计算机问题求解 – 论题2-10

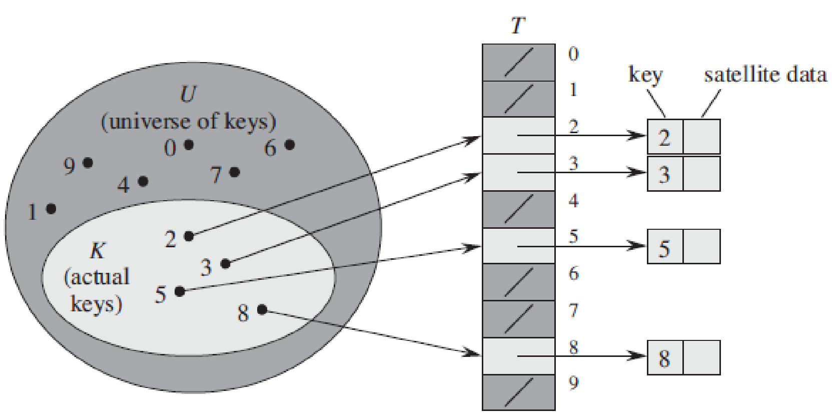
- Hashing方法

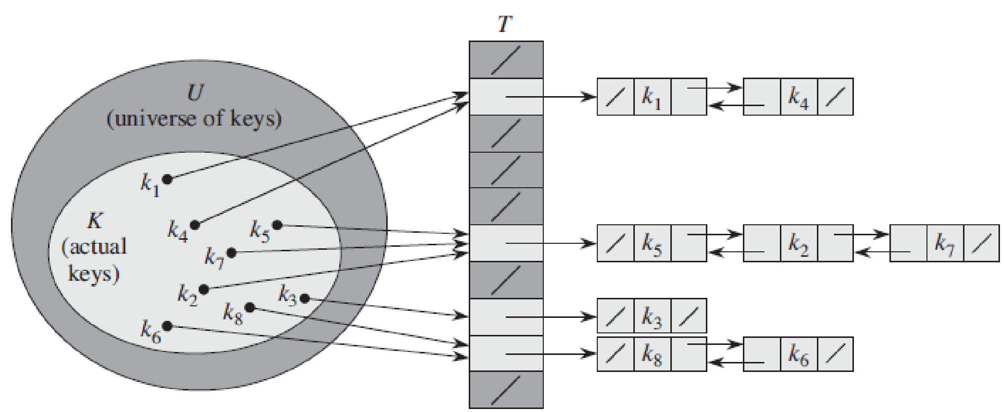
课程研讨

- 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 $O(n^2)$.
- Note the expansion is required during the ith operation only if $i=2^k$, and the cost of the ith operation

Of course NOT!

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

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

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.

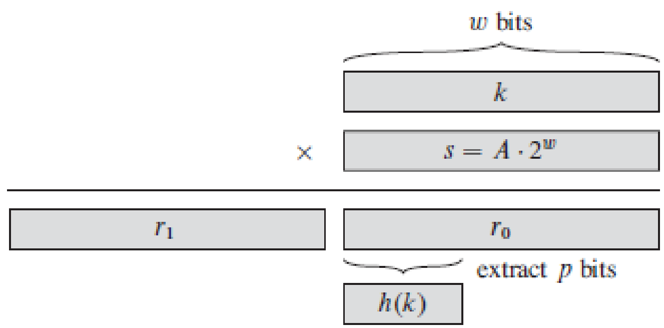
- 你如何理解simple uniform hashing？
  它对hash table为什么至关重要？

问题2： hash function (续)

- 你理解这两种hash function了吗？

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

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



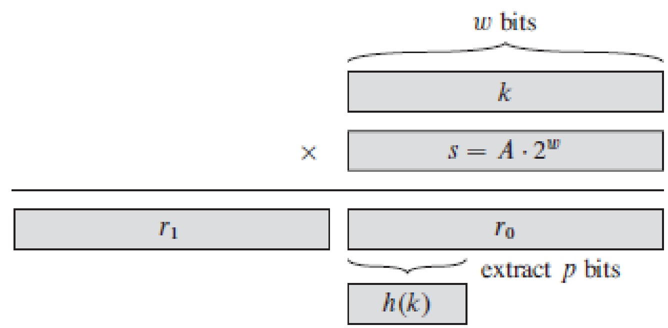
- 这些hash function在实际中能确保是simple uniform hashing吗？
  如果不能，可能的原因是什么？如何解决？
-

问题2： hash function (续)

- 你理解这两种hash function了吗？

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

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



- 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？
  - 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：闭地址散列

- 采用Hashing by Chaining 计算成功搜索与不成功搜索的代价有什么不同?

- 什么是简单一致hash？

- 不成功检索的代价？
  - $\Theta(1+\alpha)$

Hashing by Chaining: 成功搜索

- For successful search: (assuming that x_i is the ith element inserted into the table, i=1,2,...,n)
  - For each i, the probability of that x_i is searched is $1/n$.
  - For a specific x_i, the number of elements examined in a successful search is t+1, where t is the number of elements inserted into the same list as x_i, after x_i has been inserted. And for any j, the probability of that x_j is inserted into the same list of x_i is $1/m$. So, the cost is:

$$\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\frac{1}{m}\right)$$

Cost for computing hashing

Expected number of elements in front of the searched one in the same linked list.

Hashing by Chaining: 成功搜索

- The average cost of a successful search:
  - Define α=*n*/*k* as ***load factor***,

The average cost of a successful search is :

$$\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\frac{1}{m}\right)=1+\frac{1}{nm}\sum_{i=1}^{n}(n-i)=1+\frac{1}{nm}\sum_{i=1}^{n-1}i$$

$$=1+\frac{n-1}{2m}=1+\frac{\alpha}{2}-\frac{\alpha}{2n}=\Theta(1+\alpha)$$

Cost for computing hashing

Number of elements in front of the searched one in the same linked list.

问题4：开地址散列

- 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] == 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] == NIL or i == m
return NIL
```

不成功查找的平均代价

- 不成功查找的代价：1/(1-α) (α=*n*/*m*<1)

define the random variable X to be the number of probes made in an unsuccessful search, and let us also define the event A_i, for $i = 1, 2, \ldots$, to be the event that an ith probe occurs and it is to an occupied slot. Then the event $\{X \geq i\}$ is the intersection of events $A_1 \cap A_2 \cap \cdots \cap A_{i-1}$. We will bound $\Pr\{X \geq i\}$ by bounding $\Pr\{A_1 \cap A_2 \cap \cdots \cap A_{i-1}\}$.

$$\begin{aligned}
\Pr\{X \geq i\} &= \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdots \frac{n-i+2}{m-i+2} \\
&\leq \left(\frac{n}{m}\right)^{i-1} \\
&= \alpha^{i-1} .
\end{aligned}$$

$$\begin{aligned}
\mathrm{E}[X] &= \sum_{i=0}^{\infty} i \cdot \Pr\{X = i\} \\
&= \sum_{i=0}^{\infty} i(\Pr\{X \geq i\} - \Pr\{X \geq i+1\}) \\
&= \sum_{i=1}^{\infty} \Pr\{X \geq i\} .
\end{aligned}$$

$$\mathrm{E}[X] = \sum_{i=1}^{\infty} \Pr\{X \geq i\} \leq \sum_{i=1}^{\infty} \alpha^{i-1} = \frac{1}{1-\alpha} .$$

成功查找的平均代价

- 成功查找的代价：$\dfrac{1}{\alpha}\ln\dfrac{1}{1-\alpha}$

To search for the $(i+1)$th inserted element in the table,

the cost is the same as the cost for inserting it when there

are just i elements in the table. At that time, $\alpha = \dfrac{i}{m}$, so,

the cost is $\dfrac{1}{1\text{-}\dfrac{i}{m}} = \dfrac{m}{m-i}$

For your reference:

Half full: 1.387; 90% full: 2.559

So, the cost is :

$$\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i} = \frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i} = \frac{1}{\alpha}\sum_{i=m-n+1}^{m}\frac{1}{i} \le \frac{1}{\alpha}\int_{m-n}^{m}\frac{dx}{x} = \frac{1}{\alpha}\ln\frac{m}{m-n} = \frac{1}{\alpha}\ln\frac{1}{1-\alpha}$$

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