A hash function consists of:

1. hash function $h(k)$
2. array
3. collision resolution strategies

SUHA: Simple Uniform Hashing Assumption，意思大概是均匀分布，定义为$\mathrm{P}(\mathrm{h}(a)=\mathrm{h}(b))=\frac{1}{m}$

一个好的哈希函数满足：

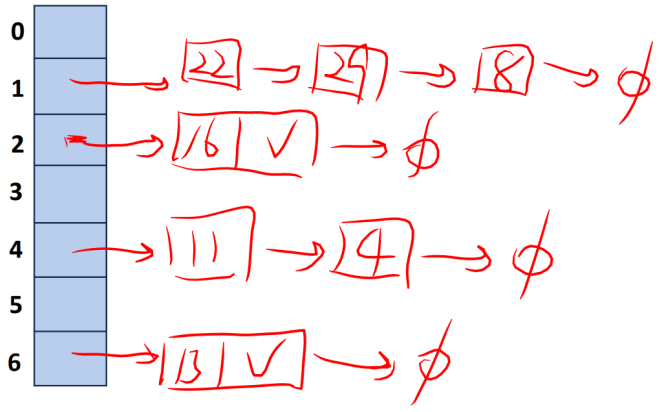
- 常数时间（$\mathrm{O}(1)$）
- 确定性（如果$k_1==k_2 \implies h(k_1)==h(k_2)$
- SUHA

Collision Handlin

Separate Chaining

使用链表来存储

假设 $S=\{16,8,4,13,29,11,22\}, h(k)=k\%7$

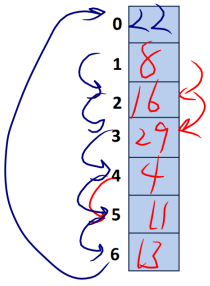


load factor $\alpha=\frac{n}{N}$

注意到会有空位没用（unbounded），所以这是一个 open hashing

Probe-based Hashing

假设 $S = \{16, 8, 4, 13, 29, 11, 22\}, h(k) = k\%7$

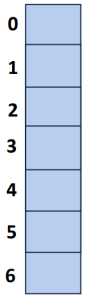


Try h(k) = (k + 0) % 7, if full...
Try h(k) = (k + 1) % 7, if full...
Try h(k) = (k + 2) % 7, if full...
Try ...

缺点：即使负载低也很容易拥挤，导致寻址变慢

Double Hashing

S = { 16, 8, 4, 13, 29, 11, 22 } |S| = n
$h_1(k)$ = k % 7 |Array| = N
$h_2(k)$ = 5 - k % 5



Try h(k) = (k + 0*$h_2(k)$) % 7, if full...
Try h(k) = (k + 1*$h_2(k)$) % 7, if full...
Try h(k) = (k + 2*$h_2(k)$) % 7, if full...
Try ...

h(k, i) = ($h_1(k)$ + i*$h_2(k)$) % 7

时间（不用记）

下面是SUHA条件下找到一个key的期望寻找次数

Linear Probing

- Successful: $\frac{1}{2}\left(1+\frac{1}{1-\alpha}\right)$
- Unsuccessful: $\frac{1}{2}\left(1+\frac{1}{1-\alpha}\right)^2$

Double Hashing

- Successful: $\frac{1}{\alpha}\times\ln\left(\frac{1}{1-\alpha}\right)$
- Unsuccessful: $\frac{1}{1-\alpha}$

Separate Chaining

- Successful: $1+\frac{\alpha}{2}$
- Unsuccessful: $1+\alpha$

重哈希

从上面知道，α大的时候哈希表会速度会退化，所以在α达到一定值我们需要进行重哈希。

我们一般吧哈希表大小扩大到原来的2倍，然后重新进行哈希映射

复杂度

时间复杂度

- 查找：
  - 平均：$O(1)$
  - 最差：$O(n)$
- 插入：
  - 平均：$O(1)$
  - 最差：$O(n)$

空间复杂度

$O(n)$

$N \approx 1.5n$

几个std ADS

std::map

是红黑树实现的

方法:

- `operator[]`
- `insert`
- `erase`
- `lower_bound(key)`: Iterator to the first element <= key
- `upper_bound(key)`: Iterator to the first element > key

std::unordered_map

方法

- `operator[]`
- `insert`
- `erase`
- `load_factor()`
- `max_load_factor(ml)`: 设置最大load factor