Hash table

Speaker : MARK

Outline

- **Introduction**
  - Direct addressing table
  - Hash table
- **Hash function**
  - Division
  - Mid-square
  - Folding
- **Collision & Overflow handing**
  - Chaining
  - Open addressing

Introduction

- Many applications require a dynamic set *S* to supports the following dictionary operations:
  - Search(*k*): check if *k* is in *S*
  - Insert(*k*): insert *k* into *S*
  - Delete(*k*): delete *k* from *S*
- Hash table: an effective data structure for implementing dictionaries

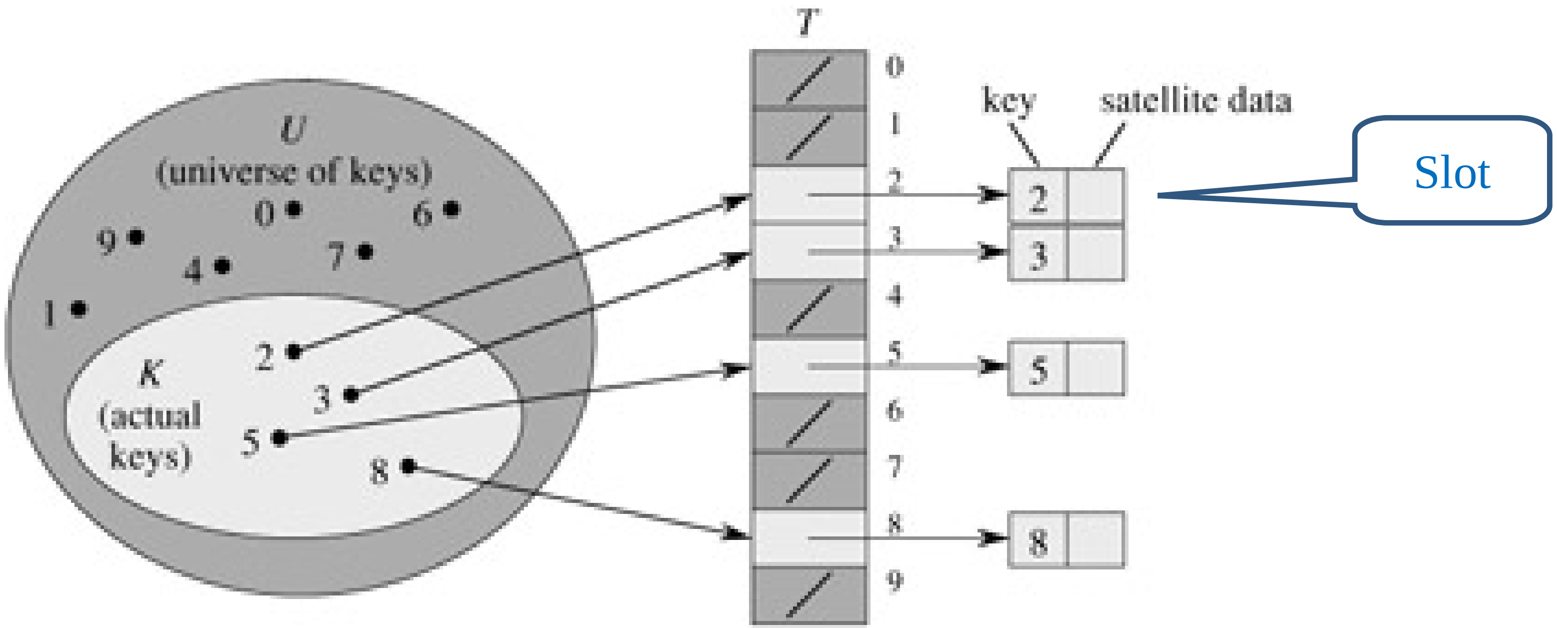
Definitions

- ***U*** : a set of universe keys
- ***K*** : a dynamic set of actual keys
  - Like an application needs in which each element has a key drawn from the universe $U = \{0, 1, ..., m\text{-}1\}$
- ***T*** : the table denoted by $T[0 \sim m\text{-}1]$,
  - in which each position, or slot, corresponds to a key in the universe *U* **.**

Direct addressing table

◈ Ex.

Key = 2	Name = John	…	…	…



◈ Search time = Insert time = Delete time = O(1)

Direct addressing table

- The difficulty with direct addressing is obvious:
- The table T size = $O(|U|)$
- If $|K| << |U|$, then use too much spaces.

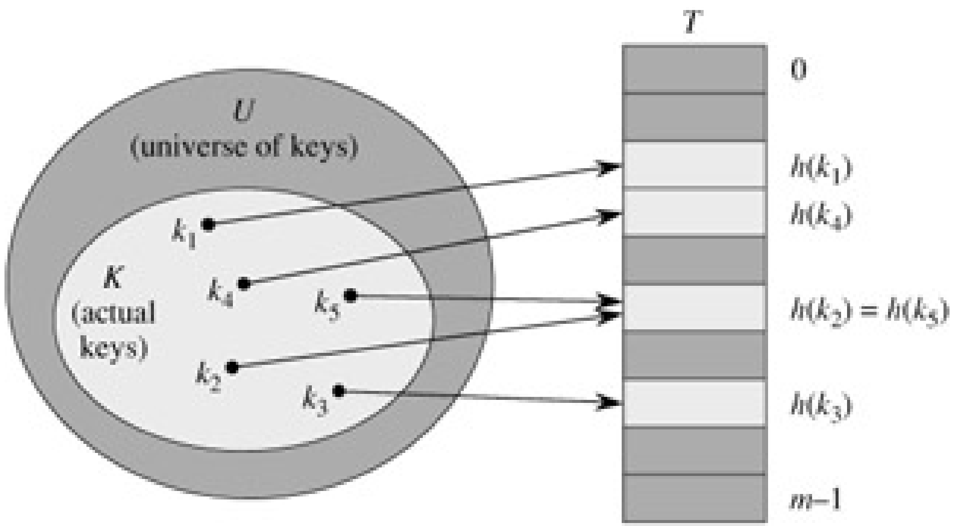
- Time is money ! Space is money, too !?

What is hashing ?

- ***Hashing*** has following advantages:
  - Use hashing to search, data need not be sorted
  - Without collision & overflow, search only takes O(1) time. Data size is not concerned
  - Security. If you do not know the hash function, you cannot get data

Hash table

- With direct addressing ,
  - an element with key *k* is stored in slot *k*
- With hashing ,
  - this element is stored in slot *h(k)*



Hash function

- A good hash function satisfies (approximately) the assumption of ***simple uniform hashing*** :

  Each key is equally likely to hash to *any of the m slots*, independently of where any other key has hashed to.

Hash function

- For example, if the keys ***k*** are known to be *random real numbers* independently and uniformly distributed in the range $0 \leq k < 1$, the hash function

$$h(k) = \lfloor km \rfloor$$

satisfies the condition of simple uniform hashing.

Hash function

- *Interpreting keys as natural numbers*
- Most hash functions assume that the universe of keys is the set N = {0, 1, 2, ...} of natural numbers.
- Ex. Key ‘pt’
  - p = 112 & t = 116 in ASCII table
  - as a radix-128 integer,
    ‘pt’ = (112·128) + 116 = 14452

(1) Division

- Mapping a key k into one of m slots by taking the remainder of k divided by m
- $h(k) = k \bmod m$
- Ex. $m = 12$, $k = 100$, then $h(k) = 4$
- Prime number m may be good choice !

(2) Mid-square

- Mapping a key k into one of m slots by get the middle some digits from value k^2
- h(k) = k^2 get middle (log m) digits
- Ex. m = 10000, k = 113586, log m = 4

h(k) = 113586^2 get middle 4 digits

= 12901779369 get middle 4 digits

= 1779

(3) Folding

- Divide k into some sections, besides the last section, have same length . Then add these sections together.
  - a. shift folding
  - b. folding at the boundaries
- H(k) = $\sum$(section divided from k) by a or b

(3) Folding

- Ex, k = 12320324111220, section length = 3

P1	P2	P3	P4	P5
1 2 3	2 0 3	2 4 1	1 1 2	2 0

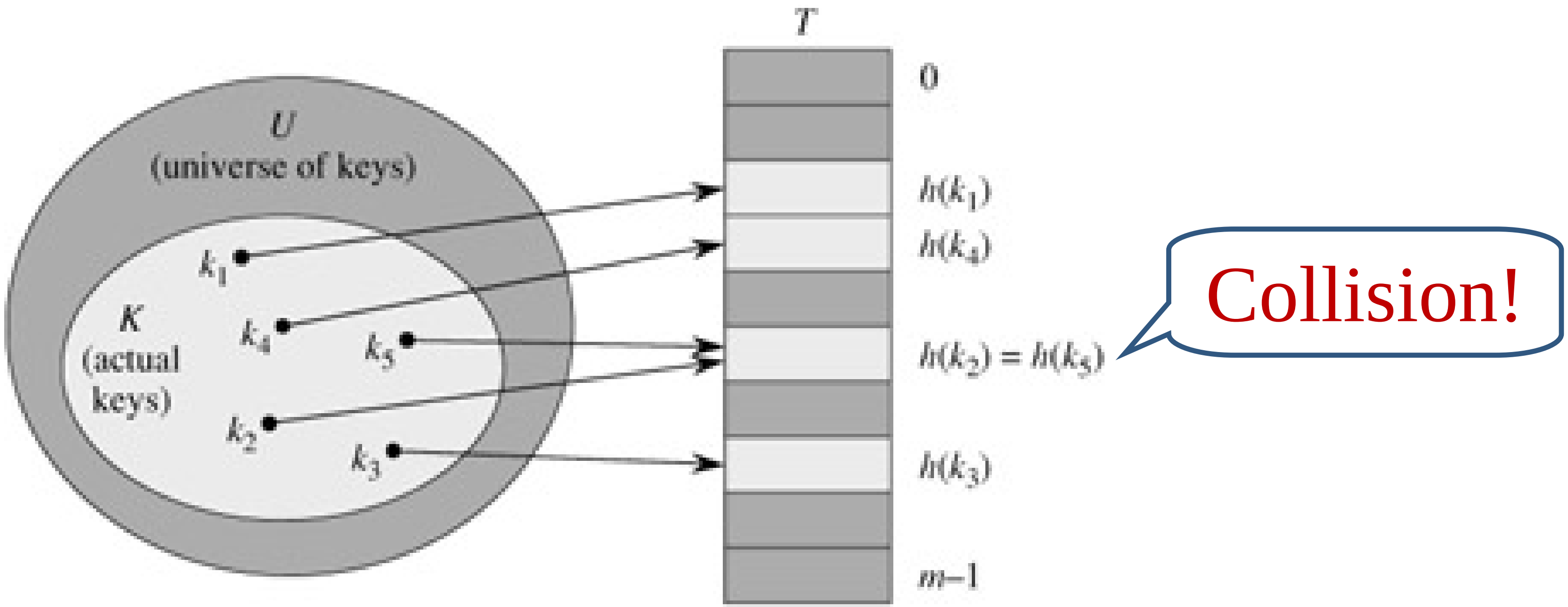
P1	123
P2	203
P3	241
P4	112
P5	20
	879

shift folding

P1	123
P2	302
P3	241
P4	211
P5	20
	897

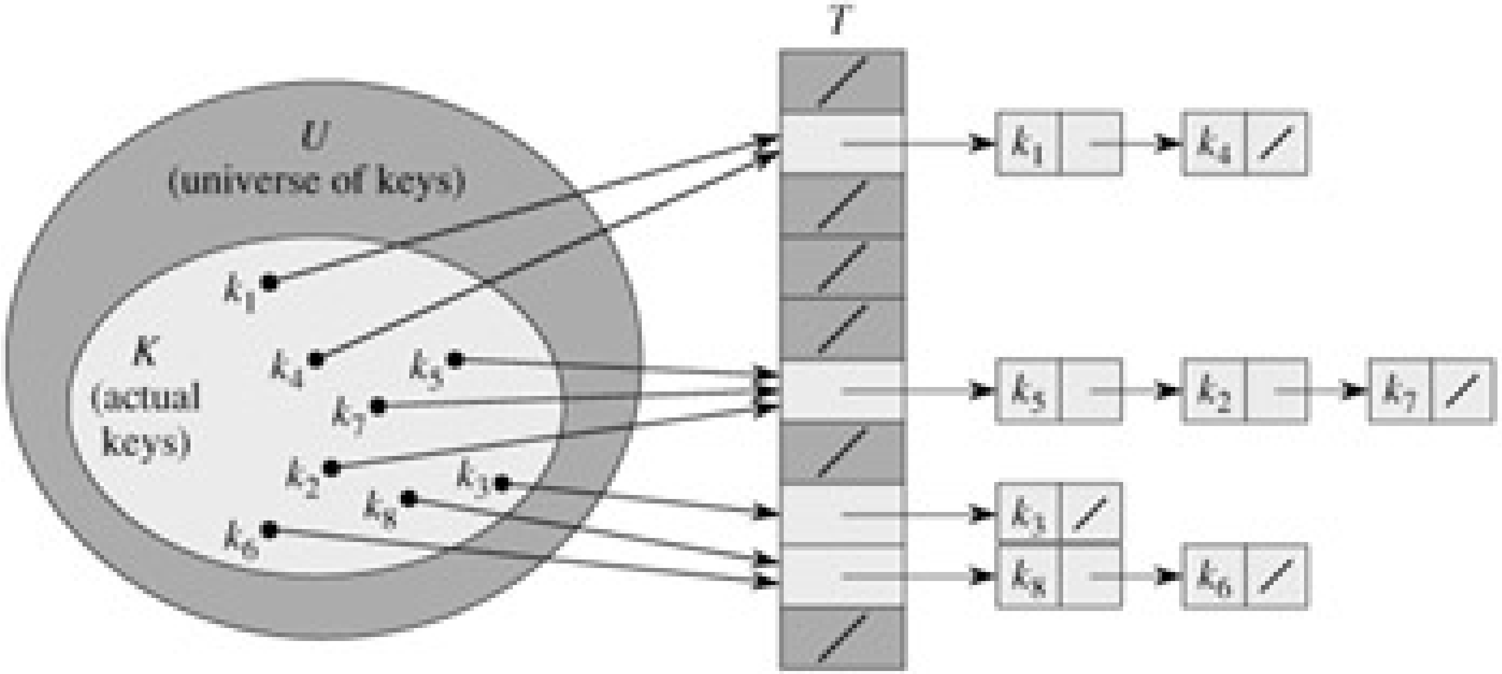
folding at the boundaries

Collision & Overflow handing



(1) Chaining

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



(1) Chaining analysis

- Worst-case insert time = O(1)
  - insert into the beginning of each link list
- Worst-case search time = Θ(n)
  - Every key mapping to the same slot

    Ex. h(1) = h(2) = h(3) = … = h(n) = x

    then search key '1'

(1) Chaining analysis

- For j = 0, 1, ..., m-1, let us denote the length of the list T[j] by n_j , so that

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

- the average value of n_j is $E[n_j] = \alpha = n/m$.
- Average search time = $\Theta(1 + \alpha)$

(1) Chaining analysis

- Unsuccessful search time = $\Theta(1 + \alpha)$
- 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 has* expected length $E[n_{h(k)}] = \alpha$.

(1) Chaining analysis

- Successful search time = $\Theta(1 + \alpha)$
- The situation for a successful search is slightly different, since *each list is not equally likely to be searched.*
- Instead, the probability that a list is searched is *proportional to the number* of elements it contains.

(1) Chaining analysis

- For keys k_i and k_j , we define *indicator random variable* $\boldsymbol{X_{ij}} = \mathrm{I}\{h(k_i) = h(k_j)\}$
- Under the assumption of simple uniform hashing, we have $Pr\{h(k_i) = h(k_j)\} = 1/m$, *and* $\boldsymbol{E[X_{ij}]} = 1/m$
- The expected number of elements examined in a *successful search* is :

(1) Chaining analysis

$$
\begin{aligned}
\mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n} X_{ij}\right)\right] \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n} \mathrm{E}\left[X_{ij}\right]\right) \quad \text{(by linearity of expectation)} \\
&= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n} \frac{1}{m}\right) \\
&= 1+\frac{1}{nm}\sum_{i=1}^{n}(n-i) \\
&= 1+\frac{1}{nm}\left(\sum_{i=1}^{n} n-\sum_{i=1}^{n} i\right) \\
&= 1+\frac{1}{nm}\left(n^2-\frac{n(n+1)}{2}\right) \quad \text{(by equation (A.1))} \\
&= 1+\frac{n-1}{2m} \\
&= 1+\frac{\alpha}{2}-\frac{\alpha}{2n} .
\end{aligned}
$$

$\Theta(2 + \alpha/2 - \alpha/2n) =$

$\boldsymbol{\Theta(1 + \alpha)}$

(1) Chaining analysis

- $\Theta(1 + \alpha)$ means ?
- If the number of hash-table slots is at least proportional to the number of elements in the table, we have

  $n = O(m)$ *and*, $\alpha = n/m = O(m)/m = O(1)$.

- *Thus, searching takes* ***constant time*** *on average.*

(2) Open addressing

- In open addressing, all elements are stored in the hash table itself.
- That is, each table slot contains either an element of the dynamic set or NIL.
- The hash table can "fill up"

  => no further insertions can be made;
- load factor $\alpha = n/m \leq 1$.

(2) Open addressing

- The assumption of ***uniform hashing :***

  we assume that each key is equally likely to have any of the *m! permutations of*

  $<0, 1, ..., m-1>$ *as its probe sequence.*

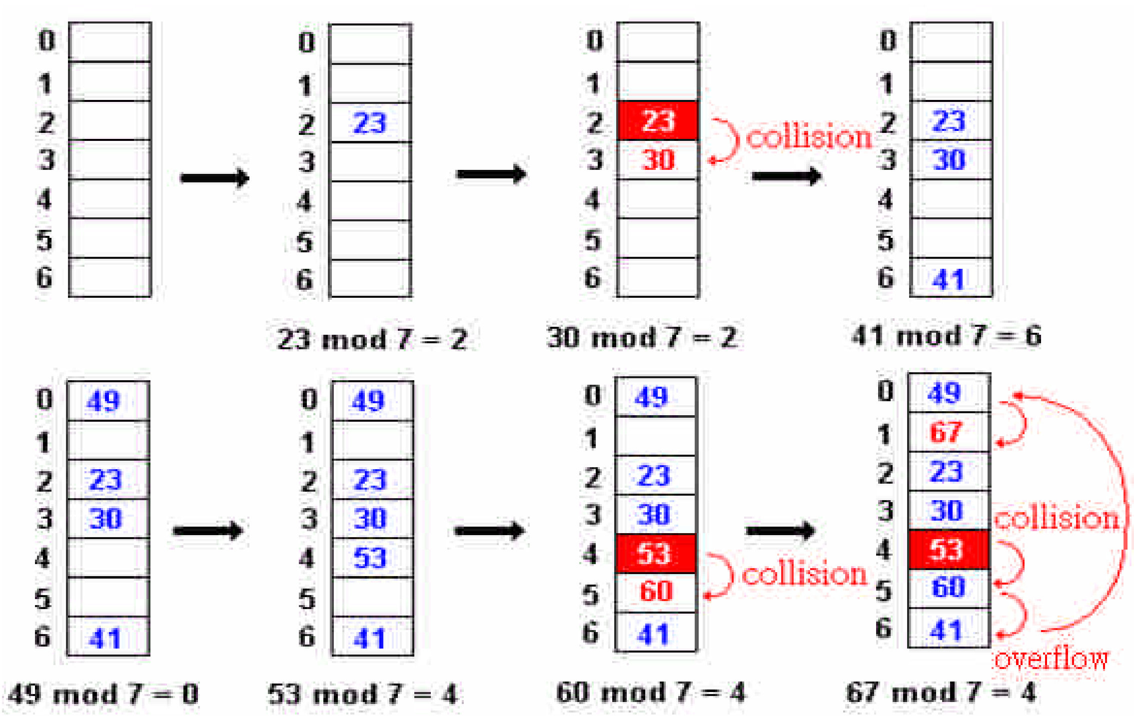
- *Linear probing*, *Quadratic probing*, and *Double hashing* are commonly used to compute the probe sequences required for open addressing.

(2.1) Linear Probing

- h(k, i) = (h'(k) + i) mod m ,

  h' : *auxiliary hash function*

  i : *0, 1, ... , m-1*



(2.2) Quadratic Probing

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

  h' : *auxiliary hash function*

  $c_1, c_2 \neq 0$: *auxiliary constants*

  i : *0, 1, ... , m-1*

- This method works much better than linear probing, but to make *full use* of the hash table,
- the values of c_1, c_2, and m are constrained.

(2.3) Double hashing

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

  h_1, h_2 : *auxiliary hash function*

  i : *0, 1, ... , m-1*

- Double hashing is one of the *best methods available for open addressing*
- because the permutations produced have ***many*** of the characteristics of randomly chosen permutations.

(2) Open addressing

- These techniques all guarantee that
  $\langle h(k, 0), h(k, 1), \ldots, h(k, m-1) \rangle$ is a permutation of $\langle 0, 1, \ldots, m-1 \rangle$ for each key k
- None of these techniques *fulfills* the assumption of uniform hashing.
- Double hashing has the greatest number of probe sequences and, as one might expect, seems to give the best results.

(2) Open addressing analysis

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.

- Define the random variable *X to be the number of probes made in an unsuccessful search.*
- Define the event A_i , *for i = 1, 2, ..., to be the event that there is an* ith *probe and it is to an occupied* slot.

(2) Open addressing analysis

- Then the event $\{X \geq i\} = A_1 \cap A_2 \cap \cdots \cap A_{i-1}$.
- *We will bound* $Pr\{X \geq i\}$ *by bounding*

Pr $\{A_1 \cap A_2 \cap \cdots \cap A_{i-1}\} = \Pr\{A_1\} \cdot \Pr\{A_2|A_1\} \cdot \Pr\{A3|A_1 \cap A_2\} \cdot \Pr\{A_{i-1}|A_1 \cap A_2 \cap \cdots \cap A_{i-2}\}$

$$\begin{aligned} \Pr\{X \geq i\} &= \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdots \frac{n-i+2}{m-i+2} \\ &\leq \left(\frac{n}{m}\right)^{i-1} \\ &= \alpha^{i-1} . \end{aligned}$$

(2) Open addressing analysis

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

- If α is a constant, an unsuccessful search runs in *O(1) time*.
- Ex. average number of probes in an *unsuccessful search :*
  - If the hash table is **half full :**
    at most 1/(1 - 0.5) = 2
  - If the hash table is **90% full :**
    at most 1/(1 - 0.9) = 10

(2) Open addressing analysis

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.

- 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 $1/(1 - \alpha)$.

(2) Open addressing analysis

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.

(2) Open addressing analysis

- 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 1/(1 - i/m) = m/(m-i).*
- *Averaging over all n keys in the hash table gives us the average number of probes in a* successful search:

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

(2) Open addressing analysis

$$\begin{aligned}\frac{1}{\alpha}(H_m - H_{m-n}) &= \frac{1}{\alpha}\sum_{k=m-n+1}^{m} 1/k \\ &\le \frac{1}{\alpha}\int_{m-n}^{m}(1/x)\,dx \quad \text{(by inequality (A.12))} \\ &= \frac{1}{\alpha}\ln\frac{m}{m-n} \\ &= \frac{1}{\alpha}\ln\frac{1}{1-\alpha}\end{aligned}$$

- Ex. the expected number of probes in a *successful search* is :
  - If the hash table is **half full :** less than 1.387
  - If the hash table is **90% full :** less than 2.559