计算机问题求解 – 论题2-10

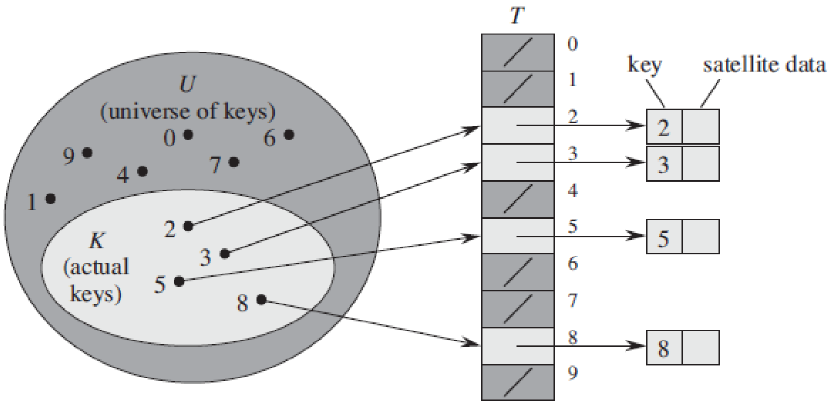
- Hashing方法

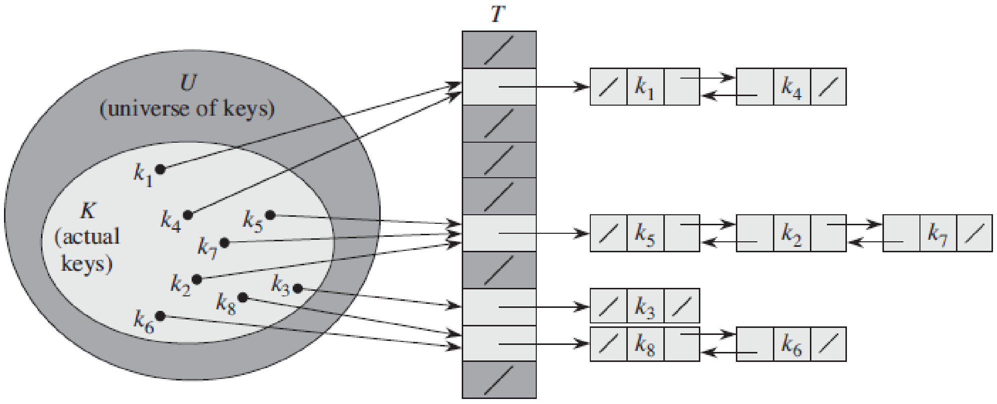
课程研讨

- TC第11章
- CS第5章第5节

问题1：dictionary

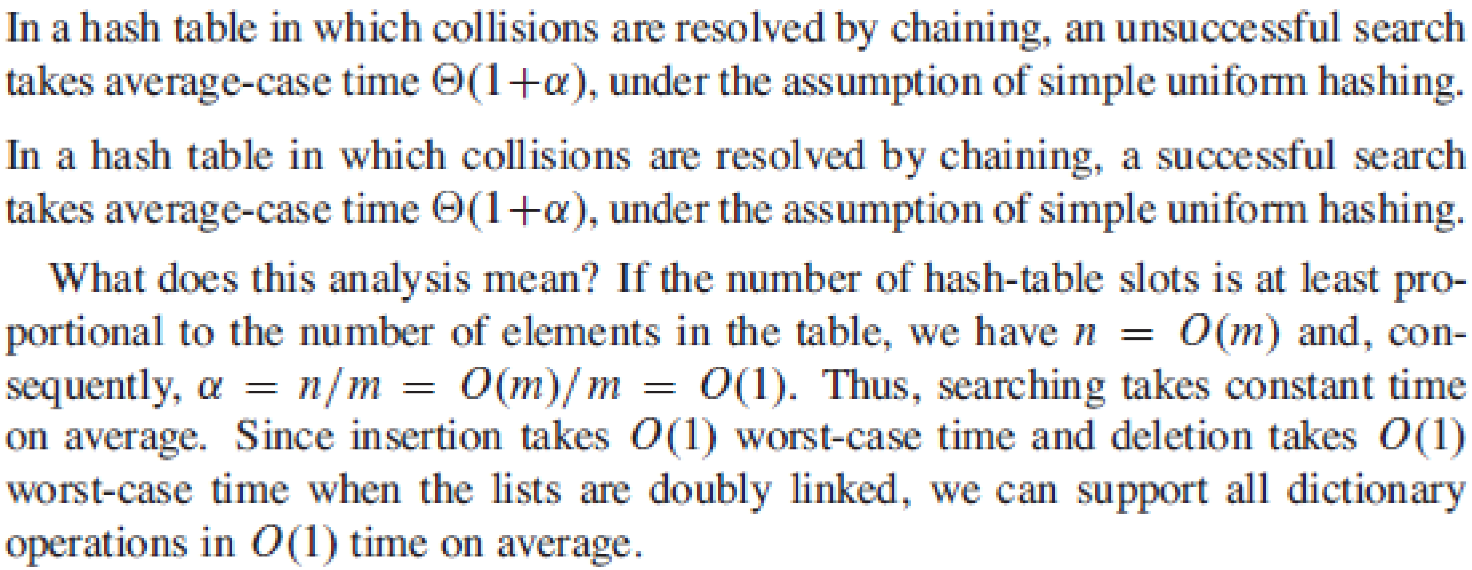
- 什么是dictionary？
- 你如何理解它的两种实现？
  - direct-address table
  - hash table
- 你能分析它们的存储空间和插入/删除/查找时间吗？
- 因此，你能对比它们的优缺点吗？





问题1： dictionary (续)

- 你理解这段话了吗?



In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

In a hash table in which collisions are resolved by chaining, a successful search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

What does this analysis mean? If the number of hash-table slots is at least proportional to the number of elements in the table, we have $n = O(m)$ and, consequently, $\alpha = n/m = O(m)/m = O(1)$. Thus, searching takes constant time on average. Since insertion takes $O(1)$ worst-case time and deletion takes $O(1)$ worst-case time when the lists are doubly linked, we can support all dictionary operations in $O(1)$ time on average.

- 对于dynamic set，如何做到那个“if”？

问题1： dictionary (续)

```
void addEntry(int hash, K key, V value, int bucketIndex) {
    if ((size >= threshold) && (null != table[bucketIndex])) {
        resize(2 * table.length);
        hash = (null != key) ? hash(key) : 0;
        bucketIndex = indexFor(hash, table.length);
    }

    createEntry(hash, key, value, bucketIndex);
}
```

Worst-case Analysis of the Insertion

- For *n* execution of insertion operations
  - A bad analysis: the worst case for one insertion is the case when expansion is required, up to *n*
  - So, the worst case cost is in $O(n^2)$.

Of course NOT!

- Note the expansion is required during the *i*th operation only if $i=2^k$, and the cost of the *i*th operation

$$c_i = \begin{cases} i & \text{if } i-1 \text{ is exactly power of } 2 \\ 1 & \text{otherwise} \end{cases}$$

$$\text{So, the total cost is: } \sum_{i=1}^{n} c_i \le n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j < n + 2n = 3n$$

问题2： hash function

- 你如何理解一个好的hash function应有的这些要素？
  - Satisfies (approximately) the assumption of simple uniform hashing.
  - Derives the hash value in a way that we expect to be independent of any patterns that might exist in the data.

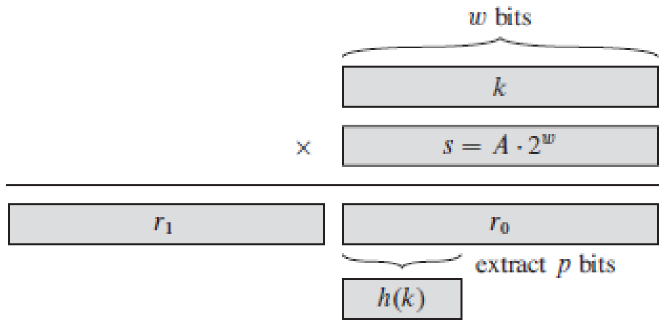
- 你如何理解simple uniform hashing？
  它对hash table为什么至关重要？

问题2： hash function (续)

- 你理解这两种hash function了吗？

$$h(k) = k \bmod m$$

$$h(k) = \lfloor m\,(kA \bmod 1)\rfloor$$



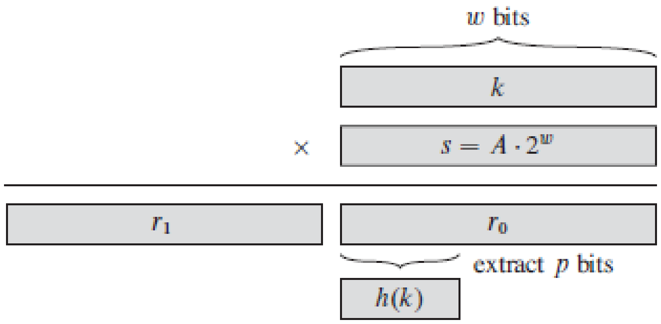
- 这些hash function在实际中能确保是simple uniform hashing吗？
如果不能，可能的原因是什么？如何解决？

问题2： hash function (续)

- 你理解这两种hash function了吗？

$$h(k) = k \bmod m$$

$$h(k) = \lfloor m\,(k A \bmod 1) \rfloor$$



- 这些hash function在实际中能确保是simple uniform hashing吗？
  如果不能，可能的原因是什么？如何解决？
  - universal hashing: to choose the hash function randomly in a way that is independent of the keys that are actually going to be stored

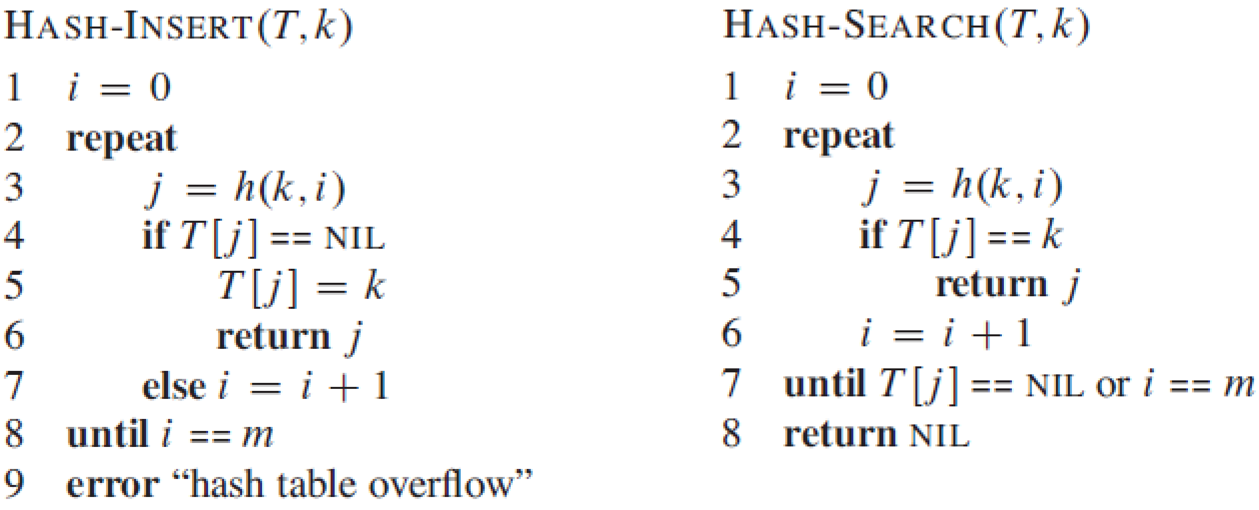
问题3：probability calculations in hashing

- 你会计算这些期望值吗?
  - expected number of items per location n/k
  - expected number of empty locations $k(1-\frac{1}{k})^n$
  - expected number of collisions $n-k+k(1-\frac{1}{k})^n$
  - expected time until all locations have at least one item

$$\sum_{j=1}^{k}\frac{k}{k-j+1}$$

问题4：collision resolution

- 你理解open addressing了吗？
  它与chaining的本质区别是什么？
  因此，它有哪些相对的优缺点？



```
HASH-INSERT($T, k$)
$i = 0$
repeat
    $j = h(k, i)$
    if $T[j] == \text{NIL}$
        $T[j] = k$
        return $j$
    else $i = i + 1$
until $i == m$
error "hash table overflow"
```

```
HASH-SEARCH($T, k$)
$i = 0$
repeat
    $j = h(k, i)$
    if $T[j] == k$
        return $j$
    $i = i + 1$
until $T[j] == \text{NIL}$ or $i == m$
return NIL
```

开地址散列

- 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开始的一个一维数组散列，函数为：H(key)=(key * 3)MOD T，处理冲突采用线性探测再散列法，要求装载因子为0.7。问题：
  - 请画出所构造的散列表。
  - 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。

问题4： collision resolution (续)

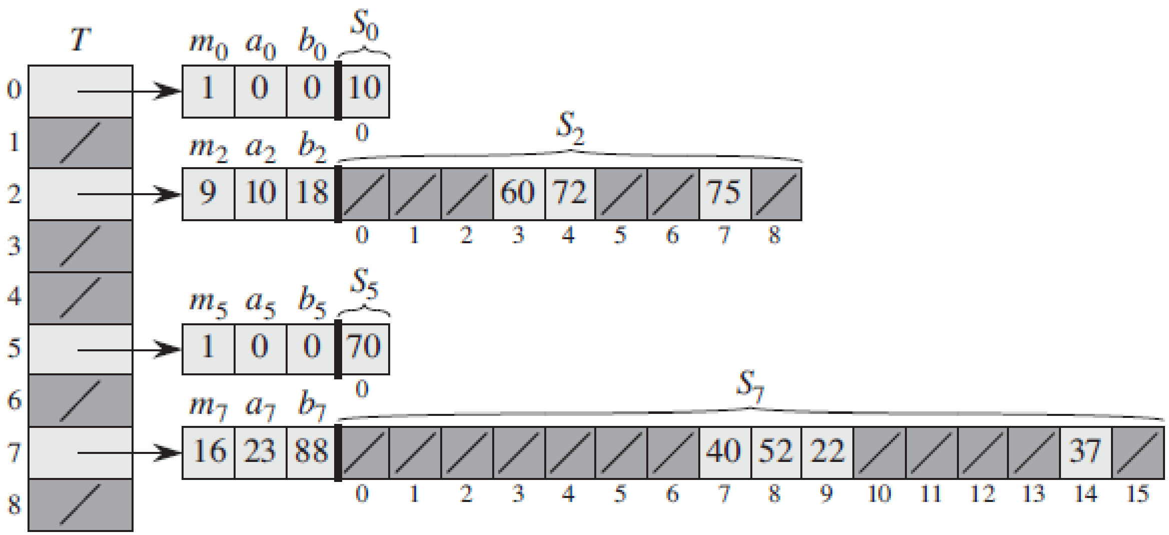
- 一个好的h函数应该具有哪些特点？
  -
  -
- 你理解这些h函数了吗？它们为什么不是最好的h函数？
  - linear probing $h(k,i) = (h'(k) + i) \bmod m$
  - quadratic probing $h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$
  - double hashing $h(k,i) = (h_1(k) + i h_2(k)) \bmod m$
- 你理解这些具体原因了吗？
  - linear probing: primary clustering
  - quadratic probing: secondary clustering

问题4： collision resolution (续)

- 一个好的h函数应该具有哪些特点？
  - The probe sequence is a permutation of <0, 1, …, m-1>.
  - uniform hashing: The probe sequence of each key is equally likely to be any of the m! permutations of <0, 1, …, m-1>.
- 你理解这些h函数了吗？ 它们为什么不是最好的h函数？
  - linear probing $h(k,i) = (h'(k) + i) \bmod m$
  - quadratic probing $h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$
  - double hashing $h(k,i) = (h_1(k) + i h_2(k)) \bmod m$
- 你理解这些具体原因了吗？
  - linear probing: primary clustering
  - quadratic probing: secondary clustering

问题4：collision resolution (续)

- 你理解perfect hashing了吗？
  它与chaining的本质区别是什么？
  因此，它有哪些相对的优缺点？



T
0
1
2
3
4
5
6
7
8
m_0 a_0 b_0 S_0
1 0 0 10
0
m_2 a_2 b_2 S_2
9 10 18
60 72 75
0 1 2 3 4 5 6 7 8
m_5 a_5 b_5 S_5
1 0 0 70
0
m_7 a_7 b_7 S_7
16 23 88
40 52 22 37
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15