Do not distribute course material

You may not and may not allow others to reproduce or distribute lecture notes and course materials publicly whether or not a fee is charged.

Lecture 3
Hash Tables

maintain a dynamic set

How are you feeling?

For this lecture we will use

0-indexing

(All other lectures use 1-indexing)

Dynamic Set Operations

We assume the items in the set, S, are indexed by its key, **k**

Queries (non modifying)

1. **Search(S, k)** - returns a pointer x, where x.key = k or nil if not found
2. **Minimum(S)/Maximum(S)** - returns pointer to item with smallest(largest) key
3. **Successor(S, x)**

   Predecessor(S, x) - assumes total order on set of keys. Given a pointer to an element x, whose key k returns element which has next largest(smallest) key.

We typically measure time to perform an operation in terms of a function of the size of the set

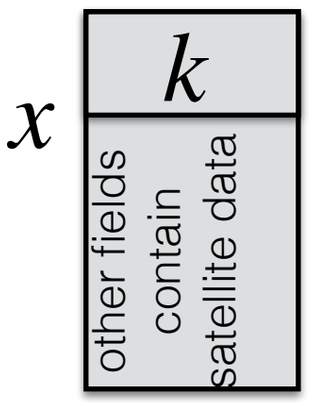
The items may contain satellite data which are carried around (and not used by the implementation)

Set

$S =$

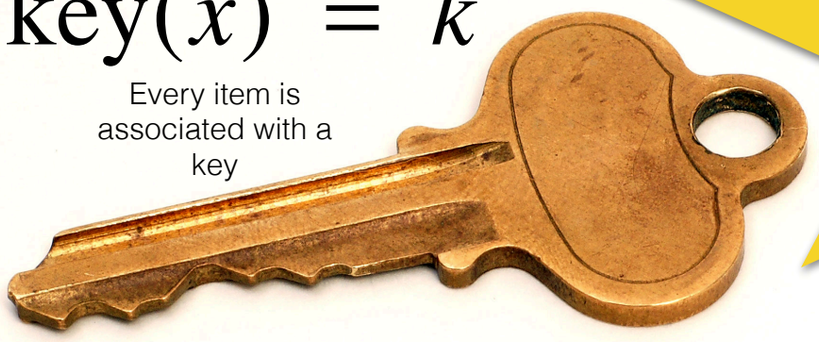
101,,Van Cortlandt Park - 242 St,,40.889248,-73.898583,,,1,
101N,,Van Cortlandt Park - 242 St,,40.889248,-73.898583,,,0,101
101S,,Van Cortlandt Park - 242 St,,40.889248,-73.898583,,,0,101
103,,238 St,,40.884667,-73.90087,,,1,
103N,,238 St,,40.884667,-73.90087,,,0,103
103S,,238 St,,40.884667,-73.90087,,,0,103
104,,231 St,,40.878856,-73.904834,,,1,
104N,,231 St,,40.878856,-73.904834,,,0,104
104S,,231 St,,40.878856,-73.904834,,,0,104
106,,Marble Hill - 225 St,,40.874561,-73.909831,,,1,
106N,,Marble Hill - 225 St,,40.874561,-73.909831,,,0,106
106S,,Marble Hill - 225 St,,40.874561,-73.909831,,,0,106
⋮

S28N,,Clifton,,40.621319,-74.071402,,,0,S28
S28S,,Clifton,,40.621319,-74.071402,,,0,S28
S29,,Stapleton,,40.627915,-74.075162,,,1,
S29N,,Stapleton,,40.627915,-74.075162,,,0,S29
S29S,,Stapleton,,40.627915,-74.075162,,,0,S29
S30,,Tompkinsville,,40.636949,-74.074835,,,1,
S30N,,Tompkinsville,,40.636949,-74.074835,,,0,S30
S30S,,Tompkinsville,,40.636949,-74.074835,,,0,S30
S31,,St George,,40.643748,-74.073643,,,1,
S31N,,St George,,40.643748,-74.073643,,,0,S31
S31S,,St George,,40.643748,-74.073643,,,S31



$\text{key}(x) = k$

Every item is associated with a key



https://en.wikipedia.org/wiki/Key_(lock)#/media/File:Standard-lock-key.jpg

Different implementations have different running times

Which implementation of these operations is best to use depends on the application

Modifying Operations

6. **Insert(S, x)** - element pointed to by x is added to S
7. **Delete(S, x)** - given a pointer x to an element in S, it is removed from S

We can implement these operations many ways!

Dictionary Operations

We assume the items in the set, S, are indexed by its key, **k**

Queries (non modifying)

1. **Search(S, k)** - returns a pointer x, where x.key = k or nil if not found

We typically measure time to perform an operation in terms of a function of the size of the set

The items may contain satellite data which are carried around (and not used by the implementation)

Set

$S =$

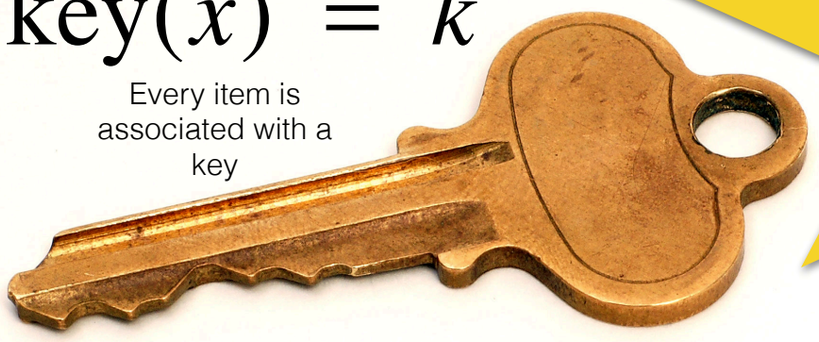
101,,Van Cortlandt Park - 242 St,,40.889248,-73.898583,,,1,
101N,,Van Cortlandt Park - 242 St,,40.889248,-73.898583,,,0,101
101S,,Van Cortlandt Park - 242 St,,40.889248,-73.898583,,,0,101
103,,238 St,,40.884667,-73.90087,,,1,
103N,,238 St,,40.884667,-73.90087,,,0,103
103S,,238 St,,40.884667,-73.90087,,,0,103
104,,231 St,,40.878856,-73.904834,,,1,
104N,,231 St,,40.878856,-73.904834,,,0,104
104S,,231 St,,40.878856,-73.904834,,,0,104
106,,Marble Hill - 225 St,,40.874561,-73.909831,,,1,
106N,,Marble Hill - 225 St,,40.874561,-73.909831,,,0,106
106S,,Marble Hill - 225 St,,40.874561,-73.909831,,,0,106
⋮

S28N,,Clifton,,40.621319,-74.071402,,,0,S28
S28S,,Clifton,,40.621319,-74.071402,,,0,S28
S29,,Stapleton,,40.627915,-74.075162,,,1,
S29N,,Stapleton,,40.627915,-74.075162,,,0,S29
S29S,,Stapleton,,40.627915,-74.075162,,,0,S29
S30,,Tompkinsville,,40.636949,-74.074835,,,1,
S30N,,Tompkinsville,,40.636949,-74.074835,,,0,S30
S30S,,Tompkinsville,,40.636949,-74.074835,,,0,S30
S31,,St George,,40.643748,-74.073643,,,1,
S31N,,St George,,40.643748,-74.073643,,,0,S31
S31S,,St George,,40.643748,-74.073643,,,S31

x | k | other fields contain satellite data

$\text{key}(x) = k$

Every item is associated with a key



https://en.wikipedia.org/wiki/Key_(lock)#/media/File:Standard-lock-key.jpg

Different implementations have different running times

Modifying Operations

6. **Insert(S, x)** - element pointed to by x is added to S
7. **Delete(S, x)** - given a pointer x to an element in S, it is removed from S

We can implement these operations many ways!

Dictionary Data Structure

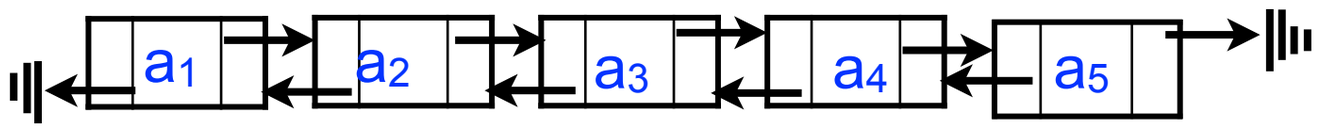
- Given a universe of keys, $\mathcal{U}$ (numbers, strings, etc.)
- A data structure that stores a subset $\mathbb{S} \subseteq \mathcal{U}$
- Operations:
  - SEARCH(S,k): given k $\in \mathcal{U}$ is k$\in \mathbb{S}$
  - INSERT(S,x): given x, a pointer to k $\notin \mathbb{S}$ add k to $\mathbb{S}$
  - DELETE(S,x): given x, a pointer to k $\in \mathbb{S}$ delete k from $\mathbb{S}$
- Best data structure may depend on if the data is static (mostly search operations) or dynamic (frequent insert and/or delete operations)

How should we implement the dictionary?

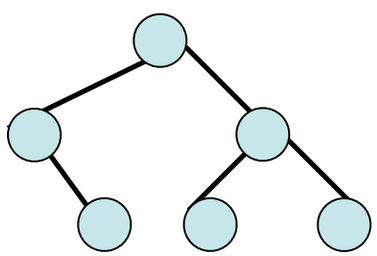
A data structure capable of inserting, deleting and testing membership

- sorted list (array)?

- unsorted list(array)?
- sorted list (linked list)?



- unsorted list(linked list)?
- queue?
- stack?
- binary search tree



- balance binary search tree

If I used a sorted array A to store the data, how long would it take to insert an new item in the worst case?

- $O(1)$
- $O(\log n)$
- $O(n)$
- $O(n \log n)$
- $O(n^2)$
- None of the above

Total Results: 0

If I used a sorted array A to store the data, how long would it take to insert an new item in the worst case?

- $O(1)$
- $O(\log n)$
- $O(n)$
- $O(n \log n)$
- $O(n^2)$
- None of the above

If I used a sorted array A to store the data, how long would it take to insert an new item in the worst case?

- $O(1)$
- $O(\log n)$
- $O(n)$
- $O(n \log n)$
- $O(n^2)$
- None of the above

If I use an unsorted array, how long does it take to search for an item in the worst case?

A) $O(1)$

B) $O(\log n)$

C) $O(n)$

D) $O(n \log n)$

E) $O(n^2)$

F) None of the above

If I use an unsorted array, how long does it take to search for an item in the worst case?

A) $O(1)$

B) $O(\log n)$

C) $O(n)$

D) $O(n \log n)$

E) $O(n^2)$

F) None of the above

If I use an unsorted array, how long does it take to search for an item in the worst case?

A) $O(1)$

B) $O(\log n)$

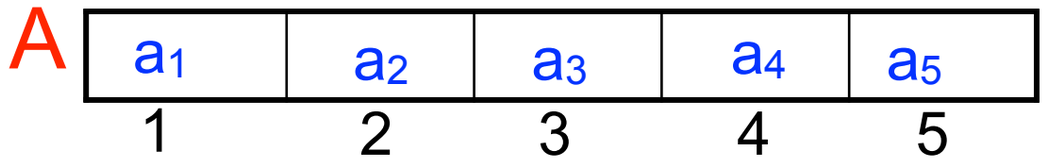
C) $O(n)$

D) $O(n \log n)$

E) $O(n^2)$

F) None of the above

Array, A[i]



1. Search(A,k)
2. Insert(A,x)
3. Delete(A,x)

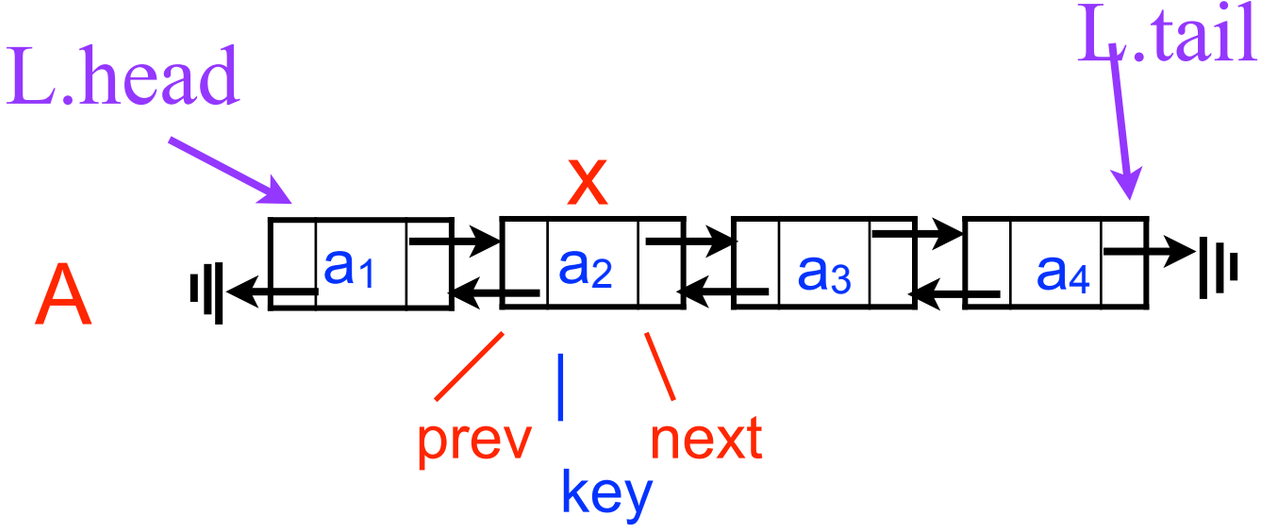
Running Time for this implementation?

Sorted array only works for universe $\mathcal{U}$ which has total order

List

$a_1, a_2, a_3, \ldots, a_n$

attributes: L.head, L.tail, L.key



1. List-Search(L,k)
2. List-Insert(L,x)
3. List-Delete(L,x)

Running Time for this implementation?

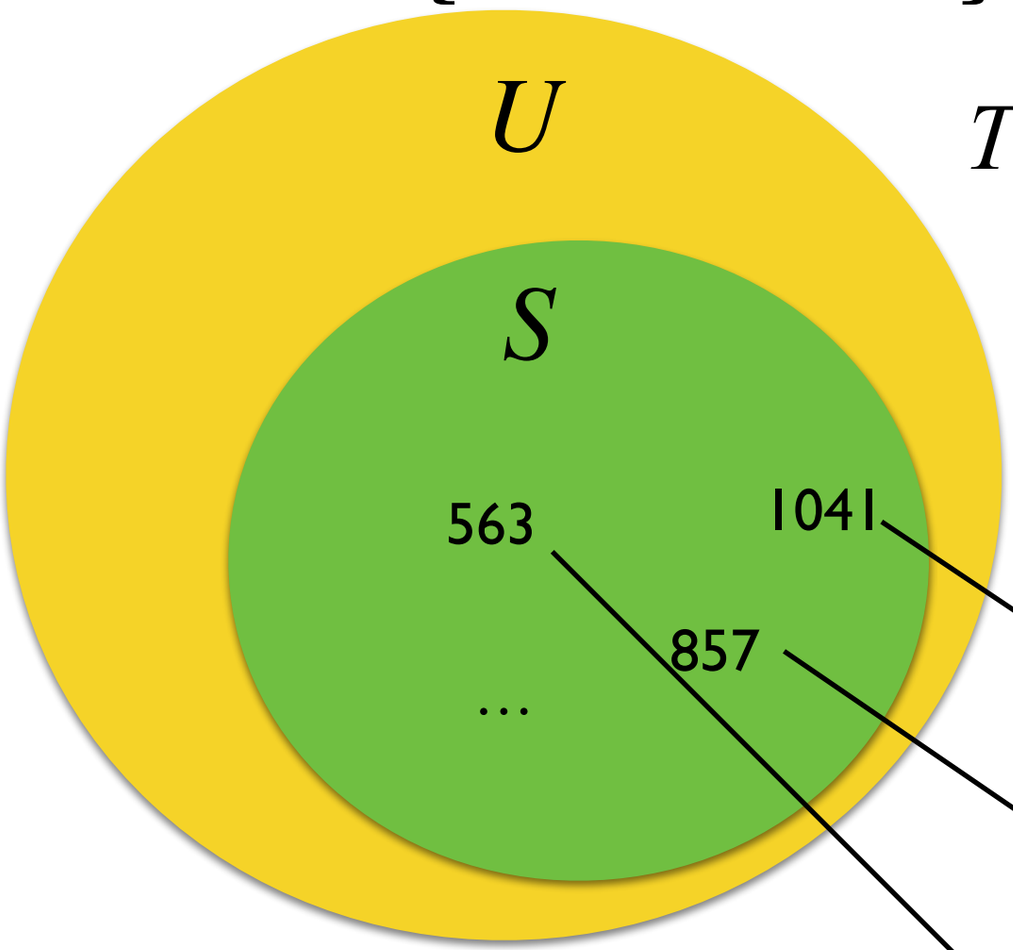
Faster search than a sorted array?

n, O(Log(n)), O(1)?

1st Approach

Direct Addressing

$$U = \{0,1,...,m-1\}$$

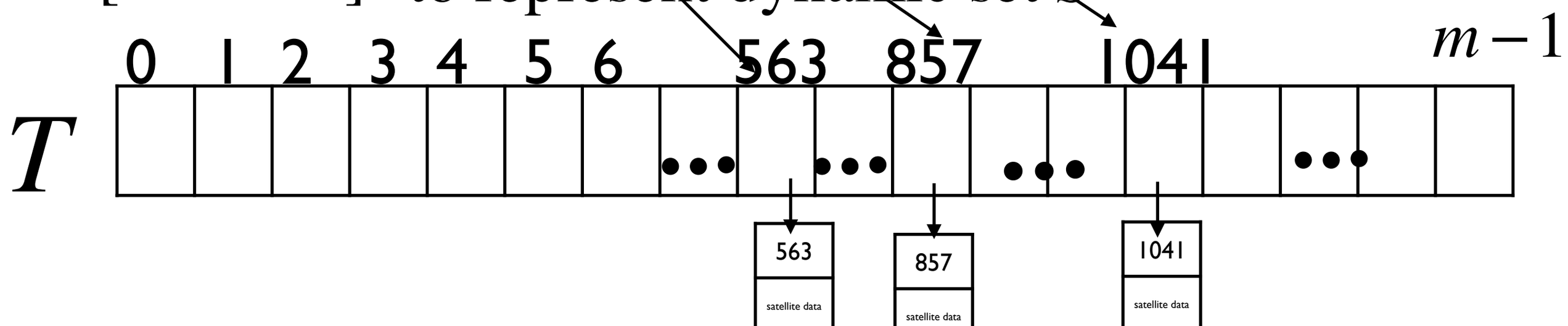


$$T[u] = \begin{cases} x & \text{if } x.\text{key} \in S \text{ and x.key} = u \\ nil & \text{ow} \end{cases}$$

Great when:

- m isn't too large
- no two elements have the same key
- assumes keys are integers

$T[0...m-1]$ to represent dynamic set S



Running Time for Direct Addressing

Queries (non modifying)

Direct-Address-Search(S,k)
return T[k]

$O(1)$

Modifying Operations

Direct-Address-Insert(S,x)
T[x.key] = x

$O(1)$

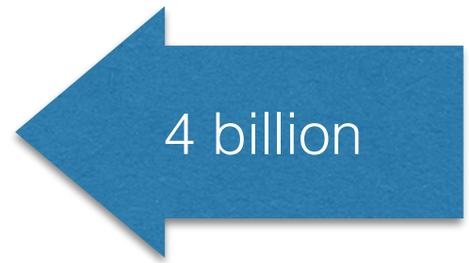
Direct-Address-Delete(S,x)
T[x.key] = NIL

$O(1)$

Suppose…

- you are asked to store a dynamic set of 300 IP (Internet protocol) addresses for the set of active customers of a Web service. (IP address are 32 bits that encode the location of a computer on the internet.)

- We could use direct addressing!
- We only need an array of $2^{32} \approx 4 \times 10^9$
- Or we could just store the IP address in a ____



Hash Tables

maintain a dynamic set

Hashing and Hash Tables

- A hash table is an array T of size m
- A hash function creates an index in the array from an k ∈ $\mathcal{U}$

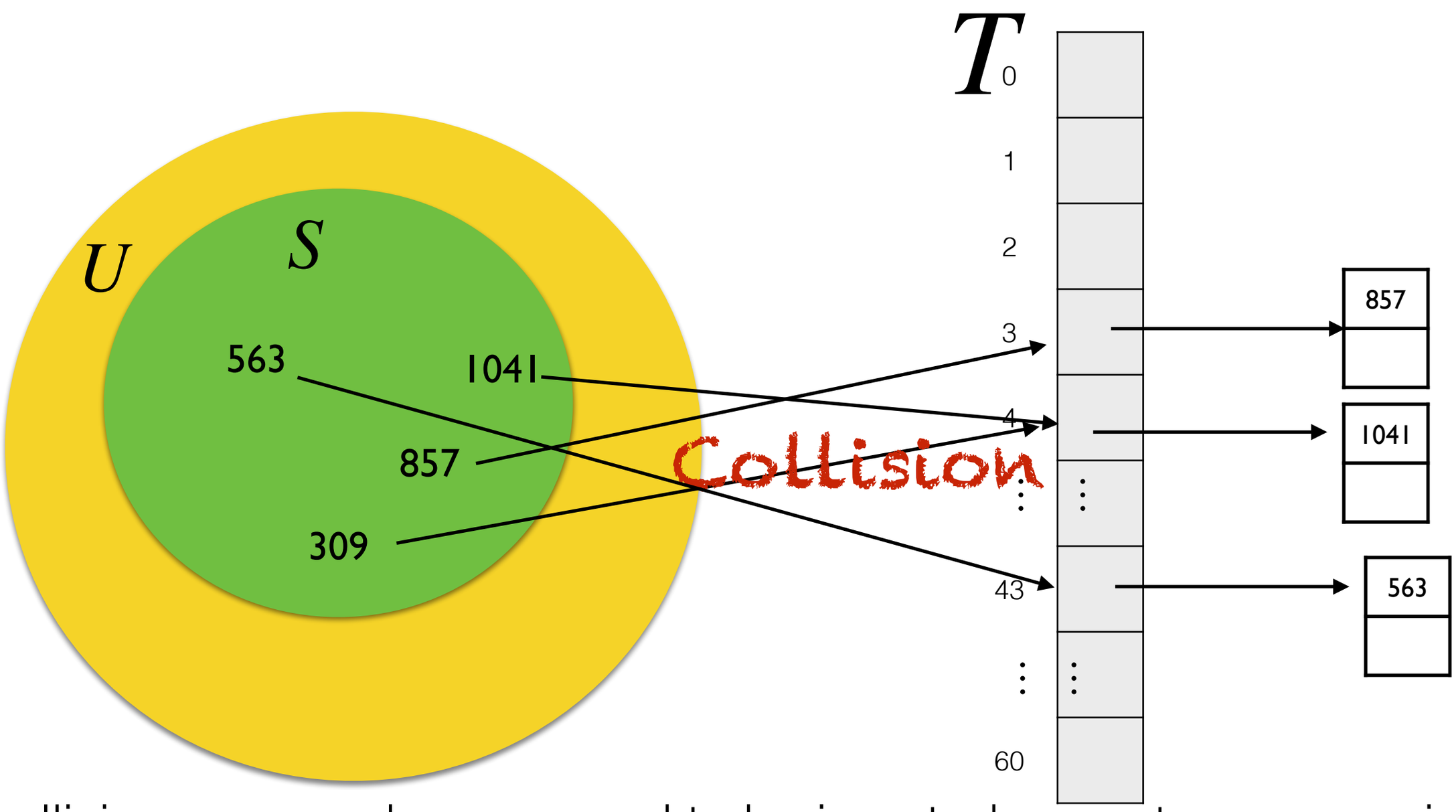
  h: $\mathcal{U}$→{0,…,m-1}

  "k ∈ $\mathcal{U}$ hashes to slot h(k) in T"
- How to we SEARCH(S,k), INSERT(S,x), and DELETE(S,x)?

Hashing

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

"k *hashes* to slot h(k)"
We call h a *hash function*



A collision occurs when a record to be inserted maps to an occupied slot

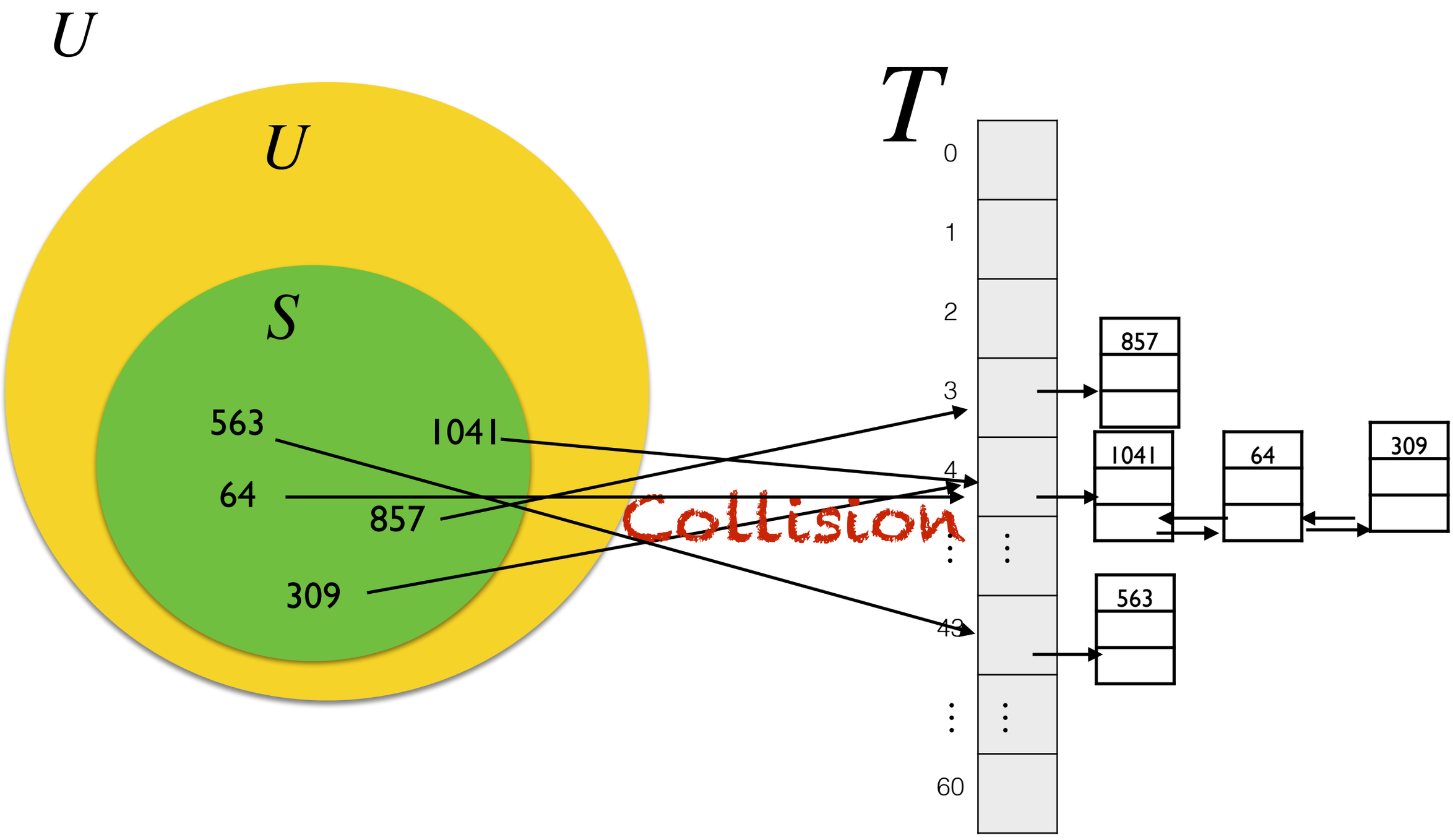
Collision Resolution!

Two common methods for collision resolution:

- chaining
- open addressing

Separate Chaining

h(1041) = h(309) = h(64) = 4



Running time?

- Complexity of computing h(k) for k ∈ $\mathcal{U}$
- $|\mathcal{U}|$ compared to | $\mathbb{S}$ |
- Size of table, m = |T|, compared to | $\mathbb{S}$ |. We usually talk about the load factor α = | $\mathbb{S}$ |/m
- Worst case? Average case?
- Usually analyze running time wrt assumptions on hash function. How do we choose a good hash function h?

Dictionary Operations using Collision Resolution by Chaining

Queries (non modifying)

Chained-Hash-Search(S,k)

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

running time proportional to the size of the list

Modifying Operations

Chained-Hash-Insert(S,x)

insert x at the head of list T[x.key]

O(1)

this running time assumes we don't need to check for duplicates

ow running time proportional to the size of the list

Chained-Hash-Delete(S,x)

delete x from the list T[x.key]

given pointer to x and a doubly linked list, O(1)

ow running time proportional to the size of the list

So… how long does it really take?

Worst case?

Average case?

Assumption:

Book has a careful analysis of a successful search time on page 260

simple uniform hashing:

any key is equally likely to hash to any of the m slots T

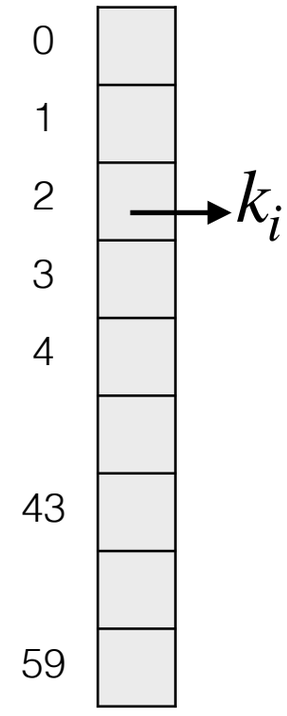
From this assumption, we can note that:

$$\text{for all } i \neq j, \quad \Pr\left[h(k_i) = h(k_j)\right] = \frac{1}{m}$$

Example:

Suppose $h(k_i) = 2$

$p[h(k_j) = 2] = 1/60$



Convenient assumption so we can analyze the run time

For any hash function there are many sets of keys which **violate** the **simple uniform hashing** assumption…

Suppose you had a hash table with 100 slots. If you assume the simple uniform hashing assumption for h, what is the probability $h(3)=h(4)$?

- 0
- 1
- 1/2
- 3/4
- 1/50
- 1/100
- None of the above

Suppose you had a hash table with 100 slots. If you assume the simple uniform hashing assumption for h, what is the probability $h(3)=h(4)$?

- 0
- 1
- 1/2
- 3/4
- 1/50
- 1/100
- None of the above

Suppose you had a hash table with 100 slots. If you assume the simple uniform hashing assumption for h, what is the probability $h(3)=h(4)$?

- 0
- 1
- 1/2
- 3/4
- 1/50
- 1/100
- None of the above

Analysis for Unsuccessful search

Assuming for all $k, k' \in U$, $\Pr[h(k) = h(k')] = \frac{1}{m}$ where m is the table size

Given x, y

Define $C_{x,y} = \begin{cases} 1 & h(x) = h(y) \\ 0 & \text{otherwise} \end{cases}$

$$E[C_{x,y}] = 1 \cdot \Pr[h(x) = h(y)] + 0 \cdot \Pr[h(x) \neq h(y)] = \frac{1}{m}$$

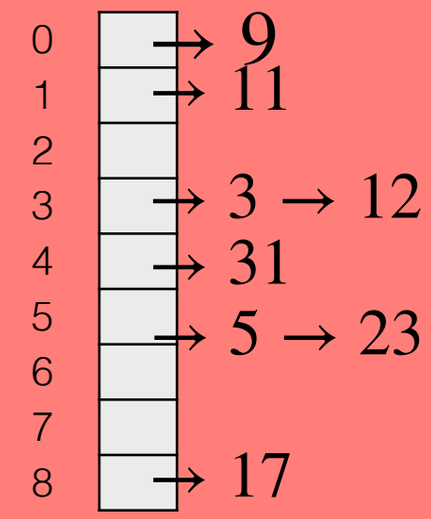
Let S be the items already in the table where x ∉ S.
How many items collide with x? Let |S| = n

Define the function:

$C_x = \sum_{y \in S, y \neq x} C_{x,y}$ = the # of items that x has a collision with

$E[C_x] = \sum_{y \neq x} E[C_{x,y}] = \frac{n}{m}$ = # items in the table, divided by the size of the table (m)

Example: $h(k) = k \mod 9$



$S = \{3,9,12,11,31,17,5,23\}$

$x = 14$

$C_{14,3} = 0 \quad C_{14,9} = 0 \quad \cdots$

$C_{14,5} = 1 \quad C_{14,23} = 1$

$C_{14} = C_{14,2} + C_{14,9} + \cdots + C_{14,5} + C_{14,23} = 2$

$x = 7$

$C_{7,3} = 0 \quad \cdots \quad C_{7,5} = 0 \quad C_{7,23} = 0$

$C_7 = C_{7,2} + C_{7,9} + \cdots + C_{7,5} + C_{7,23} = 0$

For any h

$E[C_{x,y}] = 1/9 \qquad E[C_x] = 8/9$

Running time!

if $n = O(m)$ then $\alpha = n / m = O(m) / m = O(1)$

Insertion, deletion, find all take O(1) time on average!!!!

Pro tip: if too many items are put into your table and the load factor gets too large, you can “rehash” your keys into a larger table

Assuming the Simple Uniform Hashing Assumption Average Case Running Times for Dictionary Operations using Collision Resolution by Chaining if $\alpha = n/m = O(m)/m = O(1)$

Hash function should ideally be simple to compute and should map different keys to different values.

Queries (non modifying)

Chained-Hash-Search(S,k)

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

Modifying Operations

Chained-Hash-Insert(S,x)

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

Chained-Hash-Delete(S,x)

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

Choosing a hash function

Read more in CLRS on pages 262-264

Distribute the keys uniformly in the slots of the table

- Division method: $h(k) = k \bmod m$
- Best if most keys and m are relatively prime - so don't pick m with small factors
- If $m = 2^r$, then only the hash function only uses the last r bits of k: e.g. if r = 7 and k = 01110100001110_2 then h(k) = 0111010_2
- Also don't pick m to be close to a power of 2 or 10 or a number common in the keys

Choosing a poor hash function...

- Invalidate *simple uniform hashing assumptions*
- which means we don't get constant running time...

Difficult to prove a hash function is good

- Some choices clearly don't work:
  - h(name) = |name|
  - h(phone number) = |number of digits|

- Cryptographic hash functions take too much time

Fundamental Weakness

For any hash function h we can find a set of keys, S={k_1,k_2…,k_n}, that are hashed to the same spot

$$\text{for all } k_j \in S, \quad h(k_1) = h(k_2) = \cdots = h(k_n)$$

Thus chain length is n

How can we guarantee the conditions needed to prove average constant time bounds?

How can we find a function that distributes the keys nicely?

A malicious adversary could choose the keys to have the same hash value..

Solution?

Randomness!

Would it work to create a hash function h that on input x randomly chose a number between 0 and $m-1$ each time it was used?

Yes

No

Would it work to create a hash function h that on input x randomly chose a number between 0 and $m-1$ each time it was used?

Yes

No

Would it work to create a hash function h that on input x randomly chose a number between 0 and $m-1$ each time it was used?

Yes

No

$U = \{0,1,2,\ldots,m-1\}$

Universal Hashing

Choosing the hash function independently of the keys to be stored!

choose h randomly from:

$\mathcal{H} = \{h_1, h_2, \ldots, h_w\}$ a finite collection of hash functions with certain mathematical properties

$U = \{0,1,2,\ldots,m-1\}$

Universal Class of Hash Functions ☺

$\mathcal{H} = \{h_1, h_2, \ldots, h_w\}$ is a universal hash function family if

$$\text{for all } k,k' \in U \qquad \Pr_{h \in \mathcal{H}}\left[h(k) = h(k')\right] \leq \frac{1}{m}$$

where m is the table size

HOW CAN WE CONSTRUCT A UNIVERSAL HASH FUNCTION FAMILY

$$U = \{0,1,2,\ldots,m-1\}$$

Universal Family $\mathcal{H}$

p is a prime larger than any key

Table size

Theorem $\mathcal{H}_{pm} = \left\{ h_{a,b}(x) = ((ax+b) \bmod p) \bmod m \right.$

$\left. \{ 1 \le a \le p-1, 0 \le b \le p-1 \} \right.$ is universal

p(p-1) choices for a and b

Examples of $h_{ab} \in \mathcal{H}_{pm}$: p =10007, m = 101

$h_{3,9}(x) = ((3x+9) \bmod p) \bmod m$

$h_{3,9}(100) = ((3 \cdot 100 + 9) \bmod 10007) \bmod 101$

$h_{3,9}(17) = ((3 \cdot 17 + 9) \bmod 10007) \bmod 101$

$h_{7,3}(x) = ((7x+3) \bmod p) \bmod m$

$h_{7,3}(100) = ((7 \cdot 100 + 3) \bmod 10007) \bmod 101$

$h_{4,0}(x) = ((4x) \bmod p) \bmod m$

$h_{4,0}(100) = ((4 \cdot 100) \bmod 10007) \bmod 101$

Example

- Choose two *keys*, *k, k'*, in U={1, 2, 3, 4, 5, 6,…, 1000}
- Choose a universal hash function:
  - Let p = 10007, let m = 100
  - Randomly choose *a* in {1,2,…,10006}
  - Randomly choose *b* in {0,1,2,…,10006}
- What do you think the probability that:

  $h_{10007,100}(k)=h_{10007,100}(k')$

  i.e. prob that $((ak + b) \bmod p) \bmod m = (ak'+b) \bmod p) \bmod m$

The proof has 4 main steps

The only place two distinct keys k, l can collide is after mod p or mod m

Step 1 if $h_{ab}(k)=h_{ab}(l)$ it is because they collided after mod p or mod m

Property of prime numbers

Modulo arithmetic

We prove the two distinct keys cannot collide after mod p

$$ak + b \mod p \neq al + b \mod p$$

Step 2 if $h_{ab}(k)=h_{ab}(l)$ it is because they collided after mod m

1-1 correspondence

Let

$$r = ak + b \mod p, \text{ and } s = al + b \mod p$$

We show there is a 1-1 relationship between r, s and the parameters of the hash function a, b

Step 3

There is a 1-1 correspondence between the (r,s) pairs and the (a,b) pairs

$p(p-1)$ functions in $\mathcal{H}_{pm}$

Now we show the number of collisions that can occur between r and s is at most $p(p-1)/m$. This is the same number of collisions that occur between a and b. Thus at most $1/m$ of the hash functions in $\mathcal{H}_{a,b}$ will cause k and l to collide. QED

Properties of $\lceil p/m \rceil$

Probabilities

Step 4

If I randomly choose an (r,s) pair, r≠s, the probability that they collided mod m is <= 1/m

Thus if I randomly choose an (a,b) pair the probability that $h_{ab}(k)=h_{ab}(l)$ is at most 1/m

Lemma

$U = \{0,1,2,\ldots,m-1\}$

$\mathcal{H}_{pm} = \{h_{a,b}(x) = ((ax+b) \bmod p) \bmod m \quad 1 \le a \le p-1, 0 \le b \le p-1, \text{ and } p \ge \text{all keys}\}$

is a universal family of hash functions

p is a prime larger than any key

we assume p> m

k and l are keys

proof: Let $k,l \in U$ such that $k \neq l$ (wlog $k > l$).

For $h_{ab} \in \mathcal{H}_{pm}$, what is the probability that $h_{ab}(k) = h_{ab}(l)$?

Step 1

There are two places for a collision: mod p and mod m

let $r = (ak+b) \bmod p$, let $s = (al+b) \bmod p$

Step 2

Note the collison cannot be mod p since o/w $r - s \equiv a(k-l) \pmod p$

Thus it must be that $r \neq s$

k≠l,
k,l < p so (k-l)<p and cannot divide p

a<p so a doesn't divide p!

Step 1 if $h_{ab}(k)=h_{ab}(l)$ it is because they collide after mod p or mod m

Step 2 if $h_{ab}(k)=h_{ab}(l)$ it is because they collide after mod m

Modulo arithmetic

$r - s \equiv ak + b - (al + b) \equiv a(k - l) \pmod p$

Lemma

$U = \{0,1,2,\ldots,m-1\}$

$\mathcal{H}_{pm} = \{h_{a,b}(x) = ((ax+b) \bmod p) \bmod m \quad 1 \le a \le p-1, 0 \le b \le p-1, \text{ and } p \ge \text{all keys}\}$ is a universal family of hash functions

p is a prime larger than any key

we assume p> m

proof: Let $k,l \in U$ such that $k \ne l$ (wlog $k > l$).

k and l are keys

For $h_{ab} \in \mathcal{H}_{pm}$, what is the probability that $h_{ab}(k) = h_{ab}(l)$?

Step 1

There are two places for a collision: mod p and mod m

let $r = (ak+b) \bmod p$, let $s = (al+b) \bmod p$

Step 2

Note the collison cannot be mod p since o/w $0 = r - s \equiv a(k-l) \pmod p$

k≠l, k,l < p so (k-l)<p and cannot divide p

a<p so a doesn't divide p!

Thus it must be that $r \ne s$

Step 3

Solving for a and b given r and s:

$a = ((r-s)((k-l)^{-1} \bmod p)) \bmod p$

$b = (r - ak) \bmod p$

There are $p(p-1)$ possible pairs (r,s) with $r \ne s$, thus there is a 1-1 correspondence between pairs (a,b) and pairs (r,s)

Thus if a, b is chosen randomly the pair (r,s) is equally likely to be any pair of distinct values modulo p

Step 1 if $h_{ab}(k)=h_{ab}(l)$ it is because they collide after mod p or mod m

Step 2 if $h_{ab}(k)=h_{ab}(l)$ it is because they collide after mod m

Step 3
There is a 1-1 correspondence between the (r,s) pairs and the (a,b) pairs

Step 4

Given r, the # of choices for s such that $(r \ne s)$ and $s \equiv r \pmod m$ is at most $\lceil p/m \rceil - 1$

Justification: we can list all the #'s < p that collide with r (including r). Let $r' = r \bmod m$

$r', r'+m, r'+2m, r'+3m, r'+4m, \ldots, r'+\left(\left\lfloor \frac{p}{m} \right\rfloor - 1\right)m$ are all less than p

$r'+\left(\left\lceil \frac{p}{m} \right\rceil - 1\right)m$ may be less than p

$r'+\left(\left\lceil \frac{p}{m} \right\rceil\right)m$ is greater than p

Lemma

$U = \{0,1,2,\ldots,m-1\}$

$\mathcal{H}_{pm} = \{h_{a,b}(x) = ((ax+b) \bmod p) \bmod m \quad 1 \le a \le p-1, 0 \le b \le p-1, \text{ and } p \ge \text{all keys}\}$

is a universal family of hash functions

p is a prime larger than any key

we assume p> m

k and l are keys

proof: Let $k,l \in U$ such that $k \ne l$ (wlog $k > l$).

For $h_{ab} \in \mathcal{H}_{pm}$, what is the probability that $h_{ab}(k) = h_{ab}(l)$?

Step 1

There are two places for a collision: mod p and mod m

let $r = (ak+b) \bmod p$, let $s = (al+b) \bmod p$

Step 2

Note the collison cannot be mod p since o/w $r - s \equiv a(k-l) \pmod p$

k≠l,
k,l < p so (k-l)<p and cannot divide p

a<p so a doesn't divide p!

Thus it must be that $r \ne s$

Step 3

Solving for a and b given r and s:

$a = ((r-s)((k-l)^{-1} \bmod p)) \bmod p$

$b = (r - ak) \bmod p$

There are $p(p-1)$ possible pairs (r,s) with $r \ne s$, thus there is a 1-1 correspondence between pairs (a,b) and pairs (r,s)

Thus if a, b is chosen randomly the pair (r,s) is equally likely to be any pair of distinct values modulo p

Step 4

Given r, the # of choices for s such that $(r \ne s)$ $\lceil p/m \rceil - 1$

The # of pairs (r,s) with $r \ne s$ $\quad p\left(\lceil p/m \rceil - 1\right) \le p\left((p+m-1)/m - 1\right) = p \cdot (p-1)/m$

The probality of a collision is at most $\dfrac{\text{total \# collisions}}{\text{total \# pairs}} \le \dfrac{p(p-1)/m}{p(p-1)} = 1/m$

Arithmetic

$$\frac{p+m-1}{m} - \frac{m}{m} = \frac{p+m-1-m}{m} = \frac{p-1}{m}$$

Modulo arithmetic

$r - s \equiv ak + b - (al + b) \equiv a(k-l) \pmod p$

Step 1 if $h_{ab}(k)=h_{ab}(l)$ it is because they collide after mod p or mod m

Step 2 if $h_{ab}(k)=h_{ab}(l)$ it is because they collide after mod m

between the (r,s) pairs and the (a,b) pairs

Lemma

$U = \{0,1,2,\ldots,m-1\}$

$\mathcal{H}_{pm} = \{h_{a,b}(x) = ((ax+b) \bmod p) \bmod m \quad 1 \le a \le p-1, 0 \le b \le p-1, \text{ and } p \ge \text{all keys}\}$

is a universal family of hash functions

p is a prime larger than any key

we assume p> m

k and l are keys

proof: Let $k,l \in U$ such that $k \ne l$ (wlog $k > l$).

For $h_{ab} \in \mathcal{H}_{pm}$, what is the probability that $h_{ab}(k) = h_{ab}(l)$?

Step 1

There are two places for a collision: mod p and mod m

let $r = (ak+b) \bmod p$, let $s = (al+b) \bmod p$

Step 2

Note the collison cannot be mod p since o/w $0 = r - s \equiv a(k-l) \pmod p$

k≠l, k,l < p so (k-l)<p and cannot divide p

a<p so a doesn't divide p!

Thus it must be that $r \ne s$

Step 3

Solving for a and b given r and s:

$a = ((r-s)((k-l)^{-1} \bmod p)) \bmod p$

$b = (r - ak) \bmod p$

There are $p(p-1)$ possible pairs (r,s) with $r \ne s$, thus there is a 1-1 correspondence between pairs (a,b) and pairs (r,s)

Thus if a, b is chosen randomly the pair (r,s) is equally likely to be any pair of distinct values modulo p

Step 4

Given r, the # of choices for s such that $(r \ne s)$ and $s \equiv r \pmod m$ is at

The # of pairs (r,s) with $r \ne s$ $\quad p\left(\lceil p/m \rceil - 1\right) \le p\left((p+m-1)/m - 1\right)$

The probality of a collision is at most $\dfrac{\text{total \# collisions}}{\text{total \# pairs}} \le \dfrac{p(p-1)/m}{p(p-1)}$ =

Modulo arithmetic

$r - s \equiv ak + b - (al + b) \equiv a(k-l) \pmod p$

between the (r,s) pairs and the (a,b) pairs

Step 1 if $h_{ab}(k)=h_{ab}(l)$ it is because they collide after mod p or mod m

Step 2 if $h_{ab}(k)=h_{ab}(l)$ it is because they collide after mod m

Step 4

If I randomly choose an (r,s) pair, r≠s, the probability that they collide mod m is ≤1/m

Thus if I randomly choose an (a,b) pair the probability that $h_{ab}(k)=h_{ab}(l)$ is at most 1/m

Analysis

Theorem If for all $k, k' \in U$, $\Pr_{h \in \mathcal{H}}[h(k) = h(k')] \leq \frac{1}{m}$ where m is the table size

then for any $S \subseteq U$, for any $x \in U - S$ the expected number of collisions between x and the other elements in S is at most n/m where $|S| = n$

Proof:

For our fixed x, define: Given x, y

$$C_{x,y} = \begin{cases} 1 & h(x) = h(y) \\ 0 & \text{otherwise} \end{cases}$$

Let S be the items already in the table where $x \notin S$. How many items collide with x? Let |S| = n

$$E[C_{x,y}] = \Pr[h(x) = h(y)] \leq \frac{1}{m}$$

Define the function:

$C_x = \sum_{y \in S, y \neq x} C_{x,y}$ = the # of items in S that x has a collision with

$E[C_x] = \sum_{y \in S,\ y \neq x} E[C_{x,y}] \leq \frac{n}{m}$ = the number of items in the table, divided by the size of the table (m)

Corollary 1: If for all $k, k' \in U$, $\Pr_{h \in \mathcal{H}}\left[h(k) = h(k')\right] \leq \frac{1}{m}$ where m is the table size

then for any $S \subseteq U$, for any $x \in U - S$ the expected number of comparisons for a successful search is O(1 + n/m) and for an unsuccessful search is O(n/m) where $|S| = n$

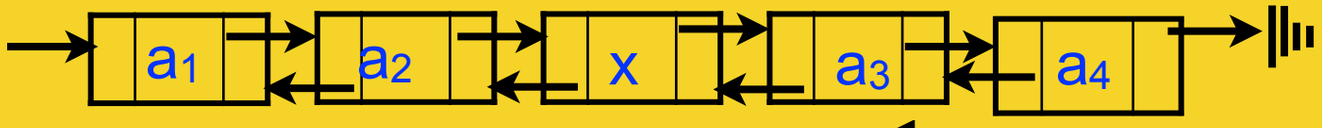
If n/m gets too large - rehash. (Typically double the table size)

Proof: Let |S| = n

From the previous theorem, we know the expected number of collisions is at most n/m.

Thus the number of comparisons for an unsuccessful search is O(n/m)

Thus the number of comparisons for a successful search is O(1+ n/2m)



The expected size of the list is 1+ n/m

$$\sum_{y \in S,\ y \neq x} E\left[C_{x,y}\right] \leq \frac{n-1}{m}$$

Corollary 2:
Using universal hashing and collision resolution by chaining in an initially empty table with m slots, it takes expected time $\Theta(n)$ to handle any sequence of n INSERT, SEARCH, and DELETE operations containing O(m) INSERT operations where m is the table size

Proof:

Since the number of insertions is O(m), we know the load factor α is O(1).

INSERT and DELETE take $\Theta(1)$ time, and using the previous corollary the expected running time of SEARCH is $\Theta(1)$ (ie. n/m is O(1)).

Using the linearity of expectation, the expected running time for each of the n operations is O(1). We also know each operation takes time $\Omega(1)$. Combining these we get the running time for n operations is $\Theta(n)$

Open Addressing Collision Resolution #2!

When a collision happens, the algorithm looks for the an available slot. Examining a slot is known as a probe.

Hash function depends on key and *probe* number

h: U×{0,1, 2,…, m-1} → {0,1, 2,…, m-1}

h(k, 0), h(k, 1), h(k, 2), …, h(k, m-1)

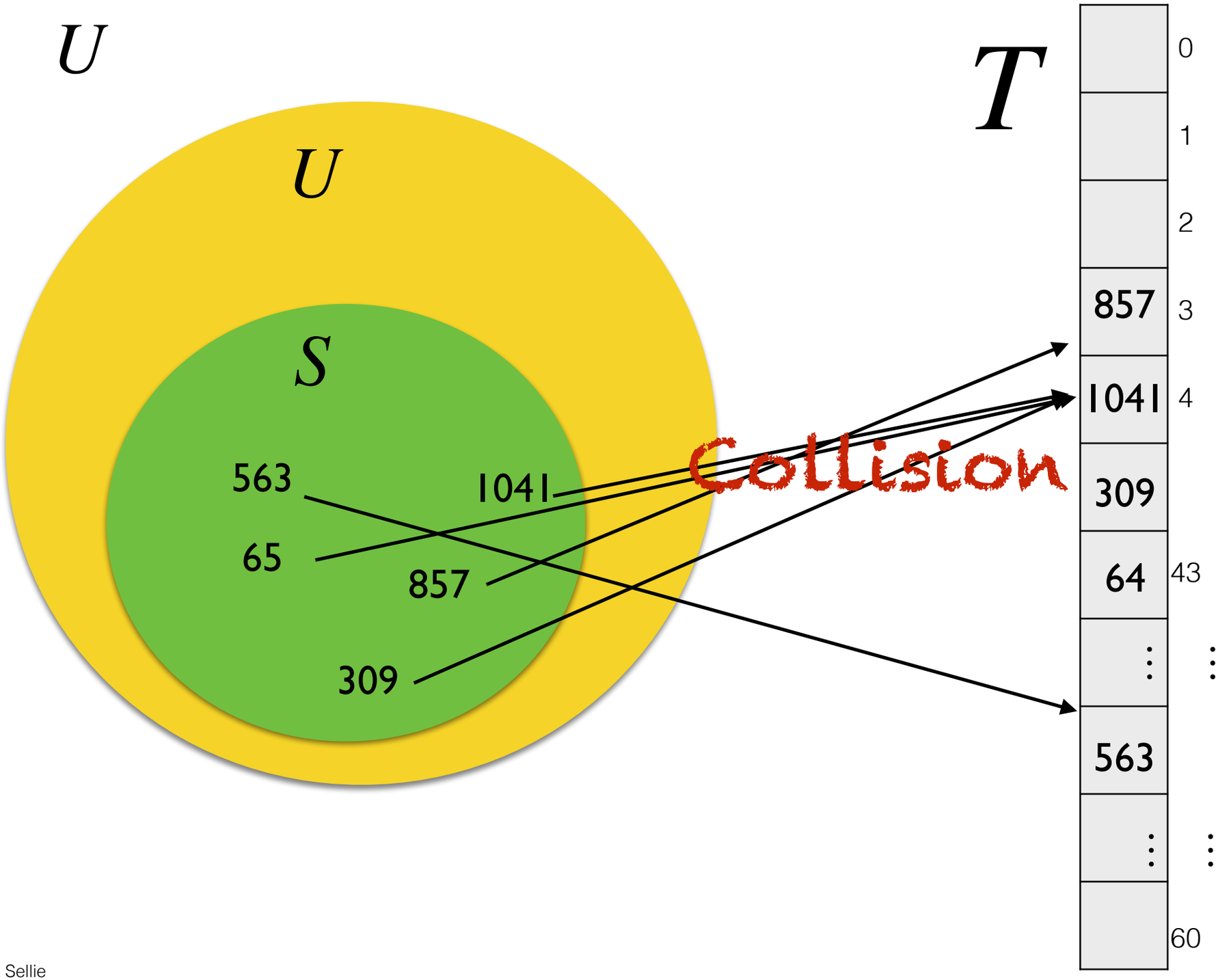
the sequence of slots *probed* should be a permutation of {0,1, 2,…, m-1}

The next item probed is determined by

- linear probing
- quadratic probing
- double hashing

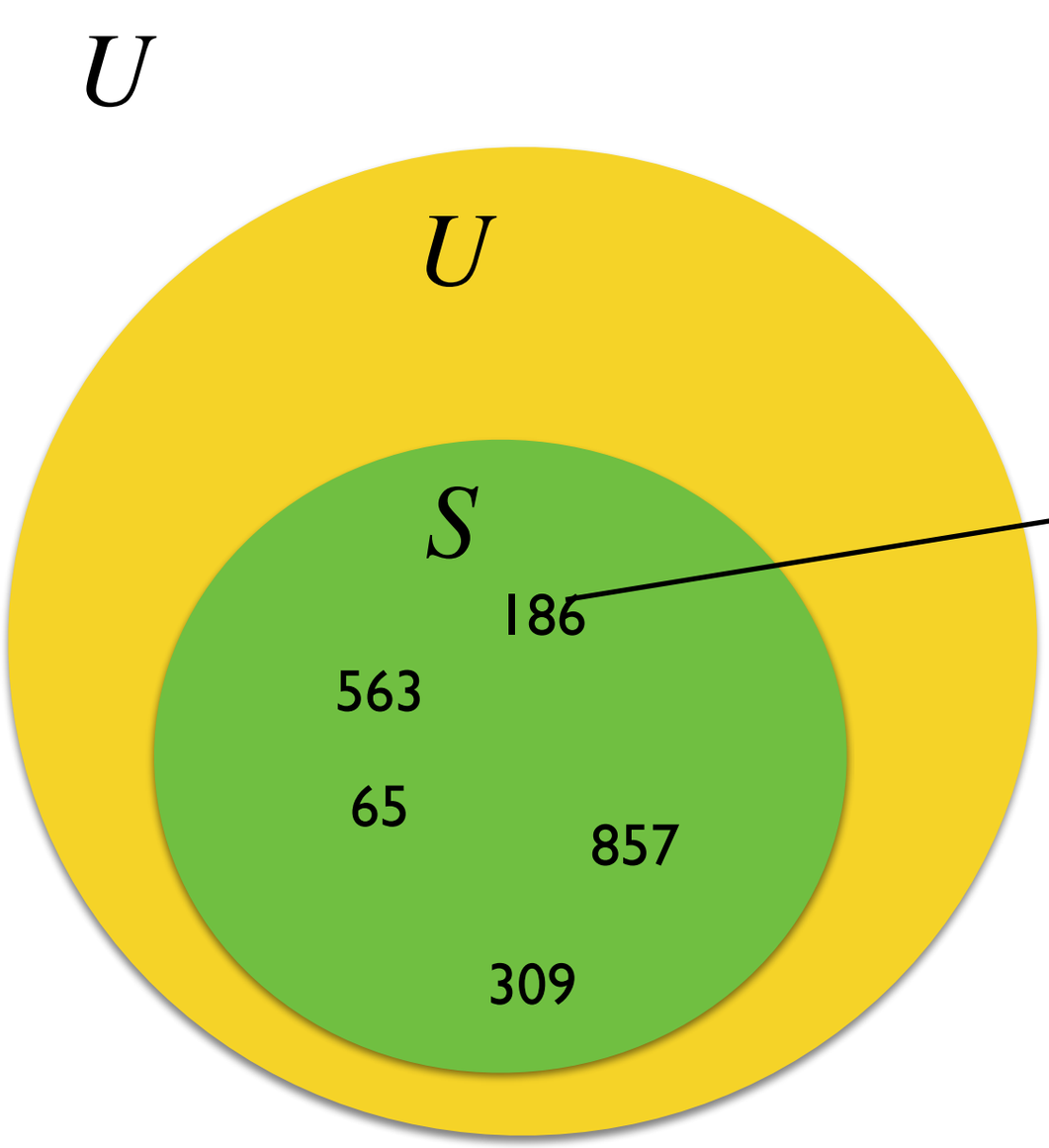
Linear Probing

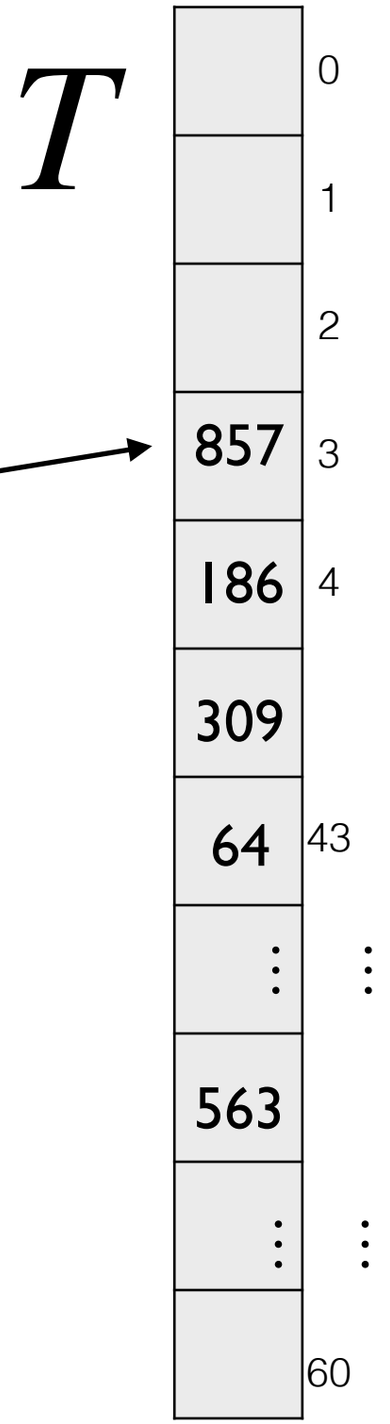
h'(1041) = h'(309) = h'(64) = 4



Linear Probing Deletion

h'(1041) = h'(309) = h'(64) = 4





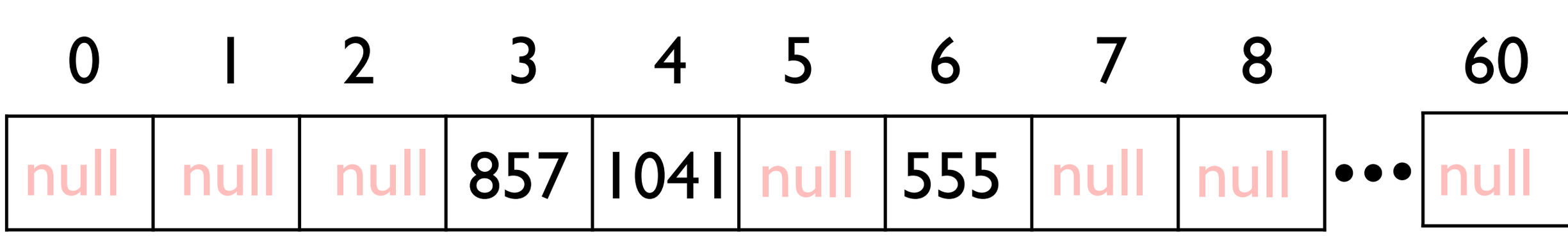
probe(h'(186),0)

probe(h'(186),1)

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

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

```
HASH-INSERT(T, k)
i =0
repeat
   j = h(k,i)
   if T[j] == NIL or T[j] == DELETED
      T[j]= k
      return j
   else i = i +1
until i == m
error "hash table overflow"
```



426?

h'(426)=60

Keys: Class list: 555, 857, 1041

Can we do better if the keys are static?

Static data:
- reserved words in a program language
- set of file names on a CD-ROM

Page 277-282 in CLRS

Perfect Hashing

- Given a fixed set of n keys we can construct a static hash table of size m = Θ(n) such that search takes Θ(1) time in the *worst* case

- We create a structure that has ***no* collisions**

Breakout rooms

How many collisions would we have if the table size was large?

$m = n^2$

Theorem Hashing n keys into $m = n^2$ slots using $h \in_R \mathcal{H}$ then $E(\#\ \text{collisions}) < 1/2$

($\mathcal{H}$ universal class of hash functions)

proof

Probability two keys collide is $1/m = 1/n^2$

pairs of keys is $\binom{n}{2}$

$$E(\#\ \text{collisions}) \le \binom{n}{2} \cdot \frac{1}{n^2} = \frac{n^2 - n}{2} \cdot \frac{1}{n^2} < \frac{1}{2}$$

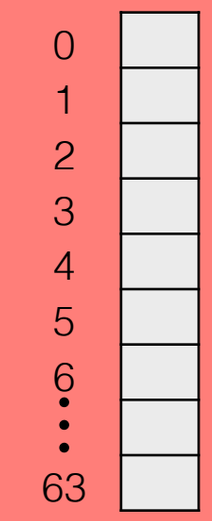
■

We are using linearity of expectation.

Example:

$S = \{3,9,12,11,31,17,5,23\}$

h randomly chosen from $H_{10007,64}$



$E[\#\text{collisons}] = \frac{8*7}{2}\frac{1}{64}$

$p\left(h(3) = h(9)\right) \le 1/64$

$p\left(h(3) = h(12)\right) \le 1/64$

$p\left(h(3) = h(11)\right) \le 1/64$

$p\left(h(3) = h(31)\right) \le 1/64$

$p\left(h(3) = h(17)\right) \le 1/64$

$p\left(h(3) = h(5)\right) \le 1/64$

$p\left(h(3) = h(23)\right) \le 1/64$

$p\left(h(9) = h(12)\right) \le 1/64$

$\vdots$

$p\left(h(5) = h(23)\right) \le 1/64$

For a fixed set S, what is the probability of choosing a hash function from $\mathscr{H}_{p,m}$ that is collision-free on S

Corollary: Probability of no collisions $> 1/2$

proof Let X be a discrete random variable holding the number of collisions, and using Markov's inequality with *t=1*

$$\Pr[X \geq 1] <= \frac{E[X]}{1} < 1/2$$

Markov's inequalilty:

For X a discrete random variable $\Pr[X \geq t] <= \frac{E[X]}{t}$

proof Let X be a discrete random variable

$$E[X] = \sum_{x=0}^{\infty} x \cdot Pr[X = x] \geq \sum_{x=t}^{\infty} x \cdot Pr[X = x] \geq \sum_{x=t}^{\infty} t \cdot Pr[X = x]$$

$$= t \cdot \sum_{x=t}^{\infty} Pr[X = x] = t \cdot Pr[X \geq t]$$

$$\therefore \Pr[X \geq t] <= \frac{E[X]}{t} \quad \blacksquare$$

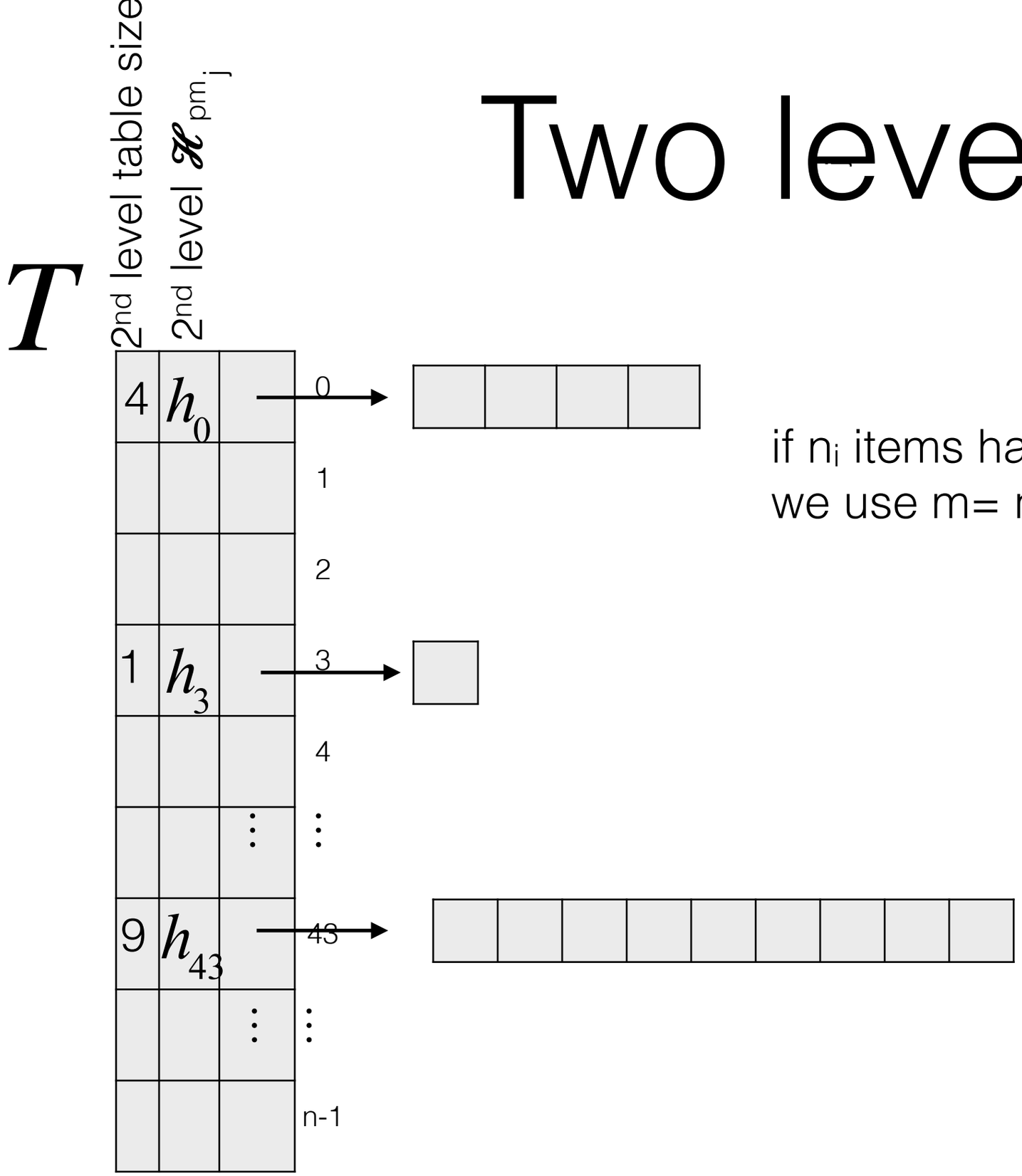
Find a collision-free hash function by trying a few $h \in_R \mathcal{H}$

Can we find a way that doesn't take so much space?

Can we do this in O(n) space?

Solution: Two level scheme using *universal hashing* at both levels

Two levels



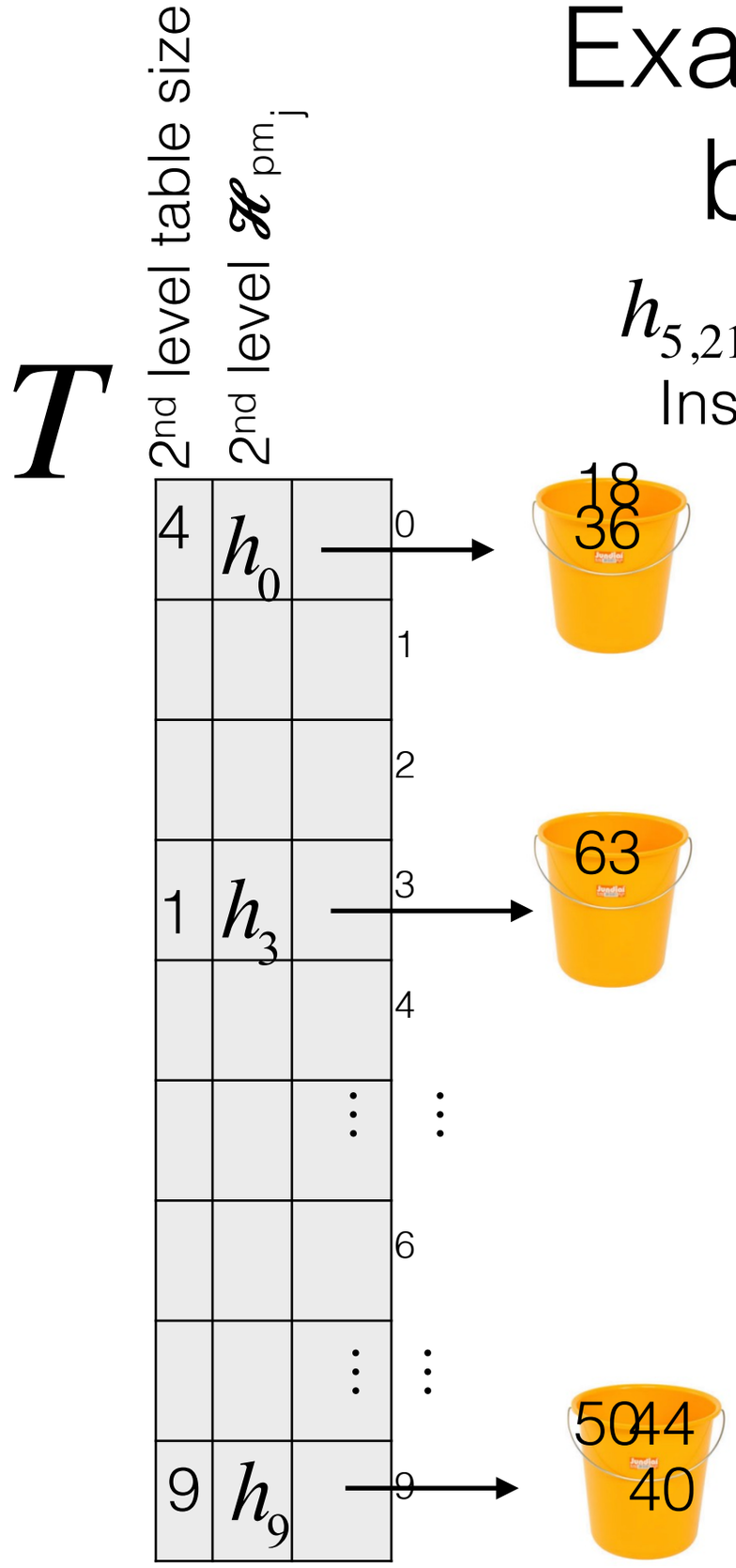
if n_i items hash to index i on level 1,
we use $m = n_i^2$ slots in the level 2 table

The first level is like hashing with chaining, but instead of chaining we use another second level hash table with another hash function that has no collisions

Example showing the steps to building perfect hashing

$$h_{5,21} = ((5x + 21) \bmod 101) \bmod 10$$

Insert: 36, 63, 44, 50, 18, 40



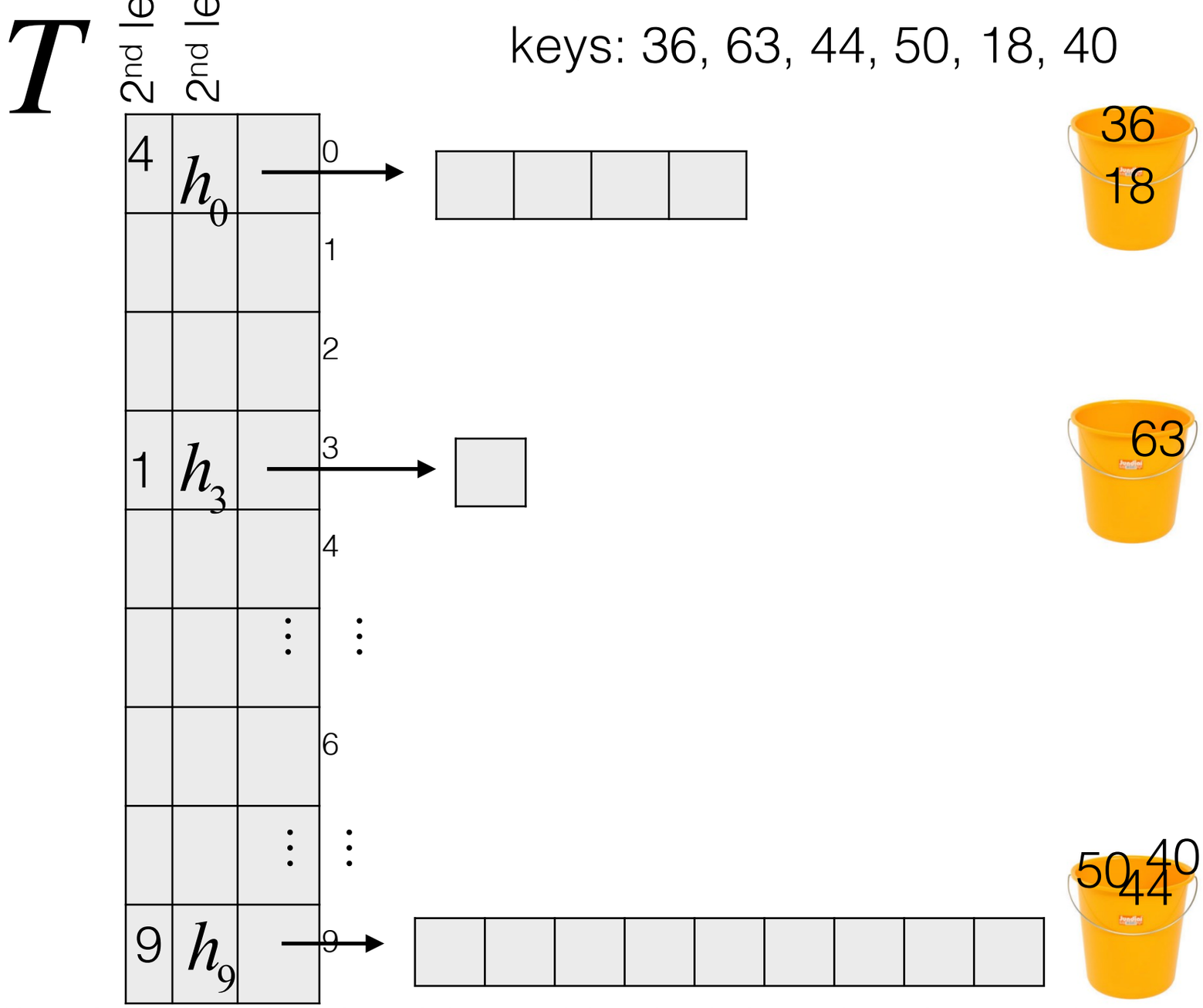
The first level is like hashing with chaining, but instead of chaining we use another second level hash table with another hash function that has no collisions

Bucket picture from https://upload.wikimedia.org/wikipedia/commons/3/31/Balde.PNG

Example continued

$$h_{5,21} = ((5x + 21) \bmod 101) \bmod 10$$

keys: 36, 63, 44, 50, 18, 40



The first level is like hashing with chaining, but instead of chaining we use another second level hash table with another hash function that has no collisions

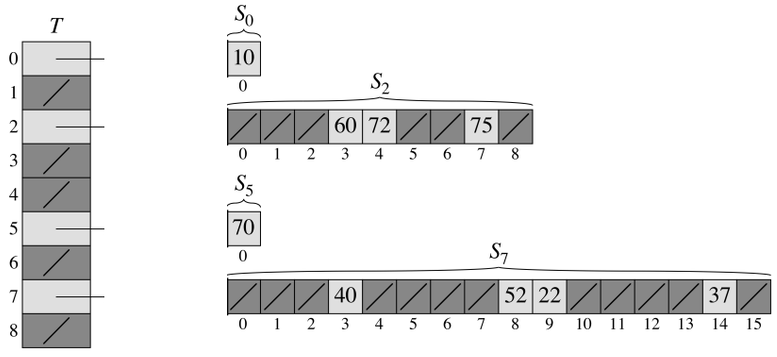
For each second level hash table its hash function h_j is chosen from a universal hash function family

How much memory is used?

Θ(n)!

*The small print disclaimer: we get Θ(n) with probability > 1/2. We might have to try multiple times to get this bound

The amount of space used in the secondary hash tables is $\sum_{j=0}^{m-1} (n_j)^2$



Keys = {10; 22; 37; 40; 52; 60; 70; 72; 75}

Picture modified from CLRS

Warning … S is used differently in the book!

Theorem: Let $m = n$ be the size of the hash table at level 1,
Let n_j^2 be the size of the hash table at index j,
Let h be randomly chosen from a universal set H, then

$$\Pr\left[\sum_{j=0}^{m-1}\left(n_j\right)^2 \geq 4n\right] < 1/2$$

proof Lets count the number of pairs of items that collide (including items that collide with themselves. This gives us the size of n_j)

$$E\left[\sum_{j=0}^{m-1}(n_j)^2\right] = E\left[\sum_{x\in S}\sum_{y\in S} C_{xy}\right] \text{ remember } C_{x,y} = \begin{cases} 1 \text{ if } h(x)=h(y) \\ 0 \text{ if } h(x)\neq h(y) \end{cases}$$

Example

If we knew
$S_9 = \{40,44,50\}$
$|S_9| = n_9 = 3$

$x = 40$

$C_{40,40} = 1$
$C_{40,44} = 1$
$C_{40,50} = 1$
$y \in S - S_9$ $C_{40,y} = 0$

$$\sum_{y\in S} C_{40,y} = n_9$$

$x = 44$

$C_{44,40} = 1$
$C_{44,44} = 1$
$C_{44,50} = 1$
$y \in S - S_9$ $C_{44,y} = 0$

$$\sum_{y\in S} C_{44,y} = n_9$$

$x = 50$

$C_{50,40} = 1$
$C_{50,44} = 1$
$C_{50,50} = 1$
$y \in S - S_9$ $C_{50,y} = 0$

$$\sum_{y\in S} C_{50,y} = n_9$$

$$\sum_{x\in S_9}\sum_{y\in S} C_{xy} = n_9 \cdot n_9$$

General case:

$$E\left[\sum_{x\in S_9}\sum_{y\in S} C_{xy}\right] = E[n_9^2]$$

This is true for all S_j

$$E\left[\sum_{j=0}^{m-1}\sum_{x\in S_j}\sum_{y\in S} C_{xy}\right]$$

Theorem: Let $m = n$ be the size of the hash table at level 1,
Let n_j^2 be the size of the hash table at index j,
Let h be randomly chosen from a universal set H, then

$$\Pr\left[\sum_{j=0}^{m-1}\left(n_j\right)^2 \geq 4n\right] < 1/2$$

proof Lets count the number of pairs of items that collide (including items that collide with themselves. This gives us the size of n_j)

$$E\left[\sum_{j=0}^{m-1}(n_j)^2\right] = E\left[\sum_{x\in S}\sum_{y\in S} C_{xy}\right] \text{ remember } C_{x,y} = \begin{cases} 1 \text{ if } h(x) = h(y) \\ 0 \text{ if } h(x) \neq h(y) \end{cases}$$

$$= n + \sum_{x\in S}\sum_{y\in S-x} E\left[C_{xy}\right] \quad \text{number of times x collides with itself is } n$$

$$\leq n + n(n-1)/m \qquad \text{if } x \neq y \text{ then } E[C_{xy}] \leq 1/m$$

$$< 2n \quad \text{since } m = n$$

Markov's inequalilty: $\Pr\left[\sum_{i=0}^{m-1}\left(n_i\right)^2 \geq 4n\right] < 2n/4n = 1/2$