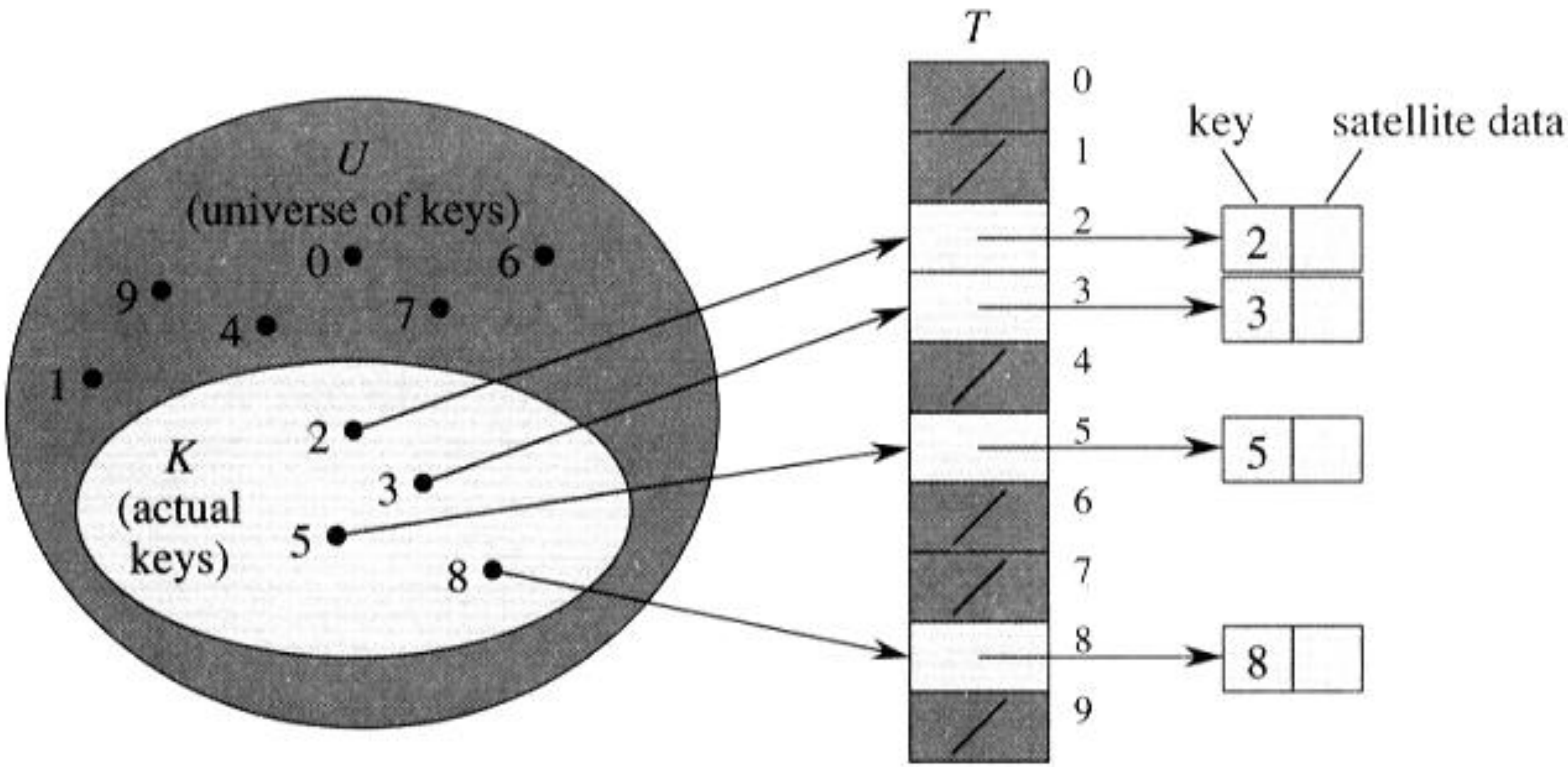
11.Hash Tables

Yu-Shuen Wang, CS, NCTU

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 $U=\{0,1,\ldots,m\text{-}1\}$ 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, $T[0..m\text{-}1]$, in which each position, or slot, corresponds to a key in the universe U.


T
0
1
2
3
4
5
6
7
8
9
key
satellite data
2
3
5
8
U
(universe of keys)
0
6
9
7
4
1
K
(actual
keys)
2
3
5
8

DIRECTED_ADDRESS_SEARCH(T,k)

return $T[k]$

DIRECTED_ADDRESS_INSERT(T,x)

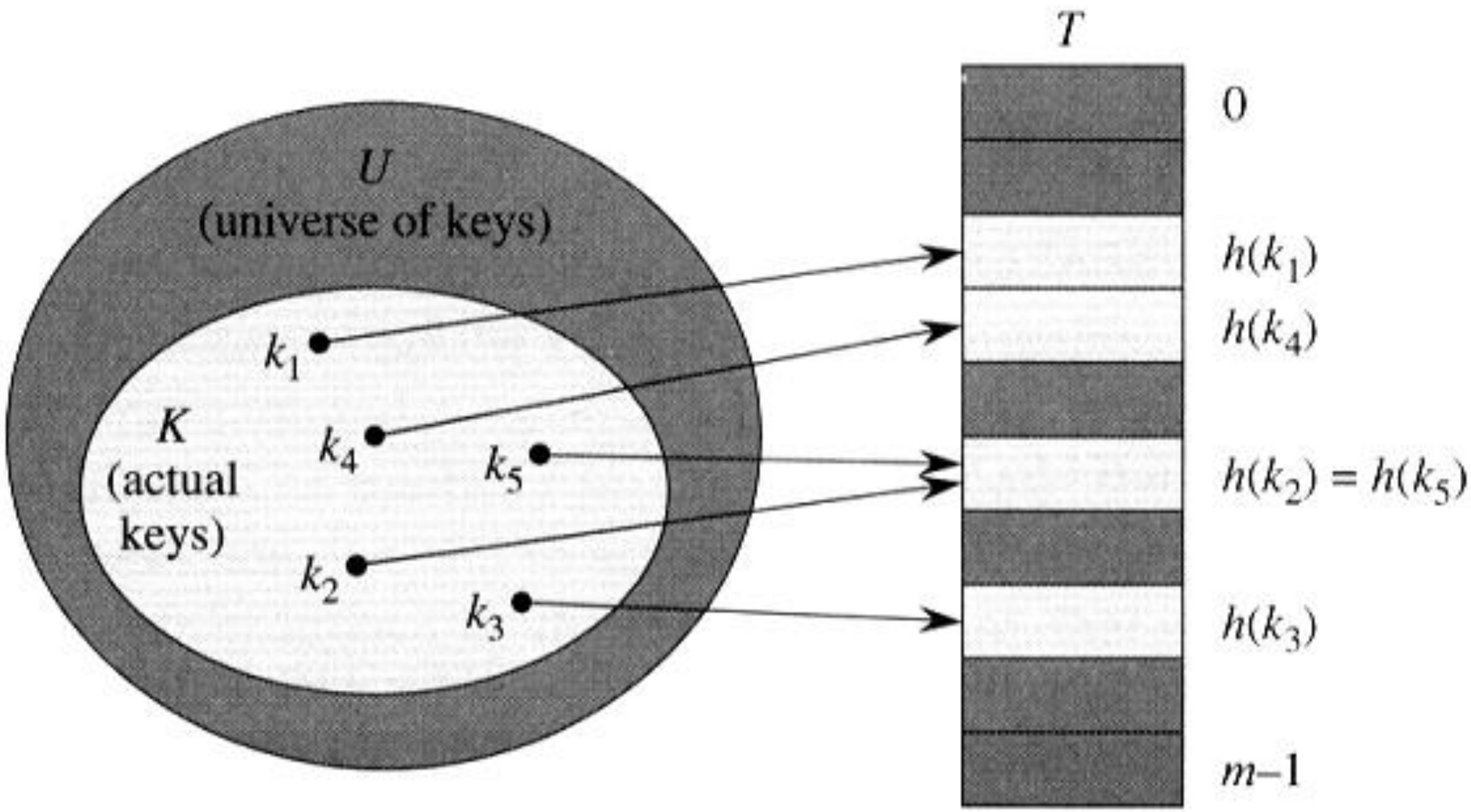
$$T[key[x]] \leftarrow x$$

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 in the hash table still requires only $O(1)$ time.



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

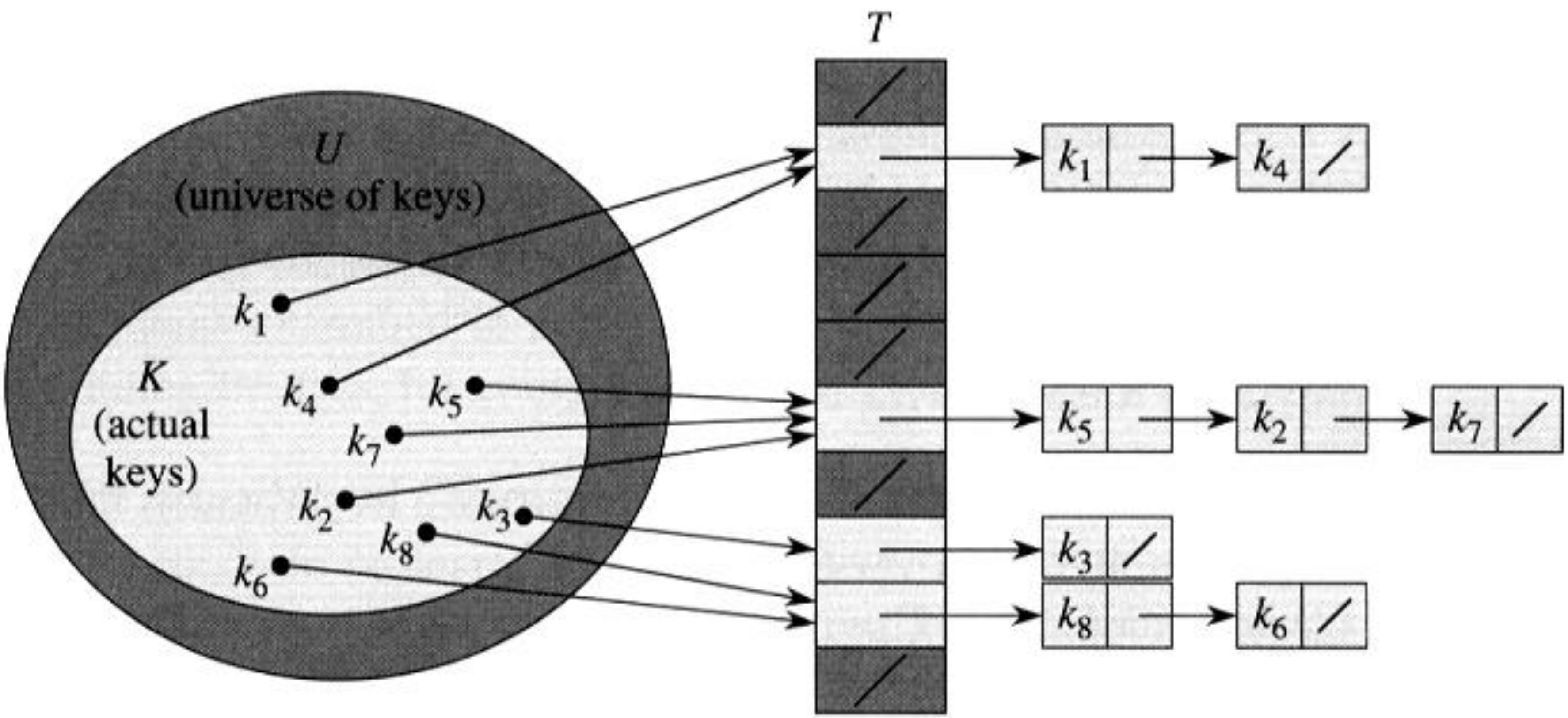
- hash function: $h: U \rightarrow \{0, 1, \ldots, m-1\}$
- hash table: $T[0 \ldots m-1]$
- k hashs to slot: $h(k)$ hash value
- collision: two keys hash to the same slot

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.


T
U
(universe of keys)
K
(actual keys)
k_1
k_4
k_5
k_7
k_2
k_3
k_8
k_6
k_1
k_4
k_5
k_2
k_7
k_3
k_8
k_6

- 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: $\alpha = \dfrac{n}{m}$
(the average number of elements stored in a chain.)

Assumption: ***simple uniform hashing***

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

  for $j = 0, 1, \ldots, m$-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 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 α.
- The total time required (including the time for computing $h(k)$) is $O(1+\alpha)$.

PS: 1 means the time for computing $h(k)$

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.

To find the expected number of elements examined, we take the average, over the n items in the table, of 1 plus the expected length of the list to which the ith element is added. The expected length of that list is (i- 1)/m, and so the expected number of elements examined in a successful search is

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

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

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

$$=1+\frac{\alpha}{2}-\frac{1}{2m}$$

Total time required for a successful search

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

PS: including the time for computing $h(k)$)

11.3 Hash functions

- What makes a good hash function?

Assume that each key is drawn independently from U according to a probability distribution *P*; that is, *P(k)* is the probability that *k* is drawn.

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

Example:

- Assume $0 \le k \le 1$ is uniformly distributed
- Set $h(k) = \lfloor km \rfloor$.

Interpreting keys as natural number

- A key that is a character string can be interpreted as an integer expressed in suitable radix notation

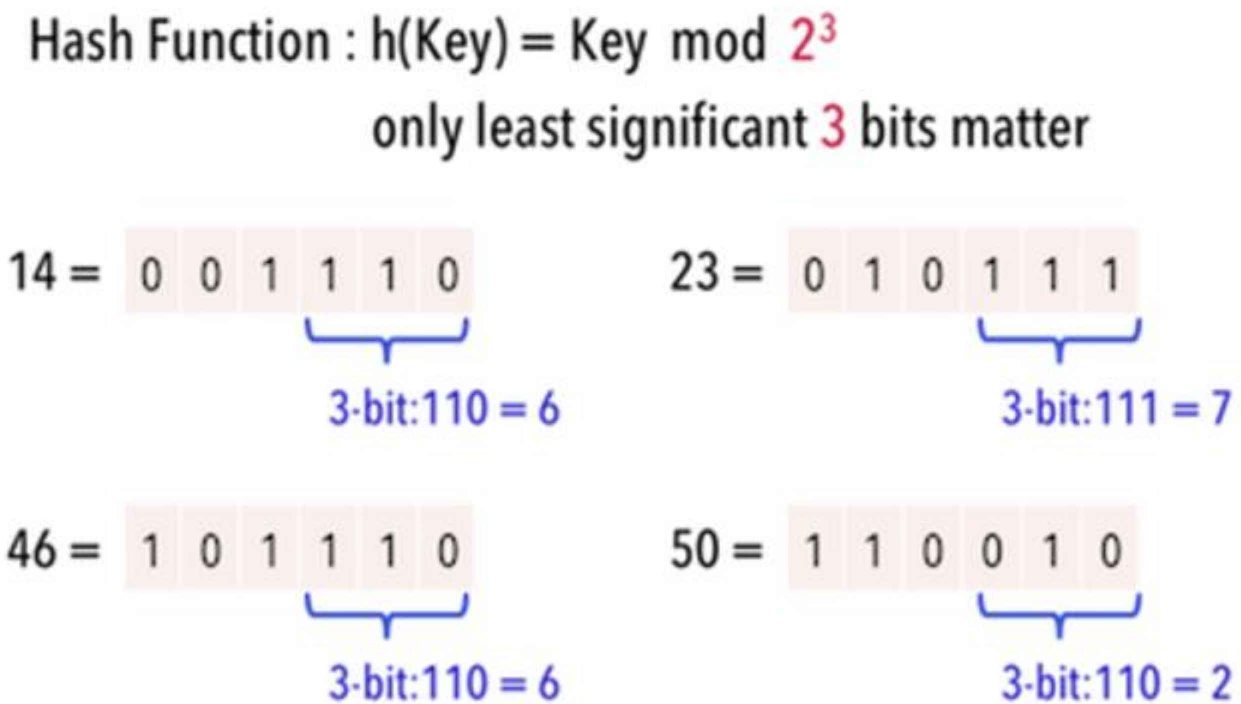
$$pt = (p, t) = (112, 116) = 112 \times 128 + 116 = 14452$$

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.

11.3.1 The division method



Hint: many words such as a_count、b_count、c_count are hashed.

11.3.2 *The multiplication method*

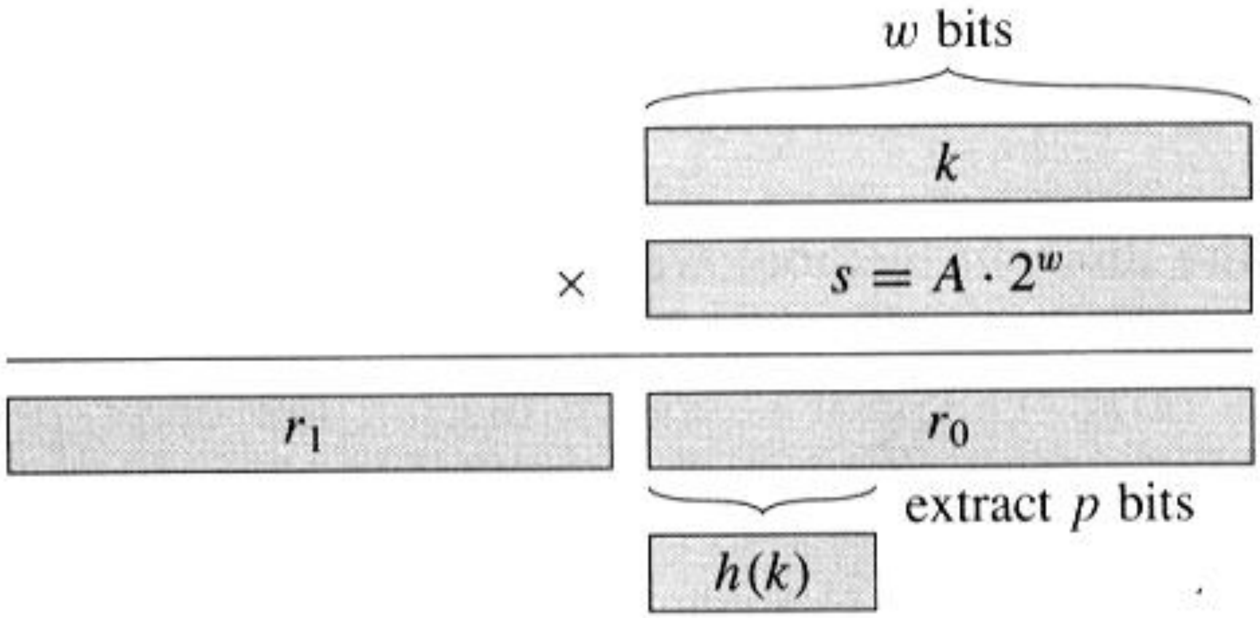
multiply the key k by a constant A in the range $0 < A < 1$ and extract the fractional part of kA. Then, we multiply this value by m and take the floor of the result.

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

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

Suggestion:

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



Example:

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

$$A = \frac{s}{2^{32}} = \frac{\sqrt{5}-1}{2} \approx 0.61803 \ldots \quad , s = 2654435769$$

$$k \times s = 327706022297664 = \left(76300 \times 2^{32}\right) + 17612864$$

$$r_1 = 76300, \qquad r_0 = 17612864$$

The 14 most important bits of r_0 *yield the value* $h(k) = 67$

A weakness of hashing

- Problem: For any hash function h, a set of keys exists that can cause the average access time of a hash table to skyrocket.
  - An adversary can pick all keys from $\{k \in U : h(k) = i\}$ for some slot i.
- IDEA: Choose the hash function at random, independently of the keys.
  - Even if an adversary can see your code, he or she cannot find a bad set of keys, since he or she doesn't know exactly which hash function will be chosen.

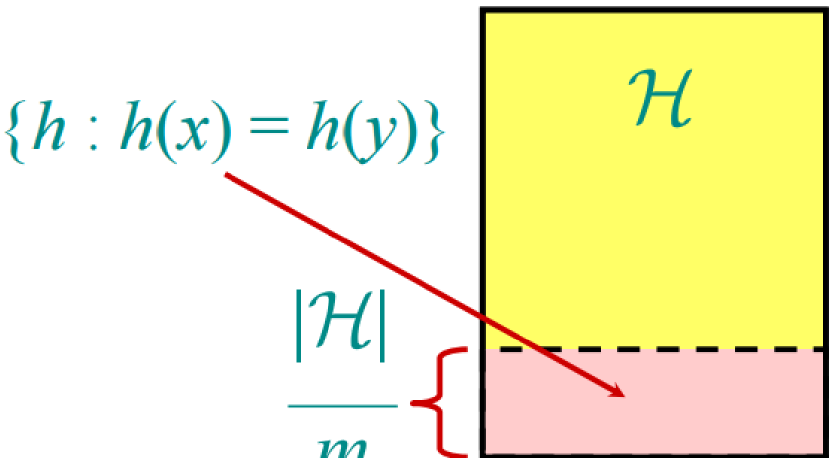
11.3.3 Universal hashing

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

11.3.3 Universal hashing

- Definition. Let U be a universe of keys, and let $\mathcal{H}$ be a finite collection of hash functions, each mapping U to {0, 1, …, m–1}. We say $\mathcal{H}$ is universal if for all x, y ∈ U, where x ≠ y, we have $|\{h \in \mathcal{H} : h(x) = h(y)\}| = |\mathcal{H}| / m$.

That is, the chance of a collision between x and y is 1/m if we choose h $|\mathcal{H}|$ randomly from $\mathcal{H}$.



Universality is good

- Theorem. Let h be a hash function chosen (uniformly) at random from a universal set $\mathcal{H}$ of hash functions. Suppose h is used to hash n arbitrary keys into the m slots of a table T. Then, for a given key x, we have
  - E[#collisions with x] < n / m.

Proof of theorem

- Proof. Let C_x be the random variable denoting the total number of collisions of keys in T with x, and let
- $C_{xy} = \begin{cases} 1 & \text{if h (x) = h(y) ,} \\ 0 & \text{otherwise} \end{cases}$
- Note: $E\ [c_{xy}]\ =\ 1/m$ and $C_x = \sum_{y=T-\{x\}} C_{xy}$

Proof.

$$E[C_x] = E\left[\sum_{y \in T-\{x\}} c_{xy}\right]$$

- Take expectation of both sides.

$$= \sum_{y \in T-\{x\}} E[c_{xy}]$$

- Linearity of expectation.

$$= \sum_{y \in T-\{x\}} 1/m$$

- $E[c_{xy}] = 1/m$.

$$= \frac{n-1}{m}. \quad \square$$

- Algebra.

Constructing a set of universal hash functions

- Let m be prime. Decompose key k into r + 1 digits, each with value in the set {0, 1, …, m–1}. That is, let $k = \langle k_0, k_1, \ldots, k_r \rangle$, where $0 \le k_i < m$.
- Randomized strategy:
  - Pick $a = \langle a_0, a_1, \ldots, a_r \rangle$ where each a_i is chosen randomly from {0, 1, …, m–1}.

Define $h_a(k) = \sum_{i=0}^{r} a_i k_i \bmod m$. ***Dot product, modulo m***

How big is $\mathcal{H} = \{h_a\}$? $|\mathcal{H}| = m^{r+1}$. ← **REMEMBER THIS!**

Universality of dot-product hash functions

- Theorem. The set $\mathcal{H} = \{h_a\}$ is universal.
- Proof. Suppose that x = ⟨x_0, x_1, …, x_r⟩ and y = ⟨y_0, y_1, …, y_r ⟩ be distinct keys. Thus, they differ in at least one digit position, without the loss of generality, position 0. For how many $h_a \in \mathcal{H}$ do x and y collide?
- We must have $h_a(x) = h_a(y)$, which implies that

$$\sum_{i=0}^{r} a_i x_i \equiv \sum_{i=0}^{r} a_i y_i \qquad (mod \;\; m)$$

Universality of dot-product hash functions

$$\sum_{i=0}^{r} a_i x_i \equiv \sum_{i=0}^{r} a_i y_i \qquad (mod\ \ m)$$

$$\sum_{i=0}^{r} a_i (x_i - y_i) \equiv 0 \qquad (mod\ \ m)$$

$$a_0 (x_0 - y_0) \equiv -\sum_{i=1}^{r} a_i (x_i - y_i) \qquad (mod\ \ m)$$

Fact from number theory

Theorem. Let m be prime. For any $z \in \mathbb{Z}_m$ such that $z \neq 0$, there exists a unique $z^{-1} \in \mathbb{Z}_m$ such that

$$z \cdot z^{-1} \equiv 1 \pmod{m}.$$

Example: $m = 7$.

z	1	2	3	4	5	6
z^{-1}	1	4	5	2	3	6

Back to proof

- We have

$$a_0(x_0 - y_0) \equiv -\sum_{i=1}^{r} a_i(x_i - y_i) \pmod{m}$$

- and since $x_0 \neq y_0$, an inverse $(x_0 - y_0)^{-1}$ must exist, which implies that

$$a_0 \equiv \left(-\sum_{i=1}^{r} a_i(x_i - y_i)\right)(x_0 - y_0)^{-1} \pmod{m}$$

- Thus, for any choices of $a_1, a_2, \ldots, a_r$, exactly one choice of a_0 causes x and y to collide.

Back to proof

- Q. How many h_a's cause x and y to collide?
- A. There are m choices for each of $a_1, a_2, \ldots, a_r$, but once these are chosen, exactly one choice for a_0 causes x and y to collide, namely

$$a_0 = \left(-\sum_{i=1}^{r} a_i(x_i - y_i)\right)(x_0 - y_0)^{-1} \pmod{m}$$

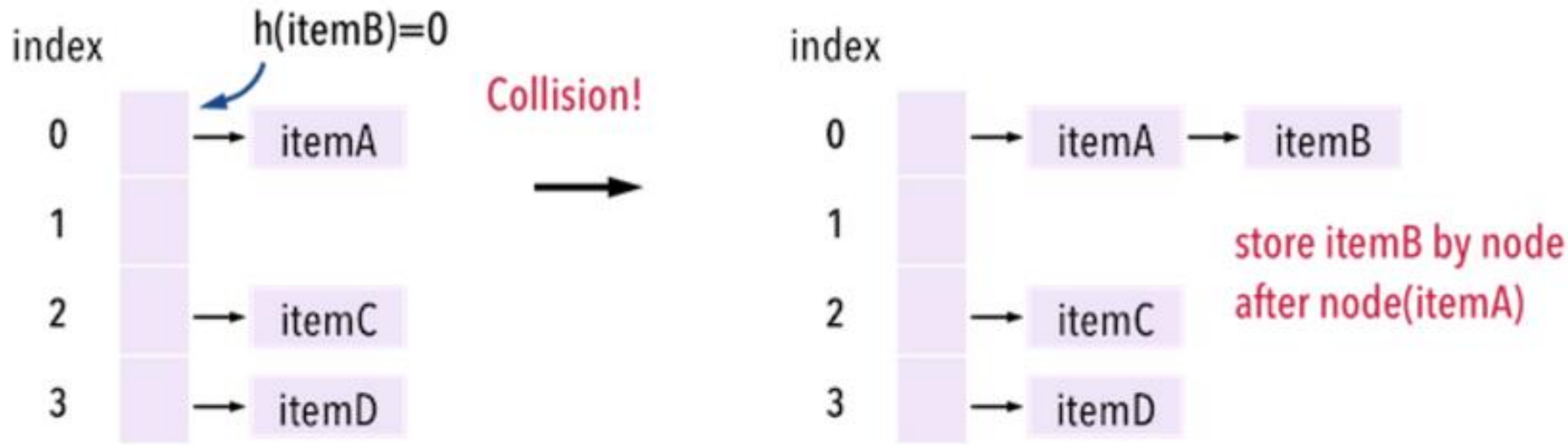
- Thus, the number of h's that cause x and y to collide is $m^r \cdot 1 = |\mathcal{H}|/m$

11.4 Open addressing

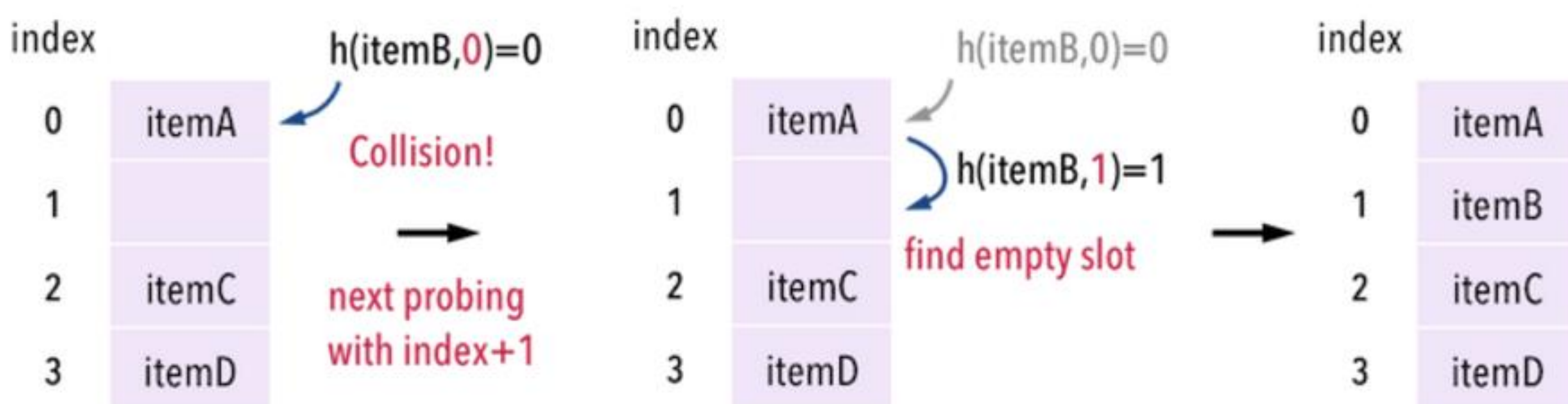
- (All elements are stored in the hash tables itself.)
- $h : \mathrm{U} \times \{0,1,\ldots,m\text{-}1\} \rightarrow \{0,1,\ldots,m\text{-}1\}$.

  With open addressing, we require that for every key k, the ***probe sequence***
  $\langle h(k,0), h(k,1), \ldots, h(k,m\text{-}1) \rangle$
  be a permutation of $\{0,1, \ldots, m\}$.

Chaining: insert itemB with h(itemB)=0



Open Addressing: insert itemB with h(itemB,0)=0



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)

$i \leftarrow 0$

repeat $j \leftarrow h(k, i)$

        if $T[j] = k$

    then return j

        $i \leftarrow i + 1$

until $T[j] = \mathrm{NIL}$ or $i = m$

return NIL

Linear probing:

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

- It suffers the primary clustering problem.

Linear Probing:

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

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

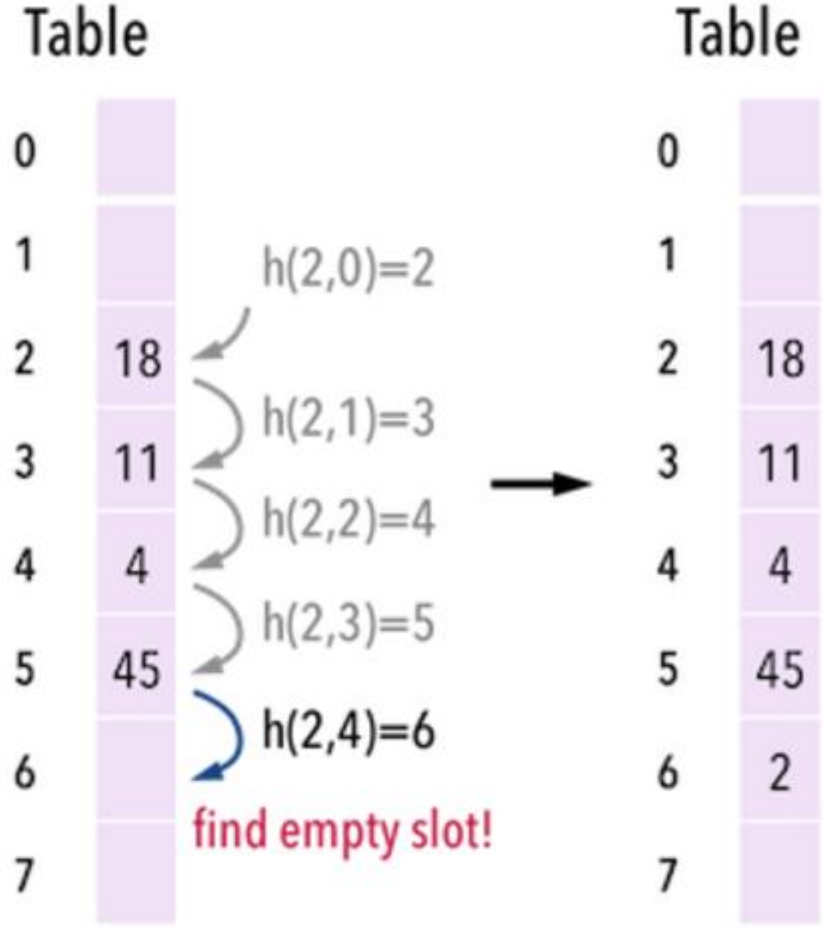
$\quad = (\, (k \bmod m) + i \,) \bmod m$

now, $m=8$, $k=2$

$h(2, i) = (\, 2 + i \,) \bmod 8$

for $i = 0 \sim 7$,

$h(2, i) = \{\, 2, 3, 4, 5, 6, 7, 0, 1 \,\}$



Quadratic probing:

$$h(k,i) = (h(k) + c_1 i + c_2 i^2) \bmod m$$

$$c_1, c_2 \neq 0$$

- It suffers the secondary clustering problem $(h(k_1)=h(k_2), k_1 \neq k_2)$.
- Note that not all c_1, c_2, and m can produce the permutation of $\{0,1,\ldots,m-1\}$

Quadratic probing:

- $c_1=c_2=0.5$, $m=2^P$

$$h(k,i) = (h(k) + c_1 i + c_2 i^2) \bmod m$$

- The values of $c_1 i + c_2 i^2$
  - 1, 3, 6, 10, 15, 21, 28
- If m=8, the probing sequence is
  - 1, 3, 6, 2, 7, 5, 4

Quadratic Probing:

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

$h(k, i) = (\, h'(k) + 0.5i + 0.5i^2 \,) \bmod m$

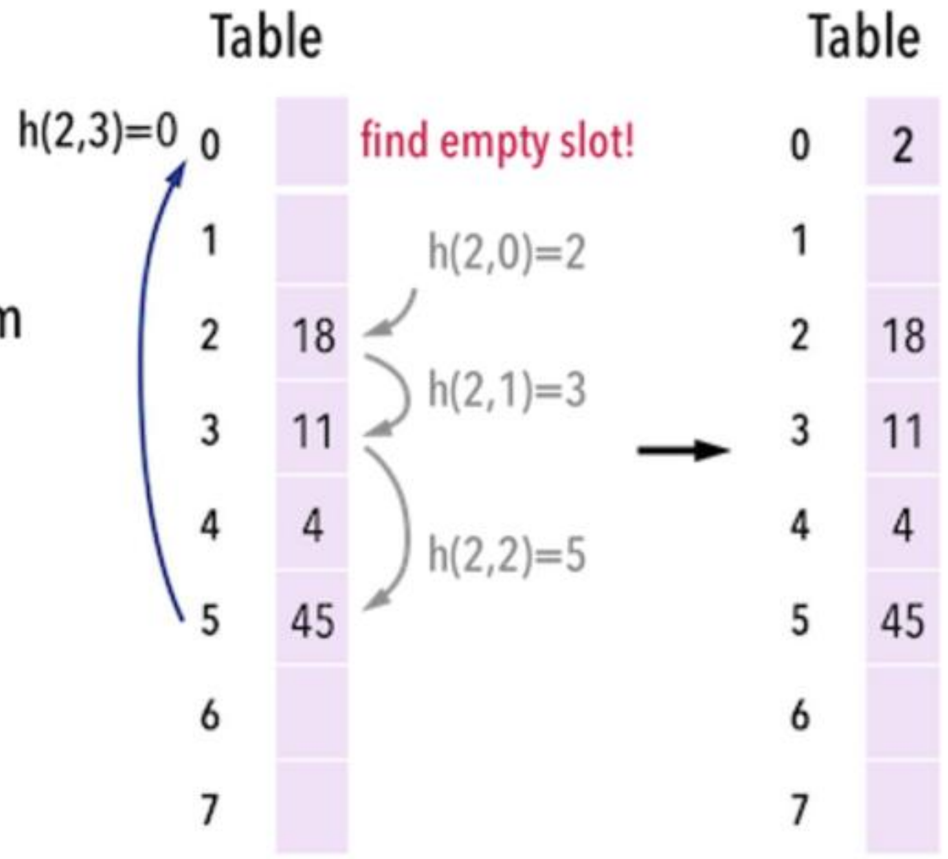
$\quad = (\, (\, k \bmod m \,) + 0.5i + 0.5i^2 \,) \bmod m$

now, $m=8$, $k=2$

$h(2, i) = (\, 2 + i \,) \bmod 8$

for $i = 0 \sim 7$,

$h(2, i) = \{\, 2, 3, 5, 0, 4, 1, 7, 6 \,\}$

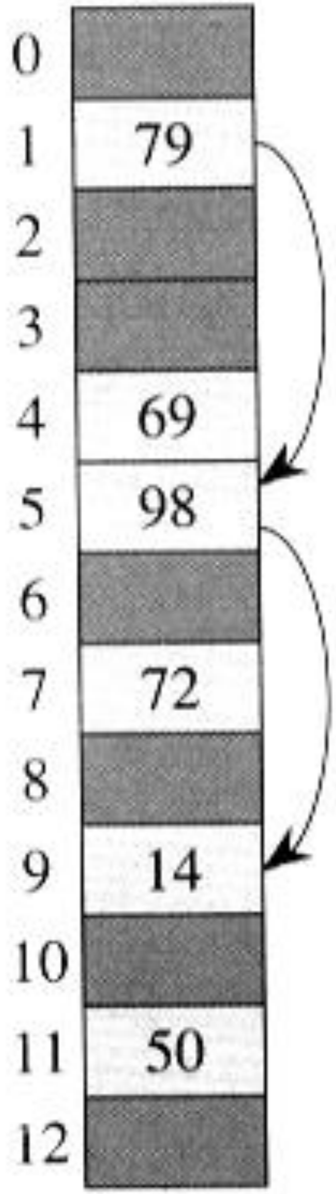


Double hashing:

$$h(k, i) = \left(h_1(k) + i \cdot h_2\,(k)\right) \quad mod \quad m$$

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

$$h(k,i) = \left(h_1(k) + i \cdot h_2(k)\right) \quad mod \quad m$$



Example:

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

$$h_2(k) = 1 + (k \quad mod \quad n)$$

$$Insert \qquad 14$$

$$h_1(k) = (k\, mod\, 13) = 1$$

$$h_2(k) = 1 + (k\, mod\, 11) = 4$$

h(k,i) = (5, 9, 0, 4, 8,…)

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.

Example:

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

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

When a random variable X takes on values from the natural numbers N = {0,1, 2, . . .}, there is a nice formula for its expectation:

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

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

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

since each term $\Pr\{X \geq i\}$ is added in i times and subtracted out $i - 1$ times (except $\Pr\{X \geq 0\}$,which is added in 0 times and not subtracted out at all).

For example

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

$i = 0$: $\quad 0(Pr\{X \geq 0\} - Pr\{X \geq 1\})$

$i = 1$: $\quad 1(Pr\{X \geq 1\} - Pr\{X \geq 2\})$

$i = 2$: $\quad 2(Pr\{X \geq 2\} - Pr\{X \geq 3\})$

$i = 3$: $\quad 3(Pr\{X \geq 3\} - Pr\{X \geq 4\})$

...

Proof.

- In an unsuccessful search, every probe but the last accesses an occupied slot that does not contain the desired key, and the last slot probed is empty.
- Define $p_i = \Pr\{\text{exactly } i \text{ probes access occupied slots}\}$ for $\mathrm{i} = 0,1,2, \dots, i \leq n$, and $p_i = 0 \;\; if \;\; i > n$
- The expected number of probes is $1 + \sum_{i=0}^{\infty} ip_i$
- Define $\mathrm{q_i} = \Pr\{\text{at least } i \text{ probes access occupied slots}\}$

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

The probability that the first probe accesses an occupied slot is $q_1 = \frac{n}{m}$

The second probe only if the first probe accesses an occupied slot; thus $q_1 = \frac{n}{m}\frac{n-1}{m-1}$

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

if $1 \leq \ i \leq n$

- $q_i = 0$, for $i > n$ *(all sluts are occupied)*.
- $$\mathbf{1} + \sum_{\boldsymbol{i}=\mathbf{0}}^{\infty} \boldsymbol{i}\boldsymbol{p}_{\boldsymbol{i}} = \mathbf{1} + \sum_{\boldsymbol{i}=\mathbf{0}}^{\infty} \boldsymbol{q}_{\boldsymbol{i}} \leq \mathbf{1} + \boldsymbol{\alpha} + \boldsymbol{\alpha}^{\mathbf{2}} + \cdots = \frac{\mathbf{1}}{\mathbf{1} - \boldsymbol{\alpha}}$$

Corollary 11.7

- Inserting an element into an open-address hash table with load factor $\boldsymbol{\alpha}$ requires at most $\frac{\mathbf{1}}{\mathbf{1}-\boldsymbol{\alpha}}$ 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 $\frac{1}{\alpha} ln\left(\frac{1}{1-\alpha}\right) + \frac{1}{\alpha}$ assuming uniform hashing and assuming that each key in the table is equally likely to be searched for.

Example:

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

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

Proof.

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

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

$$\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i}$$

Harmonic number

$$\boldsymbol{H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k}}$$

$$\boldsymbol{ln\,(n) \le H_n \le ln\,(n) + 1}$$

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

$$=\frac{1}{\alpha}(H_m-H_{m-n})\leq\frac{1}{\alpha}\big(ln(m)+1-ln(m-n)\big)$$

$$=\frac{1}{\alpha}\left(ln\left(\frac{m}{m-n}\right)+1\right)=\frac{1}{\alpha}ln\left(\frac{1}{1-\alpha}\right)+\frac{1}{\alpha}$$