# theory6

## 1.Amortized Analysis

### (a)

在一轮循环中，所有情况(2^k)下的token消耗为

(2^k)\*(2^0) + [2^(k-1)]\*(2^1) + [2^(k-2)]\*(2^2) + ... + (2^1)\*[2^(k-1)]

= 2^k + 2^k + 2^k + ... + 2^k

= (2^k)\*k

因此，均摊消耗为k，这其中包含最坏的情况，因此，最坏情况下的平均token增量为k，故复杂度为O(n)。

### (b)

由于位0每次都需要flip，所以n次增加需要n次flip，共需要n个token。

由于位0每两次flip，会使得位1进行1次flip，需要的token数为1，所以n次增加需要⌊n/2⌋次flip，共需要⌊n/2⌋个token。

### (c)

O(kn) O(n)

## 2. A New Implementation of Queues

### (a)

bool queue\_empty(queue Q)

{

return stack\_empty(Q->instack) && stack\_empty(Q->outstack);

}

void enqueue(elem e, queue Q)

{

push(e, Q->instack);

}

elem dequeue(queue Q)

//@require !queue\_empty(Q);

{

if (stack\_empty(Q->outstack)) {

while (!stack\_empty(Q->instack)) {

push(pop(Q->instack), Q->outstack);

}

}

return pop(Q->outstack);

}

### (b)

enqueue:O(1)——每次入队，直接入栈Q->instack。

dequeue:O(q)——最坏的情况是Q->outstack为空，这时需要将Q->instack中所有的元素依次出栈并入栈到Q->outstack，最后将Q->outstack栈顶元素出栈。

### (c)

每次入队均是入栈Q->instack，进行一次操作，需要1个token，也即O（1）。

而每次出队时，均摊后需要3个token，也是O（1）。

# theory7

## Dealing with Collisions

### 背景

考虑使用三种实现方式来解决哈希表的冲突。

在第一个哈希表中，我们使用**链地址法**来解决冲突。如果哈希表的大小是m，那么关键字k将会被添加到h(k) mod m的索引处，在这里使用了函数h。为了解决冲突，所有索引相同的关键字被存到相同的链表中。

在第二个哈希表中，我们使用**线性探测再散列法**解决冲突。在线性探测中，如果关键字k被插入或者查找，当冲突发生时，顺序查看表中下一单元，直到找出一个空单元或查遍全表。

在第三个哈希表中，我们使用**二次探测再散列法**来解决冲突。与线性探测再散列法不同，每次我们顺序查找时的步长为i^2，直到找出一个空单元或查遍全表。

（a）

当哈希表大小为m，关键字数为2m，关键字不均匀分布时，在最坏的情况下，查找一个key的复杂度为O(2m)——当这2m个关键字的哈希函数值均相同而且要查找的关键字在链表尾端时。

当哈希表大小为m，关键字数为2m，关键字均匀分布时，在最坏的情况下，查找一个key的复杂度为O(2)——要查找的关键字在链表尾端时。

（b）

54, 23, 67, 88, 39, 75, 49, 5对应的索引分别为2，10，2，10，0，10，10，5

故对应链表如下

|  |  |
| --- | --- |
| 0 | 39 |
| 1 | NULL |
| 2 | 54 🡪 67 |
| 3 | NULL |
| 4 | NULL |
| 5 | 5 |
| 6 | NULL |
| 7 | NULL |
| 8 | NULL |
| 9 | NULL |
| 10 | 23🡪88🡪75🡪49 |
| 11 | NULL |
| 12 | NULL |

（c）

54, 23, 67, 88, 39, 75, 49, 5对应的索引分别为2，10，2，10，0，10，10，5

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 下标 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Key | 39 | 49 | 54 | 67 |  | 5 |  |  |  |  | 23 | 88 | 75 |

（d）

54, 23, 67, 88, 39, 75, 49, 5对应的索引分别为2，10，2，10，0，10，10，5

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 下标 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Key | 39 | 75 | 54 | 67 |  | 5 | 49 |  |  |  | 23 | 88 |  |

（e）

二次探测再散列法会遭遇一个线性探测再散列法不会遇到的问题。当向一个非空的哈希表中添加值的时候，使用二次探测再散列法可能永远不会找到一个位置插入。以h(k)=k为哈希函数，以6为哈希表的大小，给出这样一个例子。

假设表中已经有36,78,12,90四个值，也即

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 下标 | 0 | 1 | 2 | 3 | 4 | 5 |
| Key | 36 | 78 |  | 90 | 12 |  |

此时再次加入关键字时，应从5^2开始，而(5^2)%6=1，无法插入，依次有(6^2)%6=0，(7^2)%6=1，(8^2)%6=4，(9^2)%6=3，而要想能够插入新值，应有(n^2)%6=2或者(n^2)%6=5，也即求同余式x^2≡2（mod 6）与x^2≡5（mod 6）的解，实际上这是无解的。因此该散列法失败。

## Strings as Keys

在某门语言中，字符串是以以下方式进行哈希存储的，其中s[i]代表第i个字符的ASCII码，p代表字符串长度，m为哈希表长度。

（a）

如果使用链地址法在容量为3021的哈希表中存放15105个字符串，在均匀分布的情况下，其装载因子为5，此时装载因子代表查找某个字符串的复杂度为O（5）

（b）

构造两个字符串如下