节点，此时二叉树为：

5

/

4

/

2

将8插入二叉树中，作为5的右子节点，此时二叉树为：

5

/ \

4 8

/

2

将6插入二叉树中，作为8的左子节点，此时二叉树为：

5

/ \

4 8

/ \

2 6

将9插入二叉树中，作为8的右子节点，此时二叉树为：

5

/ \

4 8

/ \

2 6

\

9

由于每个节点的左右子树高度差都不超过1，因此这棵二叉树是一棵平衡二叉树。

**5-10 链地址法是Hash表的一种处理冲突的方法，它是将所有哈希地址相同的数据元素都存放在同一个链表中。关于链地址法的叙述，不正确的是( C )。**

**(A) 平均查找长度较短**

**(B) 相关查找算法易于实现**

**(C) 链表的个数不能少于数据元素的个数**

**(D) 更适合于构造表前无法确定表长的情况**

解：

链地址法特点

(1)拉链法处理冲突简单，且无堆积现象，即非同义词决不会发生冲突，因此平均查找长度较短；

(2)由于拉链法中各链表上的结点空间是动态申请的，故它更适合于造表前无法确定表长的情况；

(3)开放寻址法为减少冲突，要求装填因子α较小，故当结点规模较大时会浪费很多空间。而拉链法中可取α≥1，且结点较大时，拉链法中增加的指针域可忽略不计，因此节省空间；

(4)在用拉链法构造的散列表中，删除结点的操作易于实现。只要简单地删去链表上相应的结点即可。而对开放地址法构造的散列表，删除结点不能简单地将被删结点的空间置为空，否则将截断在它之后填人散列表的同义词结点的查找路径。这是因为各种开放地址法中，空地址单元(即开放地址)都是查找失败的条件。因此在用开放地址法处理冲突的散列表上执行删除操作，只能在被删结点上做删除标记，而不能真正删除结点。

**5-11 设哈希(Hash)函数H(k)=(3k)%11，用线性探测再散列法处理冲突，di=i。已知为关键字序列22，41，53，46，30，13，01，67构造哈希表如下：**

**哈希地址 0 1 2 3 4 5 6 7 8 9 10**

**关键字 22 41 30 01 53 46 13 67**

**查找长度 1 1 2 2 1 1 2 6**

**则在等概率情况下查找成功时的平均查找长度是( A )。**

**(A) 2**

**(B) 24/11**

**(C) 3**

**(D) 3.5**

解：

22\*3%11=0

41\*3%11=2

53\*3%11=5

46\*3%11=6

30\*3%11=2

13\*3%11=6

1\*3%11=3

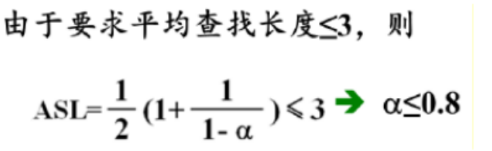
67\*3%11=3

得上表，ASL=(1+1+2+2+1+1+2+6)/8=2

**5-12 有100个不同的关键字拟存放在哈希表L中。处理冲突的方法为线性探测再散列法，其平均查找长度为。试计算L的长度(一个素数)，要求在等概率情况下，查找成功时的平均查找长度不超过3。**

**素数表：101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167。**

解：



设线性表L长度l,有:

α=100/l<=0.8 求出l>=125,即由题意选择127这个素数