查找

本章重点：

1、顺序表的查找

2、有序表的查找：折半查找法

3、二叉排序树及其查找过程（动态）

4、二叉排序树的插入（动态）

5、平衡二叉树、B-和B+树的结构

6、哈希表即哈希函数的构造方法了解

查找表：

静态查找表（查询）

动态查找表（查询、插入、删除 ）

主关键字 次关键字

平均查找长度：ASL=sum(Pi\*Ci)

为提高ASL，需要人为增加数据元素的关系，以便按照某种规则查找。

    9.1 静态查找表

    9.1.1顺序表的查找

顺序表表示：

**[cpp]** [view plaincopy](http://blog.csdn.net/jtahstu/article/details/42355001)

1. typedef int KeyType;
2. typedef struct
3. {
4. KeyType key;
5. InfoTypedata;
6. }NodeType;
7. typedef NodeType SeqList[MAXL]

顺序查找

**[cpp]** [view plaincopy](http://blog.csdn.net/jtahstu/article/details/42355001)

1. int SeqSearch(SeqList R, int n, KeyType k)
2. {
3. int i = 0;
4. while (i <n && R[i].key != k)
5. i++;
6. if (i >=n)
7. return 0;
8. else
9. return i+ 1;
10. }

    9.1.2有序表的查找

1、折半查找

**[cpp]** [view plaincopy](http://blog.csdn.net/jtahstu/article/details/42355001)

1. int BinSearch(SeqList R, int n, KeyType k)
2. {
3. int low = 0,high = n - 1, mid;
4. while (low<= high)
5. {
6. mid =(low + high) / 2;
7. if(R[mid].key == k)
8. returnmid + 1;
9. if(R[mid].key > k)
10. high= mid - 1;
11. else
12. low =mid + 1;
13. }
14. return 0;
15. }

2、斐波那契查找：按照斐波那契数列分段查找

3、插值查找：根据给定值key确定进行比较的关键字R[i].key的查找方法

    i=（key-R[i].key）/(R[h].key-R[l].key)\*(h-l+1)

    适用于关键字均匀分布的表

    9.1.4索引顺序表的查找

1、分块查找，索引顺序查找

**[cpp]** [view plaincopy](http://blog.csdn.net/jtahstu/article/details/42355001)

1. int IdxSearch(IDX I,int m,SeqList R,int n,KeyType k)
2. {
3. intlow=0,high=m-1,mid,i;
4. intb=n/m;              //b为每块的记录个数
5. while(low<=high)       //在索引表中进行二分查找,找到的位置存放在low中
6. {
7. mid=(low+high)/2;
8. if(I[mid].key>=k)
9. high=mid-1;
10. else
11. low=mid+1;
12. }
13. //应在索引表的high+1块中,再在线性表中进行顺序查找
14. i=I[high+1].link;
15. while(i<=I[high+1].link+b-1 && R[i].key!=k) i++;
16. if(i<=I[high+1].link+b-1)
17. return i;
18. else
19. return-1;
20. }

    9.2动态查找算法（树表的查找）

    9.2.1二叉排序树和平衡二叉树

1、二叉查找树（二叉排序树）及其查找过程

2、二叉排序树的查找

3、二叉排序树的插入算法（递归算法）

**[cpp]** [view plaincopy](http://blog.csdn.net/jtahstu/article/details/42355001)

1. int InsertBST(BSTNode \*&p,KeyType k)
2. {
3. if (p==NULL)                        //原树为空,新插入的记录为根结点
4. {
5. p=(BSTNode\*)malloc(sizeof(BSTNode));
6. p->key=k;
7. p->lchild=p->rchild=NULL;
8. return 1;
9. }
10. else if(k==p->key)                 //树中存在相同关键字的结点,返回0
11. return 0;
12. else if(k<p->key)
13. returnInsertBST(p->lchild,k);  //插入到\*p的左子树中
14. else
15. returnInsertBST(p->rchild,k);  //插入到\*p的右子树中
16. }

4、二叉排序树的生成算法

**[cpp]** [view plaincopy](http://blog.csdn.net/jtahstu/article/details/42355001)

1. BSTNode \*CreateBST(KeyType A[],int n)   //返回BST树根结点指针
3. {
5. BSTNode\*bt=NULL;                   //初始时bt为空树
7. int i=0;
9. while(i<n)
11. {
13. InsertBST(bt,A[i]);             //将关键字A[i]插入二叉排序树T中
15. i++;
17. }
19. returnbt;                          //返回建立的二叉排序树的根指针
21. }

5、二叉排序树的查找算法（递归）

**[cpp]** [view plaincopy](http://blog.csdn.net/jtahstu/article/details/42355001)

1. BSTNode \*SearchBST(BSTNode \*bt,KeyType k)
3. {
5. if (bt==NULL|| bt->key==k)         //递归终结条件
7. returnbt;
9. if(k<bt->key)
11. returnSearchBST(bt->lchild,k);  //在左子树中递归查找
13. else
15. return SearchBST(bt->rchild,k);  //在右子树中递归查找
17. }

    9.3 哈希表

1、确定的对应关系f

2、对应关系f就是哈希函数

3、哈希函数是一个映象，构造哈希函数的方法：

直接定址法、除留余数法、数字分析法、平方取中法、折叠法

4、冲突现象和解决冲突的方法

当k1!=k2时，f(k1)=f(k2)

开放定址法

（1）线性探测法

（2）平方探测法

拉链法