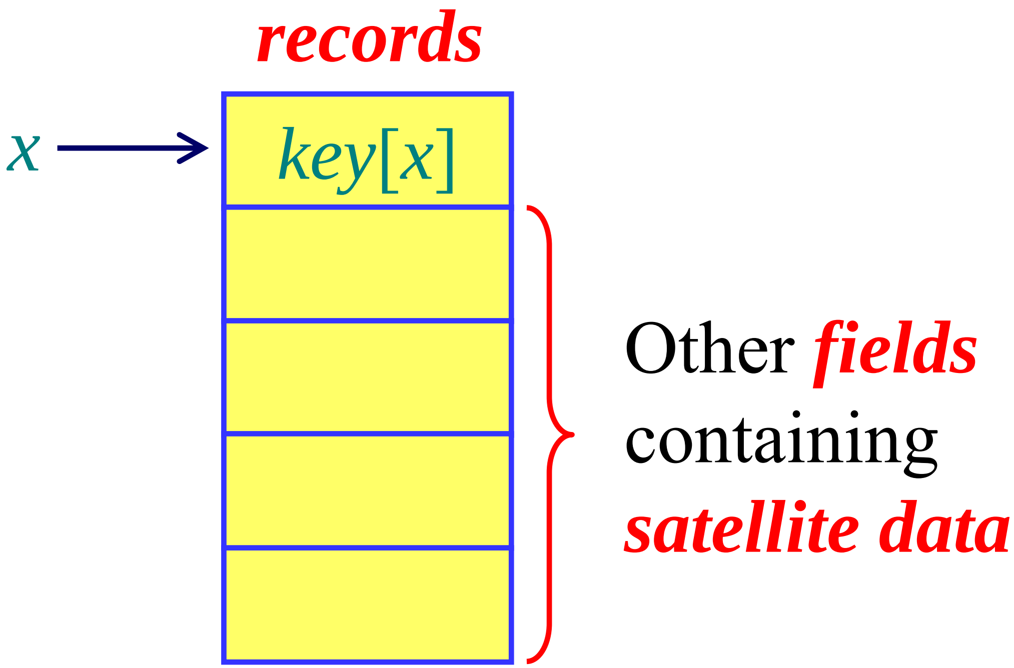
Data Structures and Algorithm

Xiaoqing Zheng
zhengxq@fudan.edu.cn

Dictionary problem

Dictionary T holding n ***records***:



Operations on T:

- INSERT(T, x)
- DELETE(T, x)
- SEARCH(T, k)

How should the data structure T be organized?

Assumptions

Assumptions:

- The set of keys is $K \subseteq U = \{1, 2, \cdots, u-1\}$
- Keys are distinct

What can we do ?

Direct access table

Create a table $T[0 \ldots u-1]$:

$$T[k] = \begin{cases} x & \text{if } k \in K \text{ and } key[x] = k, \\ \text{NIL} & \text{otherwise.} \end{cases}$$

Benefit:

– Each operation takes constant time

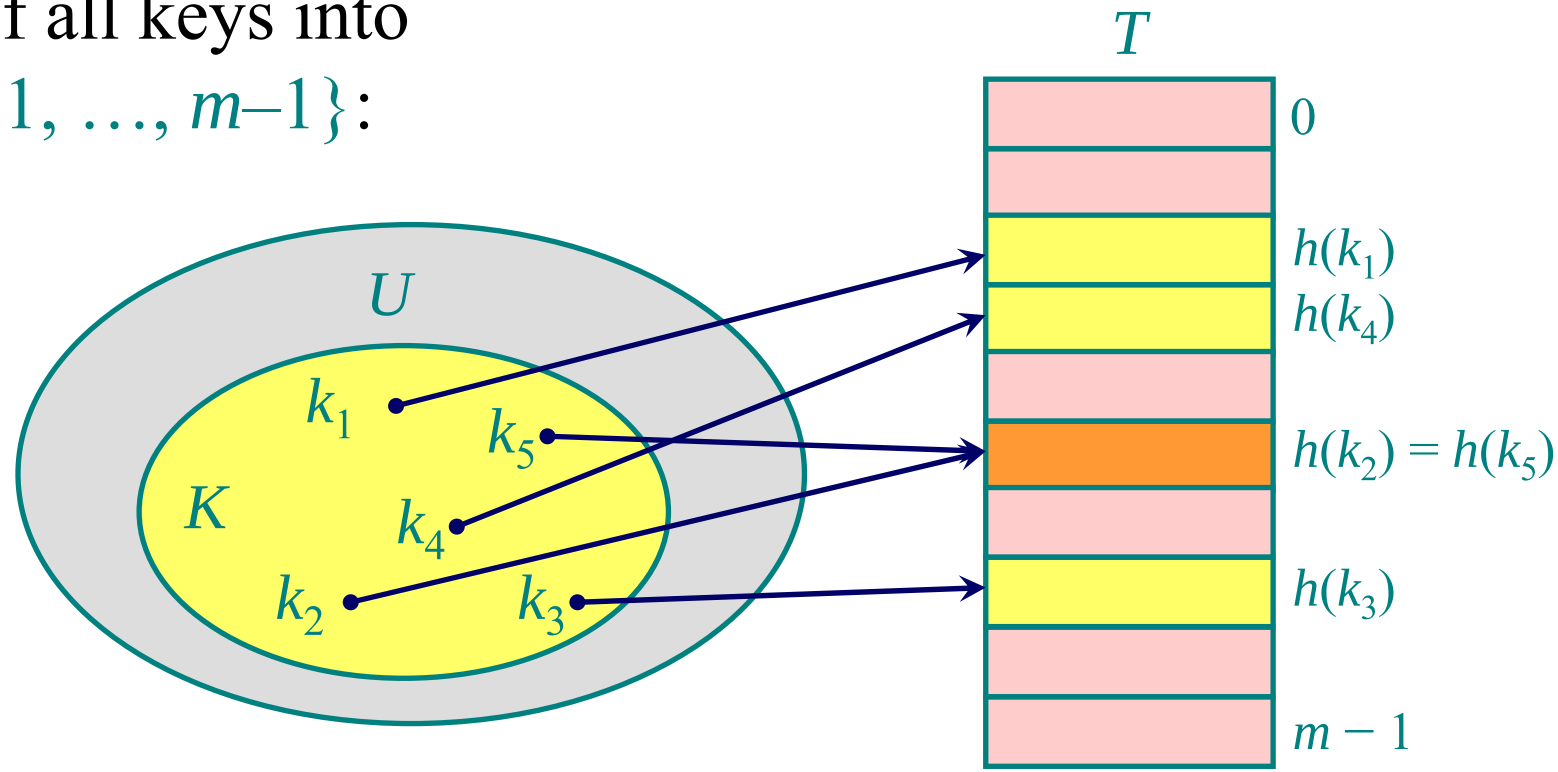
Drawbacks:

– The range of keys can be large:

- 64-bit numbers (which represent 18,446,744,073,709,551,616 different keys),
- character strings (even larger!)

Hash functions

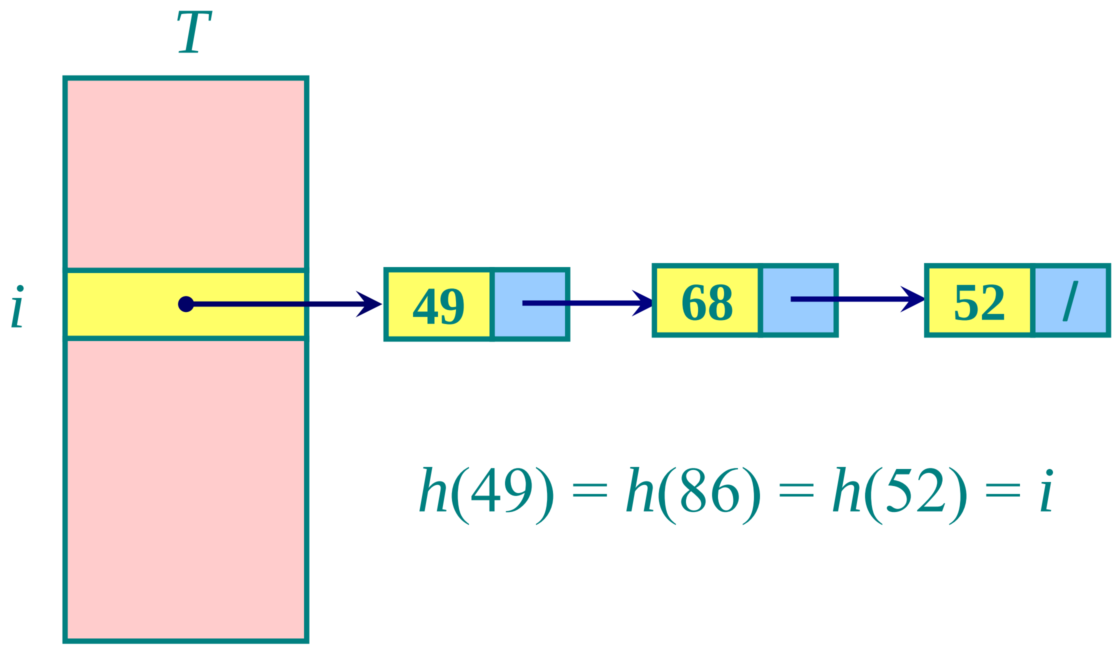
Solution: Use a ***hash function*** h to map the universe U of all keys into $\{0, 1, \ldots, m–1\}$:



When a record to be inserted maps to an already occupied slot in T, a ***collision*** occurs.

Collisions resolution by chaining

Records in the same slot are linked into a list.



$$h(49) = h(86) = h(52) = i$$

Hash functions

Designing good functions is quite nontrivial

For now, we assume they exist. Namely, we assume ***simple uniform hashing***:

– Each key $k \in K$ of keys is equally likely to be hashed to any slot of table T, independent of where other keys are hashed.

Analysis of chaining

Let n be the number of keys in the table, and let m be the number of slots.

Define the ***load factor*** of T to be

$$\alpha = n / m$$

$=$ average number of keys per slot.

The number of elements examined during a successful search for an element x is 1 more that the number of elements that appear before x in x's list.

Search cost

Expected time to search for a record with a given key

$$\Theta(1+\alpha)$$

apply hash function and access slot — *search the list*

Expected search time $= \Theta(1)$ if $\alpha = O(1)$,
or equivalently, if $n = O(m)$.

Analysis of successful search

Let x_i denote the ith element inserted into the table, for $i = 1, 2, .., n$, and let $k_i = key[x_i]$.

For keys k_i and k_j, we define the indicator variable $X_{ij} = \mathrm{I}\{h(k_i) = h(k_j)\}$

Under the assumption of simple uniform hashing, we have:

$$pr\{h(k_i) = h(k_j)\} = m \cdot (\frac{1}{m} \cdot \frac{1}{m}) = \frac{1}{m}$$

Analysis of successful search

The expected number of elements examined in a successful search is

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

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

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

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

Analysis of successful search (cont.)

The expected number of elements examined in a successful search is

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

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

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

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

$$=\Theta(1+\alpha) \qquad \square$$

Operation of hash table

CHAINED-HASH-INSERT(T, x)

1. insert x at the head of list $T[h(key[x])]$

   Running time: $\Theta(1)$

CHAINED-HASH-SEARCH(T, k)

1. search for an element with key k in list $T[h(k)]$

   Running time: $\Theta(1+\alpha)$

CHAINED-HASH-DELETE(T, x)

1. delete x from the list $T[h(key[x])]$

   Running time: $\Theta(1+\alpha)$

Dealing with wishful thinking

- A good hash function satisfies the (approximately) 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.
- The assumption of simple uniform hashing is hard to guarantee, but several common techniques tend to work well in practice as long as their deficiencies can be avoided.

Division method

Define

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

Deficiency: Don't pick an m that has a small divisor d. A preponderance of keys that are congruent modulo d can adversely affect uniformity.

Extreme deficiency: If $m = 2^r$, then the hash doesn't even depend on all the bits of k:

- If $k = 1011000111\underbrace{011010}_{h(k)}{}_2$ and $r = 6$, then $h(k) = 011010_2$.

Division method (cont.)

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

Pick m to be a prime.

This method is popular, although the next method we'll see is usually superior.

Multiplication method

Assume that all keys are integers, $m = 2^r$, and our computer has w-bit words. Define

$$h(k) = (A \cdot k \bmod 2^w) \text{ rsh } (w - r),$$

where rsh is the "bit-wise right-shift" operator and A is an odd integer in the range $2^{w-1} < A < 2^w$.

- Don't pick A too close to 2^w.
- Multiplication modulo 2^w is fast.
- The rsh operator is fast.

Multiplication method example

$$h(k) = (A \cdot k \bmod 2^w) \text{ rsh } (w - r),$$

Suppose that $m = 8 = 2^3$ and that our computer has $w = 7$-bit words:

$$
\begin{array}{r}
1\,0\,1\,1\,0\,0\,1 = A \\
\times \quad 1\,1\,0\,1\,0\,1\,1 = k \\
\hline
1\,0\,0\,1\,0\,1\,0\,\underbrace{0\,1\,1}_{h(k)}\,0\,0\,1\,1
\end{array}
$$

Open addressing

All elements are stored in the hash table itself.

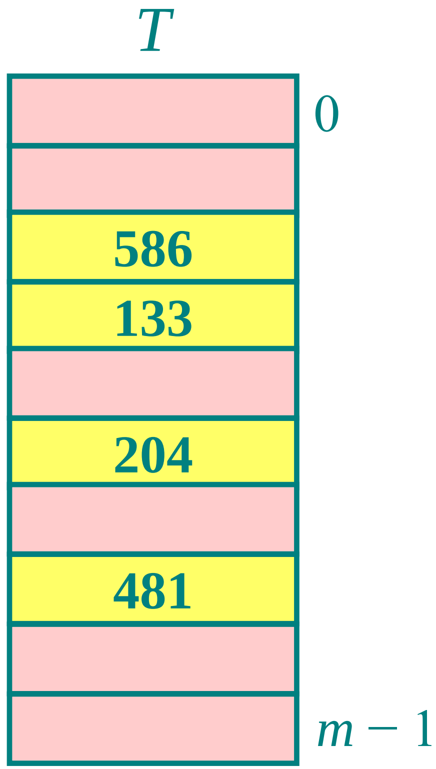
- Insertion systematically probes the table until an empty slot is found.
- The hash function depends on both the key and probe number:

  $$h: U \times \{0, 1, \ldots, m-1\} \rightarrow \{0, 1, \ldots, m-1\}.$$

- The ***probe sequence*** $\langle h(k,0), h(k,1), \ldots, h(k,m-1)\rangle$ should be a permutation of $\{0, 1, \ldots, m-1\}$.
- The table may fill up, and deletion is difficult (one solution is to mark the slot by storing in it the special value DELETED instead of NIL)

Example of open addressing

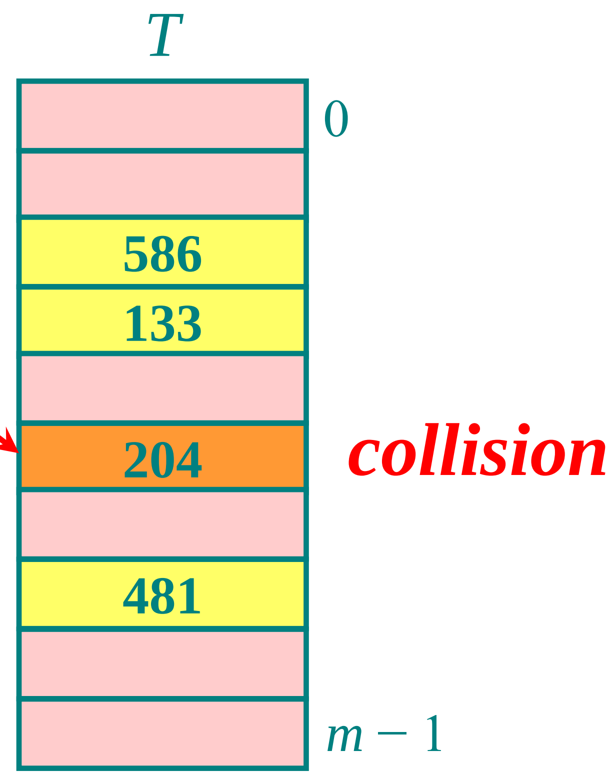
Insert key $k = 496$:



Example of open addressing

Insert key $k = 496$:

0. Probe $h(496,0)$

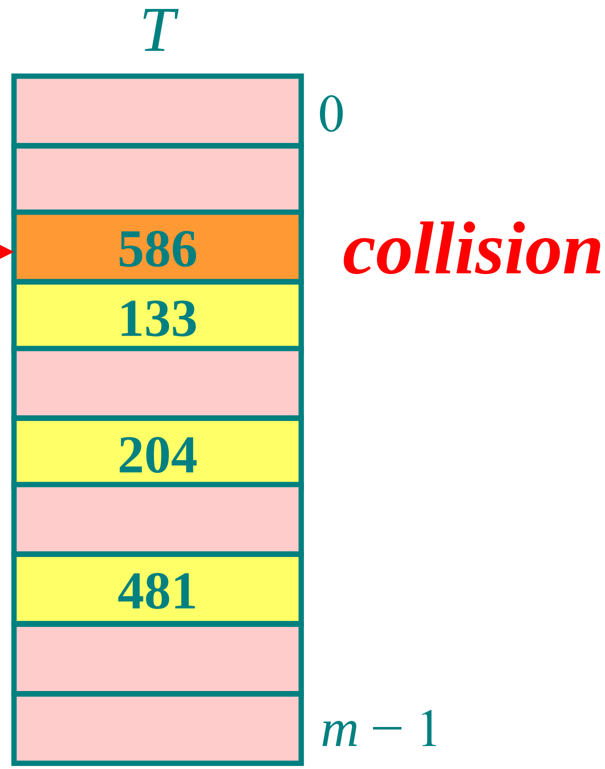


Example of open addressing

Insert key $k = 496$:

0. Probe $h(496,0)$

1. Probe $h(496,1)$



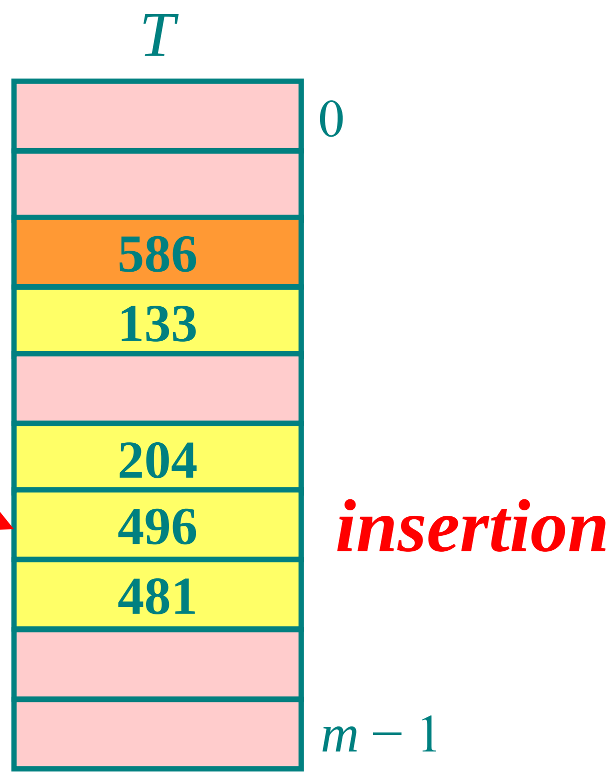
Example of open addressing

Insert key $k = 496$:

0. Probe $h(496,0)$

1. Probe $h(496,1)$

2. Probe $h(496,2)$



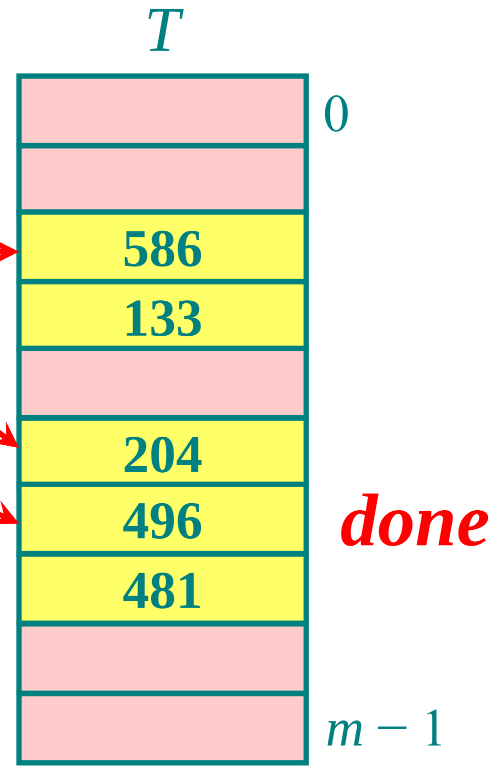
Example of open addressing

Search for key $k = 496$:

0. Probe $h(496,0)$

1. Probe $h(496,1)$

2. Probe $h(496,2)$



Search uses the same probe sequence, terminating successfully if it finds the key and unsuccessfully if it encounters an empty slot.

Probing strategies

Linear probing:

Given an ordinary hash function $h'(k)$, linear probing uses the hash function

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

This method, though simple, suffers from ***primary clustering***, where long runs of occupied slots build up, increasing the average search time. Moreover, the long runs of occupied slots tend to get longer.

Probing strategies

Quadratic probing:

Given an ordinary hash function $h'(k)$, quadratic probing uses the hash function

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

If two keys have the same initial probe position, then their probe sequences are the same, since $h(k_1, 0) = h(k_2, 0)$ implies $h(k_1, i) = h(k_2, i)$. this property leads to a milder form of clustering, called ***secondary clustering***.

Probing strategies

Double hashing:

Given two ordinary hash functions $h_1(k)$ and $h_2(k)$, double hashing uses the hash function

$$h(k,i) = (h_1(k) + i \cdot h_2(k)) \bmod m.$$

This method generally produces excellent results, but $h_2(k)$ must be relatively prime to m. One way is to make m a power of 2 and design $h_2(k)$ to produce only odd numbers.

Analysis of open addressing

We make the assumption of ***uniform hashing***:

- Each key is equally likely to have any one of the $m!$ permutations as its probe sequence.

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

Proof of the theorem

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

$$\begin{aligned}
\Pr\{X \geq i\} &= \Pr\{A_1 \cap A_2 \cap \cdots \cap A_{i-1}\} \\
&= \Pr\{A_1\} \cdot \Pr\{A_2 \mid A_1\} \cdot \Pr\{A_3 \mid A_1 \cap A_2\} \cdots \\
&\quad \Pr\{A_{i-1} \mid A_1 \cap A_2 \cap \cdots \cap A_{i-2}\} \\
&= \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdots \frac{n-i+2}{m-i+2} \\
&\leq (n/m)^{i-1} \\
&= \alpha^{i-1}
\end{aligned}$$

Proof of the theorem (cont.)

$$
\begin{aligned}
E[X] &= \sum_{i=0}^{n} i \Pr[X = i] \\
&\le \sum_{i=0}^{\infty} i \Pr[X = i] \\
&= \sum_{i=0}^{\infty} i \left(\Pr\{X \ge i\} - \Pr\{X \ge i+1\} \right) \\
&= \sum_{i=1}^{\infty} \Pr\{X \ge i\} \\
&\le \sum_{i=1}^{\infty} \alpha^{i-1} \\
&= \sum_{i=0}^{\infty} \alpha^{i} \\
&= \frac{1}{1-\alpha} \qquad \square
\end{aligned}
$$

Analysis of open addressing

Theorem. Given an open-addressed hash table with load factor $\alpha = n/m < 1$, the expected number of probes in an successful search is at most .

$$\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$$

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

Proof of the theorem

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

$$
\begin{aligned}
\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i} &= \frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i} \\
&= \frac{1}{\alpha}\sum_{k=m-n+1}^{m}\frac{1}{k} \\
&\le \frac{1}{\alpha}\int_{m-n}^{m}(1/x)\,dx \\
&= \frac{1}{\alpha}\ln\frac{m}{m-n} \\
&= \frac{1}{\alpha}\ln\frac{1}{1-\alpha} \qquad \square
\end{aligned}
$$

Any question?

Xiaoqing Zheng
Fundan University