



# 11.Hash Tables

---

Hsu, Lih-Hsing

Computer Theory Lab.

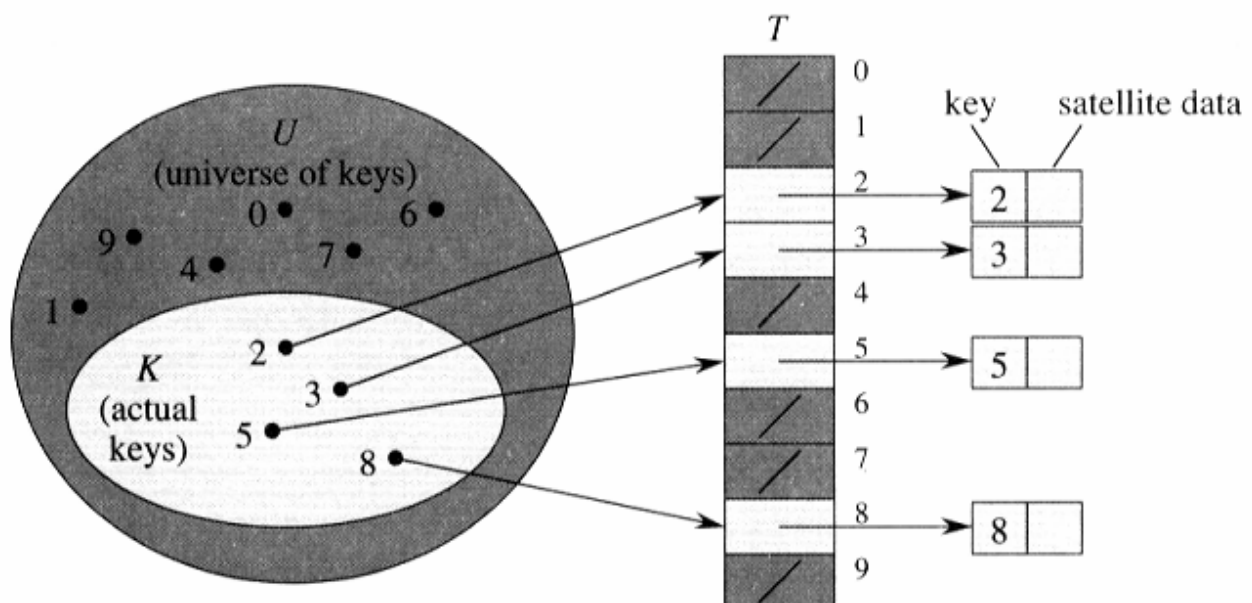


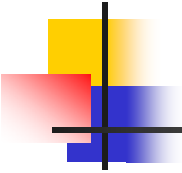
## 11.1 Directed-address tables

---

- Direct addressing is a simple technique that works well when the universe  $U$  of keys is reasonable small. Suppose that an application needs a dynamic set in which an element has a key drawn from the universe  $U=\{0,1,\dots,m-1\}$  where  $m$  is not too large. We shall assume that no two elements have the same key.

- To represent the dynamic set, we use an array, or directed-address table,  $T[0..m-1]$ , in which each position, or slot, corresponds to a key in the universe  $U$ .

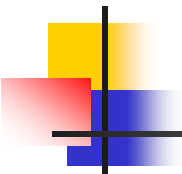




DIRECTED\_ADDRESS\_SEARCH( $T, k$ )  
**return**  $T[k]$

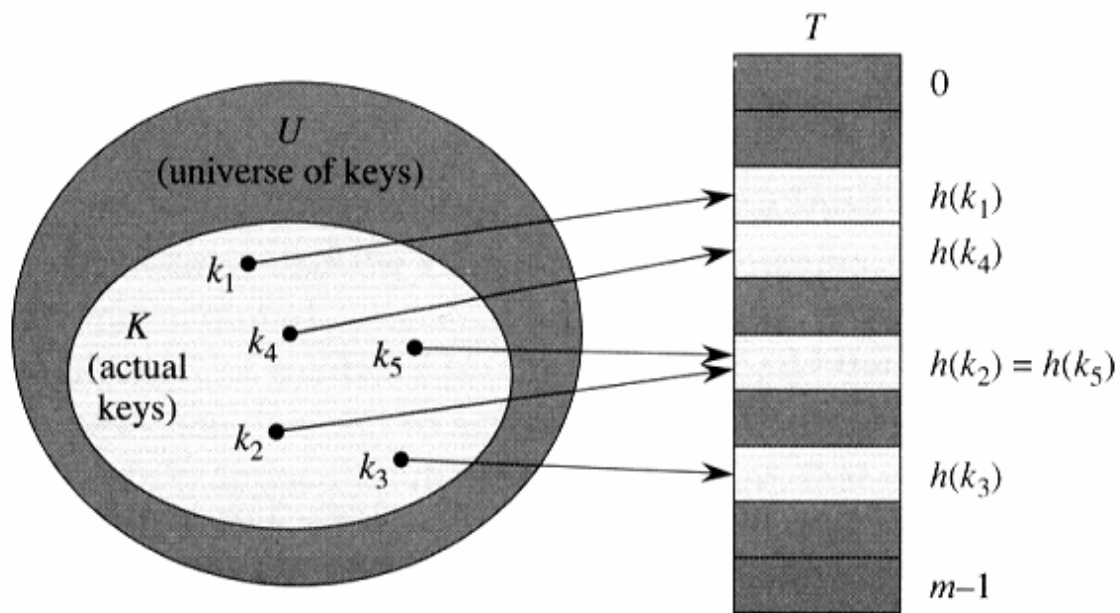
DIRECTED\_ADDRESS\_INSERT( $T, x$ )  
 $T[key[x]] \leftarrow x$

DIRECTED-ADDRESS\_DELETE( $T, x$ )  
 $T[key[x]] \leftarrow nil$



## 11.2 Hash tables

- The difficulty with direct address is obvious: if the universe  $U$  is large, storing a table  $T$  of size  $|U|$  may be impractical, or even impossible. Furthermore, the set  $K$  of keys actually stored may be so small relative to  $U$ . Specifically, the storage requirements can be reduced to  $O(|K|)$ , even though searching for an element in the hash table still requires only  $O(1)$  time.



- hash function:  $h:U \rightarrow \{0,1,\dots,m-1\}$
- hash table:  $T[0..m-1]$
- $k$  hashes to slot:  $h(k)$  hash value
- collision: two keys hash to the same slot



## Collision resolution technique:

---

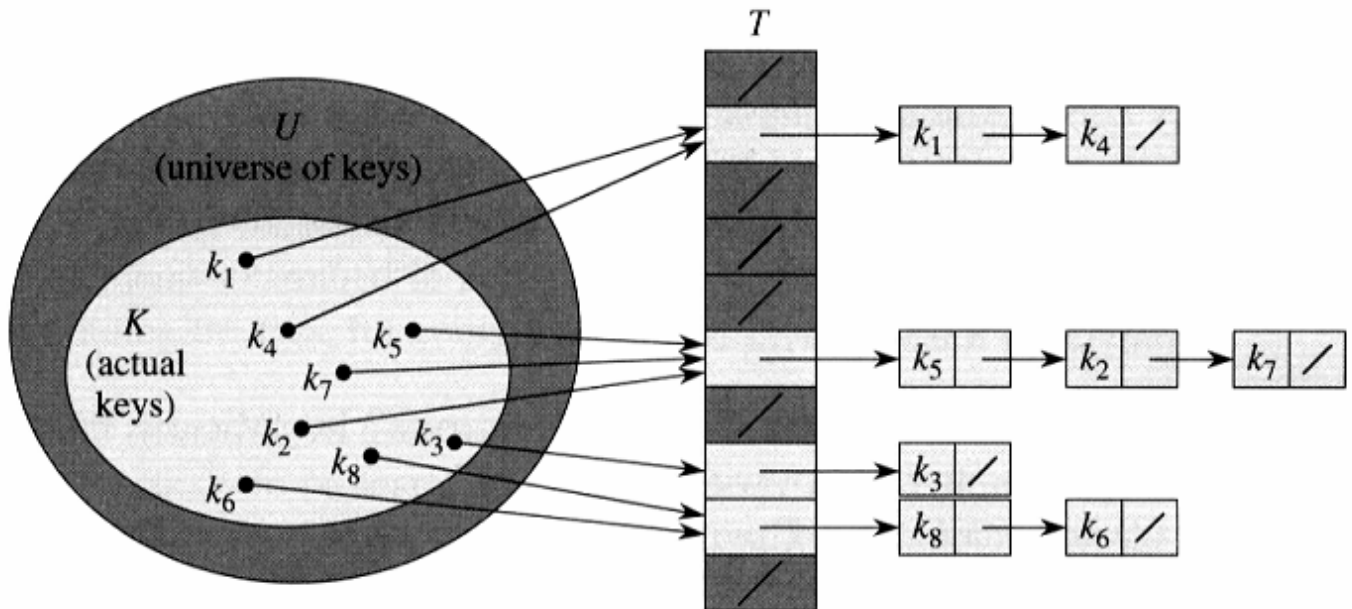
- chaining
- open addressing



## Collision resolution by chaining:

---

- In chaining, we put all the elements that hash to the same slot in a linked list.

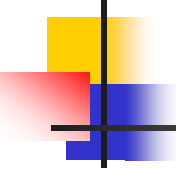


## ■ CHAINED\_HASH\_INSERT( $T, x$ )

Insert  $x$  at the head of the list  $T[h[key[x]]]$

## ■ CHAINED\_HASH\_SEARCH( $T, k$ )

Search for the element with key  $k$  in the list  $T[h[k]]$

- 
- **CHAINED\_HASH\_DELETE( $T, x$ )**  
delete  $x$  from the list  $T[h[key[x]]]$
- Complexity:**

INSERT     $O(1)$

DELETE     $O(1)$  if the list  
              are doubly linked.



## Analysis of hashing with chaining

- Given a hash table  $T$  with  $m$  slots that stores  $n$  elements.
- load factor:  $\alpha = \frac{n}{m}$   
(the average number of elements stored in a chain.)

## Assumption: *simple uniform hashing*

- uniform distribution, hashing function takes  $O(1)$  time.

for  $j = 0, 1, \dots, m-1$ , let us denote the length of the list  $T[j]$  by  $n_j$ , so that

$$n = n_0 + n_1 + \dots + n_{m-1},$$

and the average value of  $n_j$  is  $E[n_j] = \alpha = n/m$ .

## Theorem 11.1.

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





## Proof.

---

- The average length of the list is  $\alpha = \frac{n}{m}$ .
- The expected number of elements examined in an unsuccessful search is  $\alpha$ .
- The total time required (including the time for computing  $h(k)$ ) is  $O(1 + \alpha)$ .



## Theorem 11.2


---

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



# Proof.

- Assume the key being searched is equally likely to be any of the  $n$  keys stored in the table.
- Assume that CHAINED\_HASH\_INSERT procedure insert a new element at the end of the list instead of the front.



$$\begin{aligned}
 E\left[\frac{1}{n}\sum_{i=1}^n\left(1+\sum_{j=i+1}^n X_{ij}\right)\right] &= \frac{1}{n}\sum_{i=1}^n\left(1+\sum_{j=i+1}^n E[X_{ij}]\right) \\
 &= \frac{1}{n}\sum_{i=1}^n\left(1+\sum_{j=i+1}^n \frac{1}{m}\right) \\
 &= 1 + \frac{1}{nm}\sum_{i=1}^n(n-i) \\
 &= 1 + \frac{1}{nm}\left(\sum_{i=1}^n n - \sum_{i=1}^n i\right) \\
 &= 1 + \frac{1}{nm}\left(n^2 - \frac{n(n+1)}{2}\right) \\
 &= 1 + \frac{\alpha}{2} - \frac{\alpha}{2n}.
 \end{aligned}$$

**Total time required for a successful search**

$$\Theta\left(2 + \frac{\alpha}{2} - \frac{\alpha}{2n}\right) = \Theta(1 + \alpha).$$



## 11.3 Hash functions

- What makes a good hash function?

$$\sum_{k:h(k)=j} p(k) = \frac{1}{m} \quad \text{for } j = 1, 2, \dots, m$$



## Example:

- Assume  $0 \leq k \leq 1$  .
- Set  $h(k) = \lfloor km \rfloor$  .



## Interpreting keys as natural number

- ASCII code

$$(p, t) = (112, 116) = 112 \times 128 + 116 = 14452$$



### ***11.3.1 The division method***

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

- **Suggestion:** Choose  $m$  to be prime and not too close to exactly power of 2.

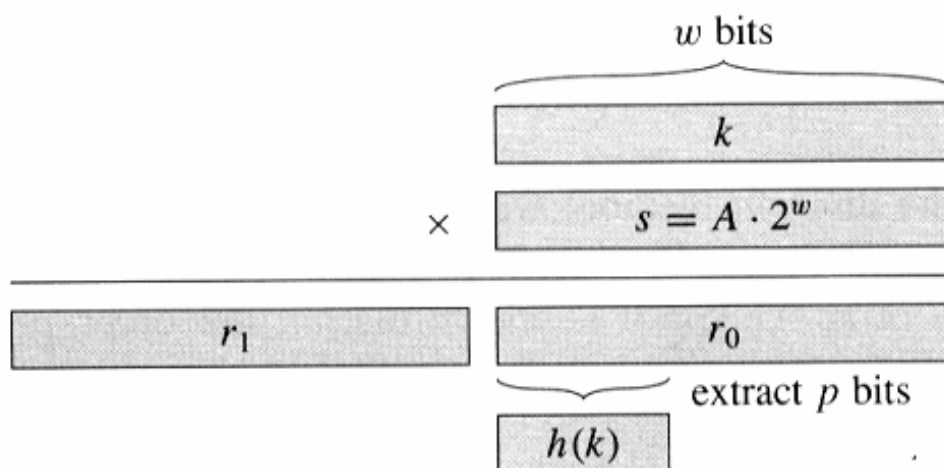
## 11.3.2 *The multiplication method*

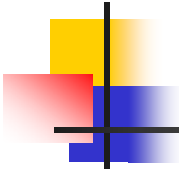
$$h(k) = \lfloor m(kA \bmod 1) \rfloor$$

$$\text{where } kA \bmod 1 = kA - \lfloor kA \rfloor$$

**Suggestion:**

choose  $m = 2^p, A = \frac{\sqrt{5}-1}{2}$





## Example:

$$k = 123456, p = 14, m = 2^{14} = 26384$$

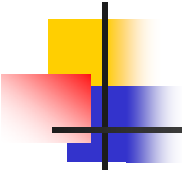
$$A = \frac{\sqrt{5} - 1}{2} \cong 0.61803\dots$$

$$\begin{aligned} h(k) &= \lfloor 10000 \times (123456 \times A \bmod 1) \rfloor \\ &= \lfloor 10000 \times 0.0041151 \rfloor \\ \lfloor 41.151 \rfloor &= 41 \end{aligned}$$

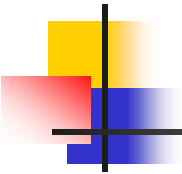


## 11.3.3 Universal hashing

- Choose the hash function randomly in a way that is independent of the keys that actually going to be stored.



- Let  $\mathcal{H}$  be a finite collection of hashing functions that maps a give universe  $U$  of keys into the range  $\{0, 1, 2, \dots, m-1\}$ . Such a collection is said to be **universal** if each pair of distinct keys  $k, l \in U$ , the number of hash functions  $h \in \mathcal{H}$  for which  $h(k) = h(l)$  is at most  $|\mathcal{H}|/m$ .



- In other words, with a hash function randomly chosen from  $\mathcal{H}$ , the chance of collision between distinct keys  $k$  and  $l$  is no more then the chance  $1/m$  of a collision if  $h(k)$  and  $h(l)$  were randomly and independently chosen from the set  $\{0, 1, \dots, m-1\}$ .



## Theorem 11.3

- Suppose that a hash function  $h$  is chosen from a universal collection of hash functions and is used to hash  $n$  keys into a table  $T$  of size  $m$ , using chaining to resolve collisions. If key  $k$  is not in the table, then the expected length  $E[n_{h(k)}]$  of the list that key  $k$  hashes to is at most  $\alpha$ . If key  $k$  is in the table, then the expected length  $E[n_{h(k)}]$  of the list containing key  $k$  is at most  $1 + \alpha$ .



## *Proof.*

- For each pair  $k$  and  $l$  of distinct keys, define the indicator random variable

$$X_{kl} = I\{h(k) = h(l)\}.$$

$$\Pr\{h(k)=h(l)\} \leq 1/m$$

$$E[X_{kl}] \leq 1/m$$



We define  $Y_k = \sum_{\substack{l \in T \\ l \neq k}} X_{kl}$ .

Thus we have

$$\begin{aligned} E[Y_k] &= E\left[\sum_{\substack{l \in T \\ l \neq k}} X_{kl}\right] \\ &= \sum_{\substack{l \in T \\ l \neq k}} E[X_{kl}] \\ &\leq \sum_{\substack{l \in T \\ l \neq k}} \frac{1}{m} \end{aligned}$$

- If  $k \notin T$ , then  $n_{h(k)} = Y_k$  and  $|\{l: l \in T \text{ and } l \neq k\}| = n$ . Thus  $E[n_{h(k)}] = E[Y_k] \leq n/m = \alpha$ .
- If  $k \in T$ , then because key  $k$  appears in list  $T[h(k)]$  and the count  $Y_k$  does not include key  $k$ , we have  $n_{h(k)} = Y_k + 1$  and  $|\{l: l \in T \text{ and } l \neq k\}| = n - 1$ . Thus  $E[n_{h(k)}] = E[Y_k] + 1 \leq (n - 1) / m + 1 = 1 + \alpha - 1/m < 1 + \alpha$ .



## Corollary 11.4

- Using universal hashing and collision resolution by chaining in a table with  $m$  slots, it takes expected time  $\Theta(n)$  to handle any sequence of  $n$  INSERT, SEARCH and DELETE operations containing  $O(m)$  INSERT operations.



## *Proof.*

Since the number of insertions is  $O(m)$ , we have  $n = O(m)$  and so  $\alpha = O(1)$ . The INSERT and DELETE operations take constant time and, by *Theorem 11.3*, the expected time for each SEARCH operation is  $O(1)$ . BY linearity of expectation, therefore, the expected time for the entire sequence of operations is  $O(n)$

# Design a universal class of hash functions

We begin by choosing a prime number  $p$  large enough so that every possible key  $k$  is in the range 0 to  $p - 1$ , inclusive. Let  $\mathbf{Z}_p$  denote the set  $\{0, 1, \dots, p - 1\}$ , and let  $\mathbf{Z}_p^*$  denote the set  $\{1, 2, \dots, p - 1\}$ .

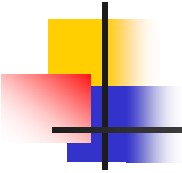
Since  $p$  is prime, we can solve equations modulo  $p$  with the methods given in Chapter 31. Because we assume that the size of the universe of keys is greater than the number of slots in the hash table we have  $p > m$ .

- We now define the hash function  $h_{a,b}$  for any  $a \in \mathbf{Z}_p^*$  and any  $b \in \mathbf{Z}_p$  using a linear transformation followed by reductions modulo  $p$  and then modulo  $m$  :

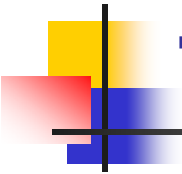
$$h_{a,b}(k) = ((ak + b) \bmod p) \bmod m.$$

For example, with  $p = 17$  and  $m = 6$ , we have  $h_{3,4}(8) = 5$ . The family of all such hash functions is

$$\mathcal{H}_{p,m} = \{h_{a,b} : a \in \mathbf{Z}_p^* \text{ and } b \in \mathbf{Z}_p\}.$$



- Each hash function  $h_{a,b}$  maps  $\mathbb{Z}_p$  to  $\mathbb{Z}_m$ . This class of hash functions has the nice property that the size  $m$  of the output range is arbitrary –not necessarily prime– a feature which we shall use in Section 11.5. Since there are  $p - 1$  choices for  $a$  and there are  $p$  choices for  $b$ , there are  $p(p - 1)$  hash functions in  $\mathcal{H}_{p,m}$ .



## Theorem 11.5

- The class  $\mathcal{H}_{p,m}$  defined above is a universal hash functions.



# Proof.

- Consider two distinct keys  $k, l$  from  $\mathbb{Z}_{p'}$ , so  $k \neq l$ . For a given hash function  $h_{a,b}$  we let

$$r = (ak + b) \bmod p,$$

$$s = (al + b) \bmod p.$$

$$r \neq s$$

$$a = ((r - s)((k - l)^{-1} \bmod p)) \bmod p,$$

$$b = (r - ak) \bmod p,$$

For any given pair of input  $k$  and  $l$ , if we pick  $(a, b)$  uniformly at random from  $\mathbb{Z}_p^* \times \mathbb{Z}_p$ , the resulting pair  $(r, s)$  is equally likely to be any pair of distinct values modulo  $p$ .



It then follows that the probability that distinct keys  $k$  and  $l$  collide is equal to the probability that  $r \equiv s \pmod{m}$  when  $r$  and  $s$  are randomly chosen as distinct values modulo  $p$ . For a given value of  $r$ , of the  $p - 1$  possible remaining values for  $s$ , the number of values  $s$  such that  $s \neq r$  and  $s \equiv r \pmod{m}$  is at most

$$\begin{aligned} \lceil p/m \rceil - 1 &\leq ((p + m - 1)/m - 1 \\ &= (p - 1)/m. \end{aligned}$$

The probability that  $s$  collides with  $r$  when reduced modulo  $m$  is at most

$$((p - 1)/m)/(p - 1) = 1/m.$$

## 11.4 Open addressing

- (All elements are stored in the hash tables itself.)
- $h : U \times \{0, 1, \dots, m-1\} \rightarrow \{0, 1, \dots, m-1\}$ .  
With open addressing, we require that for every key  $k$ , the **probe sequence**  $\langle h(k, 0), h(k, 1), \dots, h(k, m-1) \rangle$  be a permutation of  $\{0, 1, \dots, m\}$ .

### HASH\_INSERT( $T, k$ )

```

1  $i \leftarrow 0$ 
2 repeat  $j \leftarrow h(k, i)$ 
3   if  $T[j] = \text{NIL}$ 
4     then  $T[j] \leftarrow k$ 
5           return  $j$ 
6   else  $i \leftarrow i + 1$ 
7 until  $i = m$ 
8 error “hash table overflow”

```



## HASH\_SEARCH(T,k)

```

1   $i \leftarrow 0$ 
2  repeat  $j \leftarrow h(k, i)$ 
3      if  $T[j] = k$ 
4  then return  $j$ 
5       $i \leftarrow i + 1$ 
6  until  $T[j] = \text{NIL}$  or  $i = m$ 
7  return NIL

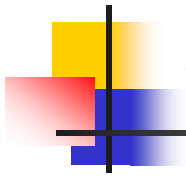
```



## Linear probing:

$$h(k, i) = (h'(k) + i) \bmod m$$

- It suffers the primary clustering problem.

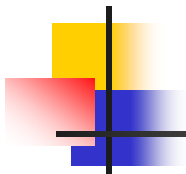


## Quadratic probing:

$$h(k,i) = (h(k) + c_1i + c_2i^2) \bmod m$$

$$c_1, c_2 \neq 0$$

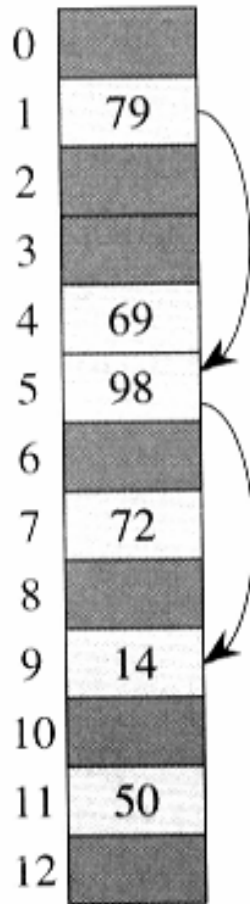
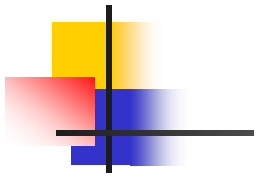
- It suffers the secondary clustering problem.



## Double hashing:

$$h(k,i) = (h_1(k) + ih_2(k)) \bmod m$$

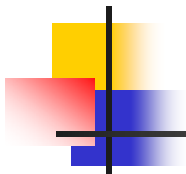




$$h_1(k) = k \mod 13$$

$$h_2(k) = 1 + (k \mod 11)$$

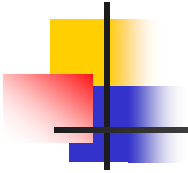
Insert 14



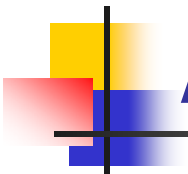
## Example:

$$h_1(k) = k \mod m$$

$$h_2(k) = 1 + (k \mod m)$$



- Double hashing represents an improvement over linear and quadratic probing in that probe sequence are used. Its performance is more closed to uniform hashing.



## Analysis of open-address hash



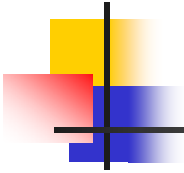
## Theorem 11.6

- Given an open-address hash-table with load factor  $\alpha = n/m < 1$ , the expected number of probes in an unsuccessful search is at most  $1/(1-\alpha)$  assuming uniform hashing.



## Proof.

- Define  $p_i = \Pr\{\text{exactly } i \text{ probes access occupied slots}\}$  for  $0 \leq i \leq n$ . And  $p_i = 0$  if  $i > n$ .
- The expected number of probes is  $1 + \sum_{i=0}^{\infty} i p_i$ .
- Define  $q_i = \Pr\{\text{at least } i \text{ probes access occupied slots}\}$ .

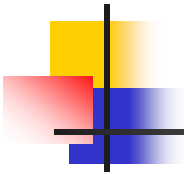


$$\sum_{i=0}^{\infty} ip_i = \sum_{i=1}^{\infty} q_i$$

$$E[X] = \sum_{i=0}^{\infty} i \Pr\{X = i\}$$

$$= \sum_{i=0}^{\infty} i(\Pr\{X \geq i\} - \sum_{i=0}^{\infty} \Pr\{X \geq i+1\})$$

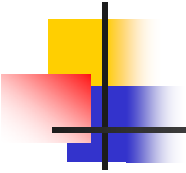
$$= \sum_{i=1}^{\infty} \Pr\{X \geq i\}$$



$$q_1 = \frac{n}{m} \quad q_2 = \left(\frac{n}{m}\right)\left(\frac{n-1}{m-1}\right)$$

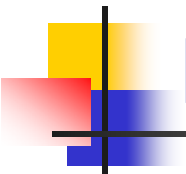
$$q_i = \left(\frac{n}{m}\right)\left(\frac{n-1}{m-1}\right)\dots\left(\frac{n-i+1}{m-i+1}\right) \leq \left(\frac{n}{m}\right)^i = \alpha^i$$

$$\text{if } 1 \leq i \leq n$$



- $q_i = 0$  for  $i > n$ .

- $1 + \sum_{i=0}^{\infty} ip_i = 1 + \sum_{i=0}^{\infty} q_i \leq 1 + \alpha + \alpha^2 + \dots = \frac{1}{1-\alpha}$



## Example:

$$\alpha = 0.5 \quad \frac{1}{1-\alpha} = 2$$

$$\alpha = 0.9 \quad \frac{1}{1-\alpha} = 10$$



## ***Corollary 11.7***

- Inserting an element into an open-address hash table with load factor  $\alpha$  requires at most  $1/(1 - \alpha)$  probes on average, assuming uniform hashing.



## **Proof.**

- An element is inserted only if there is room in the table, and thus  $\alpha < 1$ . Inserting a key requires an unsuccessful search followed by placement of the key in the first empty slot found. Thus, the expected number of probes is  $\frac{1}{1 - \alpha}$ .



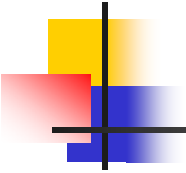
## Theorem 11.8

- Given an open-address hash table with load factor  $\alpha < 1$ , the expected number of successful search is at most  $\frac{1}{\alpha} \ln \frac{1}{1-\alpha} + \frac{1}{\alpha}$  assuming uniform hashing and assuming that each key in the table is equally likely to be searched for.

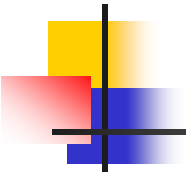


## Proof.

- A search for  $k$  follows the same probe sequence as followed when  $k$  was inserted.
- If  $k$  is the  $(i+1)$ st key inserted in the hash table, the expected number of probes made in a search for  $k$  is at most  $\frac{1}{1 - \frac{i}{m}} = \frac{m}{m-i}$ .



- Averaging over all  $n$  key in the hash table gives us the average number of probes in a successful search:

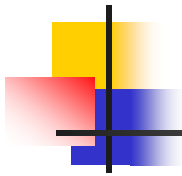


$$\begin{aligned} \frac{1}{n} \sum_{i=0}^{n-1} \frac{m}{m-i} &= \frac{m}{n} \sum_{i=0}^{n-1} \frac{1}{m-i} = \frac{1}{\alpha} (H_m - H_{m-1}) \\ &\leq \frac{1}{\alpha} (\ln m + 1 - \ln(m-n)) \end{aligned}$$

(Since  $\ln i \leq H_i \leq \ln i + 1$ )

$$= \frac{1}{\alpha} \ln \frac{m}{m-n} + \frac{1}{\alpha} = \frac{1}{\alpha} \ln \frac{1}{1-\alpha} + \frac{1}{\alpha}$$





## Example:

$$\alpha = 0.5 \quad \frac{1}{\alpha} \ln \frac{1}{1-\alpha} + \frac{1}{\alpha} \approx 3.387$$

$$\alpha = 0.9 \quad \frac{1}{\alpha} \ln \frac{1}{1-\alpha} + \frac{1}{\alpha} \approx 3.670$$