An Introduction to Hashing

(Following CLRS)

COMP 3711 - HKUST
Version of 07/05/2019
M. J. Golin

Outline

- Introduction
- Hashing with Chaining
- Open Addressing
- Hash Functions & Universal Hashing
- Odds & Ends

Introduction

Known: A set $U = \{0, 1, 2, \ldots, u - 1\}$ of the universe of possible keys that could exist.

Goal: To maintain a dictionary that permits the following operation on keys

- **Search(x):** Find the record with key x or report that it does not exist
- **Insert(x):** Insert a new record with key x
- **Delete(x):** Delete the record with key x

Introduction

Known: A set $U = \{0, 1, 2, \ldots, u-1\}$ of the universe of possible keys that could exist.

Goal: To maintain a dictionary that permits the following operation on keys

- **Search(x):** Find the record with key x or report that it does not exist
- **Insert(x):** Insert a new record with key x
- **Delete(x):** Delete the record with key x

Would like $O(1)$ (average) time per operation.

Introduction (ii)

Universe Size: U
Number of actual keys: n $(n << U)$

Introduction (ii)

Universe Size: U
Number of actual keys: n $(n << U)$

Will store keys in a size m array (m "close" to n).
Need a way to assign key k to array location.

Introduction (ii)

Universe Size: U
Number of actual keys: n $(n << U)$

Will store keys in a size m array (m "close" to n).
Need a way to assign key k to array location.

Use a *hash function* h

$$h : U \to \{0, 1, \ldots, m - 1\}$$

Introduction (ii)

Universe Size: U
Number of actual keys: n $(n << U)$

Will store keys in a size m array (m "close" to n).
Need a way to assign key k to array location.

Use a *hash function* h

$$h : U \to \{0, 1, \ldots, m-1\}$$

$\alpha = \frac{n}{m}$ is *load factor,*

Introduction (ii)

Universe Size: U
Number of actual keys: n $(n << U)$

Will store keys in a size m array (m "close" to n).
Need a way to assign key k to array location.

Use a *hash function* h

$$h : U \to \{0, 1, \ldots, m-1\}$$

$\alpha = \frac{n}{m}$ is *load factor,*
average $\#$ of existing keys with same $h(x)$

Introduction (ii)

Universe Size: U
Number of actual keys: n $(n << U)$

Will store keys in a size m array (m “close” to n).
Need a way to assign key k to array location.

Use a *hash function* h

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

$\alpha = \frac{n}{m}$ is *load factor,*
average # of existing keys with same $h(x)$

For now, assume uniform hashing, that, every key is equally likely to hash to any of the m slots,

$$\forall x, i, \quad \Pr\left(h(x) = i\right) = \frac{1}{m}.$$

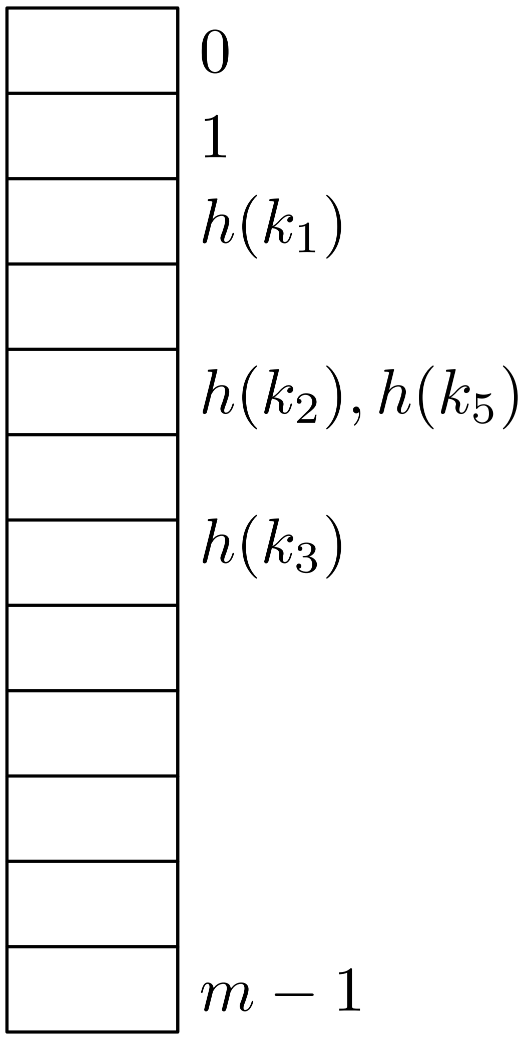
Introduction (iii)

$h : U \to \{0, 1 \ldots, m-1\}$

h maps the set of keys into a “small” table. Key k is stored in table slot $h(k)$.

Finding key k is then just a matter of going to table location $h(k)$.

Problem is that, since m is small, many keys might be mapped to the same slot, creating collision.



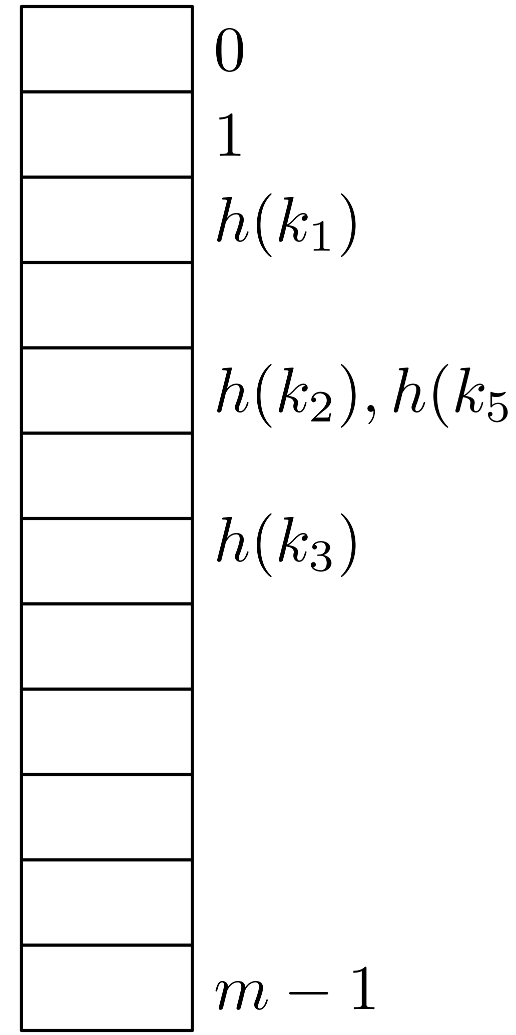
Introduction (iii)

$h : U \to \{0, 1 \ldots, m-1\}$

h maps the set of keys into a "small" table.
Key k is stored in table slot $h(k)$.

Finding key k is then just a matter of going to table location $h(k)$.

Problem is that, since m is small, many keys might be mapped to the same slot, creating collision.



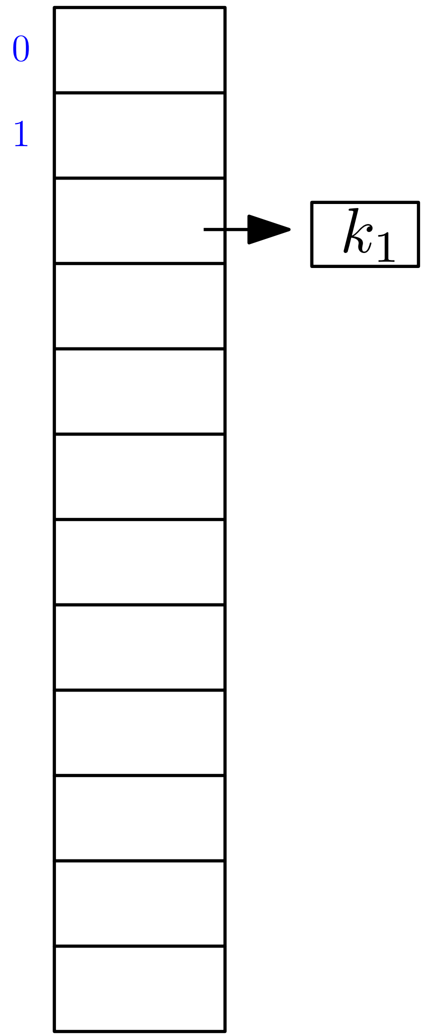
Two major approaches to addresing collisions:
(1) Chaining
(2) Open Addressing

Outline

- Introduction
- Hashing with Chaining
- Open Addressing
- Hash Functions & Universal Hashing
- Odds & Ends

Chaining

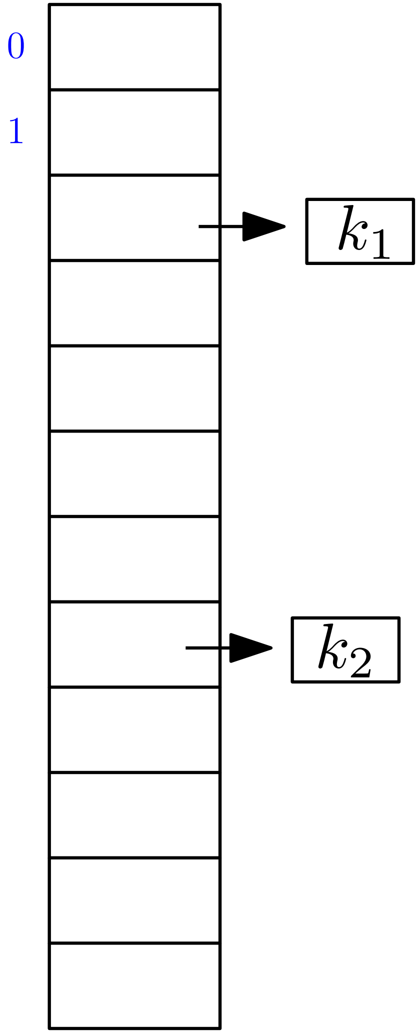
$$h : U \to \{0, 1 \ldots, m-1\}$$



All elements that hash to the same slot are put into the same linked list

Chaining

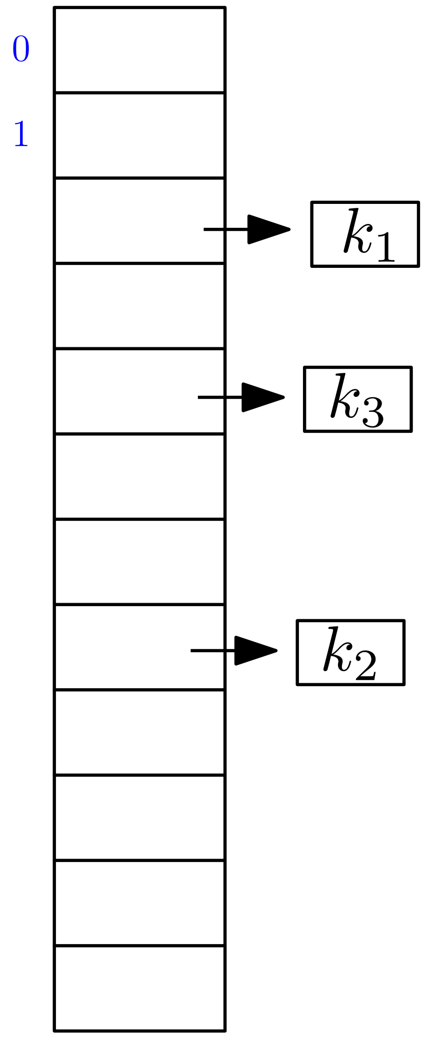
$h : U \to \{0, 1 \ldots, m-1\}$



All elements that hash to the same slot are put into the same linked list

Chaining

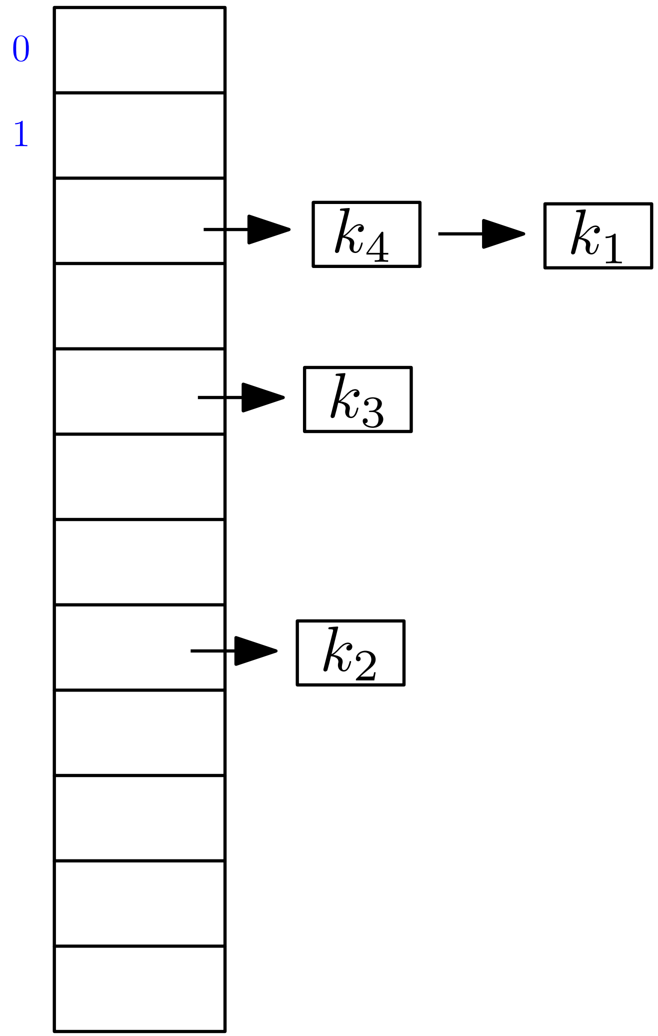
$h : U \to \{0, 1 \ldots, m-1\}$



All elements that hash to the same slot are put into the same linked list

Chaining

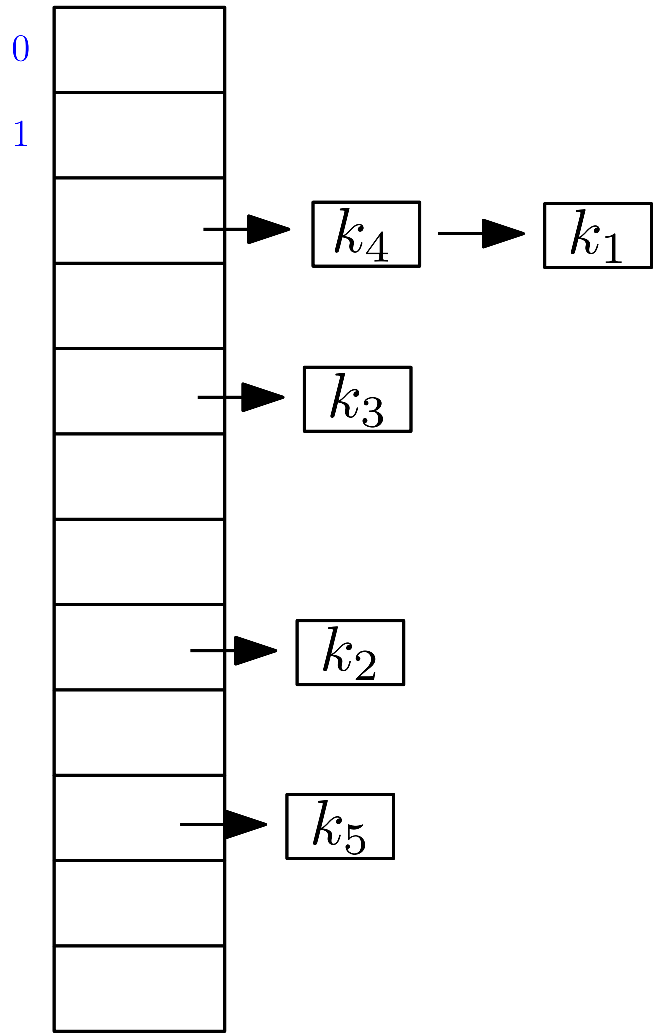
$h : U \to \{0, 1 \ldots, m-1\}$



All elements that hash to the same slot are put into the same linked list

Chaining

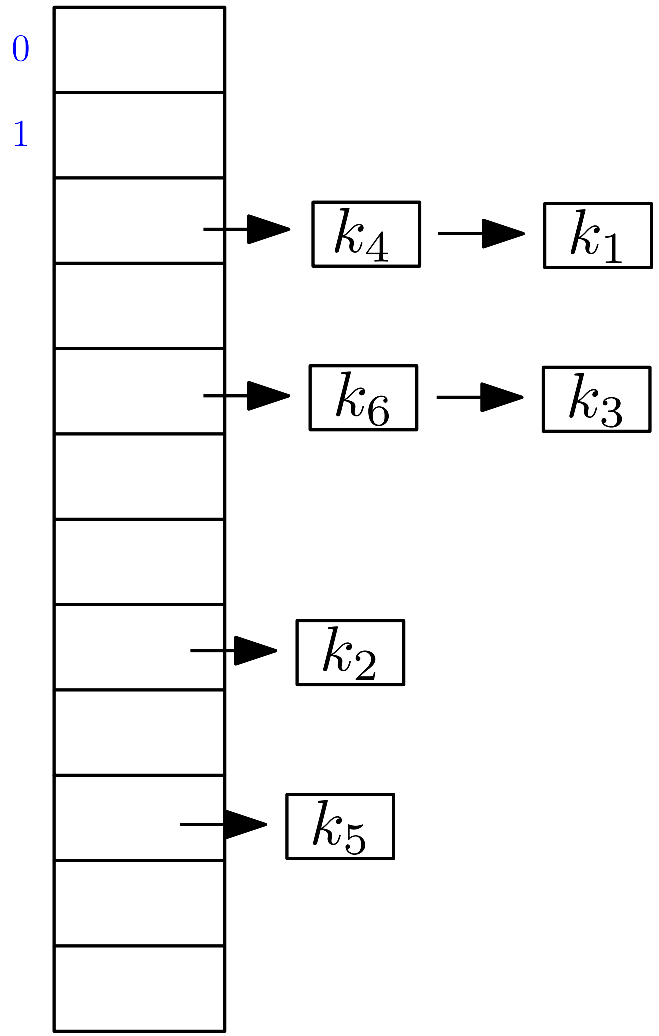
$h : U \to \{0, 1 \ldots, m-1\}$



All elements that hash to the same slot are put into the same linked list

Chaining

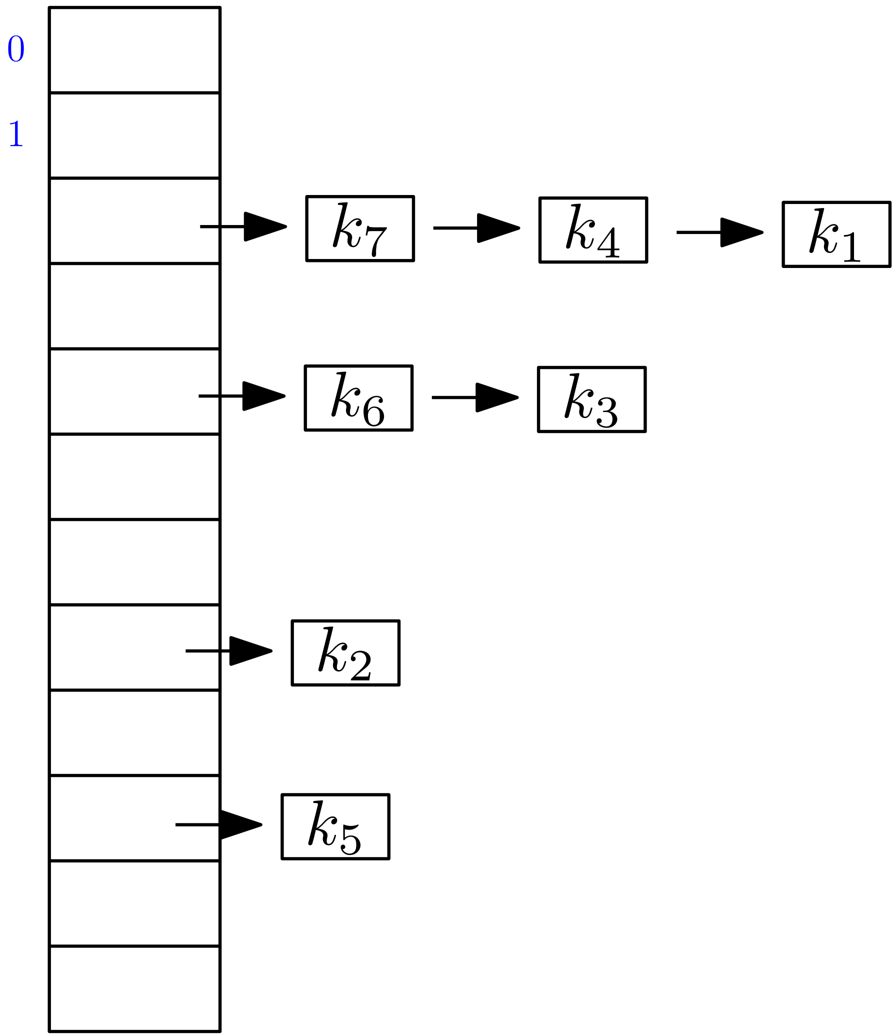
$h : U \to \{0, 1 \ldots, m-1\}$



All elements that hash to the same slot are put into the same linked list

Chaining

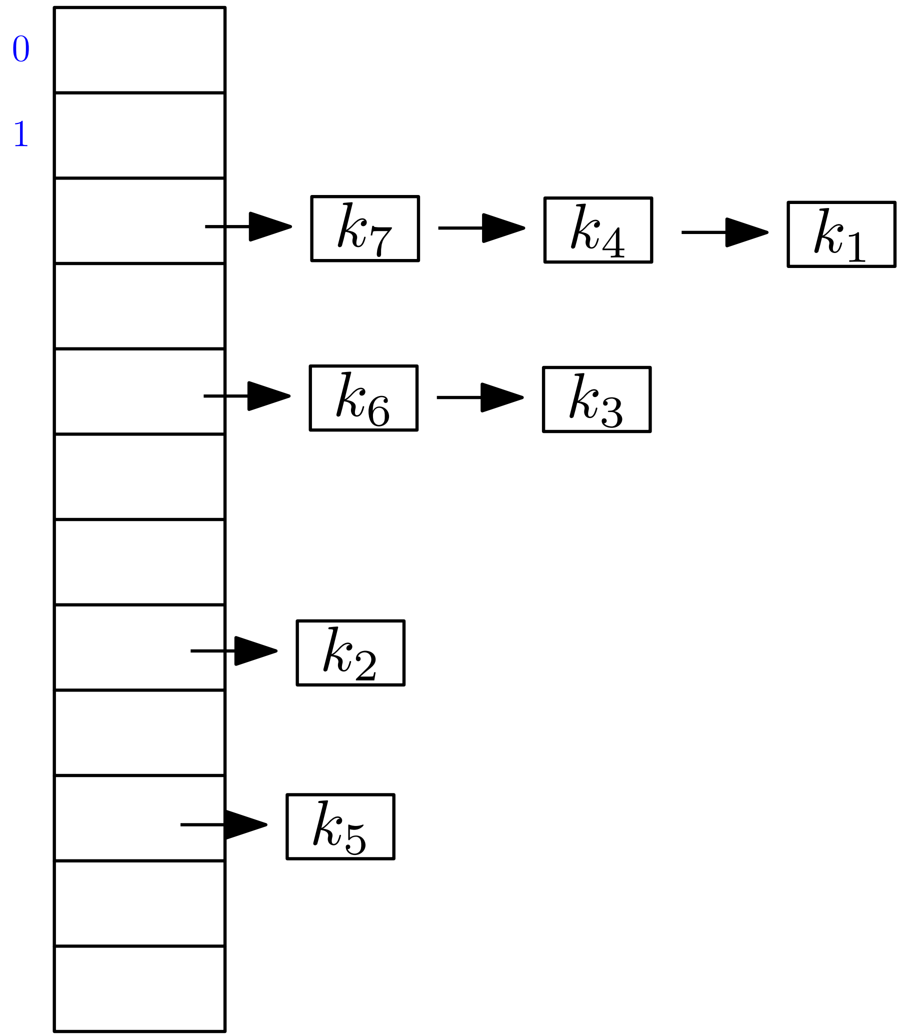
$h : U \to \{0, 1 \ldots, m-1\}$



All elements that hash to the same slot are put into the same linked list

Chaining

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



All elements that hash to the same slot are put into the same linked list

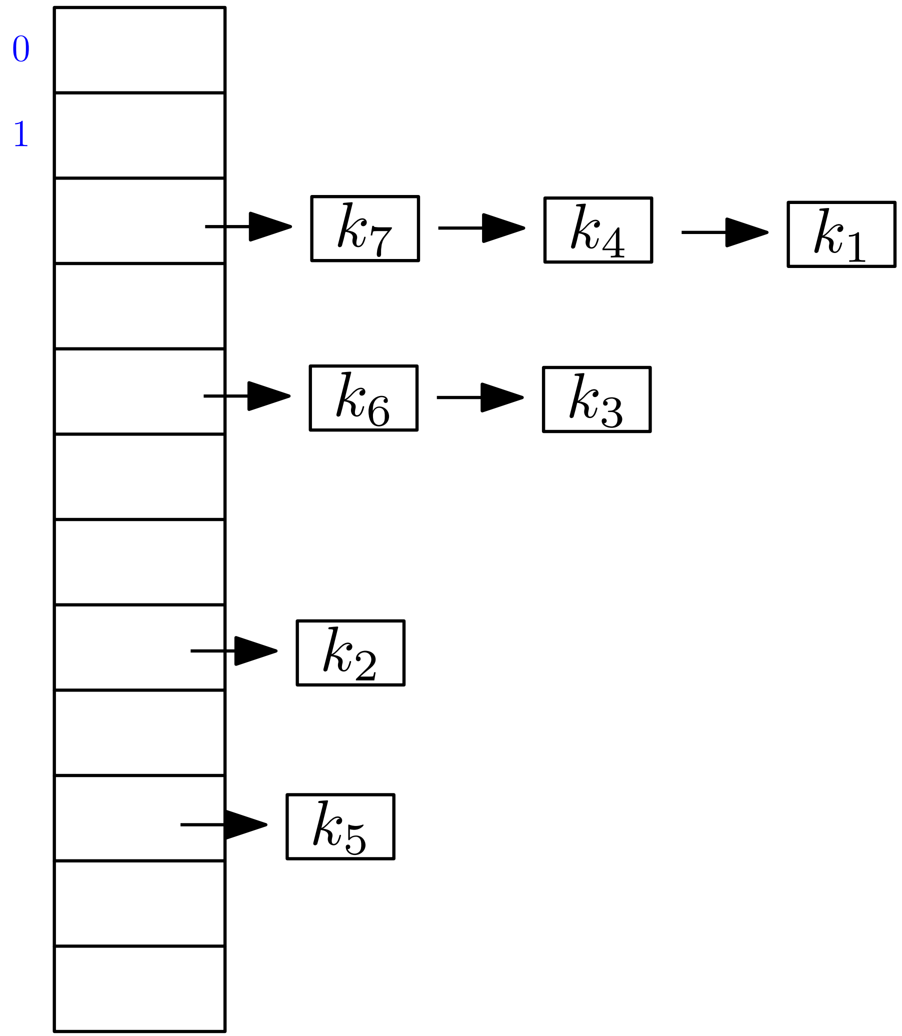
Insert(x):Insert x into front of list for slot $h(x)$

Delete(x): Delete x from list for slot $h(x)$, if it's there.
Use doubly linked lists

Search(x): Search for x in list for $h(x)$

Chaining

$h : U \to \{0, 1 \ldots, m-1\}$



All elements that hash to the same slot are put into the same linked list

Insert(x):Insert x into front of list for slot $h(x)$ $O(1)$

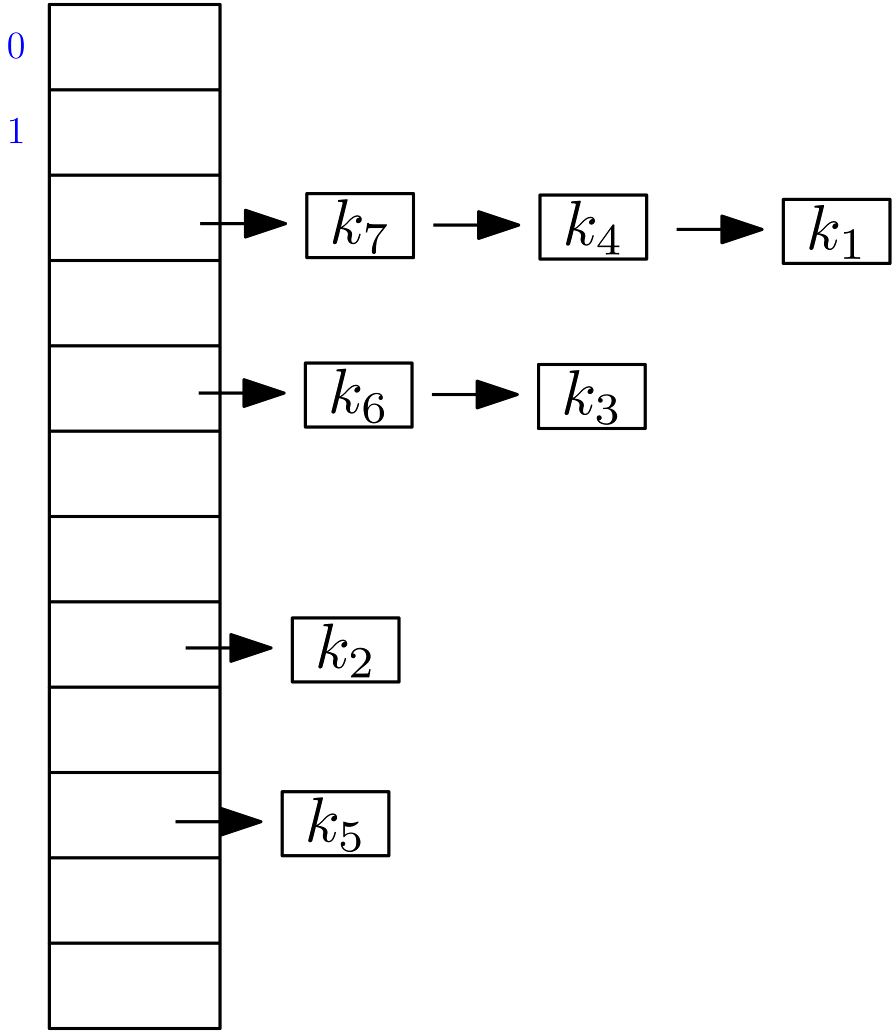
Delete(x): Delete x from list for slot $h(x)$, if it's there.
Use doubly linked lists $O(1)$

Search(x): Search for x in list for $h(x)$ $O($*length of list*$)$

Chaining: Unsuccessful Search

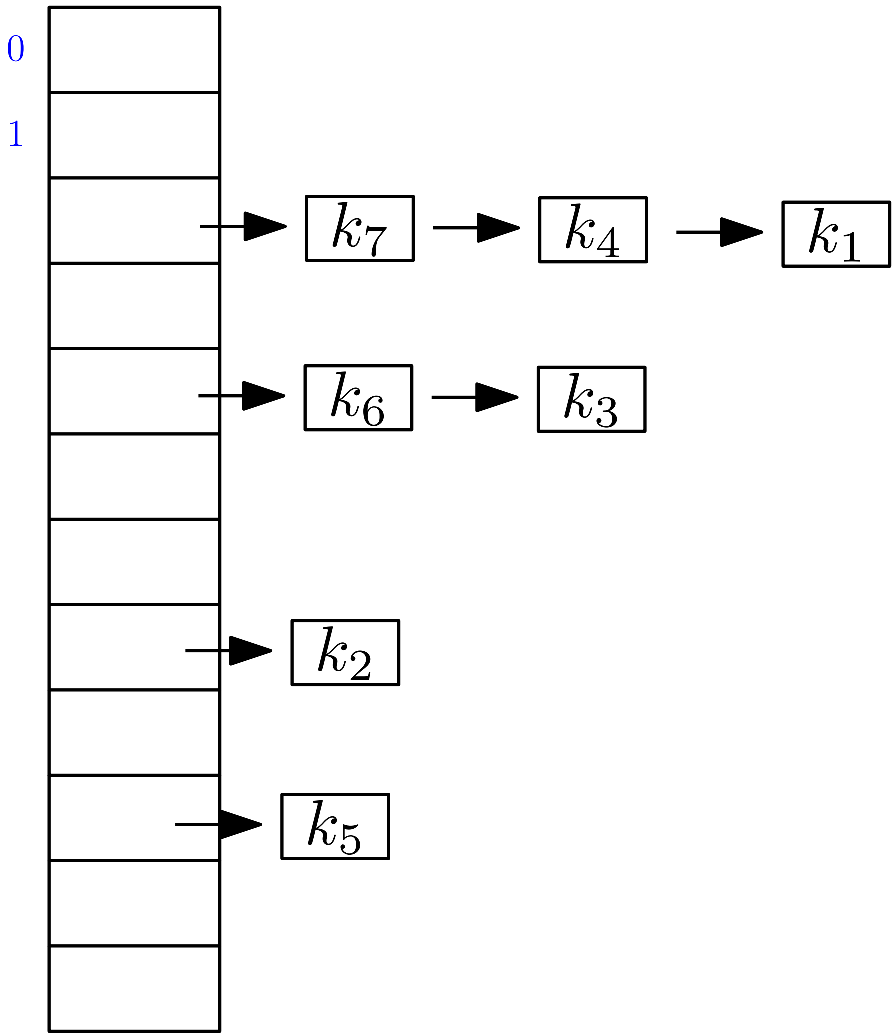
$h : U \to \{0, 1 \ldots, m-1\}$

Search(x): Search for x in list for $h(x)$ $\quad O(\textit{length of list})$



Chaining: Unsuccessful Search

$h : U \to \{0, 1 \ldots, m-1\}$



Search(x): Search for x in list for $h(x)$ $\quad$ *$O($length of list$)$*

Recall load factor $\alpha = \frac{n}{m}$.

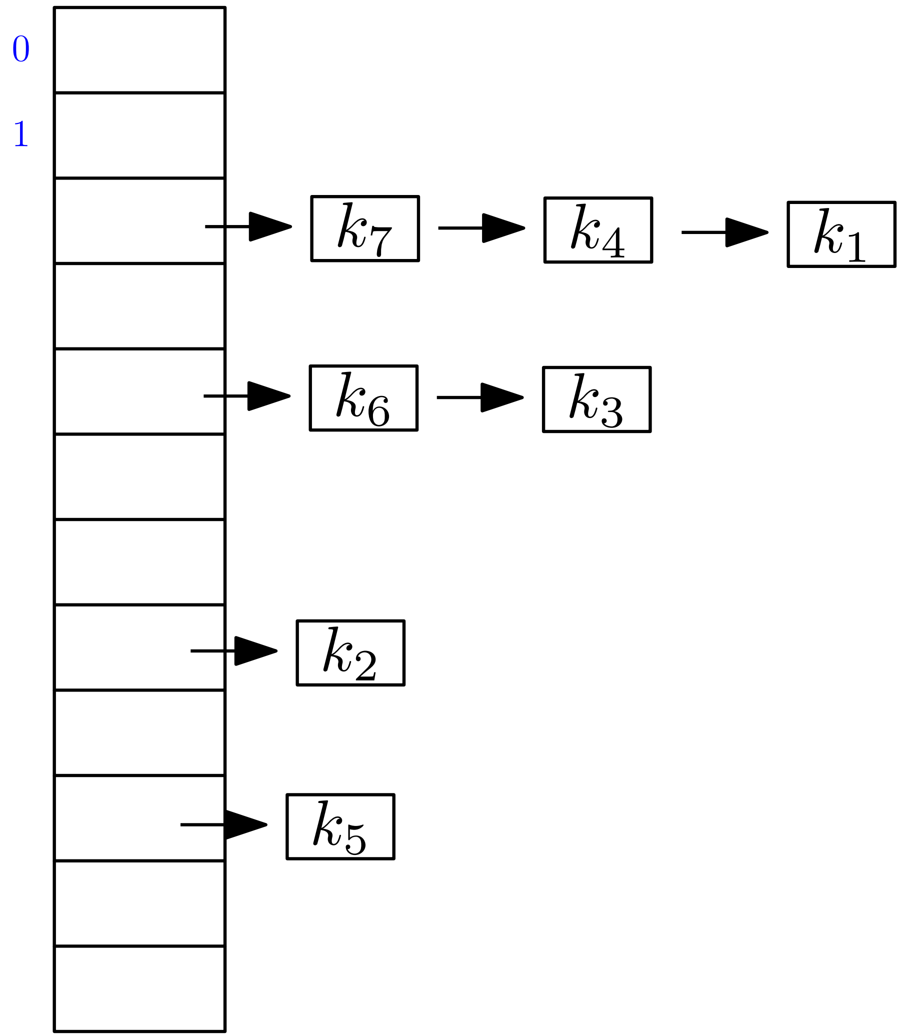
This is *average # items per list*.

Unsucessful search for x not in table will require searching entire list for $h(x)$.

Worst case length is $O(1)$.
Average case length is $O(\alpha)$.

Chaining: Unsuccessful Search

$h : U \to \{0, 1 \ldots, m-1\}$



Search(x): Search for x in list for $h(x)$ $O(\textit{length of list})$

Recall load factor $\alpha = \frac{n}{m}$.

This is *average # items per list*.

Unsucessful search for x not in table will require searching entire list for $h(x)$.

Worst case length is $O(1)$.
Average case length is $O(\alpha)$.

Average Unsucessful Search time is
$O(1+\alpha)$
where 1 is amount of time to calculate $h(x)$.

Chaining: Successful Search

For Successful Search for x:

Assume x equally likely to be any item in table

Chaining: Successful Search

For Successful Search for x:

Assume x equally likely to be any item in table

Search cost is $\#$ items ahead of x in list $h(x)$

$= \#$ of items inserted into $h(x)$ after x

Chaining: Successful Search

For Successful Search for x:

Assume x equally likely to be any item in table

Search cost is # items ahead of x in list $h(x)$

$= \#$ of items inserted into $h(x)$ after x

If x is i'th item inserted

$\Rightarrow n - i$ items inserted after x

$\Rightarrow \alpha - \frac{i}{m} = \frac{n-i}{m}$ items inserted on average into $h(x)$ after x

x is equally likely (with prob $1/n$) to be i'th inserted item.

Average # of items ahead of x in list $h(x)$ is

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

Chaining: Successful Search

For Successful Search for x:

Assume x equally likely to be any item in table

Search cost is $\#$ items ahead of x in list $h(x)$

$= \#$ of items inserted into $h(x)$ after x

If x is i'th item inserted

$\Rightarrow n - i$ items inserted after x

$\Rightarrow \alpha - \frac{i}{m} = \frac{n-i}{m}$ items inserted on average into $h(x)$ after x

x is equally likely (with prob $1/n$) to be i'th inserted item.

Average $\#$ of items ahead of x in list $h(x)$ is

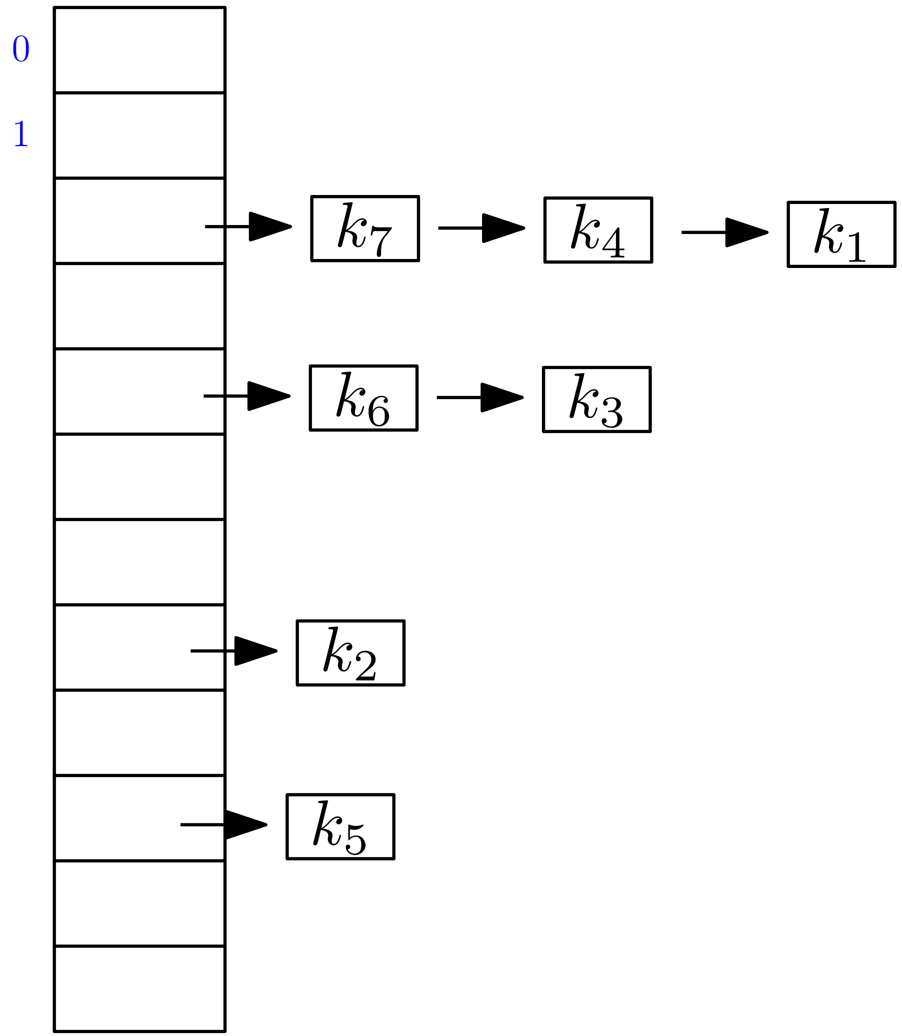
$$\frac{1}{n}\sum_{i=1}^{n}\left(\alpha - \frac{i}{m}\right) = \alpha - \frac{n(n+1)}{2nm} = \alpha - \frac{\alpha}{2} - \frac{\alpha}{2n}$$

Adding 1 unit of time to calculate $h(x)$

Average cost of successful search is $\Theta(1+\alpha)$.

Chaining

$h : U \to \{0, 1 \ldots, m-1\}$



Search(x): Search for x in list for $h(x)$ *O(length of list)*

Both Successful and Unsuccessful Search require $O(1+\alpha)$ time on average

where $\alpha = \frac{n}{m}$ is the load factor

Chaining – A Deeper Dive

For performance-tuning it helps to know not just the *average* linked list size but the *probability distribution* of how many items are on the list. Often viewed as *balls and bins problem.*

Chaining – A Deeper Dive

For performance-tuning it helps to know not just the *average* linked list size but the *probability distribution* of how many items are on the list. Often viewed as *balls and bins problem.*

- m : number of bins. These are the linked lists
- n : number of balls, These are items to be hashed
- $\alpha = n/m$.

Chaining – A Deeper Dive

For performance-tuning it helps to know not just the *average* linked list size but the *probability distribution* of how many items are on the list. Often viewed as *balls and bins problem.*

- m : number of bins. These are the linked lists
- n : number of balls, These are items to be hashed
- $\alpha = n/m$.

- Balls are thrown uniformly at random into the bins.
  A ball has probability $1/m$ of going into bin j.
- Each ball is thrown independently into a bin.

Chaining – A Deeper Dive

For performance-tuning it helps to know not just the *average* linked list size but the *probability distribution* of how many items are on the list. Often viewed as *balls and bins problem.*

- m : number of bins. These are the linked lists
- n : number of balls, These are items to be hashed
- $\alpha = n/m$.

- Balls are thrown uniformly at random into the bins.
  A ball has probability $1/m$ of going into bin j.
- Each ball is thrown independently into a bin.

Fix bin j. Let X_i be indicator random variable for event that ball i goes into bin j. $E(X_i) = 1/m$.

Let $T_j = \sum_{i=1}^{n} X_i$ be total number of balls going into bin j

Chaining – A Deeper Dive

For performance-tuning it helps to know not just the *average* linked list size but the *probability distribution* of how many items are on the list. Often viewed as *balls and bins problem.*

- m : number of bins. These are the linked lists
- n : number of balls, These are items to be hashed
- $\alpha = n/m$.

- Balls are thrown uniformly at random into the bins.
  A ball has probability $1/m$ of going into bin j.
- Each ball is thrown independently into a bin.

Fix bin j. Let X_i be indicator random variable for event that ball i goes into bin j. $E(X_i) = 1/m$.

Let $T_j = \sum_{i=1}^{n} X_i$ be total number of balls going into bin j

$$\Rightarrow E(T_j) = E\left(\sum_{i=1}^{n} X_i\right) = \sum_{i=1}^{n} E(X_i) = \frac{n}{m} = \alpha.$$

Chaining – A Deeper Dive

- m : number of bins. These are the linked lists
- n : number of balls. These are items to be hashed
- $\alpha = n/m$.

X_i is indicator random variable for event that ball i going into bin j.
$T_j = \sum_{i=1}^{n} X_i$ is total number of balls going into bin j. $E(T_j) = \alpha$.

Chaining – A Deeper Dive

- m : number of bins. These are the linked lists
- n : number of balls. These are items to be hashed
- $\alpha = n/m$.

X_i is indicator random variable for event that ball i going into bin j.
$T_j = \sum_{i=1}^{n} X_i$ is total number of balls going into bin j. $E(T_j) = \alpha$.

Another View:

Since each ball is placed independently,
T_j is a *Binomial* $B(n,p)$ *random variable with* $p = 1/m$.

Chaining – A Deeper Dive

- m : number of bins. These are the linked lists
- n : number of balls. These are items to be hashed
- $\alpha = n/m$.

X_i is indicator random variable for event that ball i going into bin j.
$T_j = \sum_{i=1}^{n} X_i$ is total number of balls going into bin j. $E(T_j) = \alpha$.

Another View:

Since each ball is placed independently,
T_j is a *Binomial $B(n,p)$ random variable with $p = 1/m$.*

$$Pr(T_j = k) = \binom{n}{k}\left(\frac{1}{m}\right)^k\left(1 - \frac{1}{m}\right)^{n-k}$$

This gives us the *distribution* of the number of items in the list.

Chaining – A Deeper Dive

- m : number of bins. These are the linked lists
- n : number of balls. These are items to be hashed
- $\alpha = n/m$.

X_i is indicator random variable for event that ball i going into bin j.
$T_j = \sum_{i=1}^{n} X_i$ is total number of balls going into bin j. $E(T_j) = \alpha$.

Another View:

Since each ball is placed independently,
T_j is a *Binomial* $B(n,p)$ *random variable with* $p = 1/m$.

$$Pr(T_j = k) = \binom{n}{k} \left(\frac{1}{m}\right)^k \left(1 - \frac{1}{m}\right)^{n-k}$$

This gives us the *distribution* of the number of items in the list.

A $B(n,p)$ random variable has average value np.

$$\Rightarrow E(T_j) = np = \frac{n}{m} = \alpha$$

is the average number of items in the list (same as before).

Chaining – A Deeper Dive

- m : number of bins. These are the linked lists
- n : number of balls, These are items to be hashed
- $\alpha = n/m$.

Another View:

T_j is a *Binomial* $B(n, p)$ *random variable with* $p = 1/m$.

Chaining – A Deeper Dive

- m : number of bins. These are the linked lists
- n : number of balls, These are items to be hashed
- $\alpha = n/m$.

Another View:

T_j is a *Binomial $B(n, p)$ random variable with $p = 1/m$.*

Known fact about Binomial Random Variable T_j:
If n grows to infinity and $np = \alpha$ stays constant, then T_j is approximated by a Poisson Random Variable with parameter α.

Chaining – A Deeper Dive

- m : number of bins. These are the linked lists
- n : number of balls, These are items to be hashed
- $\alpha = n/m$.

Another View:

T_j is a *Binomial $B(n,p)$ random variable with $p = 1/m$.*

Known fact about Binomial Random Variable T_j:
If n grows to infinity and $np = \alpha$ stays constant, then T_j is approximated by a Poisson Random Variable with parameter α.

More specifically.

$$\Pr(T_j = k) \to e^{-\alpha} \frac{\alpha^k}{k!}.$$

Chaining – A Deeper Dive

- m : number of bins. These are the linked lists
- n : number of balls, These are items to be hashed
- $\alpha = n/m$.

Another View:

T_j is a *Binomial $B(n,p)$ random variable with $p = 1/m$.*

Known fact about Binomial Random Variable T_j:
If n grows to infinity and $np = \alpha$ stays constant, then T_j is approximated by a Poisson Random Variable with parameter α.

More specifically.

$$\Pr(T_j = k) \to e^{-\alpha} \frac{\alpha^k}{k!}.$$

This gives a much deeper understanding of how many items are on each list. Helps with performance tuning.

Chaining – A Deeper Dive

- m : number of bins. These are the linked lists
- n : number of balls, These are items to be hashed
- $\alpha = n/m$.

Another View:

T_j is a *Binomial $B(n,p)$ random variable with $p = 1/m$.*

$$\Pr(T_j = k) \to e^{-\alpha}\frac{\alpha^k}{k!}.$$

This gives a much stronger understanding of how many items are on each list.

Chaining – A Deeper Dive

- m : number of bins. These are the linked lists
- n : number of balls, These are items to be hashed
- $\alpha = n/m$.

Another View:

T_j is a *Binomial $B(n,p)$ random variable with $p = 1/m$.*

$$\Pr(T_j = k) \to e^{-\alpha}\frac{\alpha^k}{k!}.$$

This gives a much stronger understanding of how many items are on each list.

For example, if $m = n/3$ then average number of items in each list is $\alpha = 3$ and

$$\Pr(T_j = 0) \sim e^{-3}, \quad \Pr(T_j = 1) \sim 3e^{-3}, \quad \Pr(T_j = 2) \sim \frac{3^2 e^{-3}}{2}, \ldots$$

Chaining – An Even Deeper Dive

- m : number of bins. These are the linked lists
- n : number of balls, These are items to be hashed
- $\alpha = n/m$.

X_i is indicator random variable for event that ball i going into bin j.
$T_j = \sum_{i=1}^{n} X_i$ is total number of balls going into bin j. $E(T_i) = \alpha$.

Chaining – An Even Deeper Dive

- m : number of bins. These are the linked lists
- n : number of balls, These are items to be hashed
- $\alpha = n/m$.

X_i is indicator random variable for event that ball i going into bin j.
$T_j = \sum_{i=1}^{n} X_i$ is total number of balls going into bin j. $E(T_i) = \alpha$.

Would like to know not just how many are on a *particular list* but the largest number on *any* list.

$$M = \max(T_1, T_2, \ldots, T_n).$$

Chaining – An Even Deeper Dive

- m : number of bins. These are the linked lists
- n : number of balls, These are items to be hashed
- $\alpha = n/m$.

X_i is indicator random variable for event that ball i going into bin j.
$T_j = \sum_{i=1}^{n} X_i$ is total number of balls going into bin j. $E(T_i) = \alpha$.

Would like to know not just how many are on a *particular list* but the largest number on *any* list.

$$M = \max(T_1, T_2, \ldots, T_n).$$

M is the worst case lookup performance after we hash the values.
Since hashing is a random procedure we really want

$$E(M) = E\left(\max(T_1, T_2, \ldots, T_n)\right).$$

Chaining – An Even Deeper Dive

- m : number of bins. These are the linked lists
- n : number of balls, These are items to be hashed
- $\alpha = n/m$.

X_i is indicator random variable for event that ball i going into bin j.
$T_j = \sum_{i=1}^{n} X_i$ is total number of balls going into bin j. $E(T_i) = \alpha$.

Would like to know not just how many are on a *particular list* but the largest number on *any* list.

$$M = \max(T_1, T_2, \ldots, T_n).$$

M is the worst case lookup performance after we hash the values.
Since hashing is a random procedure we really want

$$E(M) = E\left(\max(T_1, T_2, \ldots, T_n)\right).$$

The math is a bit complicated but, if $\alpha = 1$, using the Poisson approximation from the previous page we can show

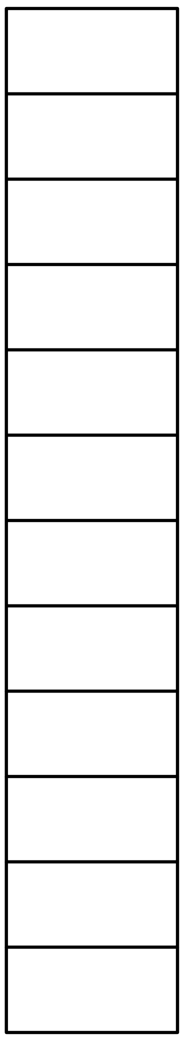
$$E(M) = \Theta\left(\frac{\log n}{\log \log n}\right).$$

Outline

- Introduction
- Hashing with Chaining
- Open Addressing
- Hash Functions & Universal Hashing
- Odds & Ends

Open Addressing

$h : U \to \{0, 1 \ldots, m-1\}$



- No lists. All keys stored in hash table itself.
- For insertion, *probe* hash table until empty slot for insertion is found.
- *Probe Sequence* is part of hash function.
- Hash function is now

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

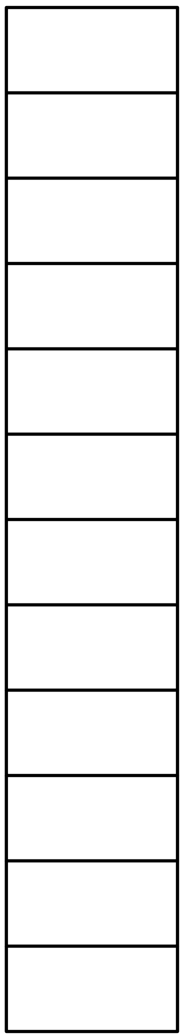
- Probe sequence for x is,

  $$h(x, 0),\ h(x, 1), \ldots,\ h(x, m-1)$$

  which is a permutation of $\{0, 1, \ldots, m\}$
- For search(x), *probe* hash table using probe sequence for $h(x)$ until either x or empty slot for insertion is found.

Open Addressing: Linear Probing

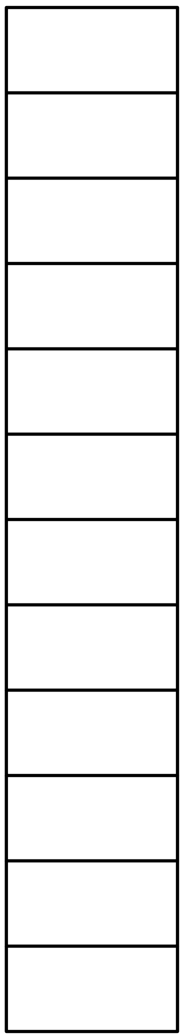
$h' : U \to \{0, 1 \ldots, m-1\}$



- Hash Function is $h(x, i) = \big(h'(x) + i\big) \bmod m$ where $h'(x)$ is original hash function.
- Insert: Attempts insertion at $h'(x)$, then $h'(x)+1$, $h'(x)+2$, etc., (wrapping around to 0 after reaching end of table) until empty slot is found and x inserted there.
- Search(x): Examines probe sequence until it finds x or an empty slot.
  If empty slot is found, then x wasn't previously inserted and the search is unsuccessful
- Deletion: More complicated.

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$



- Hash Function is $h(x,i) = \left(h'(x) + i\right) \bmod m$ where $h'(x)$ is original hash function.

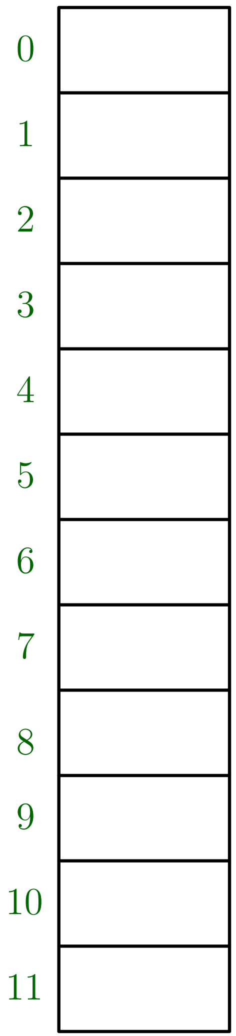
- Insert: Attempts insertion at $h'(x)$, then $h'(x)+1$, $h'(x)+2$, etc., (wrapping around to 0 after reaching end of table) until empty slot is found and x inserted there.

- Search(x): Examines probe sequence until it finds x or an empty slot.
  If empty slot is found, then x wasn't previously inserted and the search is unsuccessful

- Deletion: More complicated.
  Can't actually delete item and reset slot as 'empty"
  That would mess up Search(x).
  Can mark *slot as (used but) deleted.*
  Deletion in open addressing does cause difficulties.
  Better to use chaining.

Open Addressing: Linear Probing

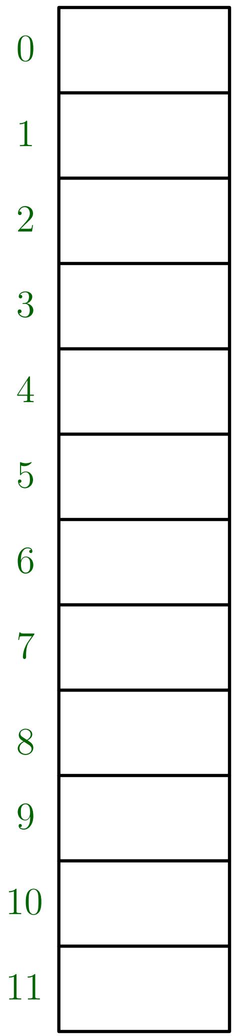
$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$



As example, let $h'(x) = x \bmod m$ with $m = 12$.

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

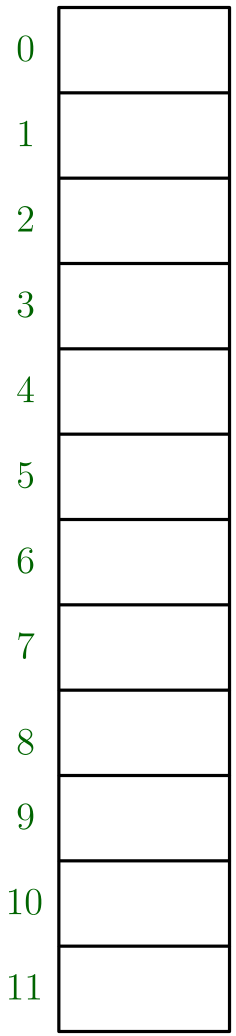


As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash function

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$



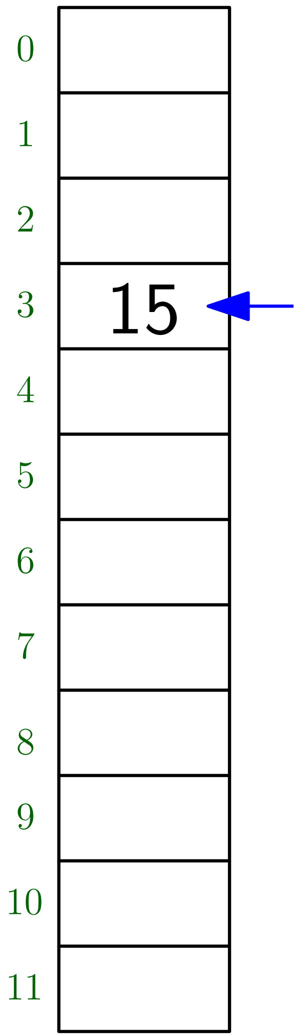
As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash function

Insert(15)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$



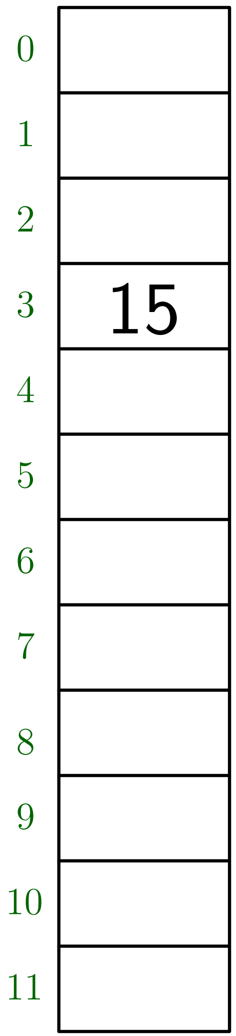
As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash function

Insert(15)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$



As example, let $h'(x) = x \bmod m$ with $m = 12$.

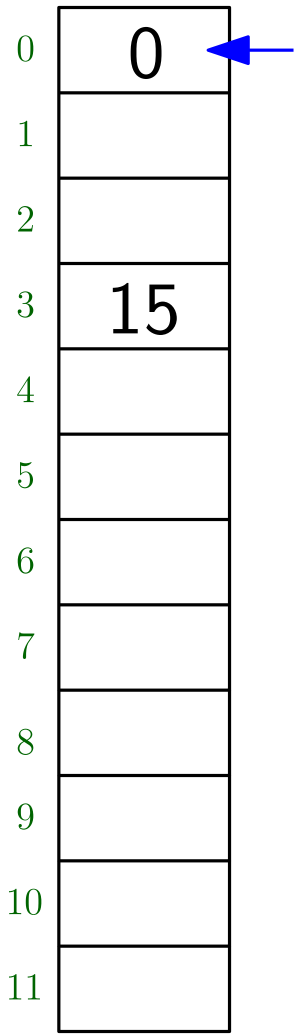
Only for illustration. This is a BAD hash function

Insert(15)

Insert(0)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\quad$ $h(x) = \big(h'(x) + 1\big) \bmod m$



As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash function

Insert(15)

Insert(0)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	
2	
3	15
4	
5	
6	
7	
8	
9	
10	
11	

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash function

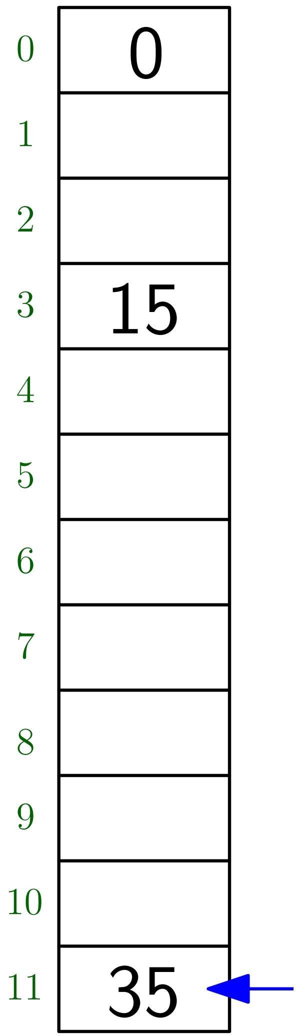
Insert(15)

Insert(0)

Insert(35)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$



As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash function

Insert(15)

Insert(0)

Insert(35)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	
2	
3	15
4	
5	
6	
7	
8	
9	
10	
11	35

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash function

Insert(15)

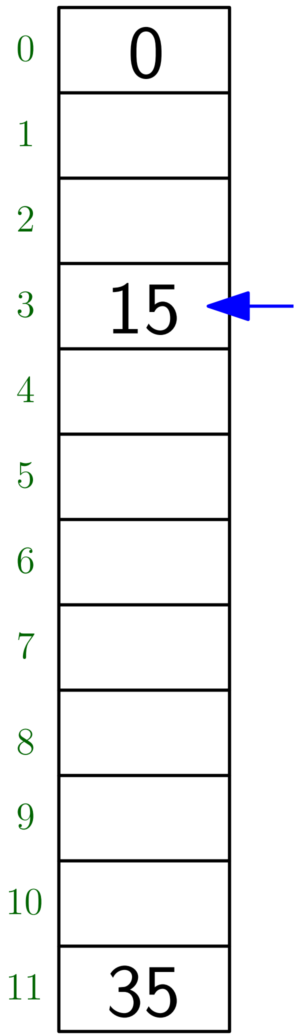
Insert(0)

Insert(35)

Insert(3)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$



As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash function

Insert(15)

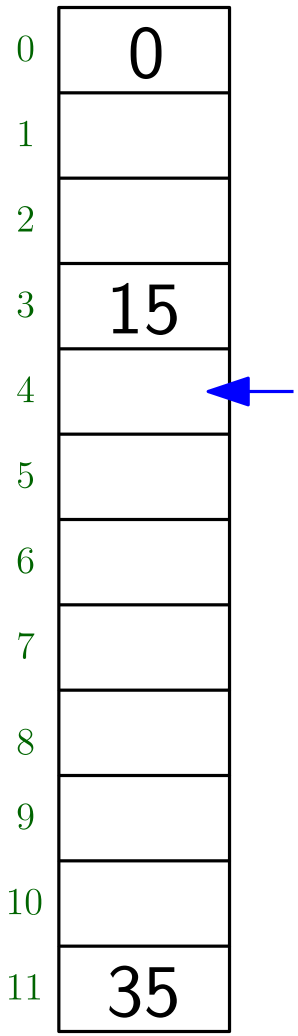
Insert(0)

Insert(35)

Insert(3)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$



As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash function

Insert(15)

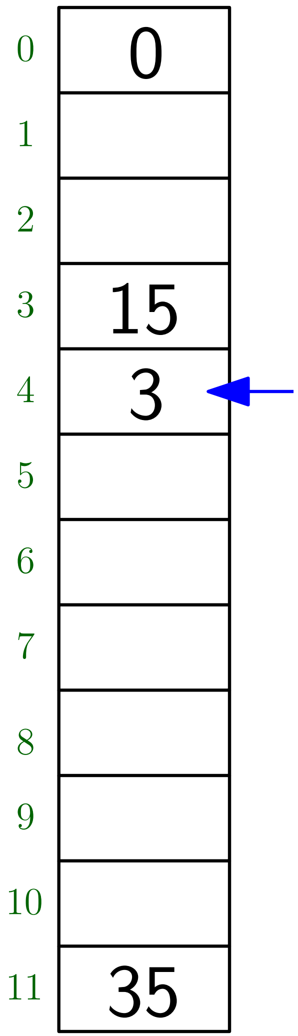
Insert(0)

Insert(35)

Insert(3)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$



As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash function

Insert(15)

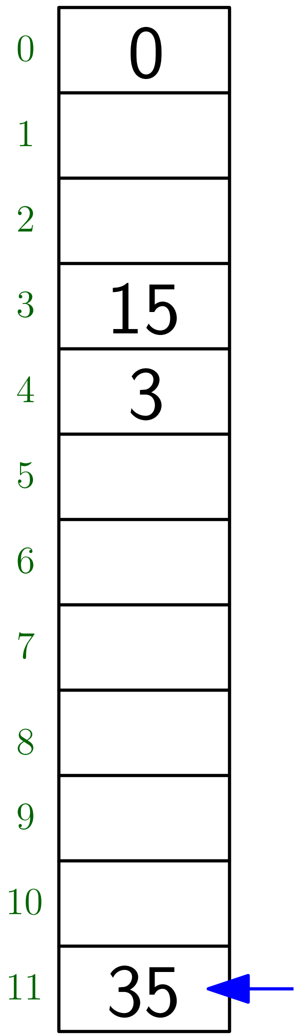
Insert(0)

Insert(35)

Insert(3)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$



As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash function

Insert(15)

Insert(0)

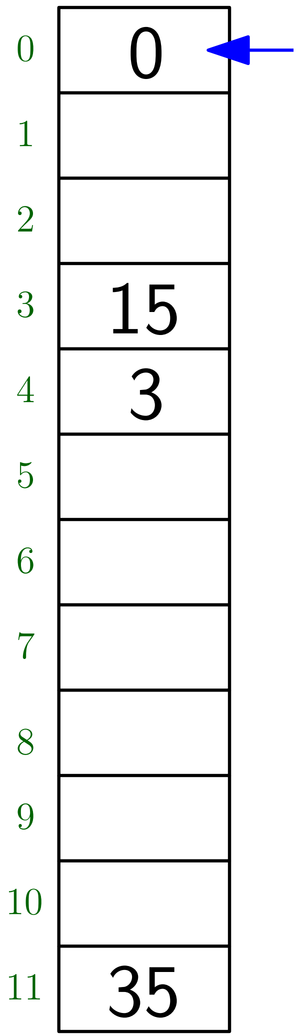
Insert(35)

Insert(3)

Insert(11)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$



As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash function

Insert(15)

Insert(0)

Insert(35)

Insert(3)

Insert(11)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	
7	
8	
9	
10	
11	35

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash function

Insert(15)

Insert(0)

Insert(35)

Insert(3)

Insert(11)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash function

Insert(15)

Insert(0)

Insert(35)

Insert(3)

Insert(11)

Insert(18)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash function

Insert(15)

Insert(0)

Insert(35)

Insert(3)

Insert(11)

Insert(18)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash functionn

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash functionn

Search(11)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash functionn

Search(11)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash functionn

Search(11)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash functionn

Search(11) $\qquad$ Exists

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash functionn

Search(11) Exists

Search(3)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash functionn

Search(11) $\qquad$ Exists

Search(3)

Open Addressing: Linear Probing

$h' : U \rightarrow \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash functionn

Search(11) Exists

Search(3) Exists

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash functionn

Search(11) Exists

Search(3) Exists

Search(9)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\quad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash functionn

Search(11)	Exists
Search(3)	Exists
Search(9)	Does not exist

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash functionn

Search(11) Exists

Search(3) Exists

Search(9) Does not exist

Search(24)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash functionn

Search(11) Exists

Search(3) Exists

Search(9) Does not exist

Search(24)

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

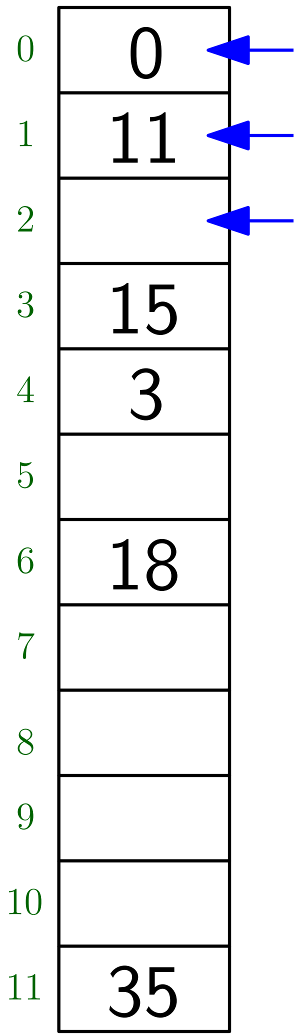
As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash functionn

Search(11)	Exists
Search(3)	Exists
Search(9)	Does not exist
Search(24)	

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$



As example, let $h'(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash functionn

Search(11)	Exists
Search(3)	Exists
Search(9)	Does not exist
Search(24)	Does not exist

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\quad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash functionn

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h(x) = x \bmod m$ with $m = 12$.

Only for illustration. This is a BAD hash functionn

Easy to code but suffers from primary clustering.
Long runs build up, increasing average search time

Open Addressing: Linear Probing

$h' : U \to \{0, 1 \ldots, m-1\}$ $\qquad$ $h(x) = \big(h'(x) + 1\big) \bmod m$

0	0
1	11
2	
3	15
4	3
5	
6	18
7	
8	
9	
10	
11	35

As example, let $h(x) = x \bmod m$ with $m = 12$.

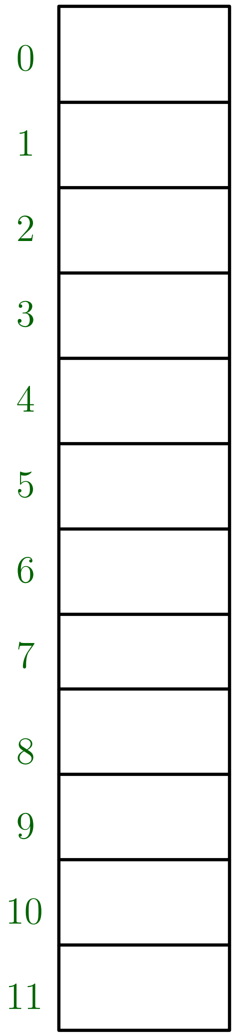
Only for illustration. This is a BAD hash functionn

Easy to code but suffers from primary clustering.
Long runs build up, increasing average search time

One fix is to change probe sequence to be *nonlinear*.

Open Addressing: Quadratic Probing

$h' : U \to \{0, 1 \ldots, m-1\}$

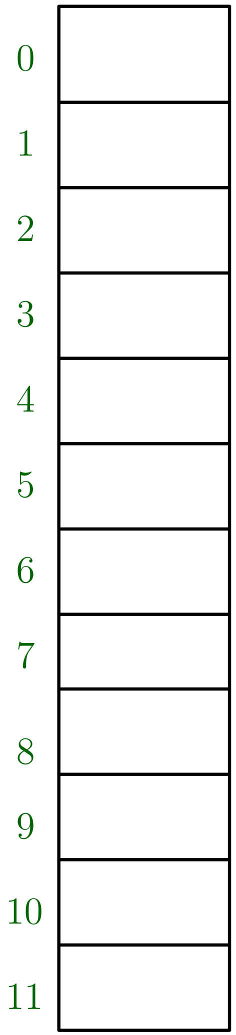


- Hash Function is $h(x,i) = \left(h'(x) + c_1 + c_2 i^2\right) \bmod m$ where $h'(x)$ is original hash function and c_1, c_2 fixed constants.

- As example we will set $h'(x) = x \bmod 12$, $c_1 = 0$ and $c_2 = 1$ so $h(x,i) = \left(x + i^2\right) \bmod 12$

Open Addressing: Quadratic Probing

$h' : U \to \{0, 1 \ldots, m-1\}$

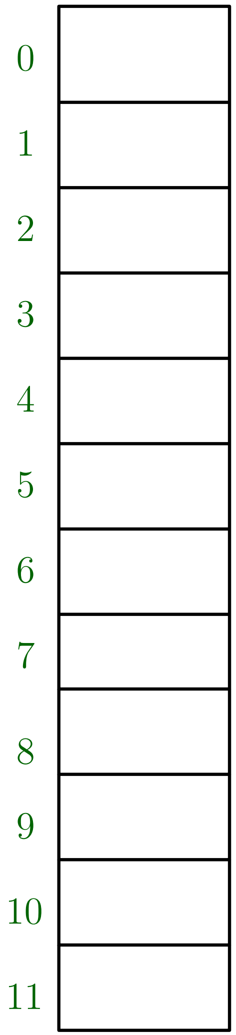


- Hash Function is $h(x, i) = \left(h'(x) + c_1 + c_2 i^2\right) \bmod m$ where $h'(x)$ is original hash function and c_1, c_2 fixed constants.

- As example we will set $h'(x) = x \bmod 12$, $c_1 = 0$ and $c_2 = 1$ so $h(x, i) = \left(x + i^2\right) \bmod 12$

Open Addressing: Quadratic Probing

$h' : U \to \{0, 1 \ldots, m-1\}$



- Hash Function is $h(x,i) = \left(h'(x) + c_1 + c_2 i^2\right) \bmod m$ where $h'(x)$ is original hash function and c_1, c_2 fixed constants.

- As example we will set $h'(x) = x \bmod 12$, $c_1 = 0$ and $c_2 = 1$ so $h(x,i) = \left(x + i^2\right) \bmod 12$

Insert(15)

Insert(0)

Insert(35)

Insert(3)

Insert(11)

Insert(18)

Open Addressing: Quadratic Probing

$h' : U \to \{0, 1 \ldots, m-1\}$

0	
1	
2	
3	15
4	
5	
6	
7	
8	
9	
10	
11	

- Hash Function is $h(x,i) = \left(h'(x) + c_1 + c_2 i^2\right) \bmod m$ where $h'(x)$ is original hash function and c_1, c_2 fixed constants.
- As example we will set $h'(x) = x \bmod 12$, $c_1 = 0$ and $c_2 = 1$ so $h(x,i) = \left(x + i^2\right) \bmod 12$

Insert(15)

Insert(0)

Insert(35)

Insert(3)

Insert(11)

Insert(18)

Open Addressing: Quadratic Probing

$h' : U \to \{0, 1 \ldots, m-1\}$

0	0
1	
2	
3	15
4	
5	
6	
7	
8	
9	
10	
11	

- Hash Function is $h(x, i) = \left(h'(x) + c_1 + c_2 i^2\right) \bmod m$ where $h'(x)$ is original hash function and c_1, c_2 fixed constants.
- As example we will set $h'(x) = x \bmod 12$, $c_1 = 0$ and $c_2 = 1$ so $h(x, i) = \left(x + i^2\right) \bmod 12$

Insert(15)

Insert(0)

Insert(35)

Insert(3)

Insert(11)

Insert(18)

Open Addressing: Quadratic Probing

$h' : U \to \{0, 1 \ldots, m-1\}$

0	0
1	
2	
3	15
4	
5	
6	
7	
8	
9	
10	
11	35

- Hash Function is $h(x,i) = \left(h'(x) + c_1 + c_2 i^2\right) \bmod m$ where $h'(x)$ is original hash function and c_1, c_2 fixed constants.
- As example we will set $h'(x) = x \bmod 12$, $c_1 = 0$ and $c_2 = 1$ so $h(x,i) = \left(x + i^2\right) \bmod 12$

Insert(15)

Insert(0)

Insert(35)

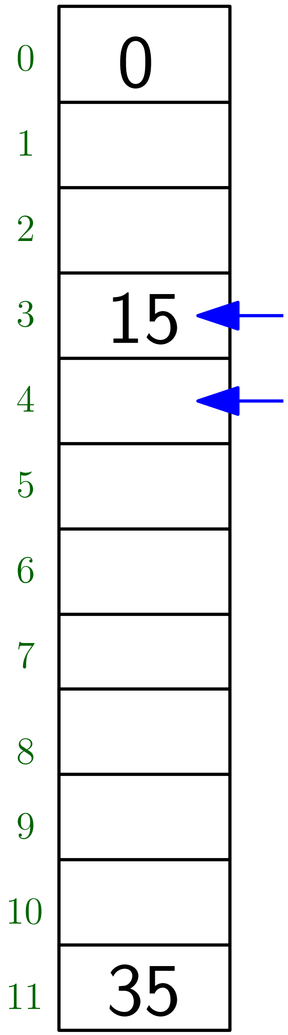
Insert(3)

Insert(11)

Insert(18)

Open Addressing: Quadratic Probing

$h' : U \to \{0, 1 \ldots, m-1\}$



- Hash Function is $h(x,i) = \left(h'(x) + c_1 + c_2 i^2\right) \bmod m$ where $h'(x)$ is original hash function and c_1, c_2 fixed constants.
- As example we will set
  $h'(x) = x \bmod 12$, $c_1 = 0$ and $c_2 = 1$ so
  $h(x,i) = \left(x + i^2\right) \bmod 12$

Insert(15)

Insert(0)

Insert(35)

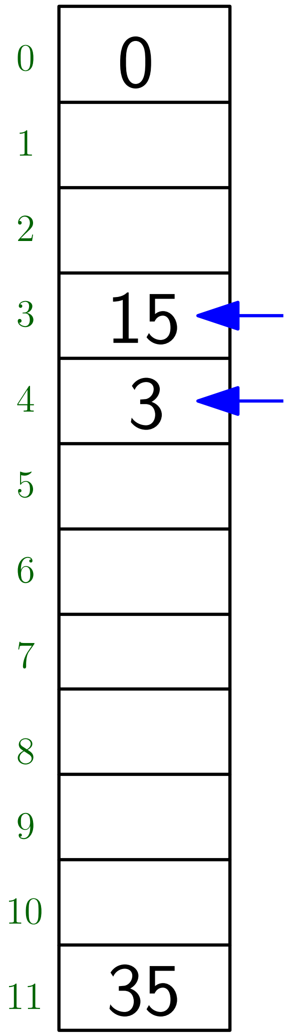
Insert(3)

Insert(11)

Insert(18)

Open Addressing: Quadratic Probing

$h' : U \to \{0, 1 \ldots, m-1\}$



- Hash Function is $h(x,i) = \left(h'(x) + c_1 + c_2 i^2\right) \bmod m$ where $h'(x)$ is original hash function and c_1, c_2 fixed constants.

- As example we will set $h'(x) = x \bmod 12$, $c_1 = 0$ and $c_2 = 1$ so $h(x,i) = \left(x + i^2\right) \bmod 12$

Insert(15)

Insert(0)

Insert(35)

Insert(3)

Insert(11)

Insert(18)

Open Addressing: Quadratic Probing

$h' : U \to \{0, 1 \ldots, m-1\}$

0	0
1	
2	
3	15
4	3
5	
6	
7	
8	
9	
10	
11	35

- Hash Function is $h(x, i) = \left(h'(x) + c_1 + c_2 i^2\right) \bmod m$ where $h'(x)$ is original hash function and c_1, c_2 fixed constants.

- As example we will set $h'(x) = x \bmod 12$, $c_1 = 0$ and $c_2 = 1$ so $h(x, i) = \left(x + i^2\right) \bmod 12$

Insert(15)

Insert(0)

Insert(35)

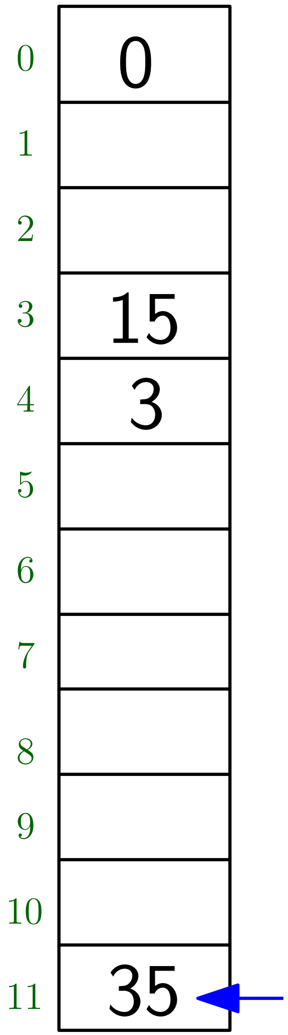
Insert(3)

Insert(11)

Insert(18)

Open Addressing: Quadratic Probing

$h' : U \to \{0, 1 \ldots, m-1\}$



- Hash Function is $h(x,i) = \left(h'(x) + c_1 + c_2 i^2\right) \bmod m$ where $h'(x)$ is original hash function and c_1, c_2 fixed constants.
- As example we will set $h'(x) = x \bmod 12$, $c_1 = 0$ and $c_2 = 1$ so $h(x,i) = \left(x + i^2\right) \bmod 12$

Insert(15)

Insert(0)

Insert(35)

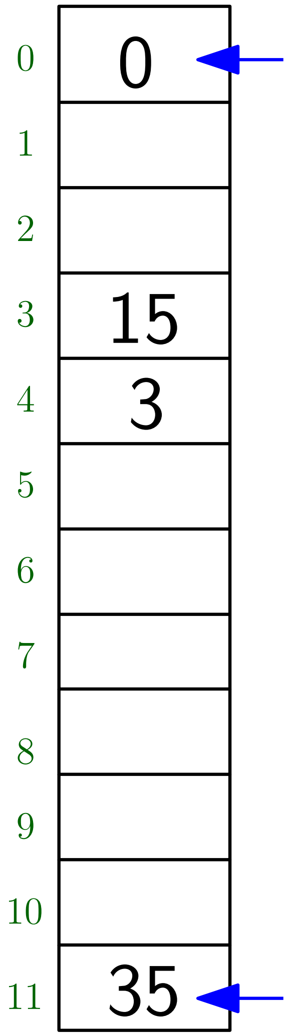
Insert(3)

Insert(11)

Insert(18)

Open Addressing: Quadratic Probing

$h' : U \to \{0, 1 \ldots, m-1\}$



- Hash Function is $h(x,i) = \left(h'(x) + c_1 + c_2 i^2\right) \bmod m$ where $h'(x)$ is original hash function and c_1, c_2 fixed constants.
- As example we will set $h'(x) = x \bmod 12$, $c_1 = 0$ and $c_2 = 1$ so $h(x,i) = \left(x + i^2\right) \bmod 12$

Insert(15)

Insert(0)

Insert(35)

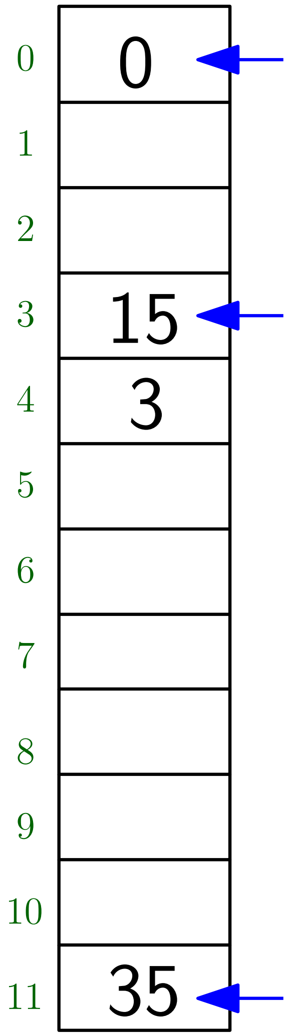
Insert(3)

Insert(11)

Insert(18)

Open Addressing: Quadratic Probing

$h' : U \to \{0, 1 \ldots, m-1\}$



- Hash Function is $h(x,i) = \left(h'(x) + c_1 + c_2 i^2\right) \bmod m$ where $h'(x)$ is original hash function and c_1, c_2 fixed constants.
- As example we will set $h'(x) = x \bmod 12$, $c_1 = 0$ and $c_2 = 1$ so $h(x,i) = \left(x + i^2\right) \bmod 12$

Insert(15)

Insert(0)

Insert(35)

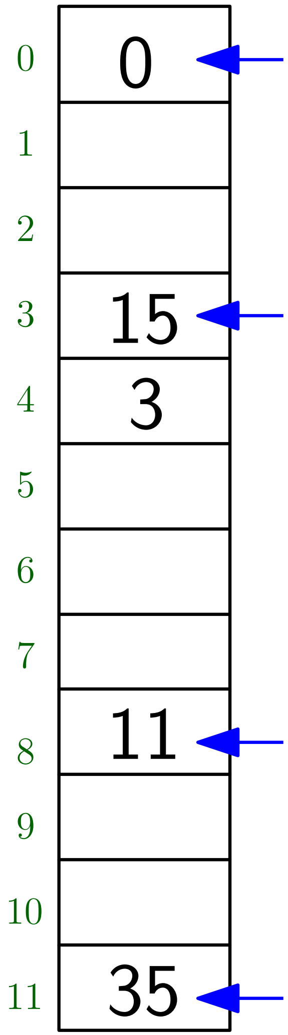
Insert(3)

Insert(11)

Insert(18)

Open Addressing: Quadratic Probing

$h' : U \to \{0, 1 \ldots, m-1\}$



- Hash Function is $h(x,i) = \left(h'(x) + c_1 + c_2 i^2\right) \bmod m$ where $h'(x)$ is original hash function and c_1, c_2 fixed constants.
- As example we will set $h'(x) = x \bmod 12$, $c_1 = 0$ and $c_2 = 1$ so $h(x,i) = \left(x + i^2\right) \bmod 12$

Insert(15)

Insert(0)

Insert(35)

Insert(3)

Insert(11)

Insert(18)

Open Addressing: Quadratic Probing

$h' : U \to \{0, 1 \ldots, m-1\}$

0	0
1	
2	
3	15
4	3
5	
6	
7	
8	11
9	
10	
11	35

- Hash Function is $h(x,i) = \left(h'(x) + c_1 + c_2 i^2\right) \bmod m$ where $h'(x)$ is original hash function and c_1, c_2 fixed constants.
- As example we will set $h'(x) = x \bmod 12$, $c_1 = 0$ and $c_2 = 1$ so $h(x,i) = \left(x + i^2\right) \bmod 12$

Insert(15)

Insert(0)

Insert(35)

Insert(3)

Insert(11)

Insert(18)

Open Addressing: Quadratic Probing

$h' : U \to \{0, 1 \ldots, m-1\}$

0	0
1	
2	
3	15
4	3
5	
6	18
7	
8	11
9	
10	
11	35

- Hash Function is $h(x,i) = \left(h'(x) + c_1 + c_2 i^2\right) \bmod m$ where $h'(x)$ is original hash function and c_1, c_2 fixed constants.
- As example we will set $h'(x) = x \bmod 12$, $c_1 = 0$ and $c_2 = 1$ so $h(x,i) = \left(x + i^2\right) \bmod 12$

Insert(15)

Insert(0)

Insert(35)

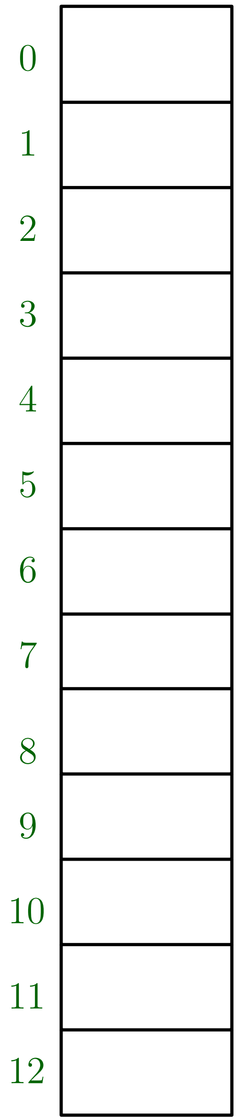
Insert(3)

Insert(11)

Insert(18)

Open Addressing: Double Hashing

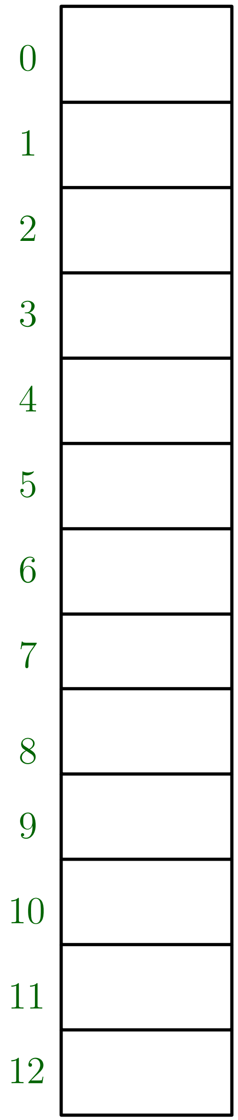
$h' : U \to \{0, 1 \ldots, m-1\}$



- Hash Function is $h(x, i) = \big(h_1(x) + ih_2(x)\big) \bmod m$
- $h_1(x)$ and $h_2(x)$ are auxillary hash functions
- Note that (unlike before) probe sequence depends upon x
- In order for probe sequence to check entire table, must have $h_2(x) \neq 0$ and be relatively prime to m, e.g.,
  - m a power of 2; $h_2(x)$ always odd
  - m prime; $h_2(x)$ always less than m.

Open Addressing: Double Hashing

$h' : U \to \{0, 1 \ldots, m-1\}$



- Hash Function is $h(x,i) = \big(h_1(x) + ih_2(x)\big) \bmod m$
- $h_1(x)$ and $h_2(x)$ are auxillary hash functions
- Note that (unlike before) probe sequence depends upon x
- In order for probe sequence to check entire table, must have $h_2(x) \neq 0$ and be relatively prime to m, e.g.,
  - m a power of 2; $h_2(x)$ always odd
  - m prime; $h_2(x)$ always less than m.

Example: $m = 13$
$h_1(x) = x \bmod m$
$h_2(x) = 1 + (x \bmod 11)$

Open Addressing: Double Hashing

$h' : U \to \{0, 1 \ldots, m-1\}$

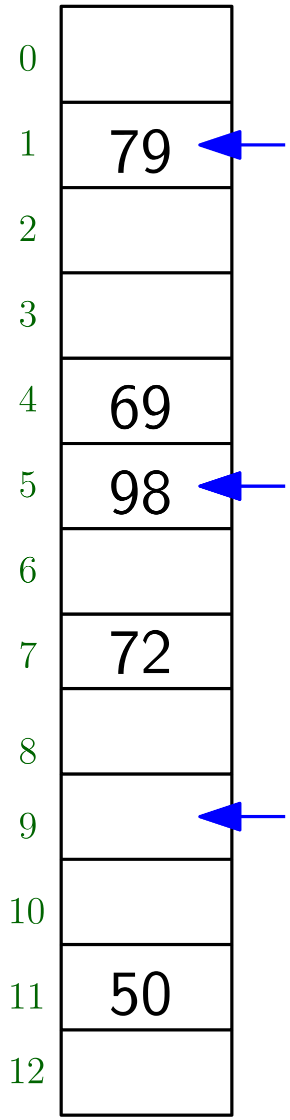
0	
1	79
2	
3	
4	69
5	98
6	
7	72
8	
9	
10	
11	50
12	

- Hash Function is $h(x,i) = \big(h_1(x) + ih_2(x)\big) \bmod m$
- $h_1(x)$ and $h_2(x)$ are auxillary hash functions
- Note that (unlike before) probe sequence depends upon x
- In order for probe sequence to check entire table, must have $h_2(x) \neq 0$ and be relatively prime to m, e.g.,
  - m a power of 2; $h_2(x)$ always odd
  - m prime; $h_2(x)$ always less than m.

Example: $m = 13$
$h_1(x) = x \bmod m$
$h_2(x) = 1 + (x \bmod 11)$

Open Addressing: Double Hashing

$h' : U \to \{0, 1 \ldots, m-1\}$



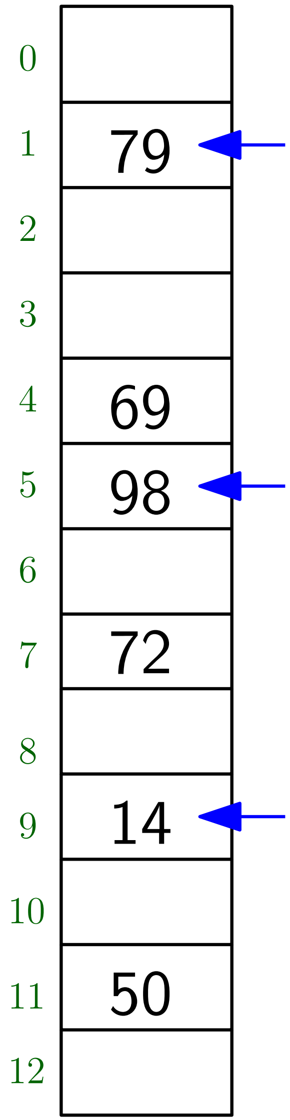
- Hash Function is $h(x,i) = \big(h_1(x) + ih_2(x)\big) \bmod m$
- $h_1(x)$ and $h_2(x)$ are auxillary hash functions
- Note that (unlike before) probe sequence depends upon x
- In order for probe sequence to check entire table, must have $h_2(x) \neq 0$ and be relatively prime to m, e.g.,
  - m a power of 2; $h_2(x)$ always odd
  - m prime; $h_2(x)$ always less than m.

Example: $m = 13$
$h_1(x) = x \bmod m$
$h_2(x) = 1 + (x \bmod 11)$

14 would have probe sequence $1, 5, 9, \ldots$
Since first 2 locations full, it will be inserted into 9.

Open Addressing: Double Hashing

$h' : U \to \{0, 1 \ldots, m-1\}$



- Hash Function is $h(x,i) = \big(h_1(x) + ih_2(x)\big) \bmod m$
- $h_1(x)$ and $h_2(x)$ are auxillary hash functions
- Note that (unlike before) probe sequence depends upon x
- In order for probe sequence to check entire table, must have $h_2(x) \neq 0$ and be relatively prime to m, e.g.,
  - m a power of 2; $h_2(x)$ always odd
  - m prime; $h_2(x)$ always less than m.

Example: $m = 13$
$h_1(x) = x \bmod m$
$h_2(x) = 1 + (x \bmod 11)$

14 would have probe sequence $1, 5, 9, \ldots$
Since first 2 locations full, it will be inserted into 9.

Open Addressing: Analysis Results

We have seen 3 different open addressing collision resolution methods:

- Linear Probing: $h(x, i) = (h'(x) + 1) \bmod m$
- Quadratic Probing $h(x, i) = (h'(x) + c_1 + c_2 x^2) \bmod m$
- Double Hashing $h(x, i) = (h_1(x) + i h_2(x)) \bmod m$

Open Addressing: Analysis Results

We have seen 3 different open addressing collision resolution methods:

- Linear Probing: $h(x,i) = (h'(x) + 1) \bmod m$
- Quadratic Probing $h(x,i) = (h'(x) + c_1 + c_2x^2) \bmod m$
- Double Hashing $h(x,i) = (h_1(x) + ih_2(x)) \bmod m$

For analysis, we often assume uniform hashing.
This states that the probe sequence

$$h(x,0),\ h(x,1), h(x,2),\ \ldots,\ h(x,m)$$

is equally likely to be any of the $m!$ permutations of $1, 2, \ldots, m$.

Open Addressing: Analysis Results

We have seen 3 different open addressing collision resolution methods:

- Linear Probing: $h(x,i) = (h'(x) + 1) \bmod m$
- Quadratic Probing $h(x,i) = (h'(x) + c_1 + c_2x^2) \bmod m$
- Double Hashing $h(x,i) = (h_1(x) + ih_2(x)) \bmod m$

For analysis, we often assume uniform hashing.
This states that the probe sequence

$$h(x,0),\ h(x,1), h(x,2),\ \ldots,\ h(x,m)$$

is equally likely to be any of the $m!$ permutations of $1, 2, \ldots, m$.

Uniform Hashing is not actually realizable.
The more random our probe sequence, though, the closer actual behavior is to theory.

Open Addressing: Analysis Results

Recall $\alpha = \frac{n}{m}$ is the *load factor*.
In what follows we assume uniform hashing.

Open Addressing: Analysis Results

Recall $\alpha = \frac{n}{m}$ is the *load factor*.
In what follows we assume uniform hashing.

Lemma: Given an open-address hash table with load factor $\alpha = n/m < 1$, the average number of probes in an unsuccessful search is at most

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

Open Addressing: Analysis Results

Recall $\alpha = \frac{n}{m}$ is the *load factor*.
In what follows we assume uniform hashing.

Lemma: Given an open-address hash table with load factor $\alpha = n/m < 1$, the average number of probes in an unsuccessful search is at most

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

Lemma: Inserting an element into an open-address hash table with load factor $\alpha = n/m < 1$, requires, on average, at most $1/(1-\alpha)$ probes.

Open Addressing: Analysis Results

Recall $\alpha = \frac{n}{m}$ is the *load factor*.
In what follows we assume uniform hashing.

Lemma: Given an open-address hash table with load factor $\alpha = n/m < 1$, the average number of probes in an unsuccessful search is at most

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

Lemma: Inserting an element into an open-address hash table with load factor $\alpha = n/m < 1$, requires, on average, at most $1/(1-\alpha)$ probes.

Lemma: Given an open-address hash table with load factor $\alpha = n/m < 1$, the average number of probes in an successful search is at most

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

Outline

- Introduction
- Hashing with Chaining
- Open Addressing
- Hash Functions & Universal Hashing
- Odds & Ends

Hash Functions & Universal Hashing

Returning to chained hashing, notice that our analysis assumed that the hashed keys were equally distributed among the slots.

- If all keys hashed to same slot, performance would be very bad.
- If the hash function $h(x)$ is given in advance and $n << U$, very easy to construct bad case in which all keys map to the same slot.

- We sidestep this issue by choosing a *random hash function*
- More specifically, we will have a *collection* of hash functions $\mathcal{H}$

Hash Functions & Universal Hashing

Returning to chained hashing, notice that our analysis assumed that the hashed keys were equally distributed among the slots.

- If all keys hashed to same slot, performance would be very bad.
- If the hash function $h(x)$ is given in advance and $n << U$, very easy to construct bad case in which all keys map to the same slot.

- We sidestep this issue by choosing a *random hash function*
- More specifically, we will have a *collection* of hash functions $\mathcal{H}$

- Given any set of keys, we will choose a random hash function $h \in \mathcal{H}$ and then hash using $h(x)$.

- *On average*, the n set of keys will be hashed so that each slot will get an average $O(n/m) = O(\alpha)$ keys.

- Our $O(1 + \alpha)$ sucessful/unsuccessful search times for chained hashing will then hold on average.

Hash Functions & Universal Hashing

Returning to chained hashing, notice that our analysis assumed that the hashed keys were equally distributed among the slots.

- If all keys hashed to same slot, performance would be very bad.
- If the hash function $h(x)$ is given in advance and $n << U$, very easy to construct bad case in which all keys map to the same slot.
- We sidestep this issue by choosing a *random hash function*
- More specifically, we will have a *collection* of hash functions $\mathcal{H}$
- Given any set of keys, we will choose a random hash function $h \in \mathcal{H}$ and then hash using $h(x)$.
- *On average*, the n set of keys will be hashed so that each slot will get an average $O(n/m) = O(\alpha)$ keys.
- Our $O(1+\alpha)$ sucessful/unsuccessful search times for chained hashing will then hold on average.

One class $\mathcal{H}$ of hash functions having this property are the *Universal* ones; they permit *Universal Hashing*

Universal Hashing

- Let $\mathcal{H}$ be a set of hash functions, such that each $h \in \mathcal{H}$ maps
  $$h : U \to \{0, 1, \ldots, m-1\}$$

Universal Hashing

- Let $\mathcal{H}$ be a set of hash functions, such that each $h \in \mathcal{H}$ maps
  $h : U \to \{0, 1, \ldots, m-1\}$

$\mathcal{H}$ is Universal if, for every two different keys k, ℓ, the number of hash functions in $\mathcal{H}$ that map k, ℓ to the same slot is at most $|\mathcal{H}|/m$, i.e.,

$$\forall k \neq \ell \in U, \quad |\{h \in \mathcal{H} \,:\, h(k) = h(\ell)\}| \leq \frac{|\mathcal{H}|}{m}.$$

Universal Hashing

- Let $\mathcal{H}$ be a set of hash functions, such that each $h \in \mathcal{H}$ maps
  $h : U \to \{0, 1, \ldots, m-1\}$

$\mathcal{H}$ is Universal if, for every two different keys k, ℓ, the number of hash functions in $\mathcal{H}$ that map k, ℓ to the same slot is at most $|\mathcal{H}|/m$, i.e.,

$$\forall k \neq \ell \in U, \quad |\{h \in \mathcal{H} : h(k) = h(\ell)\}| \leq \frac{|\mathcal{H}|}{m}.$$

Let $k_1, k_2, \ldots, k_n$ be the n keys.

Let i be any fixed index.
Then, for $j \neq i$, if $h \in \mathcal{H}$ is chosen uniformly at random,

$$\Pr(h(k_i) = h(k_j)) \leq \frac{1}{|\mathcal{H}|} \frac{|\mathcal{H}|}{m} = \frac{1}{m}.$$

Universal Hashing

- Let $\mathcal{H}$ be a set of hash functions, such that each $h \in \mathcal{H}$ maps
  $h : U \to \{0, 1, \ldots, m-1\}$

$\mathcal{H}$ is Universal if, for every two different keys k, ℓ, the number of hash functions in $\mathcal{H}$ that map k, ℓ to the same slot is at most $|\mathcal{H}|/m$, i.e.,

$$\forall k \neq \ell \in U, \quad |\{h \in \mathcal{H} \,:\, h(k) = h(\ell)\}| \leq \frac{|\mathcal{H}|}{m}.$$

Let $k_1, k_2, \ldots, k_n$ be the n keys.

Let i be any fixed index.
Then, for $j \neq i$, if $h \in \mathcal{H}$ is chosen uniformly at random,

$$\Pr(h(k_i) = h(k_j)) \leq \frac{1}{|\mathcal{H}|} \frac{|\mathcal{H}|}{m} = \frac{1}{m}.$$

From linearity of expectation, if $h \in \mathcal{H}$ is chosen uniformly at random, average # of other keys mapping to the same slot as k_i is then

$$\sum_{j \neq i; 1 \leq j \leq n} \Pr(h(k_i) = h(k_j)) \leq \frac{n-1}{m} < \alpha$$

Universal Hashing

- Let $\mathcal{H}$ be a set of hash functions, such that each $h \in \mathcal{H}$ maps
  $h : U \to \{0, 1, \ldots, m-1\}$

$\mathcal{H}$ is Universal if, for every two different keys k, ℓ, the number of hash functions in $\mathcal{H}$ that map k, ℓ to the same slot is at most $|\mathcal{H}|/m$, i.e.,

$$\forall k \neq \ell \in U, \quad |\{h \in \mathcal{H} : h(k) = h(\ell)\}| \leq \frac{|\mathcal{H}|}{m}.$$

Let $k_1, k_2, \ldots, k_n$ be the n keys.

Universal Hashing

- Let $\mathcal{H}$ be a set of hash functions, such that each $h \in \mathcal{H}$ maps

  $h : U \to \{0, 1, \ldots, m-1\}$

$\mathcal{H}$ is Universal if, for every two different keys k, ℓ, the number of hash functions in $\mathcal{H}$ that map k, ℓ to the same slot is at most $|\mathcal{H}|/m$, i.e.,

$$\forall k \neq \ell \in U, \quad |\{h \in \mathcal{H} \,:\, h(k) = h(\ell)\}| \leq \frac{|\mathcal{H}|}{m}.$$

Let $k_1, k_2, \ldots, k_n$ be the n keys.

Similarly, if k is not one of the n keys then, for all j,

$\Pr(h(k) = h(k_j)) \leq \frac{1}{|\mathcal{H}|} \frac{|\mathcal{H}|}{m} = \frac{1}{m}$.

Again from linearity of expectation, if $h \in \mathcal{H}$ is chosen uniformly at random, average $\#$ of keys mapping to same slot as k is then

$$\sum_{j=1}^{n} \Pr(h(k) = h(k_j)) \leq n/m = \alpha.$$

Universal Hashing

- Let $\mathcal{H}$ be a set of hash functions, such that each $h \in \mathcal{H}$ maps
  $h : U \to \{0, 1, \ldots, m-1\}$

$\mathcal{H}$ is Universal if, for every two different keys k, ℓ, the number of hash functions in $\mathcal{H}$ that map k, ℓ to the same slot is at most $|\mathcal{H}|/m$, i.e.,

$$\forall k \neq \ell \in U, \quad |\{h \in \mathcal{H} : h(k) = h(\ell)\}| \leq \frac{|\mathcal{H}|}{m}.$$

Let $k_1, k_2, \ldots, k_n$ be the n keys.

Universal Hashing

- Let $\mathcal{H}$ be a set of hash functions, such that each $h \in \mathcal{H}$ maps
  $h : U \to \{0, 1, \ldots, m-1\}$

$\mathcal{H}$ is Universal if, for every two different keys k, ℓ, the number of hash functions in $\mathcal{H}$ that map k, ℓ to the same slot is at most $|\mathcal{H}|/m$, i.e.,

$$\forall k \neq \ell \in U, \quad |\{h \in \mathcal{H} \,:\, h(k) = h(\ell)\}| \leq \frac{|\mathcal{H}|}{m}.$$

Let $k_1, k_2, \ldots, k_n$ be the n keys.

Combining the previous two pages,
if $h \in \mathcal{H}$ is chosen uniformly at random:

Average # of other keys mapping to same slot as key k_i is $< \alpha$.
Average # of keys mapping to same slot as non key k is $\leq \alpha$.

Construction of Universal Hash Functions

- Choose prime $p \geq |U| > m$
- Set $Z_p^* = \{1, 2, 3, \ldots, p-1\}$ and $Z_p = \{0, 1, 2, 3, \ldots, p-1\}$
- Define

$$\forall a \in Z_p^*,\, b \in Z_p, \quad h_{a,b}(x) = \Big(\big(ax + b\big) \bmod p\Big) \bmod m$$

Construction of Universal Hash Functions

- Choose prime $p \geq |U| > m$
- Set $Z_p^* = \{1, 2, 3, \ldots, p-1\}$ and $Z_p = \{0, 1, 2, 3, \ldots, p-1\}$
- Define

$$\forall a \in Z_p^*,\, b \in Z_p, \quad h_{a,b}(x) = \Big(\big(ax + b\big) \bmod p\Big) \bmod m$$

Example: Set $p = 17$, $m = 6$. Then

$$h_{3,4}(8) = ((3 \cdot 8 + 4) \bmod 17) \bmod 6 = 5$$

Construction of Universal Hash Functions

- Choose prime $p \geq |U| > m$
- Set $Z_p^* = \{1, 2, 3, \ldots, p-1\}$ and $Z_p = \{0, 1, 2, 3, \ldots, p-1\}$
- Define

$$\forall a \in Z_p^*,\ b \in Z_p, \quad h_{a,b}(x) = \Big((ax+b) \bmod p\Big) \bmod m$$

Example: Set $p = 17$, $m = 6$. Then

$$h_{3,4}(8) = ((3 \cdot 8 + 4) \bmod 17) \bmod 6 = 5$$

Lemma: The Class $\quad \mathcal{H} = \left\{ h_{a,b} \,:\, a \in Z_p^*,\ b \in Z_p \right\}$
is Universal.

Construction of Universal Hash Functions

- Choose prime $p \geq |U| > m$
- Set $Z_p^* = \{1, 2, 3, \ldots, p-1\}$ and $Z_p = \{0, 1, 2, 3, \ldots, p-1\}$
- Define

$$\forall a \in Z_p^*,\ b \in Z_p, \quad h_{a,b}(x) = \Big((ax+b) \bmod p\Big) \bmod m$$

Example: Set $p = 17$, $m = 6$. Then

$$h_{3,4}(8) = ((3 \cdot 8 + 4) \bmod 17) \bmod 6 = 5$$

Lemma: The Class $\mathcal{H} = \left\{ h_{a,b} \,:\, a \in Z_p^*,\ b \in Z_p \right\}$ is Universal.

Proof: Need to show that for all $k \neq \ell$, number of pairs (a, b) with $h_{a,b}(k) = h_{a,b}(\ell)$ is $\leq p(p-1)/m$

Construction of Universal Hash Functions (ii)

$$\forall a \in Z_p^*,\, b \in Z_p, \quad h_{a,b}(x) = \Big((ax + b) \bmod p \Big) \bmod m$$

p prime, $\quad Z_p^* = \{1, 2, 3, \ldots, p-1\}$, $\quad Z_p = \{0, 1, 2, 3, \ldots, p-1\}$

Construction of Universal Hash Functions (ii)

$\forall a \in Z_p^*,\ b \in Z_p, \quad h_{a,b}(x) = \Big((ax+b) \bmod p \Big) \bmod m$

p prime, $\quad Z_p^* = \{1, 2, 3, \ldots, p-1\}$, $\quad Z_p = \{0, 1, 2, 3, \ldots, p-1\}$

(1) Let $k \neq \ell \in U$. For given $(a, b) \in Z_p^* \times Z_p$ set

$$r = (ak + b) \bmod p, \quad s = (a\ell + b) \bmod p$$

Construction of Universal Hash Functions (ii)

$\forall a \in Z_p^*,\, b \in Z_p, \quad h_{a,b}(x) = \Big((ax+b) \bmod p\Big) \bmod m$

p prime, $\quad Z_p^* = \{1, 2, 3, \ldots, p-1\}$, $\quad Z_p = \{0, 1, 2, 3, \ldots, p-1\}$

(1) Let $k \neq \ell \in U$. For given $(a, b) \in Z_p^* \times Z_p$ set

$$r = (ak+b) \bmod p, \quad s = (a\ell + b) \bmod p$$

(2) Every different (a, b) pair generates a unique (r, s) pair

Construction of Universal Hash Functions (ii)

$\forall a \in Z_p^*,\ b \in Z_p, \quad h_{a,b}(x) = \Big((ax+b) \bmod p\Big) \bmod m$

p prime, $\quad Z_p^* = \{1, 2, 3, \ldots, p-1\}, \quad Z_p = \{0, 1, 2, 3, \ldots, p-1\}$

(1) Let $k \neq \ell \in U$. For given $(a, b) \in Z_p^* \times Z_p$ set

$$r = (ak + b) \bmod p, \quad s = (a\ell + b) \bmod p$$

(2) Every different (a, b) pair generates a unique (r, s) pair

Proof:

For a given (r, s) pair we can solve

$$a = (r - s)(k - \ell)^{-1} \bmod p, \quad b = (r - ak) \bmod p.$$

where $(k - \ell)^{-1}$ is the multiplicative inverse base p. Since, for fixed p, k, ℓ, we must have $r \neq s$, there are are $p(p-1)$ (r, s) pairs. Since there are also $p(p-1)$ (a, b) pairs, there is a one-one correspondence between them, with every (a, b) pair generating a diffferent (r, s).

Construction of Universal Hash Functions (iii)

$\forall a \in Z_p^*,\, b \in Z_p, \quad h_{a,b}(x) = \Big((ax+b) \bmod p\Big) \bmod m$

p prime, $\quad Z_p^* = \{1,2,3,\ldots,p-1\}$, $\quad Z_p = \{0,1,2,3,\ldots,p-1\}$

$k \neq \ell$. $(a,b) \in Z_p^* \times Z_p$

$r = (ak+b) \bmod p, \quad s = (a\ell + b) \bmod p$

(2) Every different (a,b) pair generates a unique (r,s) pair, $r \neq s$.

Construction of Universal Hash Functions (iii)

$\forall a \in Z_p^*,\, b \in Z_p, \quad h_{a,b}(x) = \Big((ax+b) \bmod p\Big) \bmod m$

p prime, $\quad Z_p^* = \{1,2,3,\ldots,p-1\}$, $\quad Z_p = \{0,1,2,3,\ldots,p-1\}$

$k \neq \ell$. $(a,b) \in Z_p^* \times Z_p$

$r = (ak+b) \bmod p, \quad s = (a\ell + b) \bmod p$

(2) Every different (a,b) pair generates a unique (r,s) pair, $r \neq s$.

---

(3) $\#$ of (a,b) pairs for which $h_{a,b}(k) = h_{a,b}(\ell)$ is $\leq p(p-1)/m$.

Construction of Universal Hash Functions (iii)

$\forall a \in Z_p^*,\, b \in Z_p, \quad h_{a,b}(x) = \Big((ax+b) \bmod p\Big) \bmod m$

p prime, $\quad Z_p^* = \{1, 2, 3, \ldots, p-1\}, \quad Z_p = \{0, 1, 2, 3, \ldots, p-1\}$

$k \neq \ell$. $(a, b) \in Z_p^* \times Z_p$

$r = (ak+b) \bmod p, \quad s = (a\ell + b) \bmod p$

(2) Every different (a, b) pair generates a unique (r, s) pair, $r \neq s$.

---

(3) $\#$ of (a, b) pairs for which $h_{a,b}(k) = h_{a,b}(\ell)$ is $\leq p(p-1)/m$.

Proof:

Construction of Universal Hash Functions (iii)

$\forall a \in Z_p^*,\ b \in Z_p, \quad h_{a,b}(x) = \Big((ax+b) \bmod p\Big) \bmod m$

p prime, $\quad Z_p^* = \{1, 2, 3, \ldots, p-1\}$, $\quad Z_p = \{0, 1, 2, 3, \ldots, p-1\}$

$k \neq \ell$. $(a, b) \in Z_p^* \times Z_p$

$r = (ak+b) \bmod p, \quad s = (a\ell + b) \bmod p$

(2) Every different (a, b) pair generates a unique (r, s) pair, $r \neq s$.

---

(3) $\#$ of (a, b) pairs for which $h_{a,b}(k) = h_{a,b}(\ell)$ is $\leq p(p-1)/m$.

Proof: $\quad h_{a,b}(k) = h_{a,b}(\ell) \quad$ iff $\quad r \equiv s \bmod m$.

Construction of Universal Hash Functions (iii)

$\forall a \in Z_p^*,\, b \in Z_p, \quad h_{a,b}(x) = \Big((ax+b) \bmod p\Big) \bmod m$

p prime, $\quad Z_p^* = \{1,2,3,\ldots,p-1\}, \quad Z_p = \{0,1,2,3,\ldots,p-1\}$

$k \neq \ell$. $(a,b) \in Z_p^* \times Z_p$

$r = (ak+b) \bmod p, \quad s = (a\ell+b) \bmod p$

(2) Every different (a,b) pair generates a unique (r,s) pair, $r \neq s$.

---

(3) $\#$ of (a,b) pairs for which $h_{a,b}(k) = h_{a,b}(\ell)$ is $\leq p(p-1)/m$.

Proof: $\quad h_{a,b}(k) = h_{a,b}(\ell) \quad$ iff $\quad r \equiv s \bmod m$.

For fixed r, $\#$ of $s \neq r$ with $r \equiv s \bmod m$ is
$\leq \lceil p/m \rceil - 1 \leq (p-1)/m$

Construction of Universal Hash Functions (iii)

$\forall a \in Z_p^*,\ b \in Z_p, \quad h_{a,b}(x) = \Big((ax+b) \bmod p\Big) \bmod m$

p prime, $\quad Z_p^* = \{1,2,3,\ldots,p-1\}, \quad Z_p = \{0,1,2,3,\ldots,p-1\}$

$k \neq \ell$. $(a,b) \in Z_p^* \times Z_p$

$r = (ak+b) \bmod p, \quad s = (a\ell+b) \bmod p$

(2) Every different (a,b) pair generates a unique (r,s) pair, $r \neq s$.

---

(3) # of (a,b) pairs for which $h_{a,b}(k) = h_{a,b}(\ell)$ is $\leq p(p-1)/m$.

Proof: $\quad h_{a,b}(k) = h_{a,b}(\ell) \quad$ iff $\quad r \equiv s \bmod m$.

For fixed r, # of $s \neq r$ with $r \equiv s \bmod m$ is
$\leq \lceil p/m \rceil - 1 \leq (p-1)/m$

Summing over all p possible values of r gives $\leq p(p-1)/m$ pairs (r,s) with $s \neq r$ and $r \equiv s \bmod m$,

Construction of Universal Hash Functions (iii)

$\forall a \in Z_p^*,\ b \in Z_p, \quad h_{a,b}(x) = \Big((ax+b) \bmod p\Big) \bmod m$

p prime, $\quad Z_p^* = \{1,2,3,\ldots,p-1\}, \quad Z_p = \{0,1,2,3,\ldots,p-1\}$

$k \neq \ell$. $(a,b) \in Z_p^* \times Z_p$

$r = (ak+b) \bmod p, \quad s = (a\ell+b) \bmod p$

(2) Every different (a,b) pair generates a unique (r,s) pair, $r \neq s$.

---

(3) $\#$ of (a,b) pairs for which $h_{a,b}(k) = h_{a,b}(\ell)$ is $\leq p(p-1)/m$.

Proof: $\quad h_{a,b}(k) = h_{a,b}(\ell) \quad$ iff $\quad r \equiv s \bmod m$.

For fixed r, $\#$ of $s \neq r$ with $r \equiv s \bmod m$ is
$\leq \lceil p/m \rceil - 1 \leq (p-1)/m$

Summing over all p possible values of r gives $\leq p(p-1)/m$
pairs (r,s) with $s \neq r$ and $r \equiv s \bmod m$,

i.e., $\leq p(p-1)/m = |\mathcal{H}|/m$ pairs (a,b) with $h_{a,b}(k) = h_{a,b}(\ell)$

Construction of Universal Hash Functions (iii)

$\forall a \in Z_p^*,\ b \in Z_p, \quad h_{a,b}(x) = \Big((ax+b) \bmod p\Big) \bmod m$

p prime, $\quad Z_p^* = \{1,2,3,\ldots,p-1\}, \quad Z_p = \{0,1,2,3,\ldots,p-1\}$

$k \neq \ell$. $(a,b) \in Z_p^* \times Z_p$

$r = (ak+b) \bmod p, \quad s = (a\ell+b) \bmod p$

(2) Every different (a,b) pair generates a unique (r,s) pair, $r \neq s$.

---

(3) # of (a,b) pairs for which $h_{a,b}(k) = h_{a,b}(\ell)$ is $\leq p(p-1)/m$.

Proof: $\quad h_{a,b}(k) = h_{a,b}(\ell) \quad$ iff $\quad r \equiv s \bmod m$.

For fixed r, # of $s \neq r$ with $r \equiv s \bmod m$ is $\leq \lceil p/m \rceil - 1 \leq (p-1)/m$

Summing over all p possible values of r gives $\leq p(p-1)/m$ pairs (r,s) with $s \neq r$ and $r \equiv s \bmod m$,

i.e., $\leq p(p-1)/m = |\mathcal{H}|/m$ pairs (a,b) with $h_{a,b}(k) = h_{a,b}(\ell)$

$\boxed{\Rightarrow \mathcal{H} \text{ is Universal}}$

Construction of Universal Hash Functions (iii)

$\forall a \in Z_p^*,\ b \in Z_p, \quad h_{a,b}(x) = \Big((ax+b) \bmod p\Big) \bmod m$

p prime, $\quad Z_p^* = \{1, 2, 3, \ldots, p-1\}, \quad Z_p = \{0, 1, 2, 3, \ldots, p-1\}$

$k \neq \ell.\ (a, b) \in Z_p^* \times Z_p$

$r = (ak+b) \bmod p, \quad s = (a\ell + b) \bmod p$

(2) Every different (a, b) pair generates a unique (r, s) pair, $r \neq s$.

---

(3) $\#$ of (a, b) pairs for which $h_{a,b}(k) = h_{a,b}(\ell)$ is $\leq p(p-1)/m$.

Proof: $\quad h_{a,b}(k) = h_{a,b}(\ell) \quad$ iff $\quad r \equiv s \bmod m$.

For fixed r, $\#$ of $s \neq r$ with $r \equiv s \bmod m$ is
$\leq \lceil p/m \rceil - 1 \leq (p-1)/m$

Summing over all p possible values of r gives $\leq p(p-1)/m$ pairs (r, s) with $s \neq r$ and $r \equiv s \bmod m$,

i.e., $\leq p(p-1)/m = |\mathcal{H}|/m$ pairs (a, b) with $h_{a,b}(k) = h_{a,b}(\ell)$

$$\boxed{\Rightarrow \mathcal{H} \text{ is Universal}}$$

Universal Hashing: Wrap Up

- Just saw that the set of Hash functions

$$\mathcal{H} = \left\{h_{a,b} \, : \, a \in Z_p^*, \, b \in Z_p\right\}$$

  is *Universal*

- This implies that for *any* set of n keys $K = \{k_1, k_2, \ldots, k_n\}$, an effective way of storing the keys is to
  - Choose a random pair (a, b) uniformly at random from the $p(p-1)$ pairs in $Z_p^* \times Z_p$
  - Hash the items in K using hash function $h_{a,b}$

- Because $\mathcal{H}$ is Universal, average time for storing the data will be $O(n\alpha)$ where $\alpha = n/m$ is the load factor

- Average time for doing a search will be $(1 + \alpha)$

Outline

- Introduction
- Hashing with Chaining
- Open Addressing
- Hash Functions & Universal Hashing
- Odds & Ends

Odds & Ends

- Hashing first recognized as a technique in the 1950's
- Comes from English word implying *chop and mix*
- Many different types of hashing for dictionary storage out there. This introduction only scratched the surface

Odds & Ends

- Hashing first recognized as a technique in the 1950's
- Comes from English word implying *chop and mix*
- Many different types of hashing for dictionary storage out there. This introduction only scratched the surface

- A *Cryptographic Hash Funtion* is a hash function that is *almost* impossible to invert efficiently, i.e., given $h(x)$ very difficult to find x.
  - Almost by necessity requires that function h distributes keys pretty "randomly" over $0, 1, 2, \ldots, m$. If not true, then would have first step towards guessing value of x that produces $h(x)$.

  - Example: Password protection. System password file only stores h(password) and not the password itself.
    * When user logs in and types password p, system checks $h(p)$ against file.
    * If an attacker steals the file it wouldn't be helpful, since attacker can't invert hashed password to get original one.