CENG 218
Design and Analysis of Algorithms

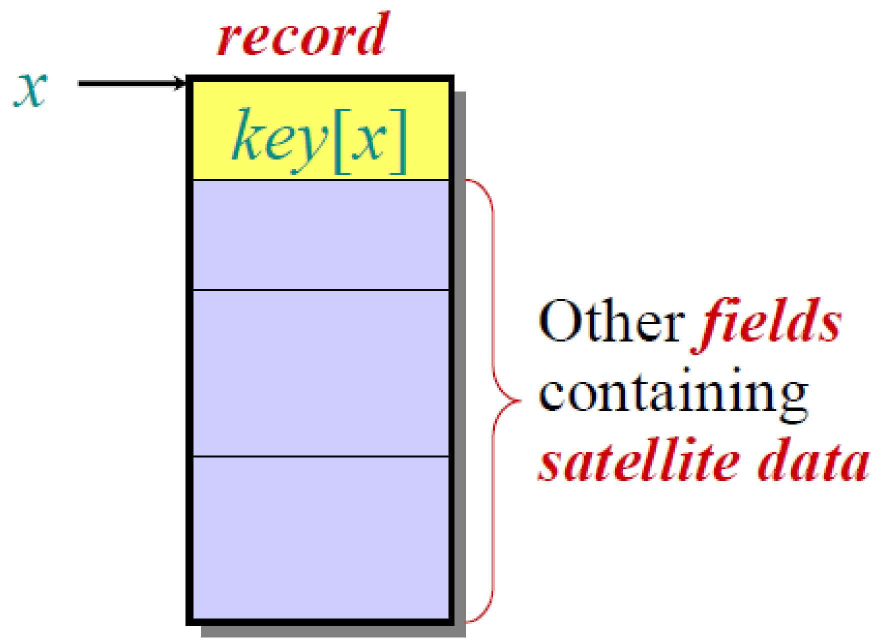
Izmir Institute of Technology

Lecture 8: Hash tables

Slides were mostly prepared using the material provided by Prof. Charles E. Leiserson and Prof. Erik Demaine from MIT

Symbol table problem

Symbol table T holding n ***records***:



Operations on T:

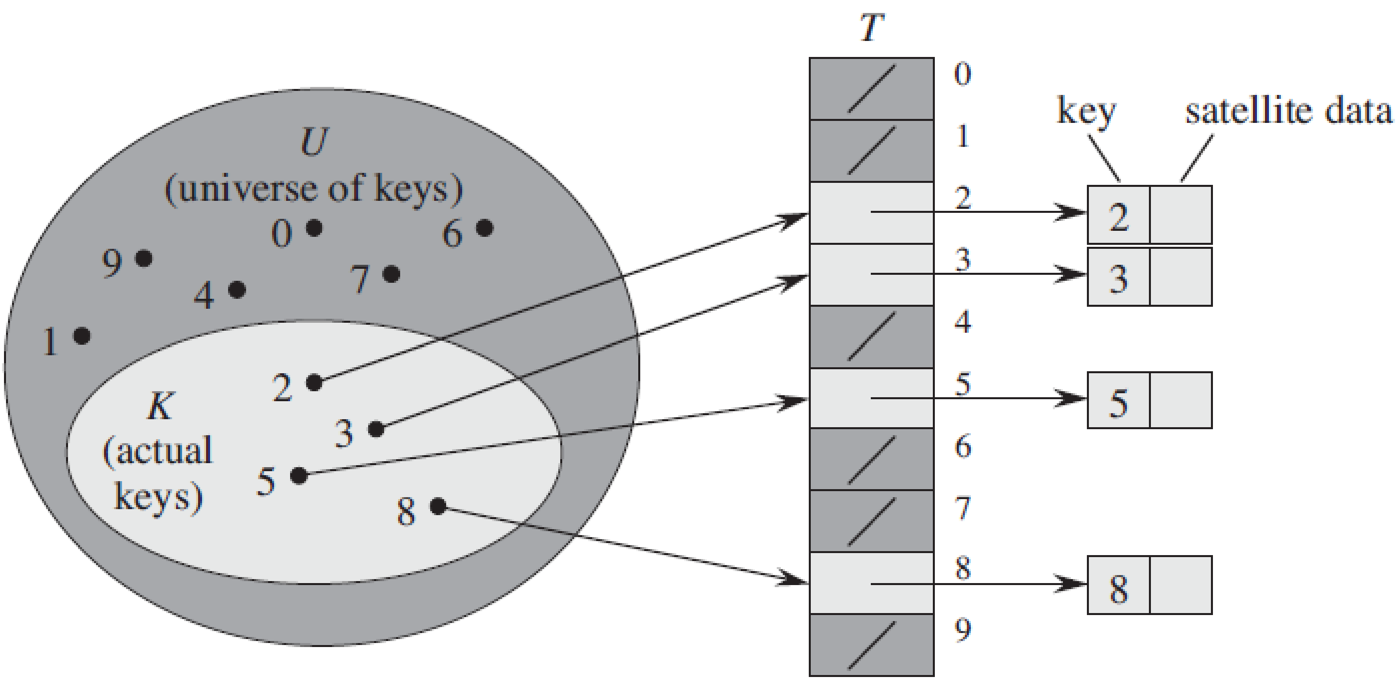
- INSERT(T, x): $T \leftarrow T \cup \{x\}$
- DELETE(T, x): $T \leftarrow T - \{x\}$
- SEARCH(T, k): returns x if $key[x]=k$

What data structure can be used to organize T ?

Direct-address table

Suppose that the set of keys is $K \subseteq \{0,1, \ldots, m-1\}$, and keys are distinct. Set up an array $T[0 \mathrel{.\,.} m-1]$:

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



Direct-address table

Then, operations take $\Theta(1)$ time. Good!

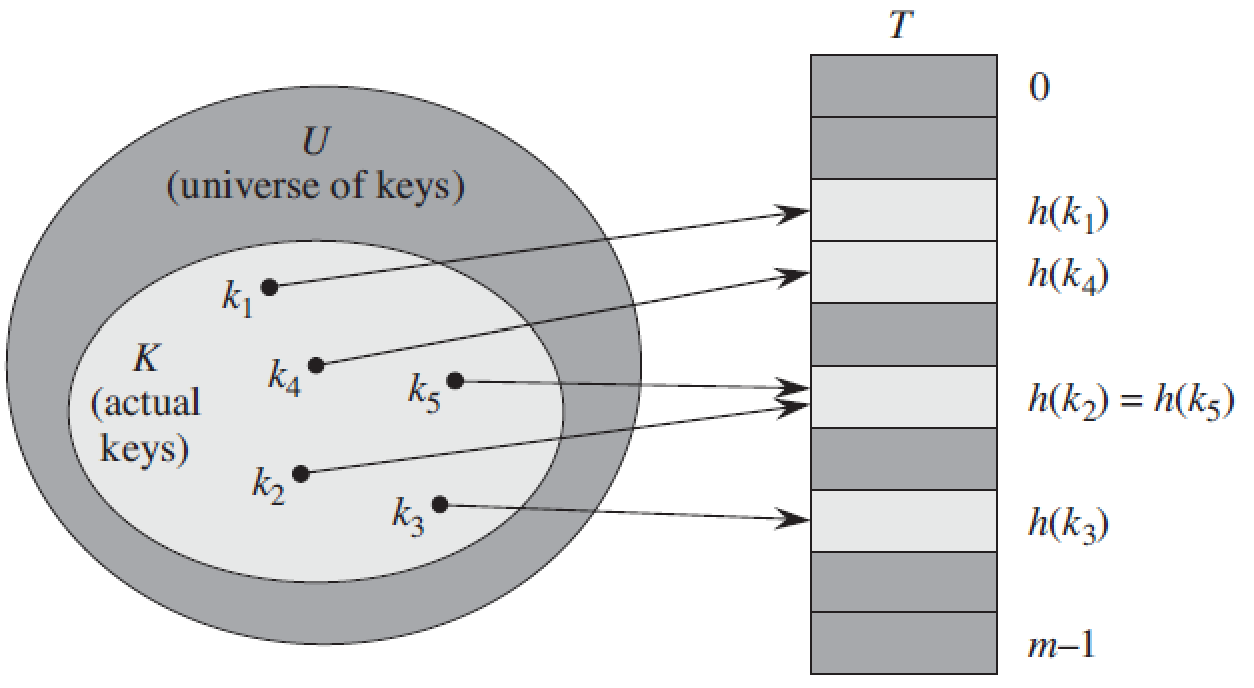
What could be a problem?

Problem: 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 tables

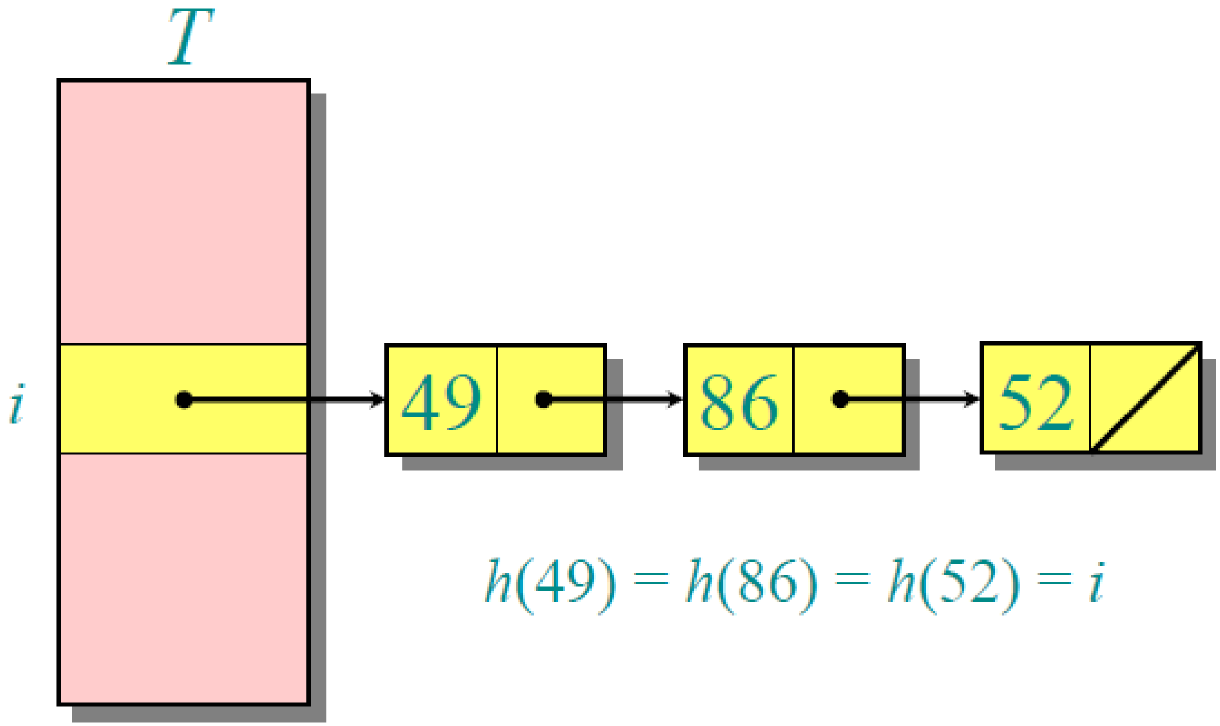
Solution: Use a ***hash function*** h that 'randomly' maps the keys into slots in T, *i.e.* $\{0, 1, \ldots, m–1\}$:



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

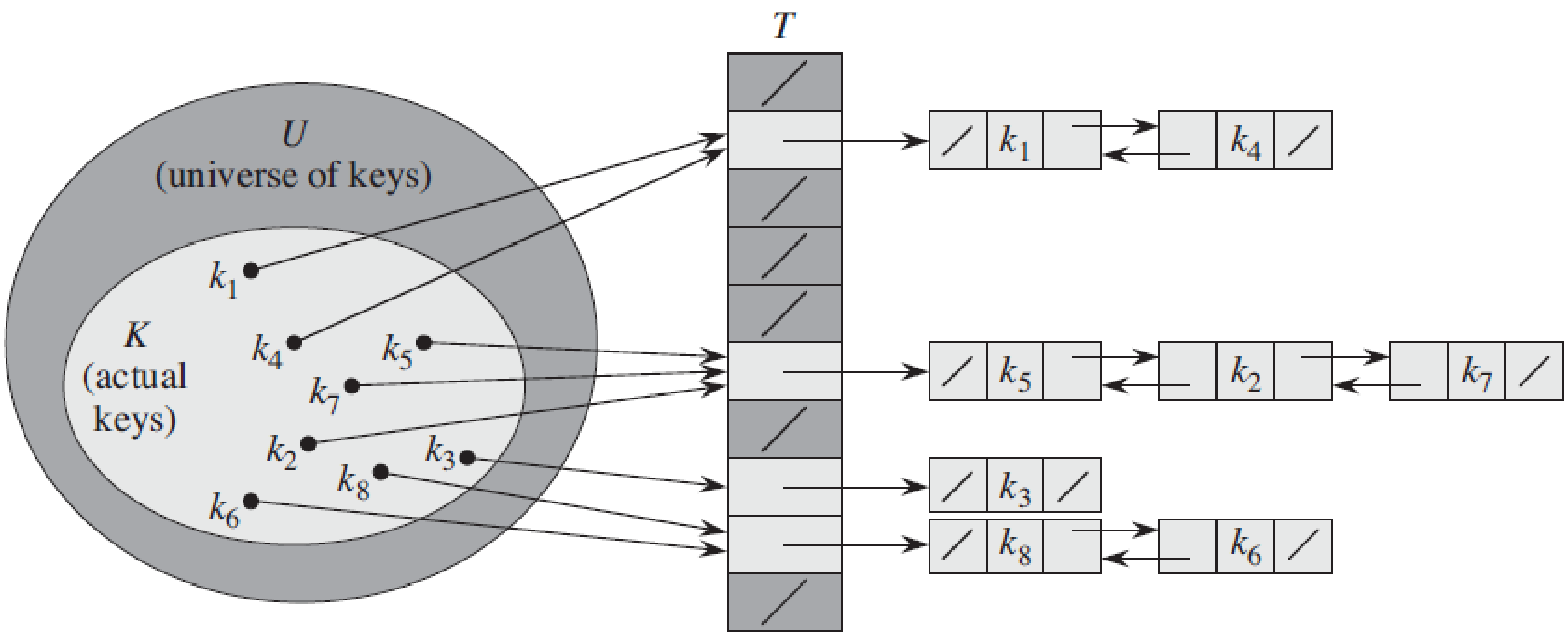
Resolving collisions by chaining

Records in the same slot are linked into a list.



Resolving collisions by chaining

Another example, now the linked list is doubly linked.



$T[j]$ contains a linked list of all the keys whose hash value is j.

Analysis of hashing and chaining

Worst-case:

Every key hashes to the same slot. Access time: $\Theta(n)$.

Average-case:

We assume ***simple uniform hashing:***

- Each key is equally likely to be hashed to a slot of table T, independent of where other keys are hashed.

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

$$\alpha = n/m$$

$$= \text{average number of keys per slot.}$$

Search cost

Expected time to search for a record with a given key = $\Theta(1 + \alpha)$.

Apply hash function and access slot

Search the linked list

Expected search time = $\Theta(1)$ if $\alpha = O(1)$.

Choosing a hash function

The assumption of simple uniform hashing is hard to guarantee

- -should distribute the keys uniformly into slots
- -regularity in the key distributions should not affect uniformity

Some techniques work well in practice if their deficiencies can be avoided.

Let’s see a few of these methods.

Division method

Assume all the keys are integers, and define $h(k) = k \bmod m$.

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

E.g: If d=2 and all k are even, odds slots are not used.

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

If $k = 10110001110\underbrace{11010}_{h(k)}{}_2$ and $r = 5$, then $h(k) = 11010_2$.

Division method

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

Pick m to be a prime not too close to a power of 2 or 10 and not otherwise used prominently in the computing environment.

Drawbacks:

- Sometimes, making the table size a prime is inconvenient.
- Although this method is popular, the next method we'll see is usually superior.

Multiplication method

Assume that all the keys are integers, $m = 2^r$, and our computer has w-bit words (32-bit or 64-bit). Define

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

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

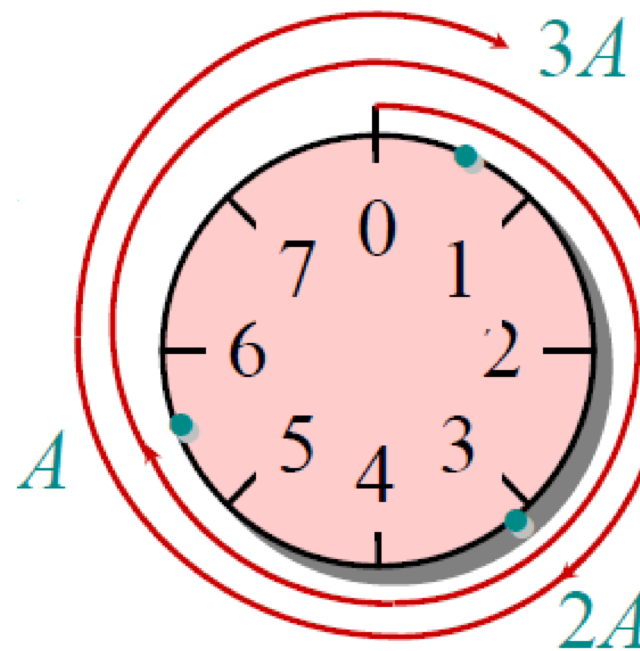
- Don’t pick A too close to 2^w.
- Multiplication modulo 2^w is faster than division method.
- The rsh operator is fast.

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:

$$\begin{array}{r l} 1\,0\,1\,1\,0\,0\,1 & = A \\ \times \qquad\qquad\qquad 1\,1\,0\,1\,0\,1\,1 & = k \\ \hline \underbrace{1\,0\,0\,1\,0\,1\,0}_{\substack{\textit{ignored} \\ (\text{mod } 2^w)}}\,\underbrace{0\,1\,1}_{h(k)}\,\underset{rhs}{\underrightarrow{0\,0\,1\,1}} & = A \cdot k \end{array}$$



Modular wheel

Resolving collisions by open addressing

No storage is used outside of the hash table itself.

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

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

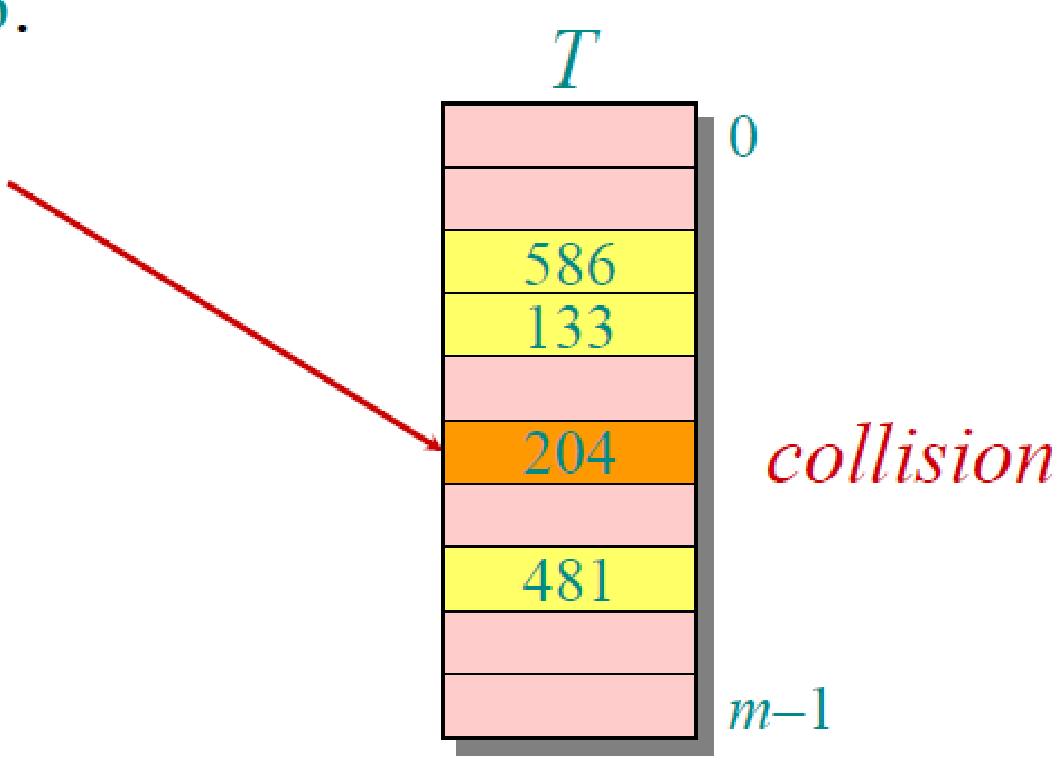
  keys probe no. slot no.

- The probe sequence: $h(k,0), h(k,1), \ldots, h(k,m\text{-}1)$

Example of open addressing

Insert key $k = 496$:

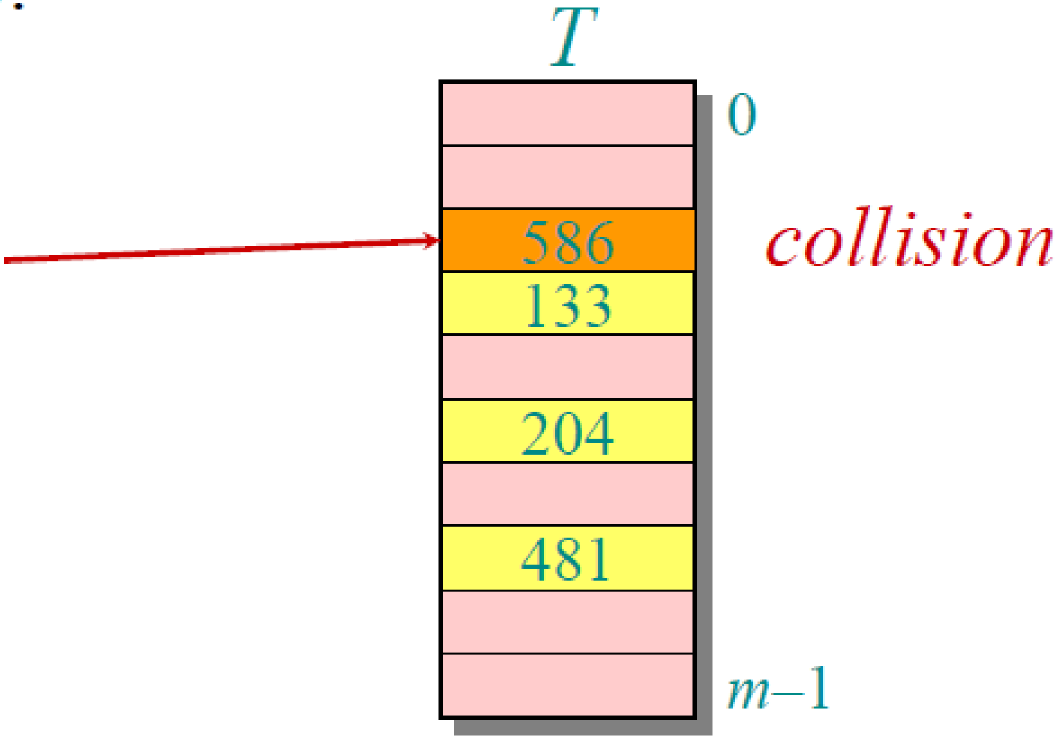
0. Probe $h(496,0)$



Example (continued)

Insert key $k = 496$:

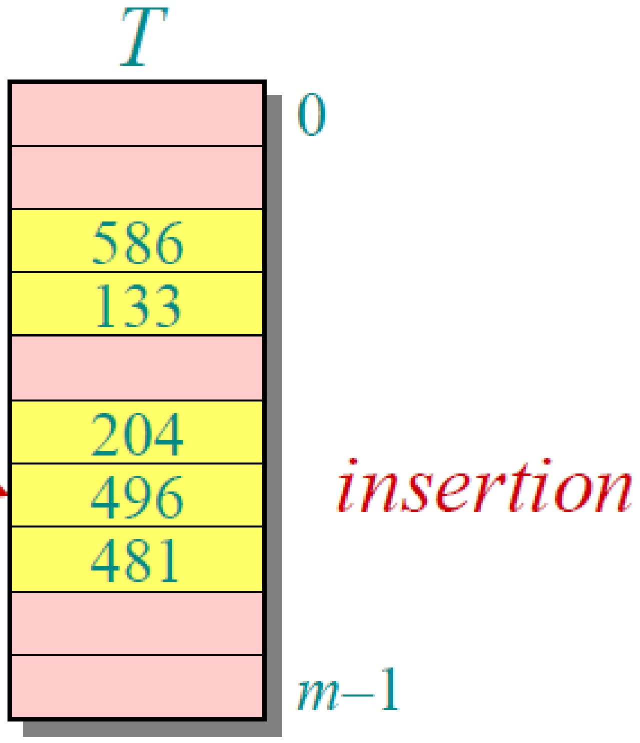
0. Probe $h(496,0)$
1. Probe $h(496,1)$



Example (continued)

Insert key $k = 496$:

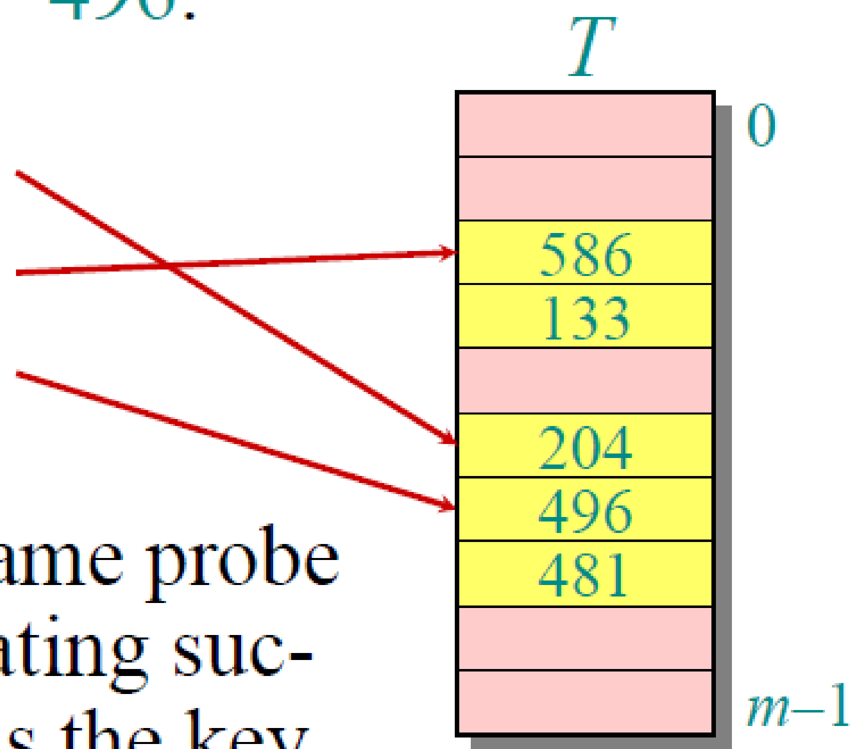
0. Probe $h(496,0)$
1. Probe $h(496,1)$
2. Probe $h(496,2)$



Example (continued)

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 suffers from ***primary clustering***, where long runs of occupied slots build up, increasing the average search time.

Probing strategies

Double hashing:

Given two ordinary hash functions $h_1(k)$ and $h_2(k)$, double hashing uses the hash function $h(k,i) = (h_1(k) + i \cdot h_2(k)) \bmod m$.

The initial probe goes to $h_1(k)$, successive one is offset by the amount $h_2(k)$ modulo m.

One way is choosing a prime m and design $h_2(k)$ so that it returns a positive integer less than m.

Probing strategies

Double hashing example:

$h_1(k) = k \bmod m$
$h_2(k) = 1 + (k \bmod m')$

where m' is chosen slightly less than m.

Let's say k=123456, take m=701, m' =700.

Hash functions give $h_1(k)$=80 and $h_2(k)$=257 which makes first probe to position 80 and then check every 257th slot after (modulo m).

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

- At least one probe is always necessary.
- With probability n/m, the first probe hits an occupied slot, and a second probe is necessary.
- With probability $(n–1)/(m–1)$, the second probe hits an occupied slot, and a third probe is necessary.
- With probability $(n–2)/(m–2)$, the third probe hits an occupied slot, etc.

Observe that $\dfrac{n-i}{m-i} < \dfrac{n}{m} = \alpha$ for $i = 1, 2, \ldots, n$.

Proof (continued)

Therefore, the expected number of probes is

$$1+\frac{n}{m}\left(1+\frac{n-1}{m-1}\left(1+\frac{n-2}{m-2}\left(\cdots\left(1+\frac{1}{m-n+1}\right)\cdots\right)\right)\right)$$

$$\le 1+\alpha(1+\alpha(1+\alpha(\cdots(1+\alpha)\cdots)))$$

$$\le 1+\alpha+\alpha^2+\alpha^3+\cdots$$

$$=\sum_{i=0}^{\infty}\alpha^i$$

$$=\frac{1}{1-\alpha}\ . \ \square$$

The textbook has a more rigorous proof.

Analysis of open addressing

- If α is constant, then accessing an open addressed hash table takes constant time.
- If the table is half full, then the expected number of probes is 1/(1–0.5) = 2.
- If the table is 90% full, then the expected number of probes is 1/(1–0.9) = 10.

The End

Read the related parts of Section 11 of the textbook.