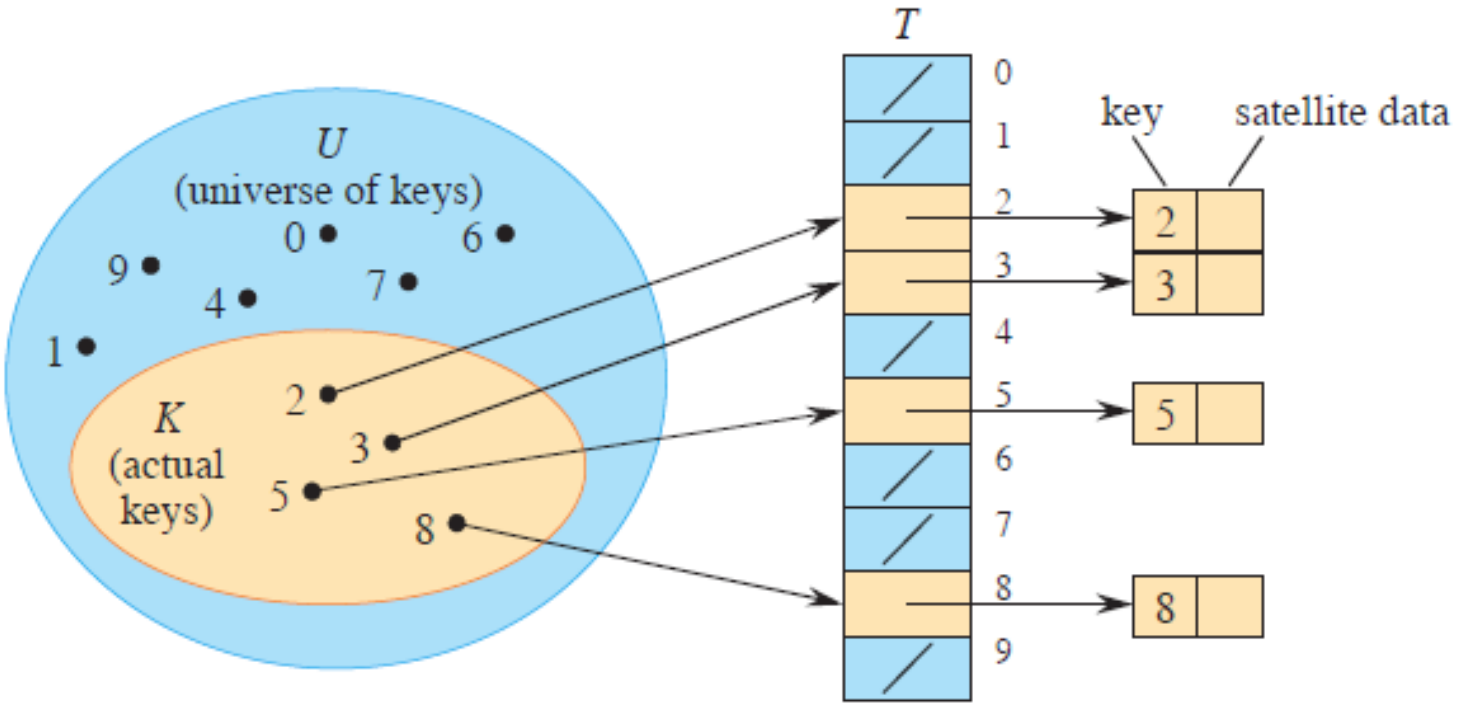
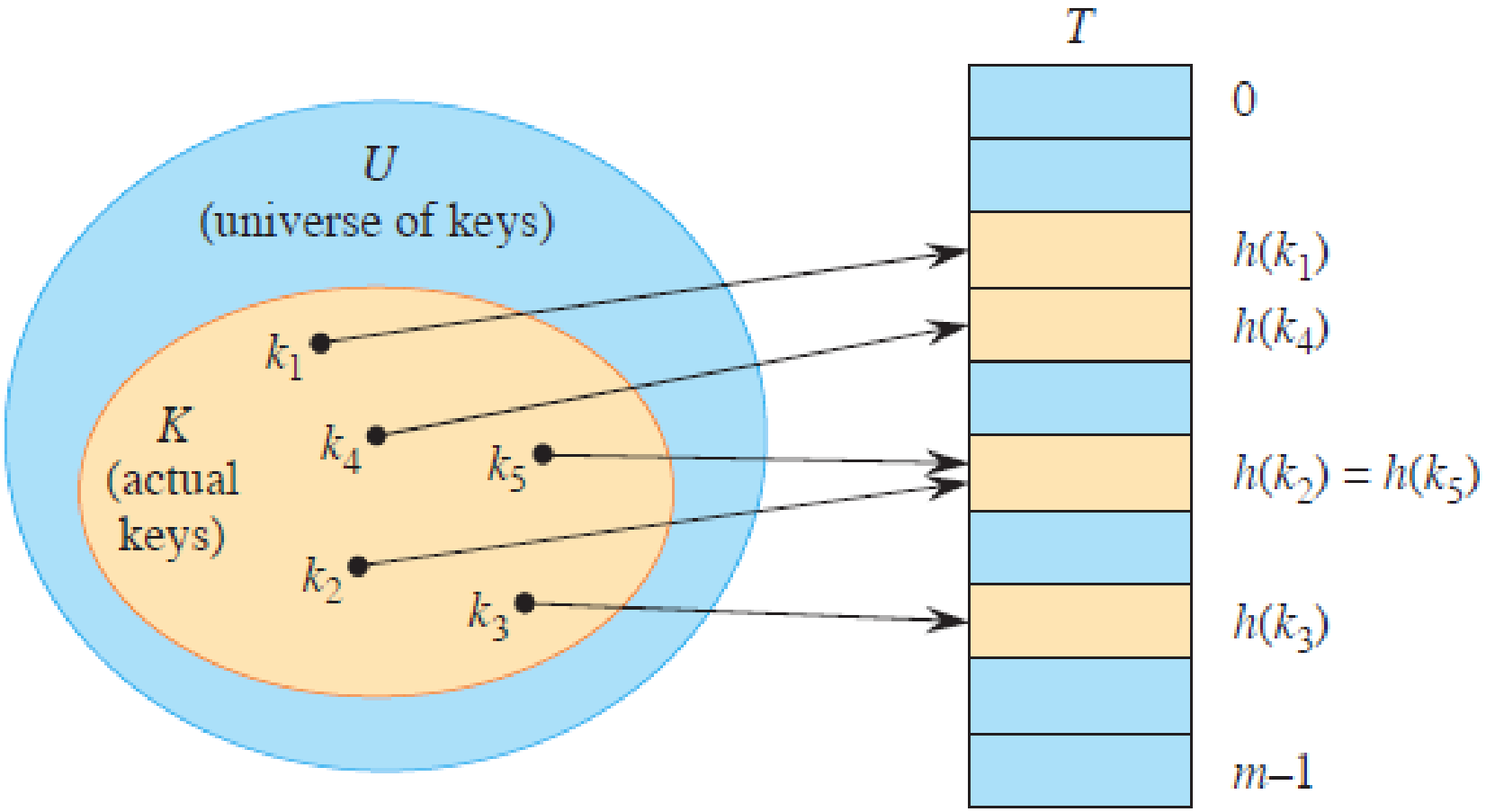
Hash Tables

Direct-address table

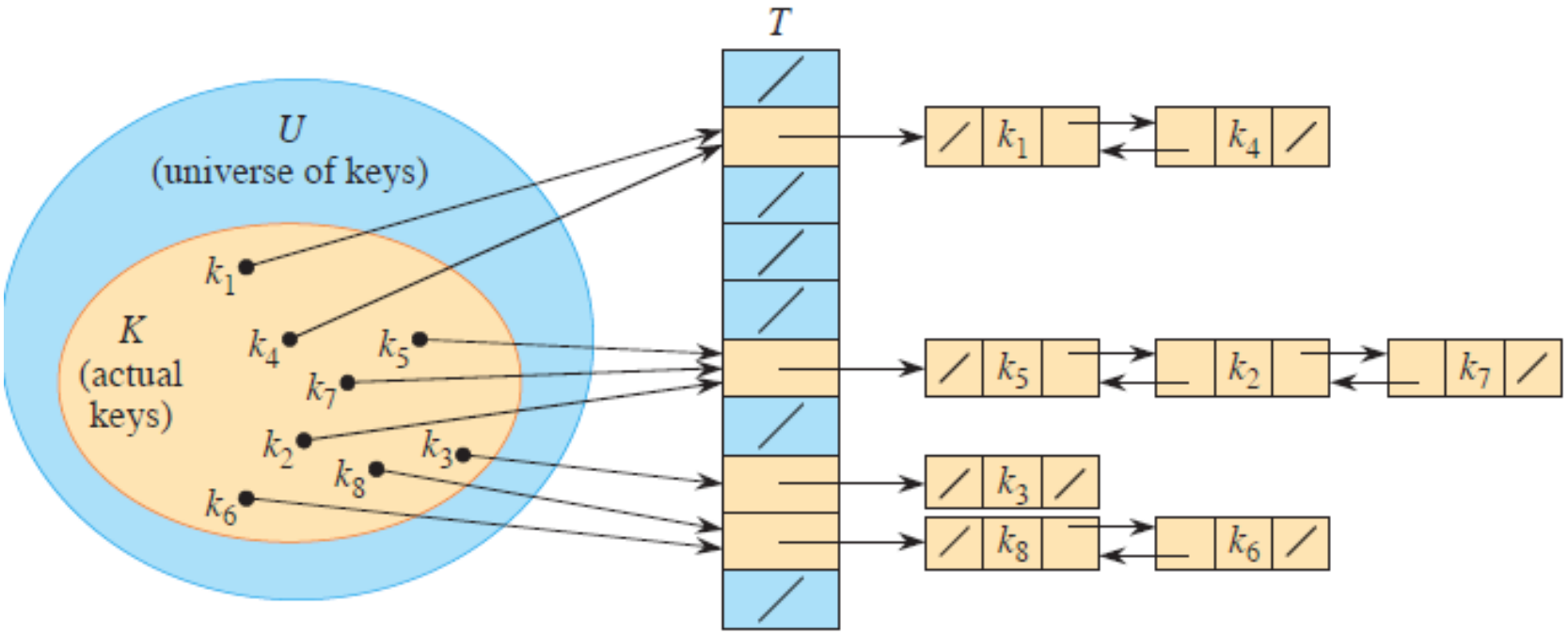



```
DIRECT-ADDRESS-SEARCH(T, k)
1  return T[k]

DIRECT-ADDRESS-INSERT(T, x)
1  T[x.key] = x

DIRECT-ADDRESS-DELETE(T, x)
1  T[x.key] = NIL
```

Hash table


T
U
(universe of keys)
K
(actual
keys)
k_1
k_4
k_5
k_2
k_3
0
$h(k_1)$
$h(k_4)$
$h(k_2) = h(k_5)$
$h(k_3)$
$m-1$

Collision resolution by chaining




```
CHAINED-HASH-INSERT(T, x)
1  LIST-PREPEND(T[h(x.key)], x)

CHAINED-HASH-SEARCH(T, k)
1  return LIST-SEARCH(T[h(k)], k)

CHAINED-HASH-DELETE(T, x)
1  LIST-DELETE(T[h(x.key)], x)
```

Analysis of hashing with chaining:

- Given a hash table with m slots and n keys, define load factor $\alpha = \mathrm{n/m}$: average number of keys per slot.

- For j=0, 1,…, m -1, denote the length of the list $T[j]$ by n_j , so that $E[n_j] = \frac{n}{m} = \alpha$.

- Assume each key is equally likely to be hashed into any slot: independently uniform hashing (IUH).

- Thm 1: In a hash table in which collisions are resolved by chaining, an unsuccessful search takes expected time $\Theta(1+\alpha)$ under IUH.

Proof:

- Under the assumption of IUH, any un-stored key is equally likely to hash to any of the m slots.
- The expected time to search unsuccessfully for a key k is the expected time to search to the end of list T[h(k)], which is exactly α.
- Thus, the total time required is $\Theta(1+\alpha)$. □

- Thm 2: In a hash table in which collisions are resolved by chaining, a successful search takes $\Theta(1+\alpha)$ time on average under IUH.

Proof:

- Let the element being searched for equally likely to be any of the n elements stored in the table.
- The expected time to search successfully for a key:

- Elements before x in the list were inserted after x was inserted.
- We want to find the expected number of elements added to x's list after x was added to the list.

- Let x_i be the ith element into the table, for i =1 to n, and let $k_i = x_i.key$.
- Define $X_{ijq} = I\{\text{search } x_i, h(k_i) = q, h(k_j) = q\}$.
- Under IUH, we have $\Pr\{h(k_i) = q\} = \Pr\{h(k_j) = q\} = 1/m$
  $\Pr\{\text{search } x_i\} = 1/n, \quad E[X_{ijq}] = 1/nm^2$.
- Let $Y_j = I\{x_j$ appear in a list prior the element being searched$\}$= $\sum_{q=0}^{m-1} \sum_{i=1}^{j-1} X_{ijq}$

- Let $Z = \sum_{j=1}^{n} Y_j$, the number of elements appear in the list prior to the being searched element.

$$
\begin{aligned}
\mathrm{E}[Z+1] &= \mathrm{E}\left[1+\sum_{j=1}^{n} Y_j\right] \\
&= 1+\mathrm{E}\left[\sum_{j=1}^{n}\sum_{q=0}^{m-1}\sum_{i=1}^{j-1} X_{ijq}\right] \\
&= 1+\mathrm{E}\left[\sum_{q=0}^{m-1}\sum_{j=1}^{n}\sum_{i=1}^{j-1} X_{ijq}\right] \\
&= 1+\sum_{q=0}^{m-1}\sum_{j=1}^{n}\sum_{i=1}^{j-1}\mathrm{E}[X_{ijq}] \qquad \text{(by linearity of expectation)} \\
&= 1+\sum_{q=0}^{m-1}\sum_{j=1}^{n}\sum_{i=1}^{j-1}\frac{1}{nm^2} \\
&= 1+m\cdot\frac{n(n-1)}{2}\cdot\frac{1}{nm^2} \\
&= 1+\frac{n-1}{2m} \\
&= 1+\frac{n}{2m}-\frac{1}{2m} \\
&= 1+\frac{\alpha}{2}-\frac{\alpha}{2n}.
\end{aligned}
$$

Thus, the total time required for a successful search (including the time for computing the hash function) is:

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

The multiplication method:

0<A<1

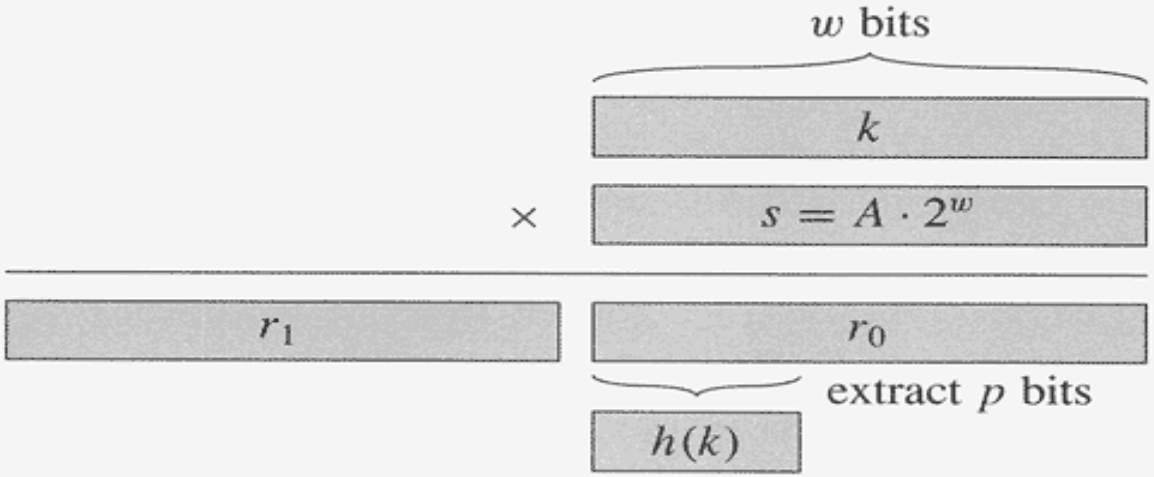


Figure 11.4 The multiplication method of hashing. The w-bit representation of the key k is multiplied by the w-bit value $s = A \cdot 2^w$. The p highest-order bits of the lower w-bit half of the product form the desired hash value $h(k)$.

Knuth suggests that $A \approx (\sqrt{5}-1)/2 = 0.6180339887\ldots$

Take $k = 123456$, $p = 14$, $m = 2^{14} = 16384$ and $w = 32$.

$s = A \cdot 2^{32} = 2654435769$

$k \cdot s = 327706022297664 = (76300 \cdot 2^{32}) + 176112864$

The 14 most significant bits of 176112864 is 67!

Random hashing

- With a fixed hash function, a malicious adversary can choose n keys that all hash to the same slot, yielding an average retrieval time of $\Theta(n)$.
- Any static hash function is vulnerable to such terrible worst-case behavior.
- The only effective way to improve the situation is to choose the hash function randomly in a way that is independent of the keys that are actually going to be stored.
- This approach is called random hashing. A special case of this approach, called universal hashing.

Universal Hashing

- $\mathcal{H} = \{ h: U \to \{0, \ldots, m-1\} \}$, which is a finite collection of hash functions.
- $\mathcal{H}$ is called "*universal*" if for each pair of distinct keys $\ell, k \in U$, the number of hash functions $h \in \mathcal{H}$ for which $h(k) = h(\ell)$ is at most $|\mathcal{H}|/m$
- Define n_i = the length of list $T[i]$

Cor 3:Using universal hashing and collision resolution by chaining in an initially empty table with m slots, it takes $\Theta(s)$ expected time to handle any sequence of s INSERT, SEARCH, and DELETE operations containing $n = O(m)$ INSERT operations.

Proof:

- Each INSERT and DELETE op take constant time.
- $\alpha = \frac{n}{m} = O(1)$.
- The expected time for each SEARCH operation is $O(1)$.
- $\Pr_h\{h(k_1) = h(k_2)\} \leq 1/m$.
- By linearity of expectation, therefore, the expected time for the entire sequence of s operations is $O(s)$.

Achievable properties of random hashing

$\mathcal{H} = \{ h: U \to \{0, \ldots, m-1\} \}$, which is a finite collection of hash functions. h can be uniformly selected from $\mathcal{H}$.

- $\mathcal{H}$ is "*uniform*" if for each key $k \in U$ and any slot q in the range $\{0, 1, \ldots, m-1\}$, $\Pr\{h(k) = q\} = 1/m$.
- $\mathcal{H}$ is "*universal*" if for each pair of distinct keys $\ell, k \in U$, $\Pr_{h \in \mathcal{H}}\{h(k) = h(\ell)\} \leq \frac{1}{m}$.
- $\mathcal{H}$ is "ϵ *−universal*" if for each pair of distinct keys $\ell, k \in U$, $\Pr_{h \in \mathcal{H}}\{h(k) = h(\ell)\} \leq \epsilon$
- $\mathcal{H}$ is "$d - independent$" if for any d distinct keys $k_1, \ldots, k_d \in U$ and any d slots $q_1, \ldots, q_d$, $\Pr_{\mathrm{h}}\{h(k_i) = q_i, i = 1..d\} = 1/m^d$.

Design a universal class of hash functions:

- Let p be a prime number.
  $Z_p = \{0, \ldots, p-1\}$, $Z_p^* = Z_p/\{0\}$

- For any $a \in Z_p^*$ and $b \in Z_P$, define:

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

$$\mathcal{H}_{p,m} = \{h_{a,b} : a \in Z_p^*\ and\ b \in Z_p\}$$

Theorem:

$\mathcal{H}_{p,\,m}$ is universal.

Pf:

- Let $k, \ell \in Z_P$ be two distinct keys.
- Given $h_{a,b}$, let

$$r = (ak + b) \bmod p,$$
$$s = (a\ell + b) \bmod p.$$

- Then $r - s \equiv a(k - \ell) \ (mod\ p))$

- For any $h_{a,b} \in \mathcal{H}_{p,m}$, distinct inputs k and ℓ map to distinct r and s modulo p.

- Each possible $p(p-1)$ choices for the pair (a,b) with $a \neq 0$ yields a different resulting pair (r,s) with $r \neq s$, since we can solve for a and b given r and s:

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

- There are $p(p-1)$ possible pairs (r,s) with $r \neq s$, there is a 1-1 correspondence between pairs (a,b) with $a \neq 0$ and (r,s), $r \neq s$.

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

- Pr[k and ℓ collide] = $\Pr_{r,s}[r \equiv s \bmod m]$

- Given r the number of s, such that $s \neq r$ and $s \equiv r \pmod m$, is at most

$$\lceil p/m \rceil - 1 \leq ((p+m-1)/m) - 1$$
$$= (p-1)/m$$
$$\because s, s+m, s+2m, \ldots, \leq p$$

- Thus,

$\Pr_{r,s}[\, r \equiv s \bmod m] \le ((p-1)/m)/(p-1) = 1/m$

- Therefore, for any pair of distinct key $k, \ell \in Z_p$:

$\Pr[h_{a,b}(k) = h_{a,b}(\ell)\,] \le 1/m,$

so that $\mathcal{H}_{p,m}$ is universal.

Hash with the multiply-shift method:

- Let $m = 2^{\ell}$ for some integer ℓ where $\ell \leq w$ and w is the number of bits in a machine word.
- Choose a fixed w-bit positive integer $a = A2^{w}$, where $0 < A < 1$ so that $0 < a < 2^{w}$.
- Assume that a key k fits into a single w-bit word and define:

$$h_a(k) = (ka \bmod 2^w) \gg (w - \ell)$$

$$\mathcal{H} = \{h_a : a \text{ is odd}, 1 \leq a < m\}, 2/m - \text{universal}$$

Hashing long inputs as vectors or strings :

- Can consider the class of vectors, such as vectors of 8-bit~ can be arbitrarily long.
- Number-theoretic approaches:

  Extend the ideas used to design universal hash functions

- Cryptographic hash functions:

  Use a hash function designed for cryptographic applications to design a good hash function for variable-length inputs.

Hashing long inputs as vectors or strings :

- Cryptographic hash functions take as input an arbitrary byte string and returns a fixed-length output.

- For example, the NIST standard deterministic cryptographic hash function SHA-256 produces a 256-bit (32-byte) output for any input.

$$h_a(k) = \text{SHA}{-}256(a \,||k) \; mod \; m$$

- Include instructions in CPU architectures to provide fast implementations of some cryptographic functions.

Advanced Encryption Standard (AES), the "AES-NI" instructions.

Open addressing:

- There is no list and no element stored outside the table.
- Advantage: avoid pointers, potentially yield fewer collisions and faster retrieval.

- $h : U \times \{0,1,\cdots,m-1\} \rightarrow \{0,1,\cdots,m-1\}$
- For every *k*, the probe sequence

  $$\langle h(k,0), h(k,1), \cdots, h(k,m-1) \rangle$$

  is a permutation of $\langle 0,1,\cdots,m-1 \rangle$.
- Deletion from an open-address hash table is difficult.
- Thus chaining is more common when keys must be deleted.

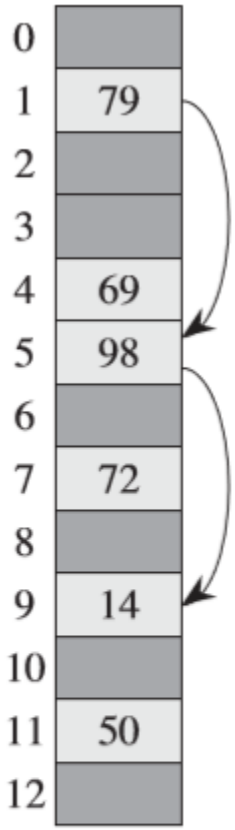
```
HASH-INSERT(T, k)
1  i = 0
2  repeat
3       q = h(k, i)
4       if T[q] == NIL
5            T[q] = k
6            return q
7       else i = i + 1
8  until i == m
9  error "hash table overflow"
```

```
HASH-SEARCH(T, k)
1  i = 0
2  repeat
3       q = h(k, i)
4       if T[q] == k
5            return q
6       i = i + 1
7  until T[q] == NIL or i == m
8  return NIL
```

- Linear probing:
  - $h' : U \to \{0, 1, \cdots, m-1\}$~ an ordinary hash function (auxiliary hash function).
  - $h(k, i) = (h'(k) + i) \; mod \; m$

- Quadratic probing:
  - $h(k,i) = (h'(k) + c_1 i + c_2 i^2) \; mod \;\; m,$ where h' is an auxiliary hash function, c_1 and $c_2 \neq 0$ and are constants.

- Double hashing:
  - $h(k,i) = \left(h_1(k) + ih_2(k)\right) \bmod m$,where h_1 and h_2 are auxiliary hash functions.
  - $h_2(k)$ must be relative prime to m to search the whole hash table.
  - There are $O(m^2)$ probe sequences when m is prime or proper numbers, since each possible $(h_1(k), h_2(k))$ pair yields a distinct probe sequence.
  - Linear and Quadratic have $O(m)$ probe sequences.



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

Figure 11.5 Insertion by double hashing. Here we have a hash table of size 13 with $h_1(k) = k \bmod 13$ and $h_2(k) = 1 + (k \bmod 11)$. Since $14 \equiv 1 \pmod{13}$ and $14 \equiv 3 \pmod{11}$, we insert the key 14 into empty slot 9, after examining slots 1 and 5 and finding them to be occupied.

- Analysis of open-addressing hashing

$$\alpha = \frac{n}{m} \quad : \text{load factor,}$$

with n elements and m slots.

- Thm:

  Given an open-address hash table with load factor $\alpha = \frac{n}{m} < 1$, the expected number of probes in an unsuccessful search is at most $1/(1-\alpha)$, assuming independent uniform permutation hashing and no deletion.

- Pf:
  - Define the r.v. X to be the number of probes made in an unsuccessful search.
  - Define A_i: the event there is an i-th probe and it is to an occupied slot.
  - Event $\{X \geq i\} = A_1 \cap A_2 \cap \cdots \cap A_{i-1}$

- $\Pr\{X \geq i\}$
- $= \Pr\{A_1 \cap A_2 \cap \cdots \cap A_{i-1}\}$

  $= \Pr\{A_1\} \cdot \Pr\{A_2|A_1\} \cdot$

  $\Pr\{A_3|A_1 \cap A_2\} \cdots \Pr\{A_{i-1}|A_1 \cap A_2 \cap \cdots \cap A_{i-2}\}$
- $\Pr\{A_1\} = n/m$
- The probability that there is a jth probe and it is to an occupied slot, given that the first j-1 probes were to occupied slots is $(n - j + 1)/(m - j + 1)$. Why?

- ∵$n < m$, $(n-j)/(m-j) \leq n/m$ for all $0 \leq j < m$.
-

$$\Pr[X \geq i] = \frac{n}{m} \cdot \frac{n-1}{m-1} \cdots \frac{n-i+2}{m-i+2}$$

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

$$E[X] = \sum_{i=1}^{\infty} \Pr[X \geq i] \leq \sum_{i=1}^{\infty} \alpha^{i-1} = \sum_{i=0}^{\infty} \alpha^{i} = \frac{1}{1-\alpha}$$

Cor: Inserting an element into an open-addressing hash table with load factor α requires at most $1/(1-\alpha)$ probes on average, assuming independent uniform permutation hashing and no deletion.

Theorem:

Given an open-address hash table with load factor $\alpha<1$, the expected number of probes in a successful search is at most $\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$, assuming independent uniform permutation hashing with no deletion and that each key in the table is equally likely to be searched for.

Pf: Suppose we search for a key k.

If k was the (i+1)st key inserted into the hash table, the expected number of probes made in a search for k is at most

$$\frac{1}{1-\frac{i}{m}} = \frac{m}{m-i}.$$

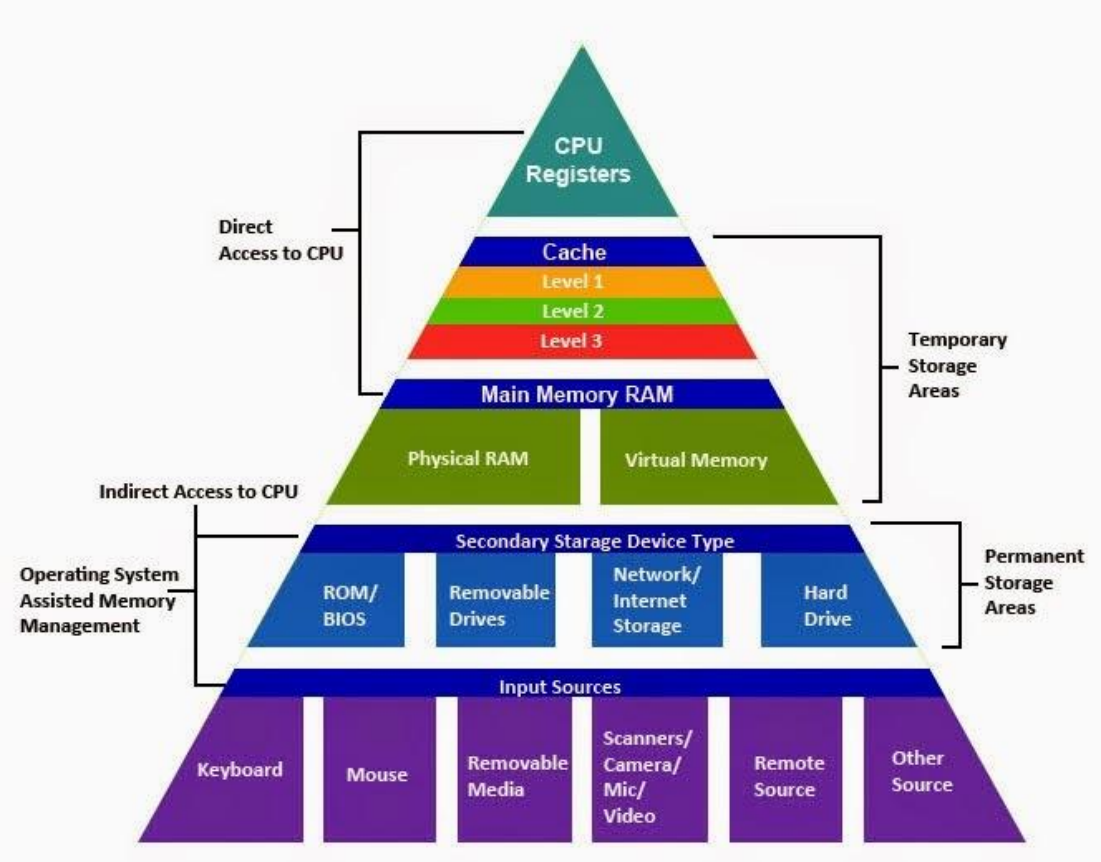
- Averaging over all n keys in the hash table gives us the average number of probes in a successful search:

$$\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i} = \frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i} = \frac{1}{\alpha}(H_m - H_{m-n})$$

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

11.5 Practical considerations

- Efficient hash table algorithms are not only of theoretical interest, but also of immense practical importance.
- Memory hierarchies: cache memory, cache blocks
- Advanced instruction sets

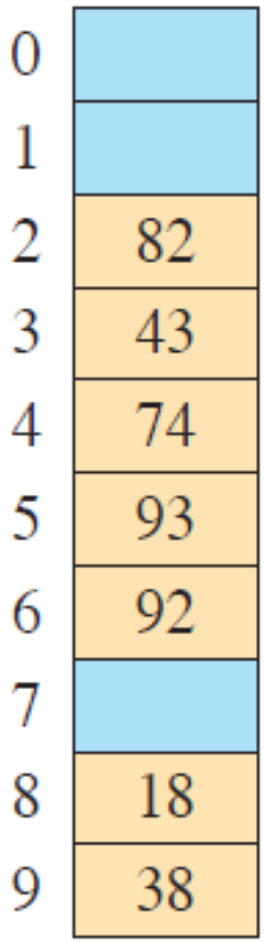


Linear probing with deletion

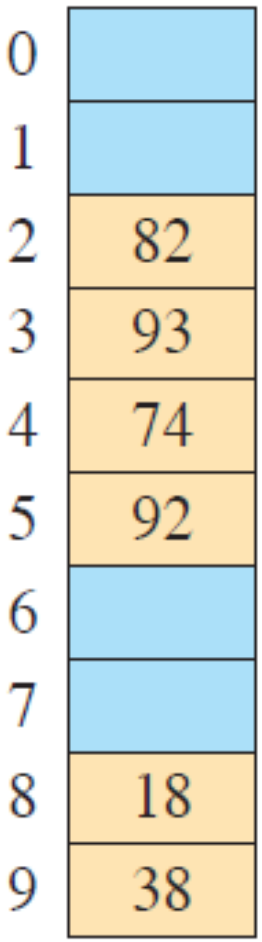
- $h(k,i) = (h_1(k) + i) \bmod m$
- The inverse function g maps a key k and a slot number q ($0 \leq q < m$) to the probe number that reaches slot q:

$$g(k,q) = \big(q - h_1(k)\big) \bmod m$$

- If $h(k,i) = q$, then $g(k,q) = i$, and so $h(k, g(k,q)) = q$.
- If $g(k',q) < g(k',q')$ then during the insertion of k' into the table, slot q was examined but found to be already occupied.



(a)



(b)

- $h(k,i) = h_1(k) + i,$
  $h_1(k) = k \bmod 10$
- (a) the hash table after inserting keys in the order 74, 43, 93, 18, 82, 38, 92
- (b) the hash table after deleting the key 43 from slot 3

```
LINEAR-PROBING-HASH-DELETE(T, q)
 1  while TRUE
 2      T[q] = NIL                          // make slot q empty
 3      q' = q                              // starting point for search
 4      repeat
 5          q' = (q' + 1) mod m             // next slot number with linear probing
 6          k' = T[q']                      // next key to try to move
 7          if k' == NIL
 8              return                      // return when an empty slot is found
 9      until g(k', q) < g(k', q')          // was empty slot q probed before q'?
10      T[q] = k'                           // move k' into slot q
11      q = q'                              // free up slot q'
```

Theorem: If h_1 is 5-independent and $\alpha \leq 2/3$, then it takes expected constant time to search for, insert, or delete a key in a hash table using linear probing.

(M. Thorup. Linear probing with 5-independent hashing. http://arxiv.org/abs/1509.04549, 2015.)

Hash functions for hierarchical memory models: *wee hash function*

- $swap(x) = (x \gg (\frac{w}{2})) + (x \ll (\frac{w}{2}))$

  w: even word size (e.g., $w = 64$)
- $f_a(k) = swap((2k^2 + ak) \bmod 2^w)$, a is odd
- $f_a^{(r)}(k)$: result of iterating f_a a total of r times starting with t-bit input value k. 1-1 function?
- *wee hash function*:

$$h_{a,b,t,r}(k) = \left(f_{a+2t}^{(r)}(k + b) \right) mod\ m$$

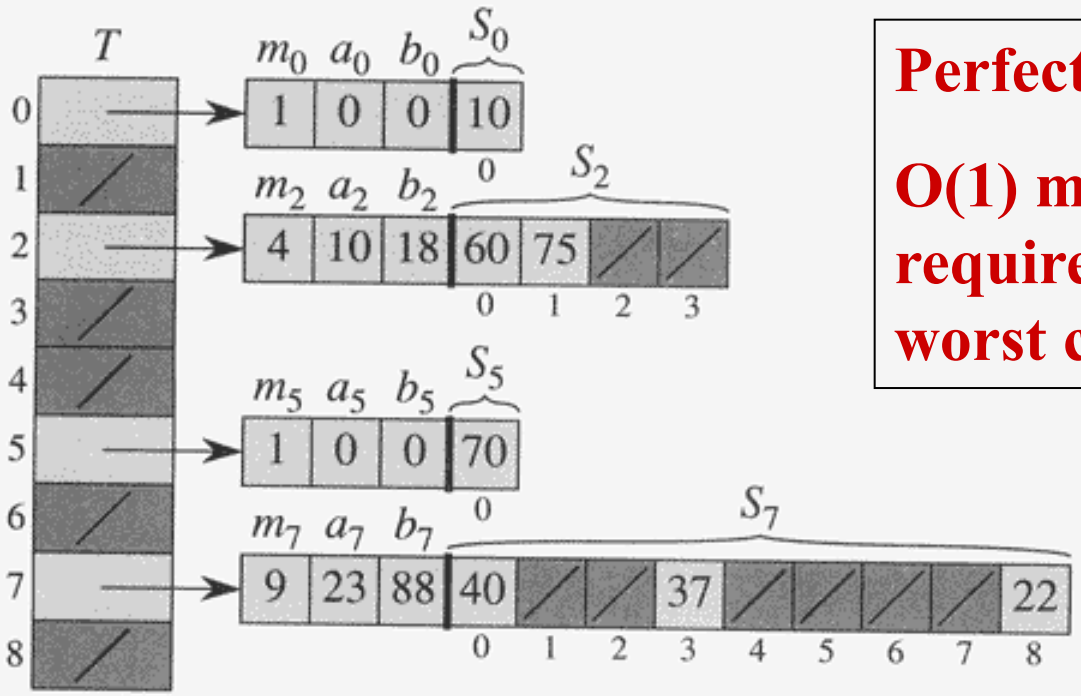
WEE hash function for variable-length inputs

$$f_a(k) = swap((2k^2 + ak) \bmod 2^w)$$

- Some k may be $> w$-bit word in length or have variable length.
- $chop(k) = \langle k_1, \cdots, k_u \rangle$, $u = \left\lceil \frac{t}{w} \right\rceil$, padding 0′s if necessary
- $h_{a,b,t,r}(k) = WEE(k, a, b, t, r, m)$

WEE(k, a, b, t, r, m)

1 $u = \lceil t/w \rceil$
2 $\langle k_1, k_2, \ldots, k_u \rangle = \text{chop}(k)$
3 $q = b$
4 **for** $i = 1$ **to** u
5 $\quad q = f^{(r)}_{a+2t}(k_i + q)$
6 **return** $q \bmod m$



Perfect Hashing:

O(1) memory accesses are required for a search in the worst case

Figure 11.6 Using perfect hashing to store the set $K = \{10, 22, 37, 40, 60, 70, 75\}$. The outer hash function is $h(k) = ((ak + b) \bmod p) \bmod m$, where $a = 3$, $b = 42$, $p = 101$, and $m = 9$. For example, $h(75) = 2$, so key 75 hashes to slot 2 of table T. A secondary hash table S_j stores all keys hashing to slot j. The size of hash table S_j is m_j, and the associated hash function is $h_j(k) = ((a_j k + b_j) \bmod p) \bmod m_j$. Since $h_2(75) = 1$, key 75 is stored in slot 1 of secondary hash table S_2. There are no collisions in any of the secondary hash tables, and so searching takes constant time in the worst case.

- Perfect hashing :
  good for when the keys are static; i.e. , once stored, the keys never change, e.g. CD-ROM, the set of reserved word in programming language.

- **Thm** : If we store n keys in a hash table of size $m = n^2$ using a hash function h randomly chosen from a universal class of hash functions, then the probability of there being any collisions < ½ .

- **Proof:** Let h be chosen from an universal family. Then each pair collides with probability $1/m$, and there are $\binom{n}{2}$ pairs of keys.

  Let X be a r.v. that counts the number of collisions. When $m = n^2$,

$$E[X] = \binom{n}{2}\frac{1}{m} = \frac{n^2 - n}{2}\cdot\frac{1}{n^2} < \frac{1}{2}$$

$$By\ Markov's\ inequality,\quad \Pr[X \geq t] \leq E[X]/t,$$

$$and\ take\ t = 1.$$

Theorem:

If we store n keys in a hash table of size $m = n$ using a hash function h randomly chosen from universal class of hash functions, then $E[\sum_{j=0..m-1} n_j^2] < 2n$, where n_j is the number of keys hashing to slot j.

- Pf:
  - It is clear for any nonnegative integer a,

$$a^2 = a + 2\binom{a}{2}$$

  -

$$E[\sum_{j=0}^{m-1} {n_j}^2] = E[\sum_{j=0}^{m-1} \left(n_j + 2\binom{n_j}{2} \right)]$$

$$= E[\sum_{j=0}^{m-1} n_j] + 2E[\sum_{j=0}^{m-1} \binom{n_j}{2}]$$

$$= E[n] + 2E[\sum_{j=0}^{m-1} \binom{n_j}{2}] = n + 2E[\underbrace{\sum_{j=0}^{m-1} \binom{n_j}{2}}]$$

total number of collisions

$$\binom{n}{2}\frac{1}{m} = \frac{n(n-1)}{2m} = \frac{n-1}{2}, \text{ since } m = n.$$

$$E[\sum_{j=0}^{m-1} {n_j}^2] \le n + 2 \cdot \frac{n-1}{2} = 2n - 1 < 2n.$$

Corollary:

If store n keys in a hash table of size $m = n$ using a hash function h randomly chosen from a universal class of hash functions and we set the size of each secondary hash table to $m_j = {n_j}^2$ for $j = 0, \ldots, m - 1$, then the expected amount of storage required for all secondary hash tables in a perfect hashing scheme is $< 2n$.

Corollary: Same as the above,

$\Pr\{\text{total storage} \geq 4n\} < 1/2$

Proof:

– By Markov's inequality, $\Pr\{X \geq t\} \leq E[X]/t$.

$$\text{Take } X = \sum_{j=0}^{m-1} m_j \text{ and } t = 4n:$$

$$\Pr\{\sum_{j=0}^{m-1} m_j \geq 4n\} \leq \frac{E[\sum_{j=0}^{m-1} m_j]}{4n} < \frac{2n}{4n} = \frac{1}{2}.$$