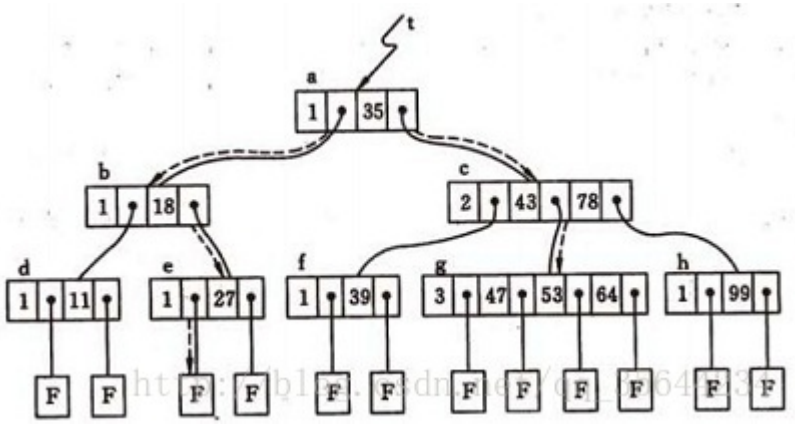
B-树

B-树的基本操作-查找

我们先给出如下的一个4阶的B-树结构。



如上图所示，这是我们的一个4阶的B-树，现在假设我们需要查找45这个数是否在B-树中。

- 从根节点出发，发现根节点a有1个关键字为35，其中45>35，往右子树走，进入节点c
- 发现结点c有2个关键字，其中其中43<45<78,所以进入结点g。
- 发现结点g有3个关键字，其中3<45<47,所以继续往下走，发现进入了结束符结点：F，所以45不在B-树中

从上面的查找过程中，我们可以得出基本的查找过程为：

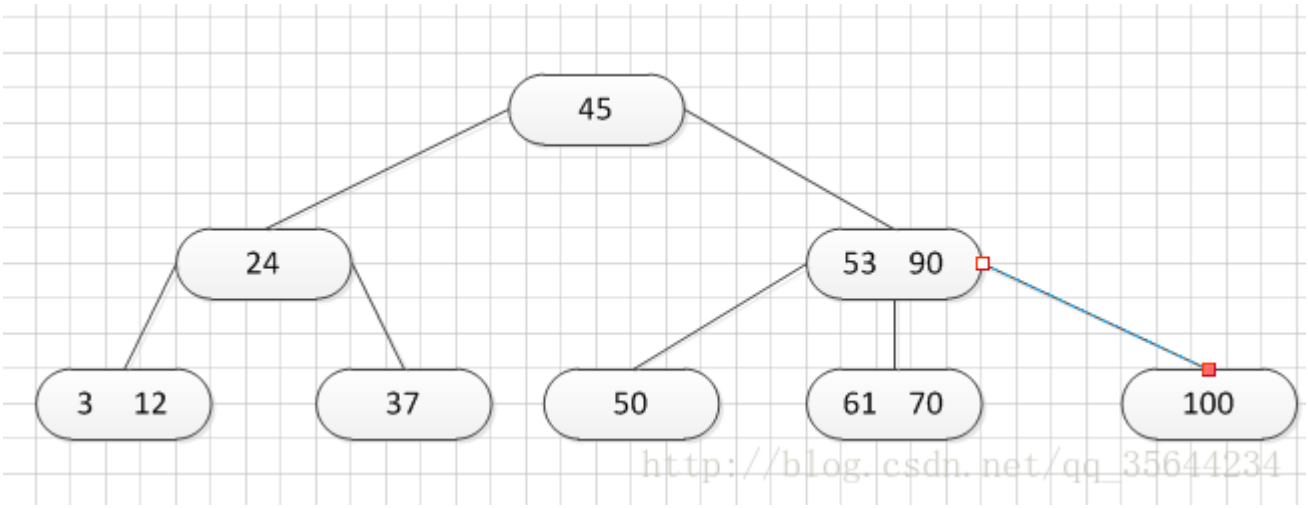
- 在 B- 树中查找节点
- 在节点中查找关键字

B-树的基本操作-插入

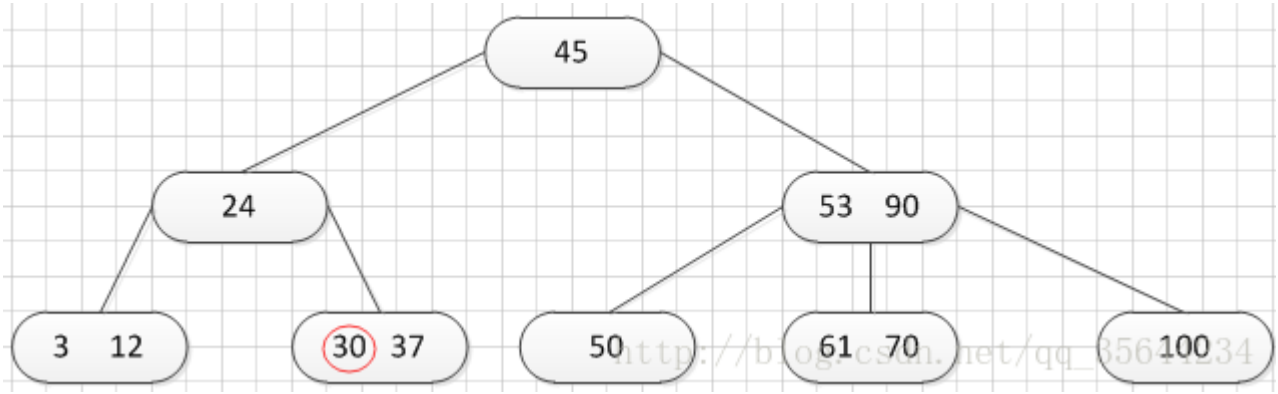
- 使用之前介绍的查找算法查找出关键字的插入位置，如果我们在B-树中查找到了关键字，则直接返回。否则它一定会失败在某个最底层的终端结点上。
- 然后，我就需要判断那个终端结点上的关键字数量是否满足：n<=m-1,如果满足的话，就直接在该终端结点上添加一个关键字，否则我们就需要产生结点的“分裂”。

> 分裂的方法是：生成一新结点。把原结点上的关键字和k（需要插入的值）按升序排序后，从中间位置把关键字（不包括中间位置的关键字）分成两部分。左部分所含关键字放在旧结点中，右部分所含关键字放在新结点中，中间位置的关键字连同新结点的存储位置插入到父结点中。如果父结点的关键字个数也超过（m-1），则要再分裂，再往上插。直至这个过程传到根结点为止。

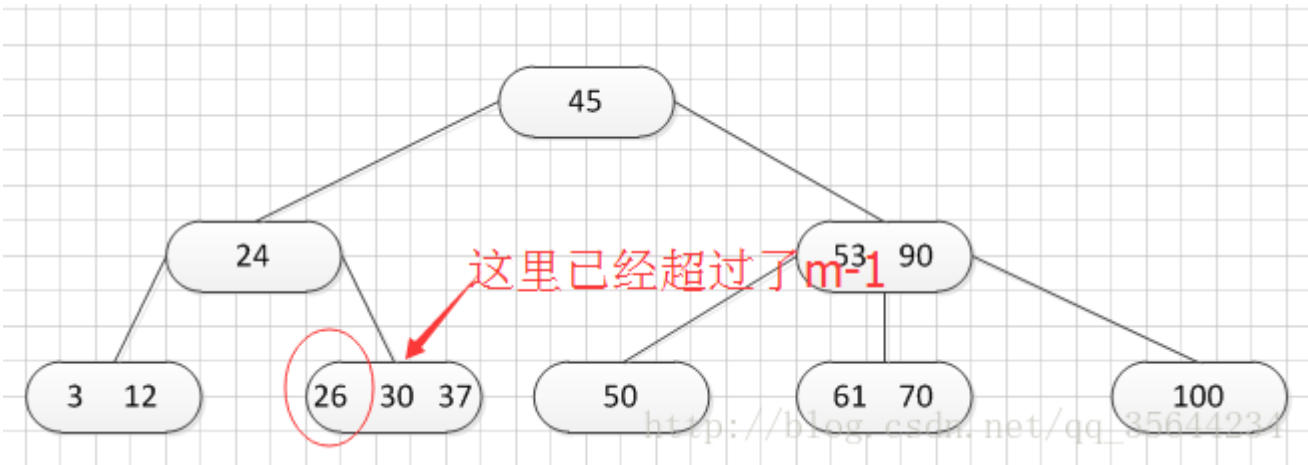
下面我们来举例说明，首先假设这个B-树的阶为：3。树的初始化时如下：



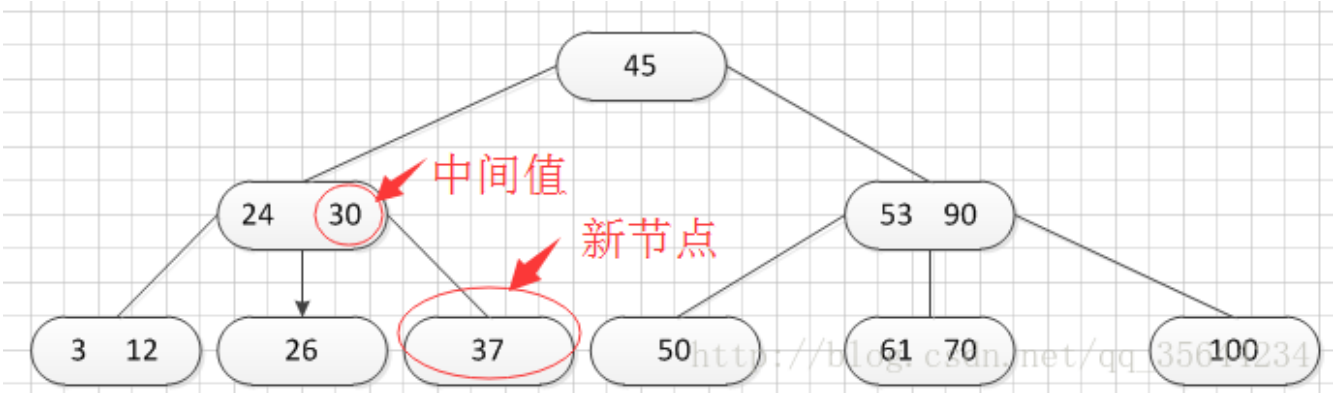
首先，我需要插入一个关键字：30，可以得到如下的结果：



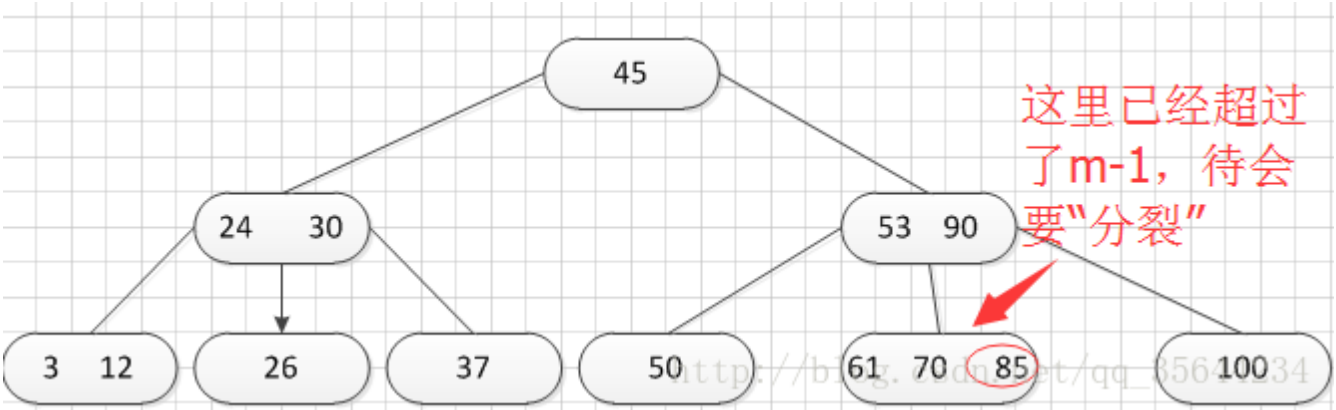
再插入26，得到如下的结果：



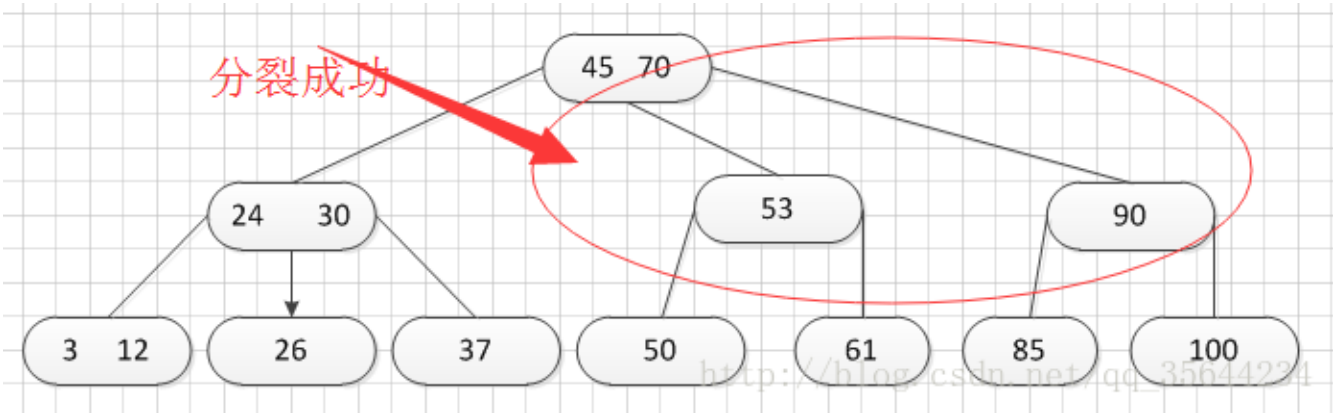
此时如图所示，在插入的那个终端结点中，它的关键字数已经超过了m-1=2，所以我们需要对结点进分裂，所以我们先对关键字排序，得到：26 30 37 ，所以它的左部分为（不包括中间值）：26，中间值为：30，右部为：37，左部放在原来的结点，右部放入新的结点，而中间值则插入到父结点，并且父结点会产生一个新的指针，指向新的结点的位置，如下图所示：



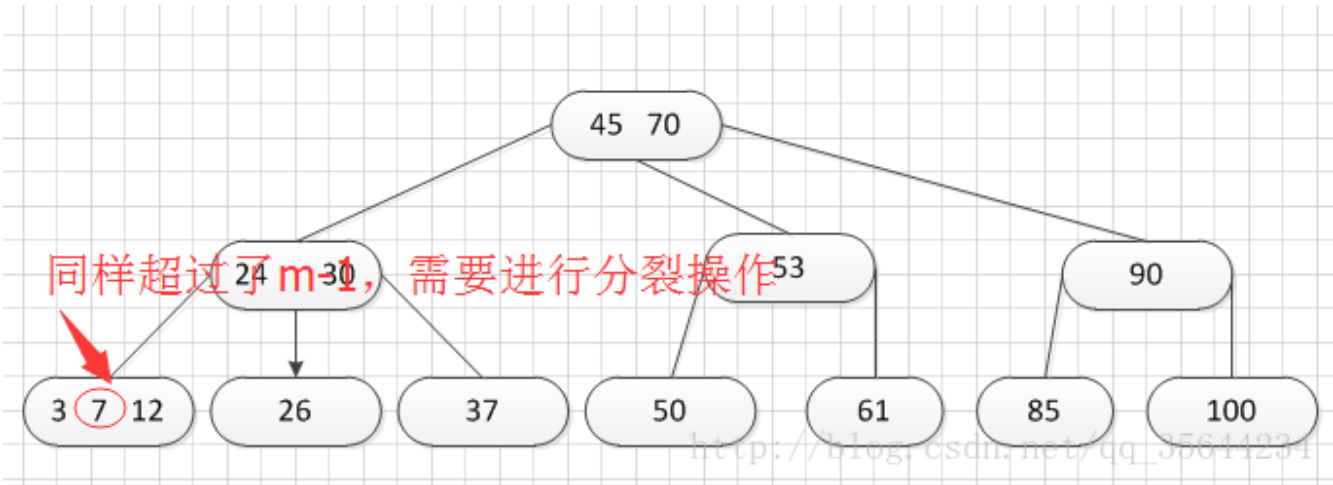
然后我们继续插入新的关键字：85，得到如下图结果：



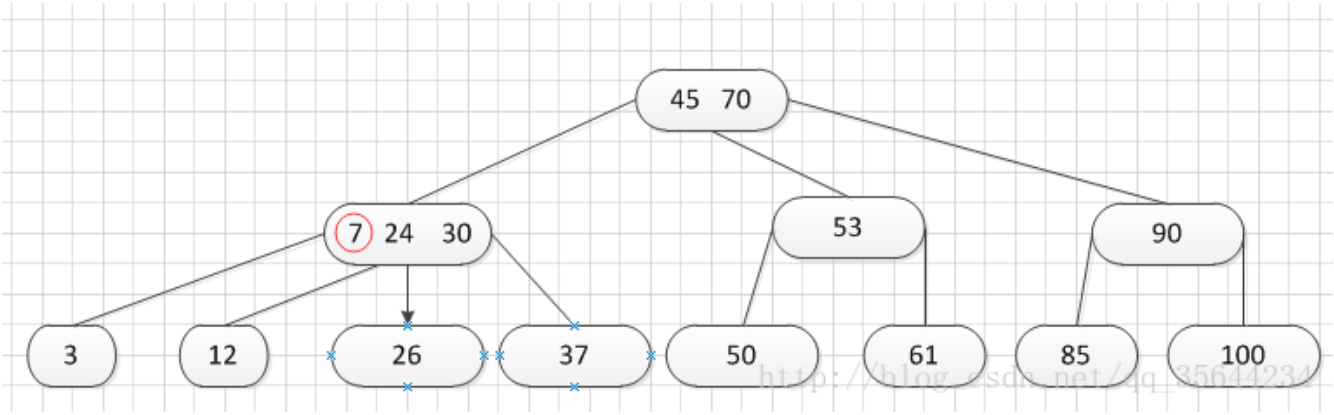
正如图所示，我需要对刚才插入的那个结点进行"分裂"操作，操作方式和之前的一样，得到的结果如下：



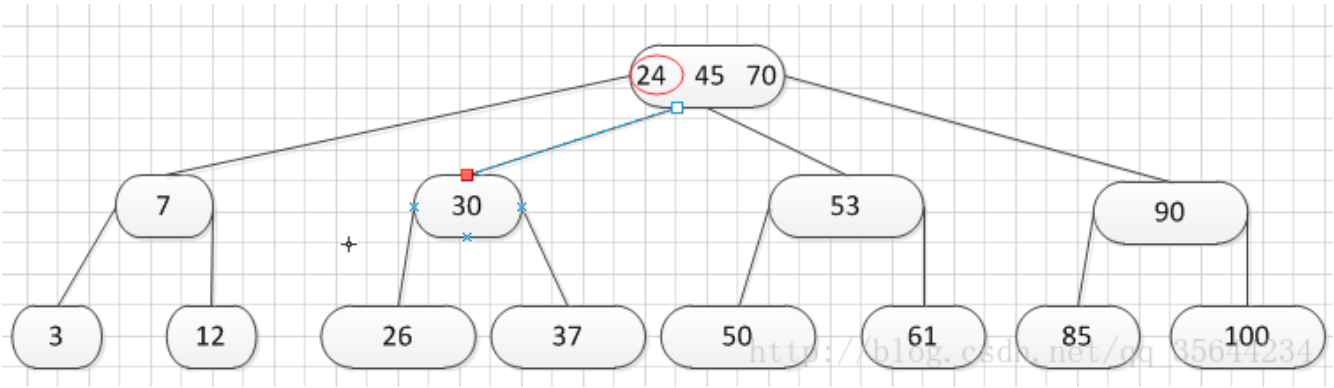
我们继续插入一个新的关键字：7，得到如下结果：



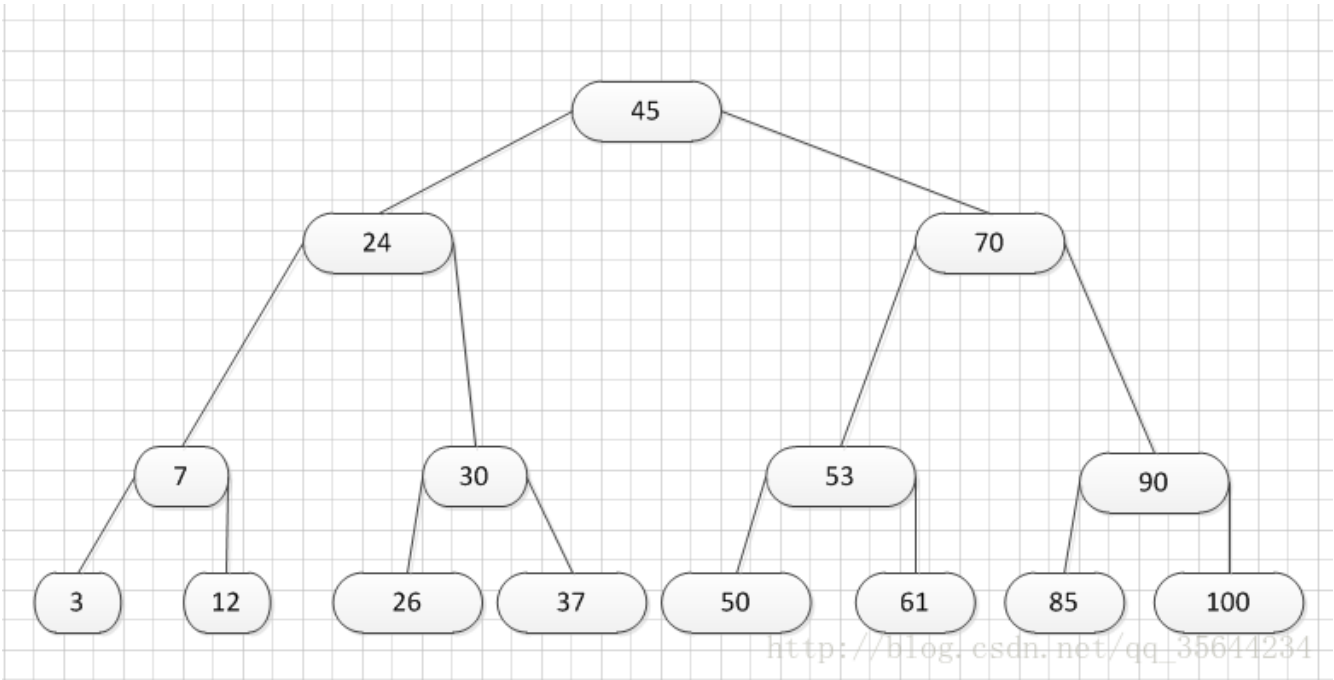
同样，需要对新的结点进行分裂操作，得到如下的结果：



到了这里，我就需要继续对我们的父亲结点进行分裂操作，因为它的关键字数超过了：m-1.



这个时候我们需要对父亲结点进行分列操作，但是根结点没父亲，所以我们需要重新创建根结点了。



B-树的基本操作-删除

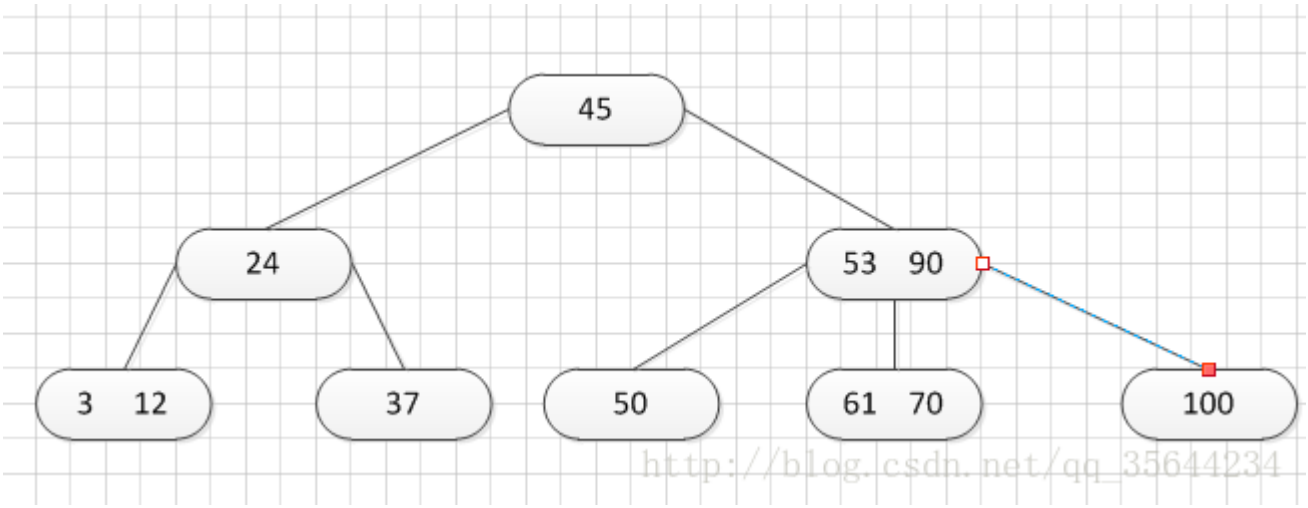
B-树的删除操作同样是分为两个步骤：

- 利用前述的B-树的查找算法找出该关键字所在的结点。然后根据 k（需要删除的关键字）所在结点是否为叶子结点有不同的处理方法。如果没有找到，则直接返回。

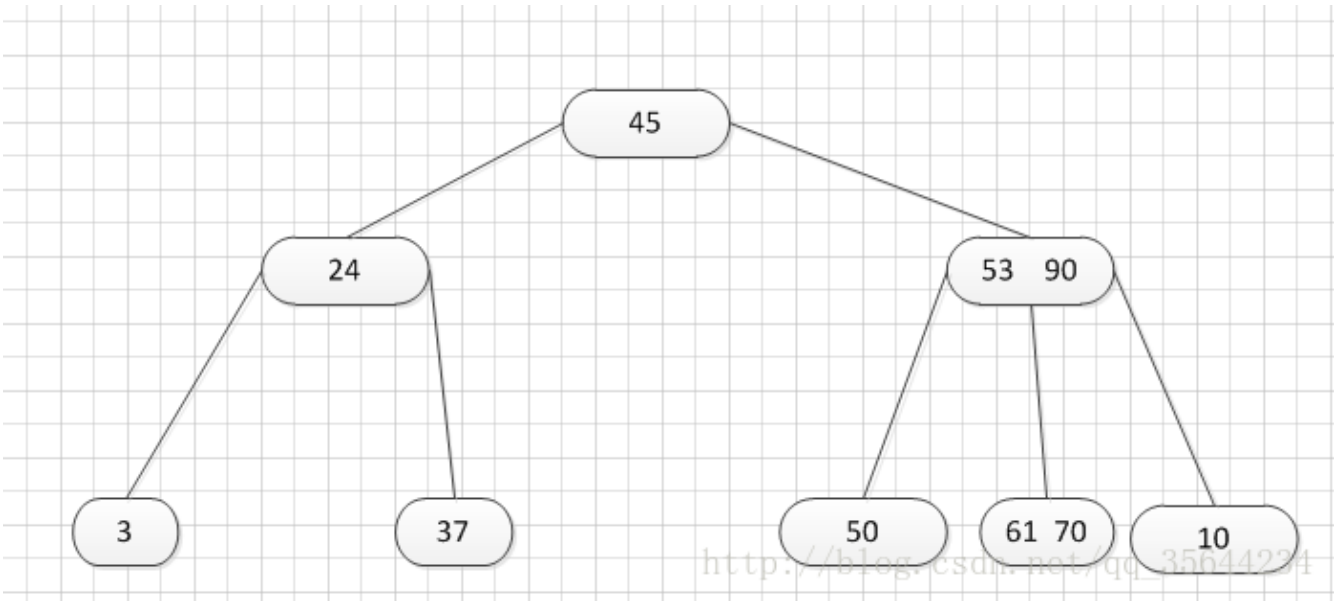
- 若该结点为非叶结点，且被删关键字为该结点中第i个关键字key[i]，则可从指针son[i]所指的子树中找出最小关键字Y，代替key[i]的位置，然后在叶结点中删去Y。

下面，我们给出删除叶子结点的三种情况：

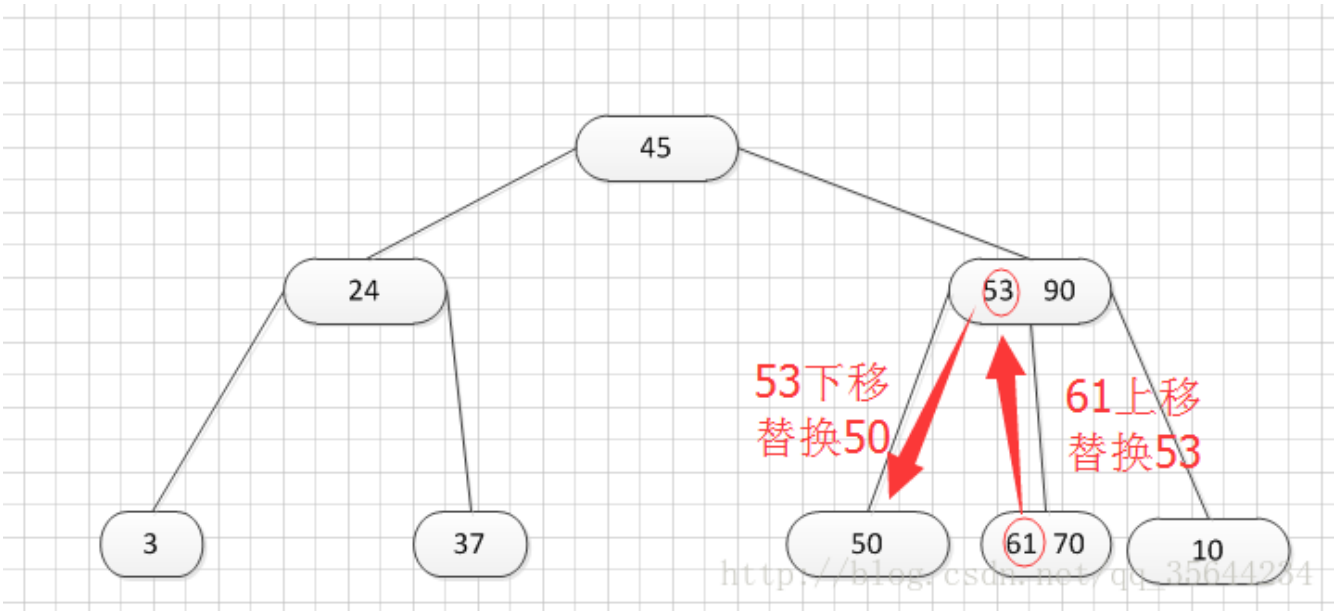
第一种：关键字的数不小于$\lceil m/2 \rceil$，如下图删除关键字：12



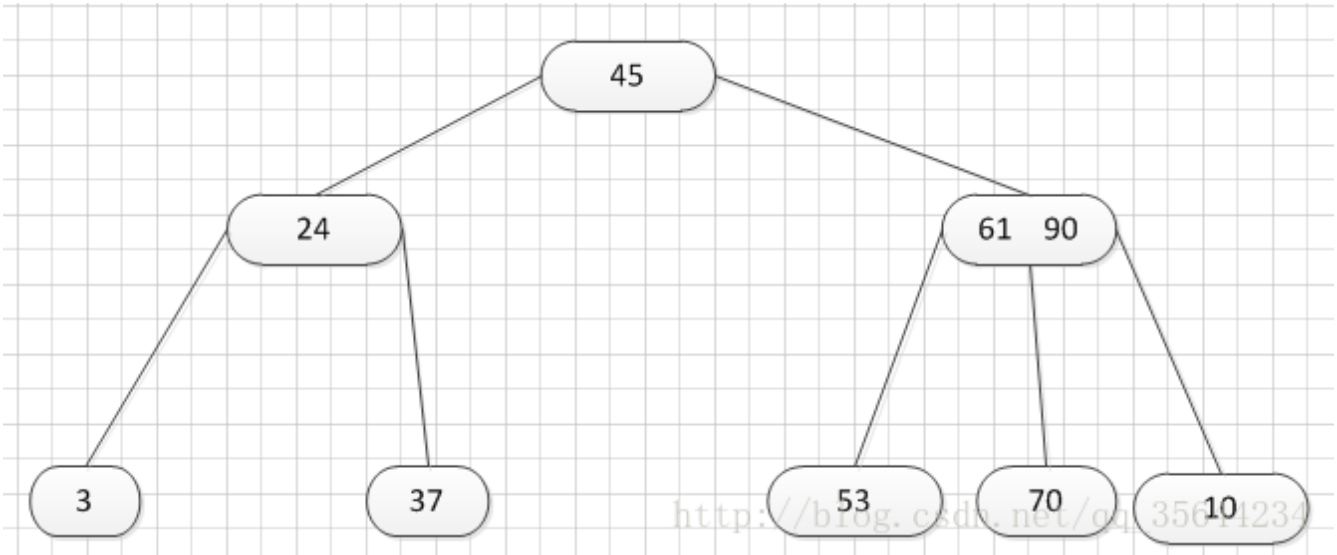
删除12后的结果如下，只是简单的删除关键字12和其对应的指针。



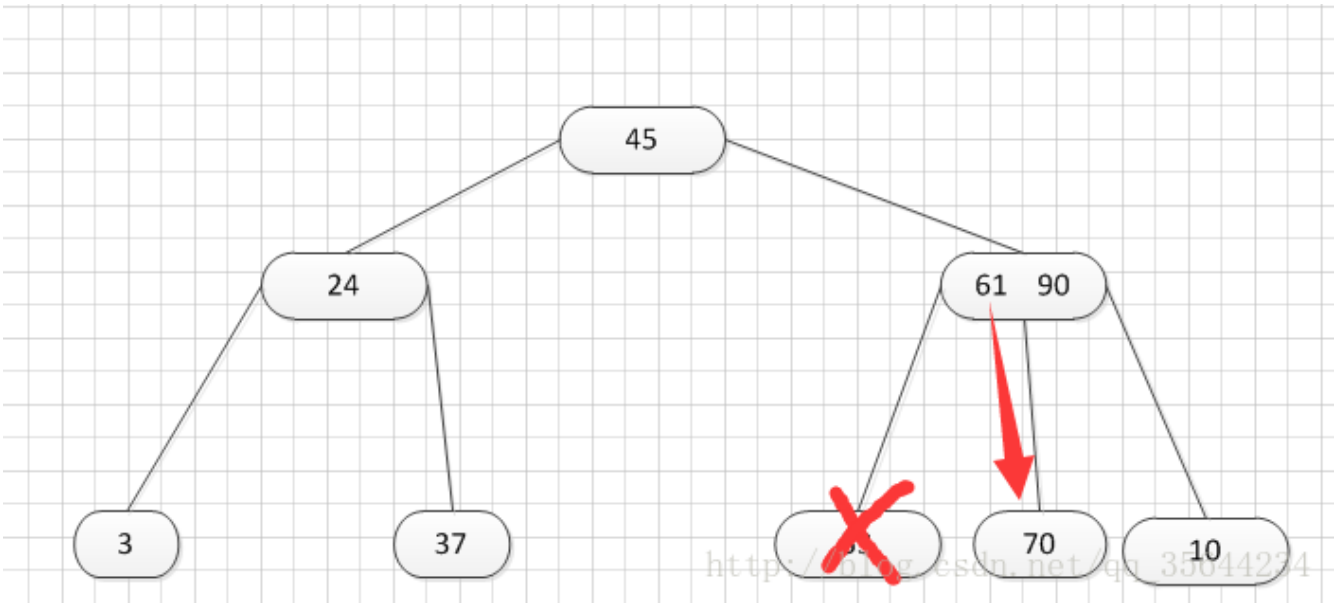
第二种：关键字个数n等于$\lceil m/2 \rceil - 1$，而且该结点相邻的右兄弟(或左兄弟)结点中的关键字数目大于$\lceil m/2 \rceil - 1$。



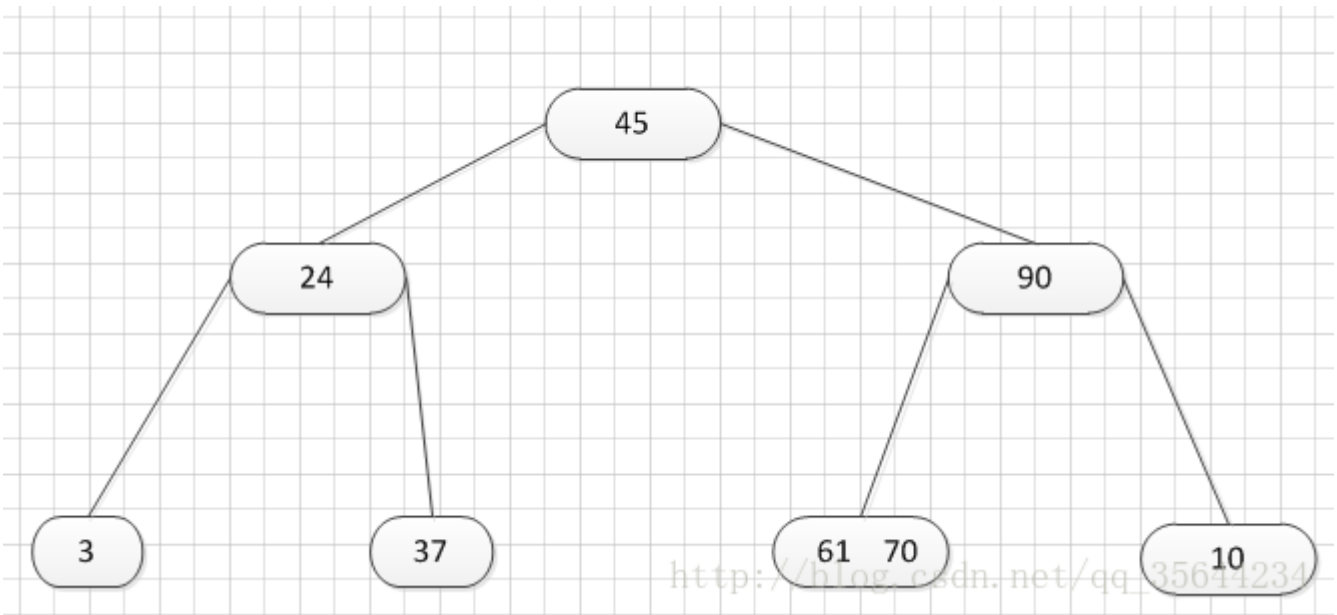
如上图，所示，我们需要删除50这个关键字，所以我们需要把50的右兄弟中最小的关键字：61上移到其父结点，然后替换小于61的关键字53的位置，53则放至50的结点中。然后，我们可以得到如下的结果：

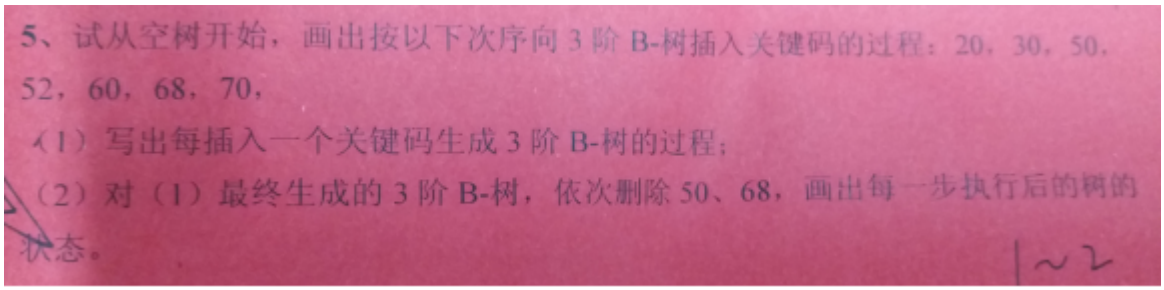


第三种：关键字个数n等于$\lceil m/2 \rceil - 1$，而且被删关键字所在结点和其相邻的兄弟结点中的关键字数目均等于$\lceil m/2 \rceil - 1$



如上图所示，我们需要删除53，那么我们就要把53所在的结点其他关键字（这里没有其他关键字了）和父亲结点的61这个关键字一起合并到70这个关键字所占的结点。得到如下所示的结果：

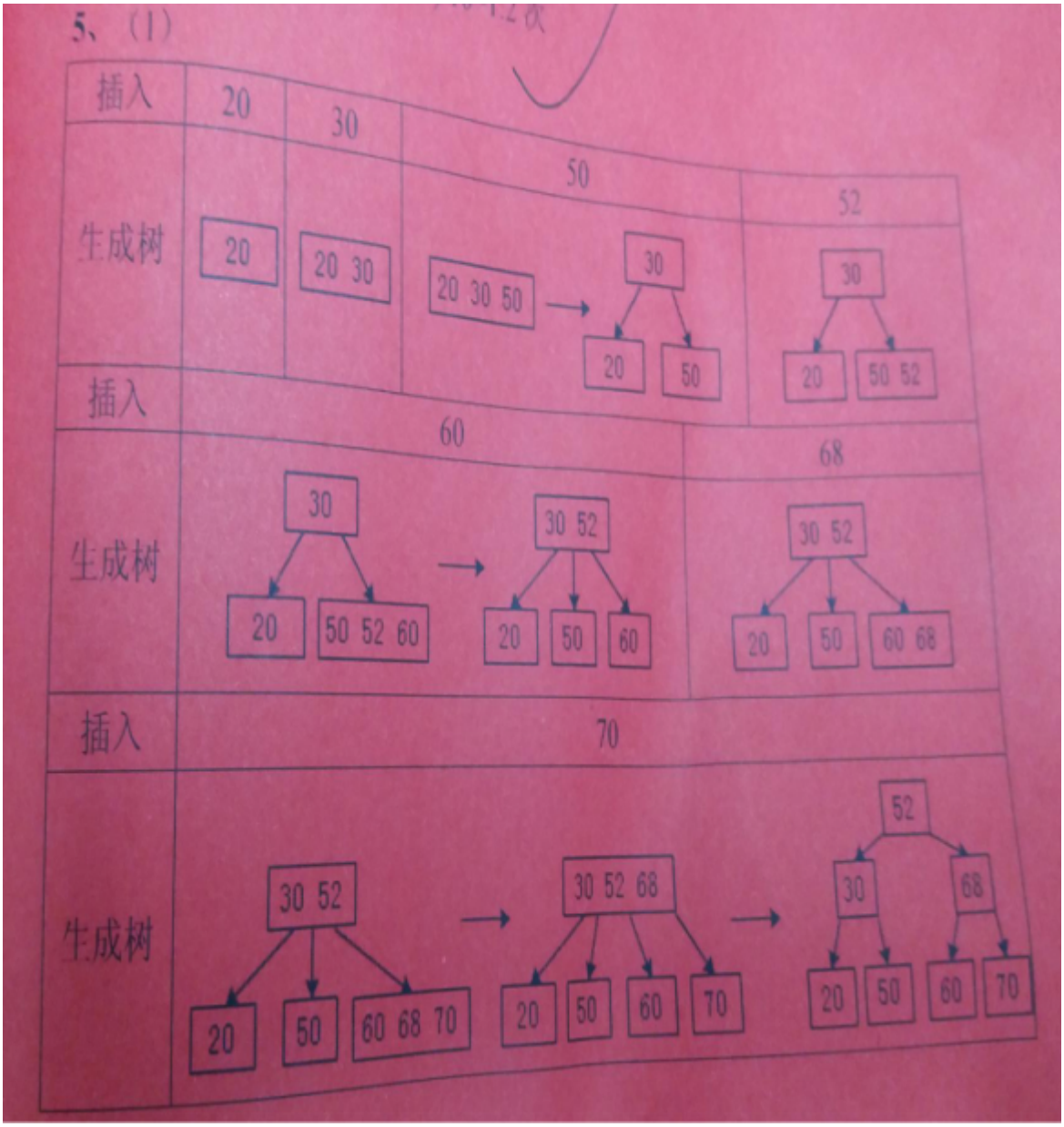


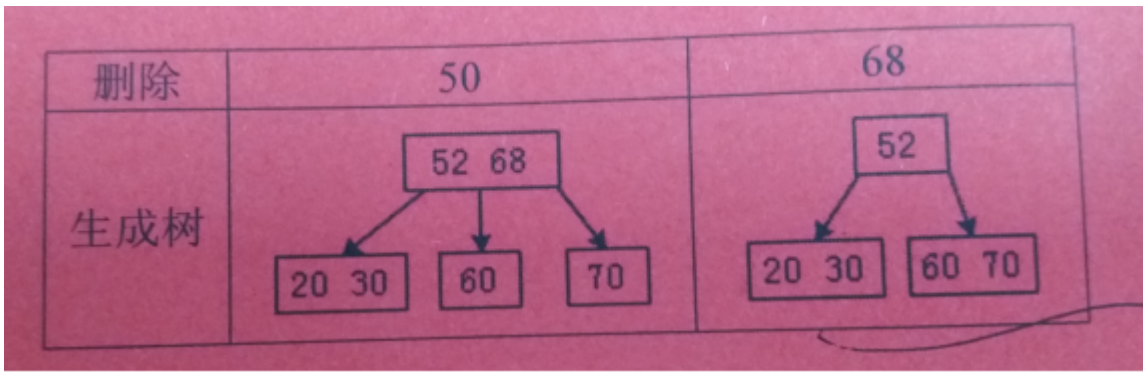


5、试从空树开始，画出按以下次序向 3 阶 B-树插入关键码的过程：20，30，50，52，60，68，70，

（1）写出每插入一个关键码生成 3 阶 B-树的过程；

（2）对（1）最终生成的 3 阶 B-树，依次删除 50、68，画出每一步执行后的树的状态。





哈希

在散列表中。插入、删除和查找都会用到散列函数。散列函数的计算速度直接影响散列表的性能。好的散列函数的一个标准就是：**计算速度快。**另一点就是：结点的**散列地址尽可能均匀。使得冲突的机会尽可能少。**

常用的散列函数包括直接定址法、保留余数法、数字分析法、平方取中法和折叠法等。

(1)直接定址法

直接取关键字的值或关键字的某个线性函数的值作为散列地址。设**关键字为x**，那么**散列地址**可表示为：

$H(x) = x$ 或 $H(x) = ax + b$ (a，b 为常数)

例如：关键字集合为{100, 400，600， 200， 800， 900}，利用直接定址法，若选取散列函数为H(x) = x/100,则散列表如下：

0	1	2	3	4	5	6	7	8	9
	100	200		400		600		800	900

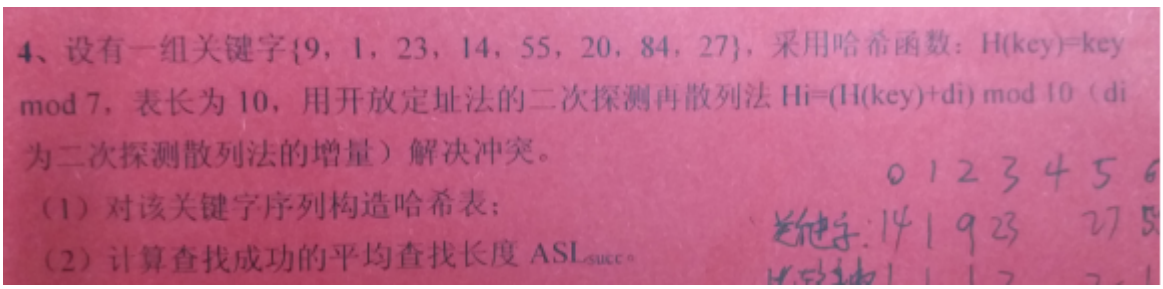
(2)保留余数法

如果M是散列表的大小，关键字 x 的数据元素的散列地址为：$H(x) = xmodM$ 在保留余数法中，选取合适的余数M很重要，如果选取不当，则导致大量的碰撞。

经验表明：M为素数（除了1和它本身以外不再有其他因数。）时，散列表的分布比较均匀。

在选取散列函数时，由于很难选取一个既均匀分布又简单，同时保证关键字和散列地址一一对应的散列表，所以冲突时不可避免的。如果具有不同关键字的 k 个数据元素的散列地址完全相同，就必须为 k-1个数据元素重新分配存储单元。通常称其为“溢出”的数据元素。在对溢出的数据元素进行处理时，通常使用的方法有：线性探测法，二次探测法等。

- **线性探测法**： 在数组中从映射到的位置开始顺序查找空位置，如果需要，从最后一个位置绕回第一个位置。这种方法碰撞可能引起连锁反应，使表中形成一些较长的连续被占用的单元，如：从1---10的地址全部被使用。从而使散列表的性能降低。
- **二次探测法**： 不直接检查下一个单元，而是检查远离初始探测点的某一单元，以消除线性探测法中的初始聚集的问题。



4、由题可知，H(key)=key mod 11，则有对所有线性表值取余如下表：

值	9	1	23	14	55	20	84	27
余数	2	1	2	0	6	6	0	6

（1）采用二次探测再散列法处理冲突，散列表：

散列值	0	1	2	3	4	5	6	7	8	9
key	14	1	9	23		27	55	20		84
比较次数	1	1	1	2		3	1	2		3

（2）平均查找长度 $ASL_{succ}=(4*1+2*2+2*3)/8=7/4$ 次

[注]二次探测再散列法 Hi=(H(key)+di) mod 10（di 为二次探测散列法的增量）的 mod 值为 10，其中增量为 1^2、-1^2、2^2、-2^2...，（-1）mod 10=9。

4、设哈希表的地址范围是 0~15，哈希函数为：H(key)=key mod 13，其中 key 为关键字，用线性探测再散列法处理冲突，输入关键字序列：

（23，11，6，30，44，17，20，21，15，39，62）

（1）构造哈希（Hash）表，画出其示意图；

（2）若查找关键字 17，需要依次与哪些关键字进行比较？

（3）假定每个关键字的查找概率相等，求查找成功时的平均查找长度 ASL_{succ}。

（1）采用线性探测再散列法处理冲突，散列表：

散列值	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
key	39		15		30	44	6	17	20	21	23	11	62			
次数	1		1		1	1	1	4	2	2	1	1	3			

（2）查找关键字 17，需要依次比较的关键字为：30、44、6。

（3）平均查找长度 $ASL_{succ}=(7*1+2*2+1*3+1*4)/11=18/11$。

4、假定一个待散列存储的线性表为（21，86，29，52，48，83，25，57，40，81），散列地址空间为 HT[11]，

（1）若采用除留余数法处理冲突，画出散列表并求出平均查找长度；

（2）若采用链地址法处理冲突，画出散列表并求出平均查找长度。

4、由题可知，H(key)=key mod 11，则有对所有线性表值取余如下表：

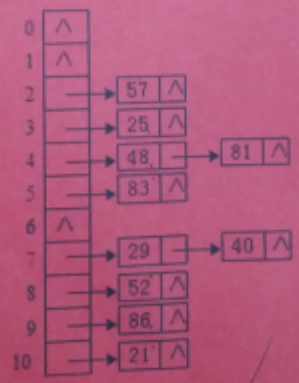
值	21	86	29	52	48	83	25	57	40	81
余数	10	9	7	8	4	6	3	2	7	4

（1）采用除留余数法处理冲突，散列表：

散列值	0	1	2	3	4	5	6	7	8	9	10
key	40		57	25	48	81	83	29	52	86	21
比较次数	5		1	1	1	2	1	1	1	1	1

平均查找长度 $ASL_1=(1*5+8*1+1*2)/10=1.5$ 次

（2）采用链地址法处理冲突，散列表：



平均查找长度 $ASL_2=(8*1+2*2)/10=1.2$ 次