BUPT

第七章　查找

7.1 基本概念与术语

查找表 同一类型的记录(数据元素)的**集合**。eg.成绩单

查找 指定某个值，在查找表中确定是否存在一个记录，该记录的**关键字**等于给定值。

老师统计>90分数

关键字 记录(数据元素)中的某个数据项的值。eg.学号/姓名

主关键字 该关键字可以唯一地标识一个记录。

次关键字 该关键字不能唯一标识一个记录。eg.重名

静态查找表 对查找表的查找仅是以查询为目的，不改动查找表中的数据。

动态查找表 在查找的过程中同时插入不存在的记录，或删除某个已存在的记录。

查找过程中数据元素发生改动

查找成功 查找表中存在满足查找条件的记录。

查找不成功 查找表中不存在满足查找条件的记录。

决定程序返回的状态

内查找 整个查找过程都在内存中进行。

外查找 在查找过程中需要访问外存。

刻画平均情况下的时间表达度

平均查找长度ASL——查找方法时效的度量

为确定记录在查找表中的位置，需将关键字和给定值比较次数的期望值。

查找成功时的ASL计算方法：$$ASL = \sum_{i=1}^{n} p_i c_i$$

n：记录的个数

pi：查找第i个记录的概率，

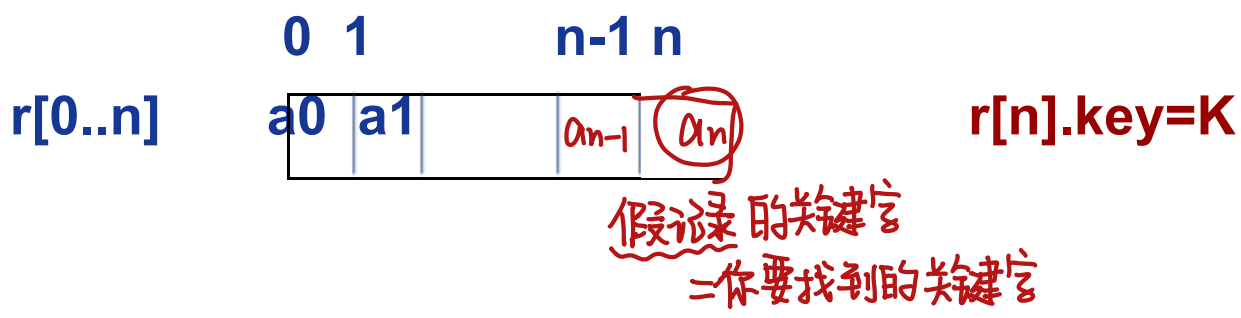
（不特别声明时认为等概率 pi =1/n ）

ci：找到第i个记录所需的比较次数

关键字和给定值比较的数学期望

约定：无特殊说明，一般默认关键字的类型为整型

7.2 顺序表的查找



[算法描述]

```
int seqsearch (int *a, const int n, const int K )
{
   int i = 0; a[n] = K;
   while (a[i] != K) i++;
   return i;
}
```

O(n)

0 ⋮ n-1

次 1 ⋮ n

[程序设计技巧] 设置监视哨，提高算法效率。

[性能分析]

- 空间：一个辅助空间。
- 时间：

1) 查找成功时的平均查找长度

设表中各记录查找概率相等

$ASLs(n)=(1+2+ \dots +n)/n =(n+1)/2$

2)查找不成功时的平均查找长度　$ASLf =n+1$

[算法特点]

- 算法简单，对表结构无任何要求
- n很大=>查找效率较低
- 改进措施：非等概率查找时，可将查找概率高的记录尽量排在表前部。

7.3 二分查找

满足 r[i].key ≤ r[i+1].key, 0 ≤ i <n-1 前提：关键字有序

仍可用顺序查找，但在找不到时不需比较到表尾，只需比较到比给定值大的记录就可终止。

模拟折纸过程
比较区间中值与查找关键字大小

[折半(二分)查找法] 缩小一半查找范围

1	2	3	4	5	6	7	8	9	10	11
05	13	19	21	37	56	64	75	80	88	92
↑low					↑mid					↑high

=⌊(low+high)/2⌋ 取整

K=21

l	h	m
1	11	6
1	5	3
4	5	4

K=85

l	h	m
1	11	6
7	11	9
10	11	10
10	9	

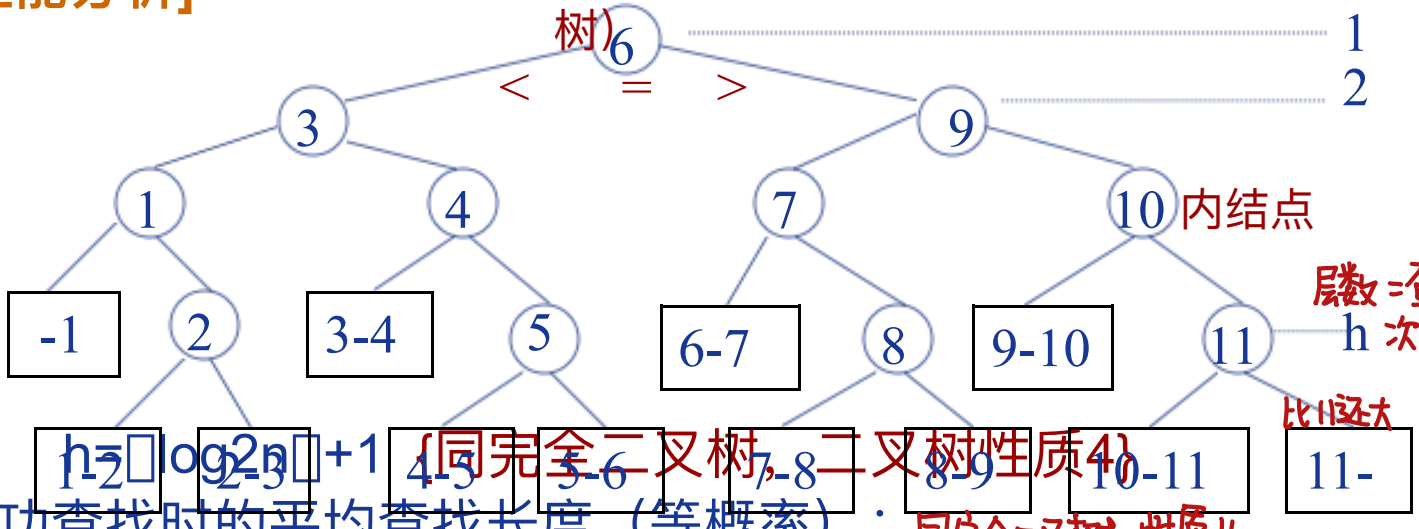
low>high 查找不出来

[算法描述]

```
int binsearch ( int K, int v[ ], int n )
{
   int low, high, mid;
   low = 1;
   high = n;
   while ( low <= high )
   {
      mid = ( low + high ) / 2;
      if ( K < v[mid] )
         high = mid - 1;
      else if ( K > v[mid] )
         low = mid + 1;
      else /* 找到了匹配的值*/
         return mid;
   }
   return -1; /* 没有查到*/
}
```

[性能分析]

判定树（描述查找过程的二叉树）



$h=\lfloor \log_2 n \rfloor +1$ {同完全二叉树，二叉树性质4}

成功查找时的平均查找长度（等概率）：

ASLs=

$$\frac{1}{n}\sum_{j=1}^{h} j \cdot 2^{j-1} = \frac{n+1}{n}\log_2(n+1) - 1 \approx \log_2(n+1) - 1$$

例：ASLS=(1*1+2*2+3*4+4*4)/11=3

不成功查找时的查找长度：h-1或h次

1. 具有12个关键字的有序表，在等概率情况下，折半查找的平均查找长度（A）

A. 3.1　　B. 4　　C. 2.5　　D. 5

2. 折半查找的时间复杂度为（　）

D

A. O（n2）　　B. O（n）　　C. O（nlogn）　　D. O（logn）

3. 用二分（对半）查找表的元素的速度比用顺序法(　)

D

A. 必然快　　B. 必然慢　　C. 相等　　D. 不能确定

4.在顺序表（8,11,15,19,25,26,30,33,42,48,50）中，用二分（折半）法查找关键码值20，需做的关键字比较次数为____.

4

在有序表A[1..12]中，采用二分查找算法查等于A[12]的元素，所比较的元素下标依次为_____。

6,9,11,12

7.4 二叉排序树BST(二叉查找/搜索树)

[定义] 动态

二叉排序树或者是空树，或者是满足如下性质的二叉树：

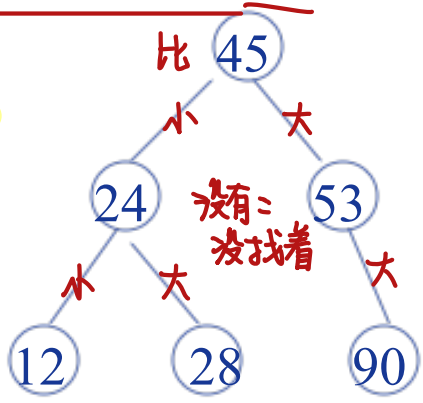
1)若其左子树非空，则左子树上所有结点的值均小于根结点的值；

2)若其右子树非空，则右子树上所有结点的值均大于等于根结点的值；

3)其左右子树本身又各是一棵二叉排序树

[性质]

中序遍历一棵二叉排序树，将得到一个以关键字递增排列的有序序列



判别给定二叉树是否为二叉排序树的算法如何实现？？

(用二叉链表存储)

方法1：根据二叉排序树中序遍历所得结点值为增序的性质，在遍历中将当前遍历结点与其前驱结点值比较，即可得出结论。

```
void  JudgeBST(BTree t,int  flag)
{ //全局指针pre，初值为NULL；调用时变量flag=TRUE
   if (t!=NULL && flag)
   {
      JudgeBST(t->lc,flag)；// 中序遍历左子树
      if(pre==NULL)
          pre=t；// 中序的第一个结点不判断
     else  if(pre->data<t->data)
          pre=t；//前驱指针指向当前结点
      else
           flag=FLASE；    //不是完全二叉树
     JudgeBST(t->rc,flag)；// 中序遍历右子树
   }
```

方法2：照定义，二叉排序树的左右子树都是二叉排序树，根结点的值大于左子树中所有值而小于右子树中所有值，即根结点大于左子树的最大值而小于右子树的最小值。

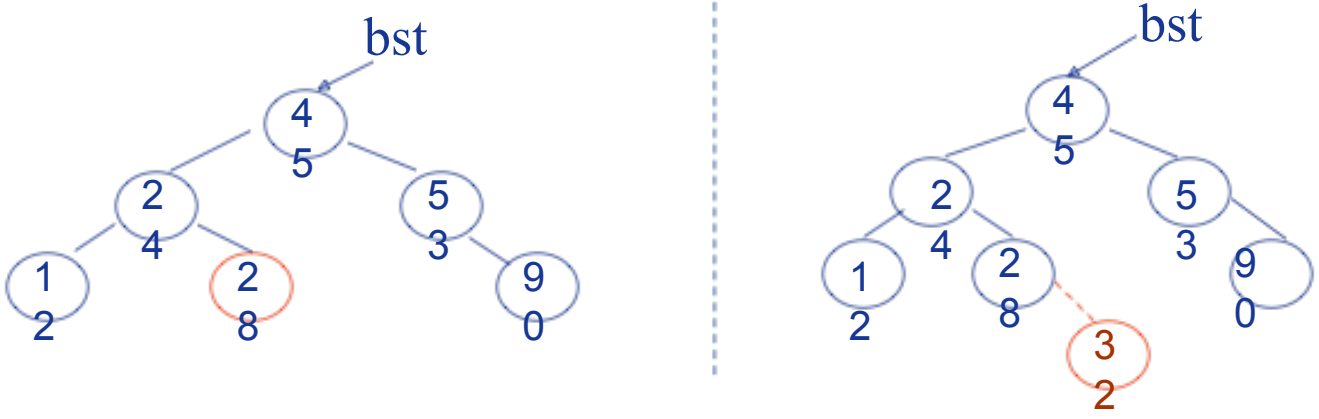
```
bool   JudgeBST(BTree t)
{
  if(t==NULL) return  TRUE;
  if (JudgeBST(t->lc) && JudgeBST(t->rc))
  {
     m=max(t->llink);
     n=min(t->rlink); //左子树中最大值和右子树中最小值
    return (t->data>m && t->data<n);
  }
 else  return   FALSE; //不是
}
```

```
int   max(BTree p)//求左子树的最大值
 {if (p==NULL) return   -maxint;
                  //返回机器最小整数
   else{
       while(p->rc！=null) p=p->rc;
       return p->data; }
}

int   min(BTree p)//求右子最小值
 {if (p==NULL)  return    maxint;

   else{
    while(p->lc!=NULL) p=p->lc;
```

[在二叉排序树上的操作]

[1.查找]

[例] K=28 K=32



查找步骤：若根结点的关键字值等于查找的关键字，成功。

否则，若小于根结点的关键字值，查其左子树。

若大于根结点的关键字值，查其右子树。

[查找算法]

```
Bitree  SearchBST ( BiTree T,  KeyType key )
  // 在二叉分类树查找关键字之值为 key 的结点，找到返回该结
  // 点的地址，否则返回空。T 为根结点的指针。
{
    if  ( ( T==NULL) ||  ( key==T ->data) )
        return ( T ) ;
    else if (key <T ->data. key  )
        return (SearchBST (T ->lc, key ));
     else
        return (SearchBST ( T -> rc, key ));
 }
```

[2.插入]

- 首先执行查找算法，找出被插结点的父亲结点。
- 判断被插结点是其父亲结点的左、右儿子。将被插结点作为叶子结点插入。
- 若二叉树为空。则首先单独生成根结点。

注意：新插入的结点总是叶子结点。

[3.生成]

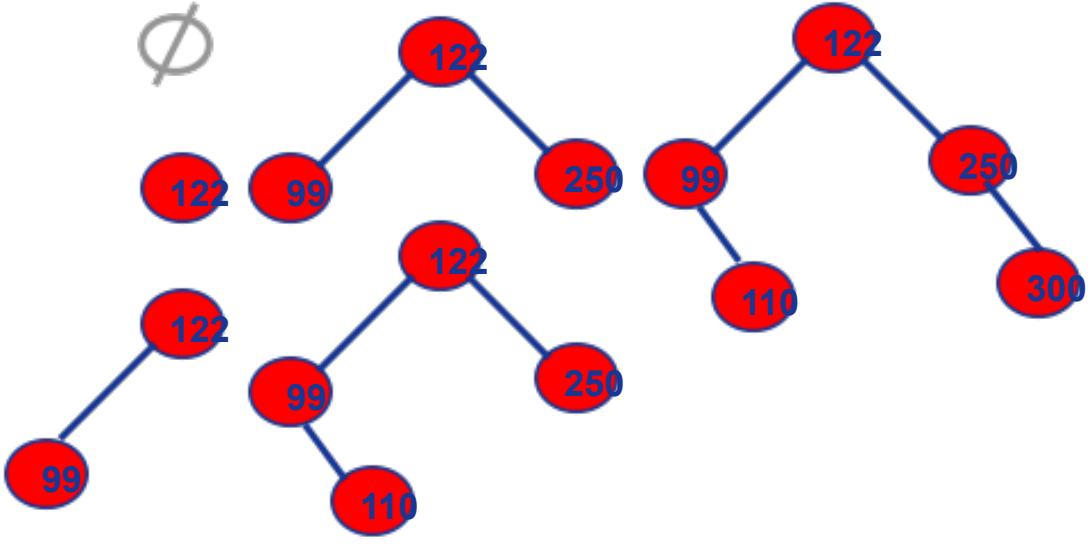
[算法步骤] 反复执行以下操作

a. 读入一个记录，设其关键字为K；

b. 调用查找算法，确定插入位置；

c. 调用插入算法，实施插入结点的操作；

例：将序列122、99、250、110、300、280 作为二叉排序树的结点的关键字值，生成二叉排序树。



[4.删除]

依据被删除结点p的不同情况分析：

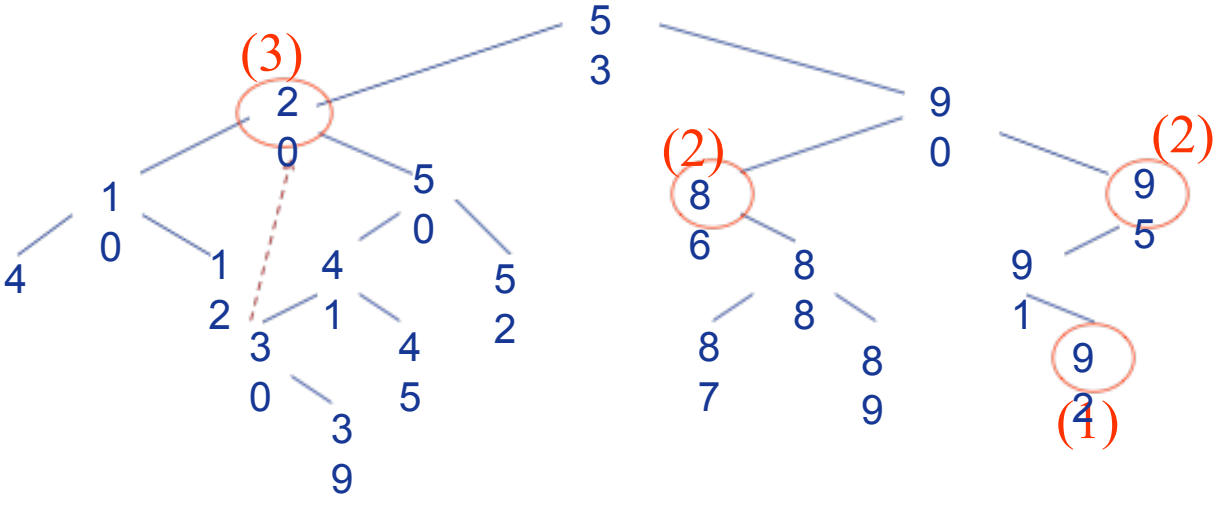
- 1. p是叶子结点：修改其双亲指针即可
- 2. p只有左孩子：用p的左子树代替以p为根的子树

  p只有右孩子：用p的右子树代替以p为根的子树
- 3. p有两个孩子：找到p的中序后继(或前趋)结点q，

  q的数据复制给p，

  删除只有右子(或左子)/无孩子的q

例：



```
void Delete(BSTree T, keytype  X) //在二叉排序树T上，删除为X的结点。
{
    BSTree f, p=T;
   while (p && p->key!=X) //查找值为X的结点
        if (p->key>X) {f=p; p=p->lc;}//f为p的双亲
        else {f=p; p=p->rc;}
   if (p==NULL) {printf("无关键字为X的结点\n"); exit(0);}
   if (p->lc==NULL) //被删结点无左子树
        if (f->lc==p) f->lc=p->rc;//将被删结点的右子树接到其双亲上
       else f->rc=p->rc;
  else {//被删结点有左子树
       q=p; s=p->lc;
       while (s->rc !=NULL) { //查左子树中最右下的结点（中序最后结点）
           q=s; s=s->rc;}
       p->key=s->key;          //结点值用其左子树最右下的结点的值代替
       if (q==p) p->lc=s->lc;//被删结点左子树的根结点无右子女
       else q->rc=s->lc;     //s是被删结点左子树中序序列最后一个结点
       free(s);
  }
```

[4. 性能分析]

- 给定树的形态，等概率查找成功时的ASL=∑ci /n
  - 最差(单支树)：(n+1)/2
  - 最好(类似折半查找的判定树)：log2(n+1)-1
  - 随机：≈1+4log2n
- 关键字有序出现时，构造出“退化”的二叉排序树，树深为n，各种操作代价O(n)。避免方法：采用平衡二叉树

7.5 哈希表

[哈希表的基本思想] 散列 → 哈希 Hash

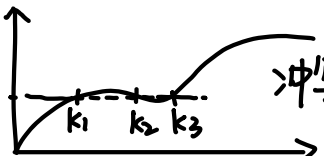
在记录的存储地址和它的关键字之间建立一个确定的对应关系；这样，理想状态不经过比较，一次存取就能得到所查元素。 通过函数建立关键字通过运算得存储位置

[术语]

哈希表 一个有限的连续的地址空间，用以容纳按哈希地址存储的记录。 (顺序存储)

哈希函数 记录的存储位置与它的关键字之间存在的一种对应关系。 Loc(ri)=H(keyi) 自变量 没有落在(0, m-1)表明哈希表长申请不合适

冲突 对于keyi≠keyj， i ≠j，出现的H(keyi) = H(keyj)的现象。 0~m-1 值域



冲突：两个同关键字通过H函数算出了一样的值

同义词 在同一地址出现冲突的各关键字。k_1, k_2, k_3

哈希(散列)地址 根据设定的哈希函数H(key)和处理冲突的方法确定的记录的存储位置。

装填因子 表中填入的记录数n和哈希表表长 m之比。

特殊方法下 $\alpha > 1$

$\alpha = n/m$ 越大记录越多.

[设计哈希表的过程]

1)明确哈希表的地址空间范围。即确定哈希函数的值域。

哈希函数值域

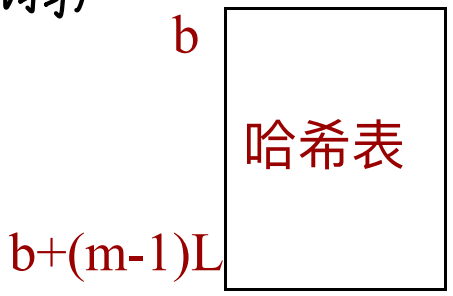
计算量小./复杂度小

① 连续.足够. ②选择 H函数. 构造.

2)选择合理的哈希函数。该函数要保证所有可能的记录的哈希地址均在指定的值域内，并使冲突的可能性尽量小。

③落在值域. 减小冲突 (概率分布 均匀)

3)设定处理冲突的方法。



[哈希函数的基本构造方法]

构造原则： 算法简单，运算量小；

均匀分布，减少冲突。

[1. 直接定址法]

H(key)=a *key + b 线性 a、b为常数

e.g: 令：a、b为1，有两个关键字k1, k2 值为 10 、1000；

选10 、1000 作为存放地址。包含.11. 1001

特点：计算简单，冲突最少。不好设计

[2. 数字分析法/基数转换法]

取关键字在某种进制下的若干数位作为哈希地址。

e.g: key = 236076 基数为10，看成是 13 进制的。

则：(236075)13 = 8 4154 7；选取 4154 作为散列地址(假设 表长m =10000)。

[3. 平方取中法]

取关键字平方后的中间几位作为哈希地址。

e.g: (4731)2 = 223 82 361 ； 选取 82 （假设 m = 100）。

[4. 随机数法]

H(key) = random(key)

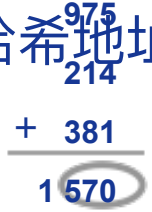
特点：较适于关键字长度不等时的情况。

[5. 折叠法]

将关键字分割成位数相同的几部分(最后一部分的位数可以不同)，然后取这几部分的叠加和(舍去进位)作为哈希地址。

e.g: key = 381，412，975

选取 768 或 570 作为散列地



[6. 除留余数法]

H(key) = key MOD p　　(p≤m)

m: 哈希表的表长；　p: 最好为素数

最常用，余数总在 0 ～ p-1 之间。通常p选 < m 的最大质数

如：m = 1024, 则 p = 1019。

e.g: 选取 p 为质数的理由：

设 key 值都为奇数，选 p 为偶数；

则 H(key) = key MOD p ，结果为奇数，　一半单元被浪费掉。

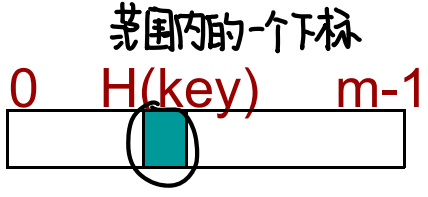
设 key 值都为 5 的倍数，选 p 为 95；

则 H(key) = key MOD p ，结果为：0、5、10、15、……

[处理冲突的常用方法]

[1. 开放定址法 (空缺编址法)]

Hi = (H(key)+di) MOD m



i=1,2, ..., k (k≤m-1)

m:哈希表的表长； di:增量序列

选取

H_1 无存放：存在这里
有存放，用d_2计算得H_2 无 有

1)线性探测再散列

di= 1,2, ..., m-1 线性增量

缺陷：有聚集(堆积)现象—非同义词地址冲突。

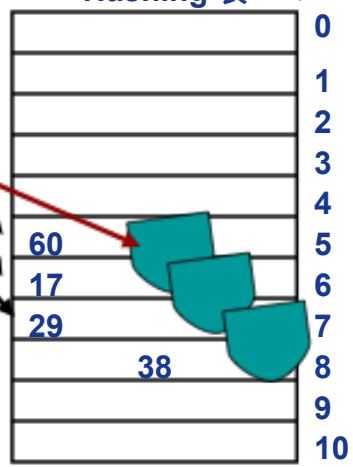
e.g: 假定采用的 hashing 函数为： H(key) = key MOD 11 假定的关键字序列为： 17、60、29、38 …… ； 它们的散列过程为：

H(17) = 6

H(60) = 5

H(29) = 7

H(38) = 5



冲突： (5+1) MOD 11 = 6 又冲突

7 MOD 11 = 7 又冲突

(5+3) MOD 11 = 8 OK!

初级冲突：不同关键字值的结点得到同一个散列地址。 二次聚集：同不同散列地址的结点争夺同一个单元。

结果：冲突加剧，最坏时可能达到 O(n)级代价。

当散列 38 时发生冲突，

同 60 争夺第 5 个单元

解决办法： 探测下一个 空单元

Hi = (H(key)+di) MOD 11

思考：假定有k个关键字互为同义词，若用线性探测再散列法把这k个关键字存入散列表中，至少要进行多少次

2)二次探测再散列

di= 12, -12, 22, -22, 32,...,k2

k ≤ m/2

缺陷：不易探查到整个散列空间。

3)伪随机探测再散列

di = 伪随机数序列

4) 双重散列函数探查法

di= i*h1(key)　　0< i ≤m- 1

要求：h1(key)的值与m互素。

m为素数时， h1(key)可取1到m-1之间的任何数，

建议：

h1(key)=key mod (m-2) +1

m为2的方幂时， h1(key)可取1到m-1之间的任何奇数

注意：开放定址法不宜执行删除操作

[2. 再哈希法] 设计足够多个哈希函数.

Hi = RHi(key) i=1,2, ... , k

出现冲突时，采用多个 hashing 函数计算散列地址，直到找到空单元为止。

如：H1(key) = key MOD 11

H2(key) = (key+ MOD 9) ＋ 1

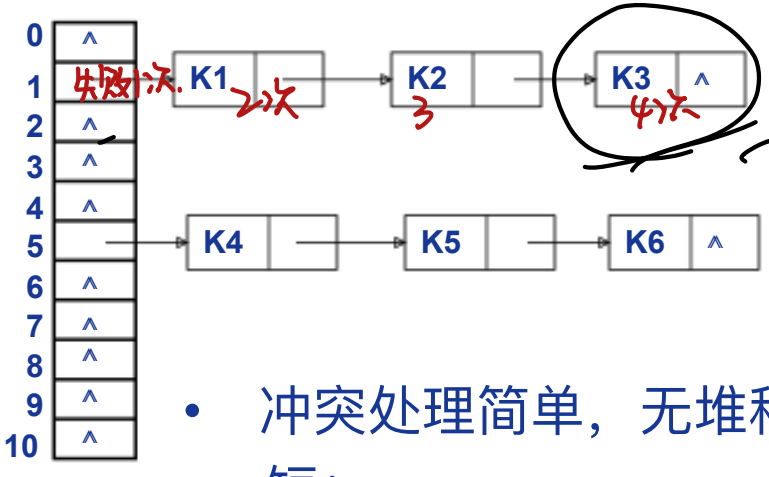
H3(key)= ……..

……..

H10(key)=…….

[3. 链地址法]

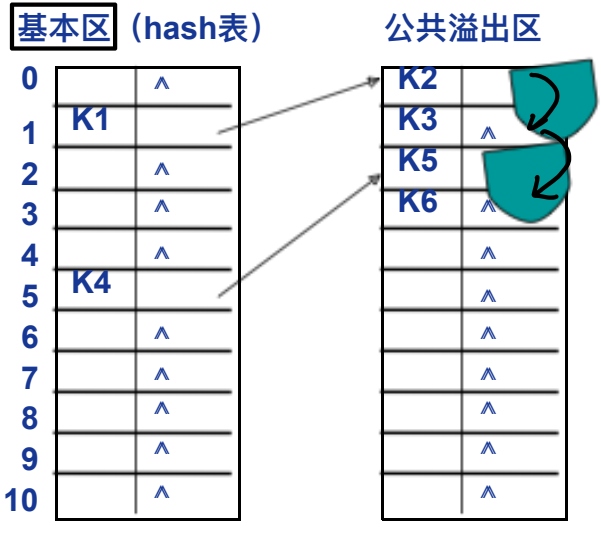
为每个哈希地址建立一个单链表，存储所有具有同义词的记录。



- 冲突处理简单，无堆积现象，平均查找长度较短；
- 较适合于事先无法确定表长的情况；
- 可取$\alpha \geq 1$，当结点信息规模较大时，节省空间
- 删除结点的操作易于实现

[4. 建立公共溢出区]

将发生冲突的结点都存放在一个公共溢出区内。通常用于组织存在于外存设备上的数据文件。hash表(基本区)只存放一个记录。发生冲突的记录都存放在公共溢出区内。



[哈希表的查找及其分析]

查找方法同 hashing 函数的产生办法。

例：key = en to tre fire fem seks syv

H(key) = 2 0 3 0 4 8 1 → 5

解决冲突方法：线性探测再散列

0	1	2	3	4	5	6	7	8
to	fire	en	tre	fem	syv			seks

精确

en to tre fem 算上"|"那次比较

ASLs= (1+1+1+2+1+1+5)/7 =12/7

是1 fire seks 0

ASLf = (7+6+5+4+3+2+1+1+8)/9=37/9

0都是0 是6 是7

0~7 8种

0~8 9种

α = 7/9

[平均查找长度与哈希表的装填因子的关系]

$$\alpha = \frac{\text{表中填入的记录数}}{\text{哈希表长度}}$$

大约情况.

在一般情况下，□□，冲突的可能性□，ASL□，但空间的浪费□。

	成功ASL	失败ASL
线性探测再散列	$\frac{1}{2}(1+\frac{1}{1-\alpha})$	$\frac{1}{2}(1+\frac{1}{(1-\alpha)^2})$
伪随机探测再散列	$-\frac{1}{\alpha}\ln(1-\alpha)$	$\frac{1}{1-\alpha}$
链地址法	$1+\frac{\alpha}{2}$	$\alpha+e^{-\alpha}$

作业

1. 已知二叉树排序树中某结点指针p,其双亲结点指针为fp,p为fp的左孩子，试编写算法，删除p所指结点。

```
typedef struct node
{int data; struct node *left, *right;
}BiTNode, *BSTree;
void Delete(BSTree root,p,fp)
```

2. 已知含12个关键字的有序表及其相应权值为：

	1	2	3	4	5	6	7	8	9	10	11	12
关键字	A	B	C	D	E	F	G	H	I	J	K	L
权值	4	6	3	4	9	3	2	6	1	5	3	4

(1)画出对以上有序表进行折半查找的判定树，求折半查找时查找成功的平均查找长度ASL。

(2)若为等概率查找，求折半查找时查找成功的平均查找长度ASL。

3. 选取哈希函数H(k)=(3k) MOD 11。

1) 用线性探测再散列法处理冲突，

2) 用链地址法处理冲突

试在0~10的散列地址空间中对关键字序列(22, 41, 53, 46, 30, 13, 01, 67)造哈希表，并求等概率情况下查找成功时的平均查找长度和不成功时的平均查找长度以及装填因子 。