<https://blog.csdn.net/yankai0219/article/details/8185847>

转载请注明出处<http://blog.csdn.net/yankai0219/article/details/8185847>

问题：

1. hash算法主表实现为什么不直接用数组,而使用malloc动态申请?

2. 另外每个桶的使用 线性队列 和 双向队列 以及 二级hash的区别以及好处是什么?

答案:

     1.

          1）hash表大小如果是固定的，当然可以采用数组；

          struct hash\_head  \*hash\_list[1024];

          2）如果hash大小没有确定，在程序中动态变化的，那就需要使用malloc动态申请。

|  |
| --- |
| int hash\_size = get\_hash\_size(); struct  hash\_head  \*\*hash\_list; \*hash = malloc(hash\_size); memset(\*hash, 0, hash\_size); |

     2.

          线性链表只能单相查找结点，即知道一个已知结点，不能找到它的前驱。

          双向链表可以向前向后二个方向查找结点。

          还有双向循环链表，我在项目中用的比较多。直接用内核里扒下来用的。

          二级hash我到觉的没什么必要。使结构太复杂。可以用一级hash来代替。设计好hash算法就行了。

问题：

3.

1）已知一个线性表(38，25，74，63，52，48)，采用的散列函数为H(Key)=Key%7，将元素散列到表长为7的哈希表中存储。若采用线性探测的开放定址法解决冲突，则在该散列表上进行等概率成功查找的平均查找长度为 \_\_\_\_ ；

2）若利用拉链法解决冲突，则在该散列表上进行等概率成功查找的平均查找长度为 \_\_\_\_；

答案:1）11/6 ，2）8/6.对于1）中，答案给出的是12/6=2，经过验证是错误的。

分析过程：

1）采用开放定址法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| key | 38 | 25 | 74 | 63 | 52 | 48 |
| 地址 | 3 | 4 | 4 | 0 | 3 | 6 |
| 解决冲突过程 |  |  | 5 |  | 4 5 6 | 7 |
| 查找次数 | 1 | 1 | 2 | 1 | 4 | 2 |

平均查找长度：（1\*3+2\*2+4）/6=11/6

2)采用拉链法

|  |  |  |
| --- | --- | --- |
| 拉链 | 链表中元素 | 查找次数 |
| 0 | 63 | 1 |
| 3 | 38   52 | 38（1次） 52（2次） |
| 4 | 25 74 | 25（1次） 74（2次） |
| 6 | 48 | 1 |

平均查找长度（1\*4+2\*2）/6=4/3

4.设一哈希表长为13，采用线性探测法解决冲突，哈希函数H(key)=key%13

1 ）画出在空表中依次插入关键字25.20.36.15.41.52.29.72.67后的哈希表

2 ）求在等概率情况下，查找成功和查找不成功的平均查找长度

答案：

1）Hash表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| hash表下标（地址） | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| hash表元素 | 52 |  | 15 | 41 | 29 | 67 |  | 20 | 72 |  | 36 |  | 25 |
| 解决冲突过程 |  |  |  | 2 3 | 3 4 | 2 3 4 5 |  |  | 7 8 |  |  |  |  |
| 查找次数 | 1 |  | 1 | 2 | 2 | 4 |  | 1 | 2 |  | 1 |  | 1 |

2）查找成功的平均查找长度：（1\*5 +2\*3 +4\*1）/9 = 15/9 。查找成功的平均查找长度为**SUM(查找次数)/SUM(元素个数)**

3)  查找失败的平均查找长度

计算长度为m的哈希表中装填有n个记录的查找不成功时的平均查找长度相当于计算在这张表中填入第n+1个记录时所需要的比较次数的期望值。  
第N+1个记录可能是0~12  
若 N+1个记录是0，由于位置0已经被占用，位置1没有占用，所以比较2次，依次类推。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| hash表下标 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| hash表元素 | 52 |  | 15 | 41 | 29 | 67 |  | 20 | 72 |  | 36 |  | 25 |
| 解决冲突过程 |  |  |  | 2 3 | 3 4 | 2 3 4 5 |  |  | 7 8 |  |  |  |  |
| 查找次数 | 1 |  | 1 | 2 | 2 | 4 |  | 1 | 2 |  | 1 |  | 1 |
| 查找不成功 | 2 | 1 | 5 | 4 | 3 | 2 | 1 | 3 | 2 | 1 | 2 | 1 | 3 |

故查找不成功的平均查找长度为（2+1+5+4+3+2+1+1+2+2+1+2+1+3）/13=30/13

5.

将关键字序列（7、8、30、11、18、9、14）散列存储到散列表中。散列表的存储空间是一个下标从0开始的一维数组，散列函数为：      H(key) = (keyx3) MOD 7，处理冲突采用线性探测再散列法，要求装填（载）因子为0.7。

(1) 请画出所构造的散列表。

(2) 分别计算等概率情况下查找成功和查找不成功的平均查找长度。

答案：

（1）

（2）**查找成功的平均查找长度**

(表3)

        所以ASLsuccess= （1+1+1+1+3+3+2）/ 7 = 12/7

**查找不成功的平均查找长度**

**计算查找不成功的次数就直接找关键字到第一个地址上关键字为空的距离即可， 但根据哈希函数地址为MOD7，因此初始只可能在0~6的位置**

    因此查找不成功的次数表如下表所示

(表4)

       所以ASLunsuccess= （3+2+1+2+1+5+4）/ 7 = 18/7。

参考文献  
<http://zhan.renren.com/astart?gid=3602888498031032593>

<http://shijuanfeng.blogbus.com/logs/172381700.html>

转载请注明出处<http://blog.csdn.net/yankai0219/article/details/8185847>