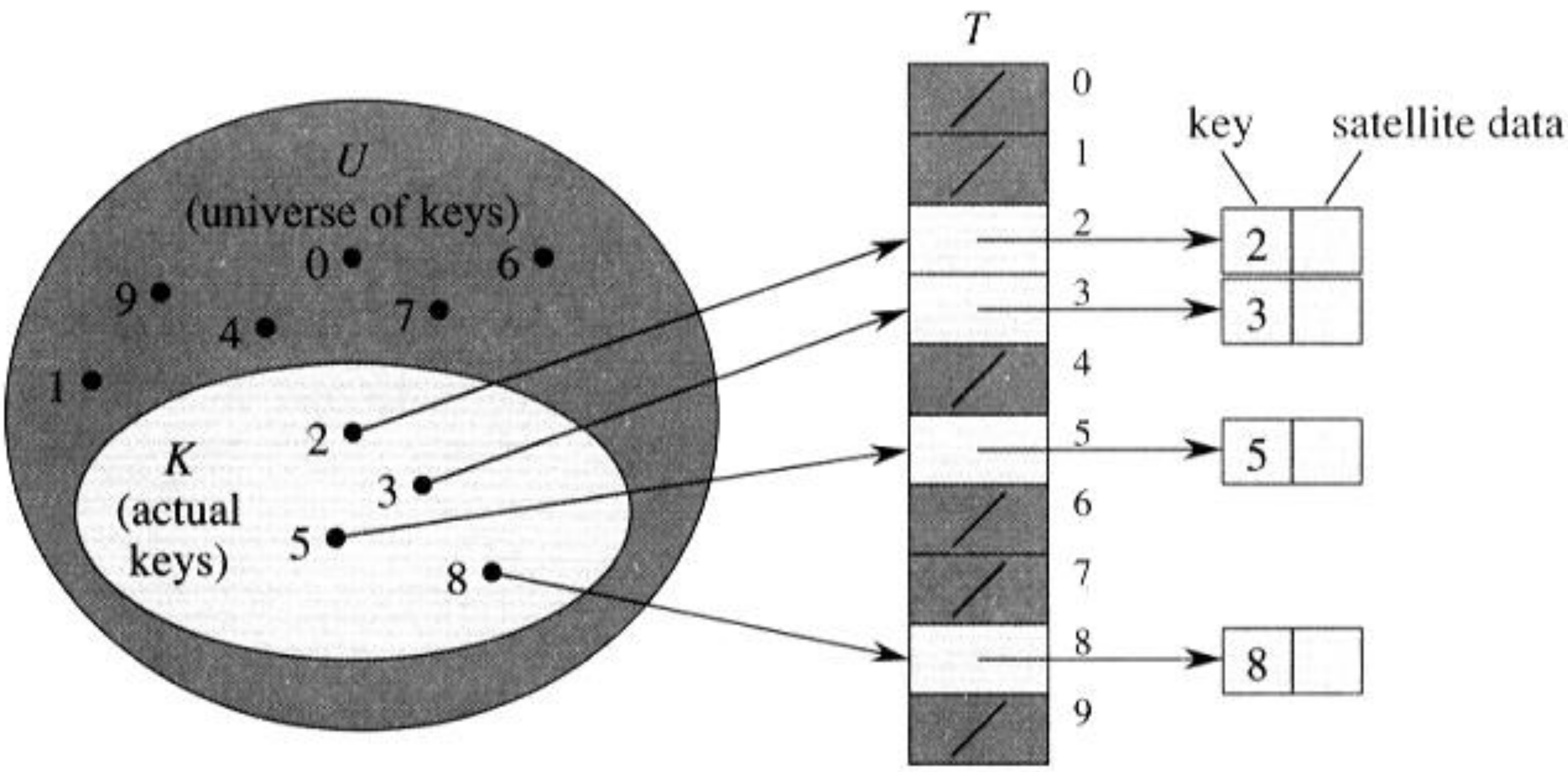
11. Hash Tables

11.1 Directed-address tables

- Direct addressing is a simple technique that works well when the universe U of keys is reasonably small. Suppose that an application needs a dynamic set in which each element has a key drawn from the universe $U = \{0,1,\ldots, m-1\}$ 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**, $T[0,\ldots, m-1]$, in which each position, or **slot**, corresponds to a key in the universe U.

Implementation of direct-address table



- $U = \{0, 1, \ldots, 9\}$, $K = \{2, 3, 5, 8\}$

Functions of direct addressing

DIRECTED-ADDRESS-SEARCH(T, k)
  return $T[k]$

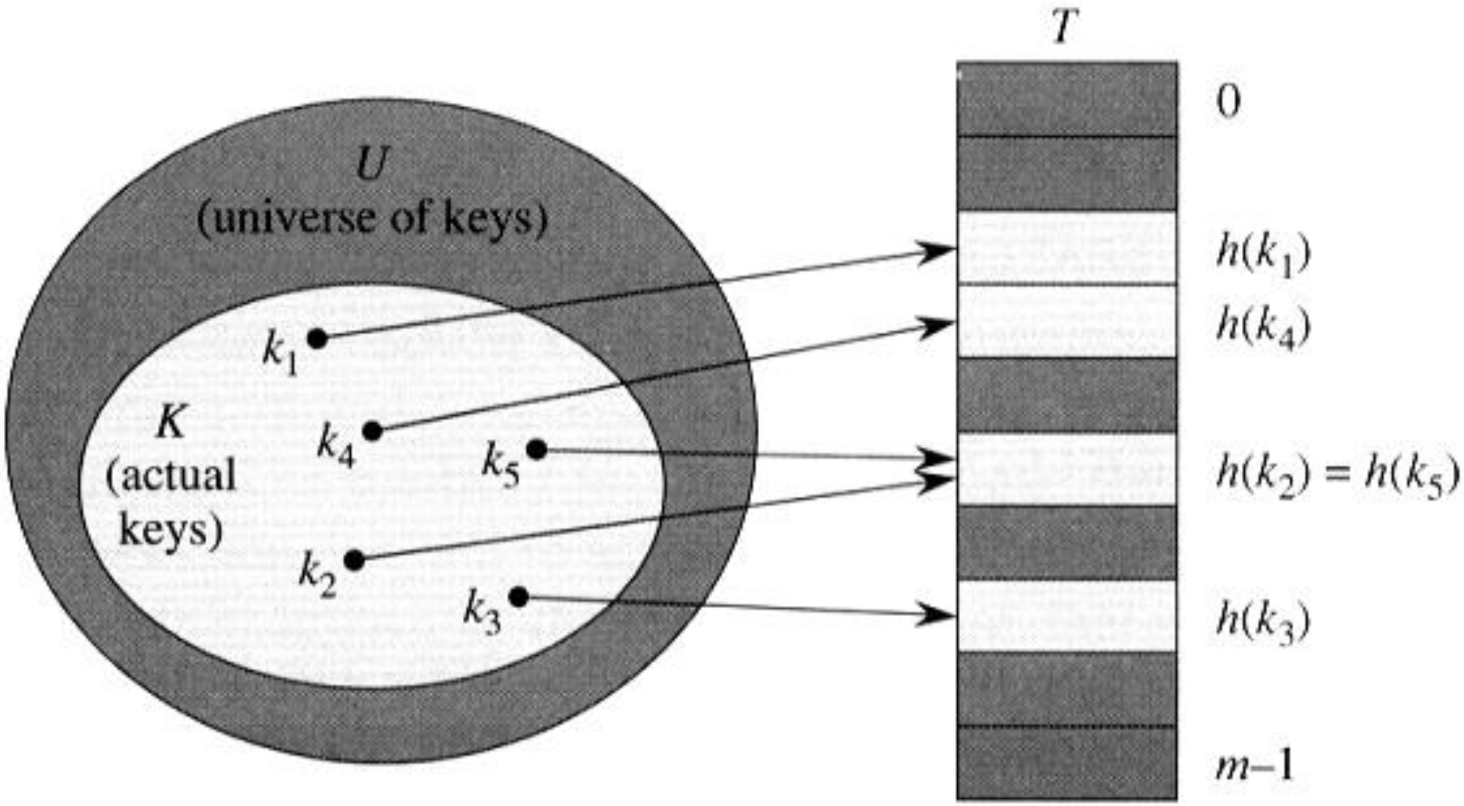
DIRECTED-ADDRESS-INSERT(T, x)
  $T[key[x]] \leftarrow x$

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 (sometime unbounded), 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 $\Theta(|K|)$, while searching for an element in the hash table still requires only $O(1)$ time.

Implementation of hash table



- Using a hash function h to map keys to hash-table slots. Keys k_2 and k_5 map to the same slot, so they collide.

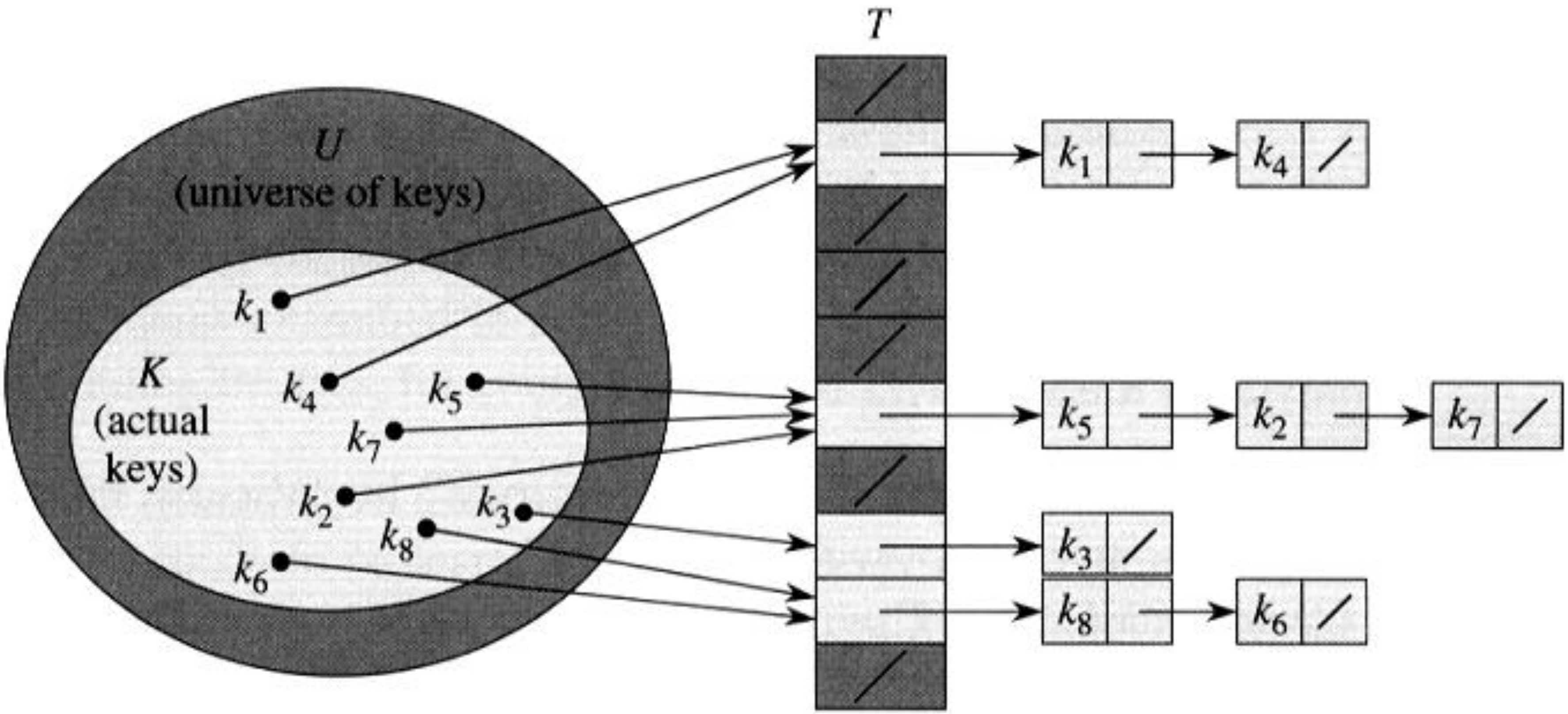
Some terminology and principles

- **Universe set**: U, the set of all possible key values.
- **Hash function**: $h: U \rightarrow \{0, 1, \ldots, m-1\}$
- **Hash table**: $T[0, \ldots, m-1]$
- We say an element with key k hashes to slot $h(k)$; we also say that $h(k)$ is the hash value of key k.
- *Collision*: two keys hashed to the same slot in the hash table.
- *Trade-off*: smaller hash table may introduce more collisions.

Collision resolution techniques:

- Chaining
  - Putting all the elements that hash to the same slot in a linked list.
- Open addressing
  - One element in one position!

Implementation of chained hash



Functions of chained hash

- 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 of chained-hash functions

- INSERT: $O(1)$ (the worst case), assume the element x being inserted is not already present in the table.
- SEARCH: in the worst case $\Rightarrow$ proportional to the length of the list.
- DELETE: $O(1)$, if the list are doubly linked and a pointer to the element is given; similar to the case of searching if the list is *singly* linked.

Analysis of hashing with chaining

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

Assumption: ***simple uniform hashing***

- Simple uniform hashing: any given element is equally likely to hash into any of the m slots, independently of where any other element has hashed to.
- We assume the case of simple uniform hashing; also, computing hashing function takes $O(1)$ time.

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

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

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

Theorem 11.1

- If a hash table in which collisions 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 $\alpha = \dfrac{n}{m}$.
- The expected number of elements examined in an unsuccessful search is α.
- 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.
  - Assume that CHAINED-HASH-INSERT procedure inserts a new element at the front of the list instead of the end.

$$E\left[\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}X_{ij}\right)\right]=\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}E[X_{ij}]\right)$$

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

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

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

$$=1+\frac{1}{nm}\left(n^2-\frac{n(n+1)}{2}\right)$$

$$=1+\frac{\alpha}{2}-\frac{\alpha}{2n}.$$

X_{ij} : the random variable indicates that the i-th and j-th element are hashed into the same slot.

Total time required for a successful search

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

11.3 Hash functions

- What makes a good hash function?

$$\sum_{k:h(k)=j} \Pr(k) = \frac{1}{m} \qquad \text{for } j = 1, 2, \ldots, m$$

- Example:
  - Assume $0 \le k < 1$
  - Set $h(k) = \lfloor km \rfloor$

Interpreting keys as natural number

- In many cases, we can assume the universe of keys is the set **N** = {0, 1, 2, …}.

- Example: ASCII coding

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

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:
  - $m = 2^p$ => $h(k)$ is just the p lowest-order bits of k
  - $m = 2^p - 1$ => $h(k_1) = h(k_2)$ if k_1 is a permutation of k_2 (exercise 11.3-3)

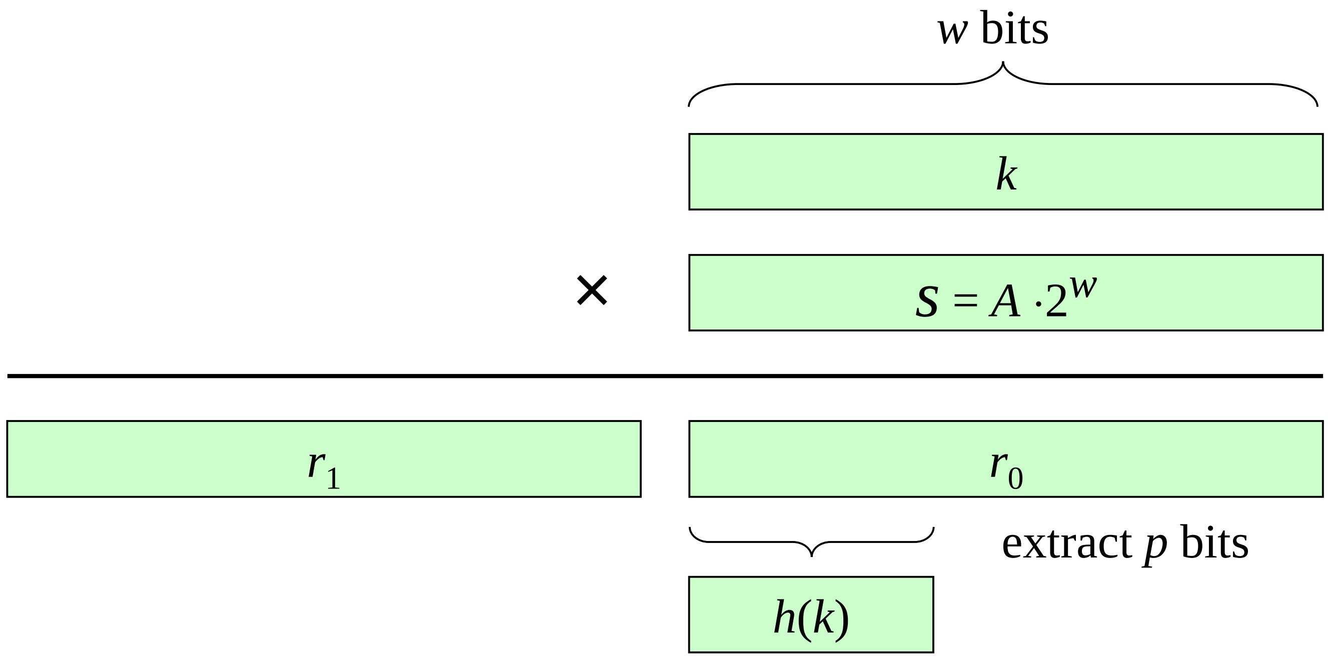
11.3.2 *The multiplication method*

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

where $kA \bmod 1 = kA - \lfloor kA \rfloor$

Suggestion:

choose $m = 2^p,\ A = \dfrac{\sqrt{5}-1}{2}$ (Knuth)



Example:

$$k = 123456 \;,\; p = 14 \,,\, m = 2^{14} = 16384$$

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

$$h(k) = \lfloor 16384 \times (123456 \times A \bmod 1) \rfloor$$

$$= \lfloor 16384 \times 0.004115107 \rfloor$$

$$= \lfloor 67.4219 \rfloor = 67$$

11.4 Open addressing

- All elements are stored in the hash tables itself (no chains).
- $h : U \times \{0,1,\ldots, m-1\} \rightarrow \{0,1,\ldots, m-1\}$.

  With open addressing, we require that for every key k, the ***probe sequence***

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

  be a *permutation* of $\{0,1, \ldots, m-1\}$,

  or at least $h(k, i)\in\{0,1,\ldots, m-1\}\ \forall k, i$

HASH-INSERT(T, k)

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

HASH-SEARCH(T, k)

```
i ← 0
repeat j ← h(k, i)
       if T[j] = k
              then return j
       i ← i + 1
until T[j] = NIL or i = m
return NIL
```

Linear probing:

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

- It suffers the *primary* clustering problem.

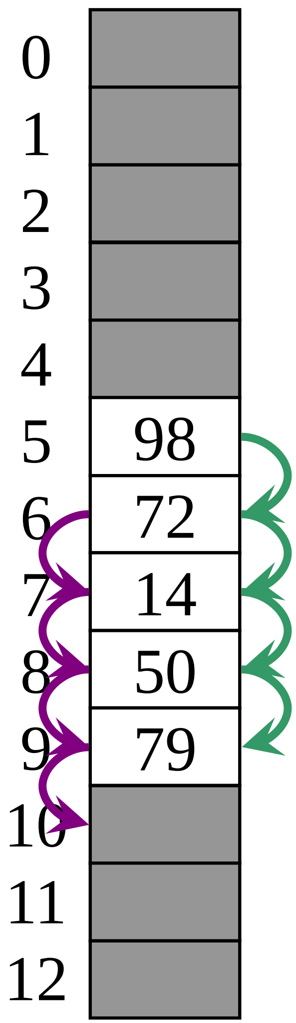
Quadratic probing:

$$h(k, i) = (h(k) + c_1 i + c_2 i^2) \ \mathrm{mod} \ m$$

$$c_1, c_2 \neq 0$$

- It suffers the *secondary* clustering problem.

Linear probing vs. Quadratic probing



$h(k_1, 0) = h(k_2, 0)$

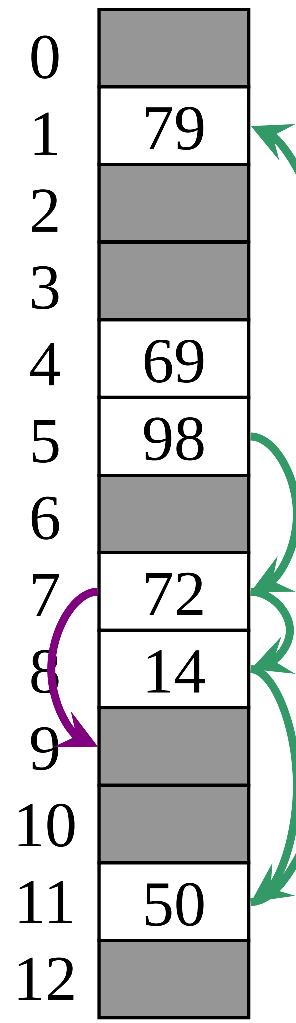
=>

$h(k_1, i) = h(k_2, i)$

$h(k_1, i) = h(k_2, j)$

likely to have

$h(k_1, i+1) = h(k_2, j+1)$



$h(k_1, 0) = h(k_2, 0)$

=>

$h(k_1, i) = h(k_2, i)$

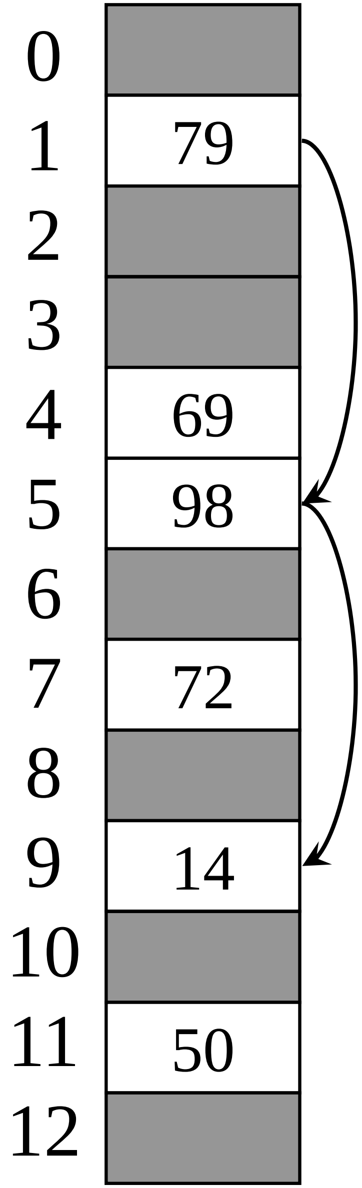
$h(k_1, i) = h(k_2, j)$

unlikely to have

$h(k_1, i+1) = h(k_2, j+1)$

Double hashing:

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



$$h_1(k) = k \text{ mod } 13$$

$$h_2(k) = 1 + (k \text{ mod } 11)$$

INSERT 14

Example:

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

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

Double hashing vs linear or quadratic probing

- Double hashing represents an improvement over linear and quadratic probing in that $\Theta(m^2)$ probe sequence are used, rather than $\Theta(m)$. Its performance is very close to uniform hashing.

Analysis of open-address hashing

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(exactly i probes access occupied slots)

  for $0 \le i \le n$. 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{at least i probes access occupied slots}.

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

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

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

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

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

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

if $1 \le i \le n$

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

$$1 + \sum_{i=0}^{\infty} i p_i = 1 + \sum_{i=1}^{\infty} q_i \le 1 + \alpha + \alpha^2 + \ldots = \frac{1}{1-\alpha}$$

Example:

$$\alpha = 0.1 \qquad \frac{1}{1-\alpha} = 1.1$$

= 5 times

~ 2 times

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

~ 2 times

= 5 times

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

Corollary 11.7

- Inserting an element into an open-address hash table with load factor α 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 at most $1/(1 - \alpha)$.

Theorem 11.8

- 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 uniform hashing and assuming that each key in the table is equally likely to be searched for.

Proof.

- A search for a key k follows the same probe sequence as was followed when the element with key k was inserted.
- 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 $\dfrac{1}{1-\frac{i}{m}} = \dfrac{m}{m-i}$.

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

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

$$\leq \frac{1}{\alpha}\int_{m-n}^{m}(1/x)\,dx$$

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

$$H_i = \sum_{j=1}^{i} 1/j$$

(harmonic numbers)

Example:

$$\alpha = 0.1 \qquad \frac{1}{\alpha}\ln\frac{1}{1-\alpha} \approx 1.054$$

$$\alpha = 0.5 \qquad \frac{1}{\alpha}\ln\frac{1}{1-\alpha} \approx 1.386$$

$$\alpha = 0.9 \qquad \frac{1}{\alpha}\ln\frac{1}{1-\alpha} \approx 2.558$$