Problem Solving

2-12 Hashing

MA Jun

Institute of Computer Software

May 14, 2020

Contents

Applications of Hashing

There are many applications of hashing (not limited to hash table), including modern day cryptography hash functions. Some of these applications are listed below:

- Message Digest
- Password Verification
- Data Structures (Programming Languages)
- Compiler Operation
- Rabin-Karp Algorithm
- Linking File Name and Path Together

https://www.geeksforgeeks.org/applications-of-hashing/

Contents

Many applications require a **Dynamic Set** that supports only the **dictionary** operations INSERT, SEARCH, and DELETE.

Many applications require a **Dynamic Set** that supports only the **dictionary** operations INSERT, SEARCH, and DELETE.

Searching for an element in a hash table can take as long as searching for an element in a linked list $\Theta(n)$ time in the worst case.

Many applications require a **Dynamic Set** that supports only the **dictionary** operations INSERT, SEARCH, and DELETE.

Searching for an element in a hash table can take as long as searching for an element in a linked list $\Theta(n)$ time in the worst case.

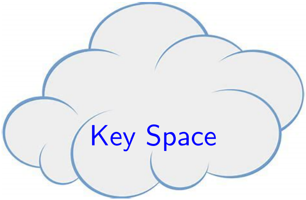
Under reasonable assumptions, the average time to search for an element in a hash table is $\Theta(1)$.

Hashing: the idea

Q : When should we use hash table?
What situations is the hash table suitable for?

Hashing: the idea

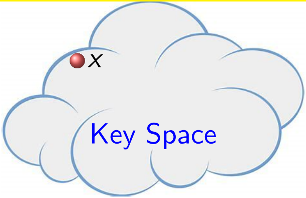
Q : When should we use hash table?
What situations is the hash table suitable for?



Hashing: the idea

Q : When should we use hash table?
What situations is the hash table suitable for?

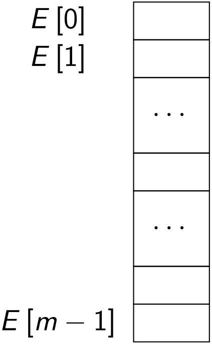
Very large, but only a small part is used in an application at a certain time.



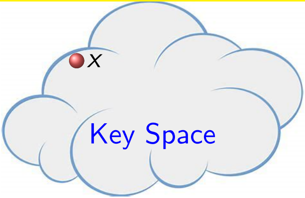
Hashing: the idea

Q : When should we use hash table?
What situations is the hash table suitable for?

Feasible size

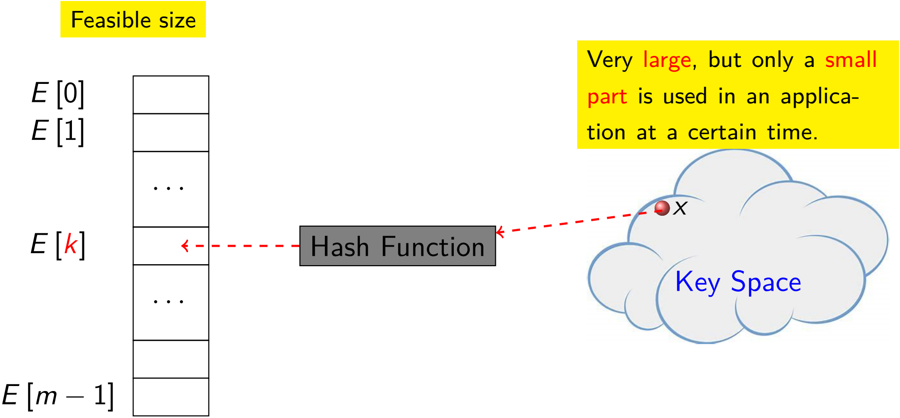


Very large, but only a small part is used in an application at a certain time.



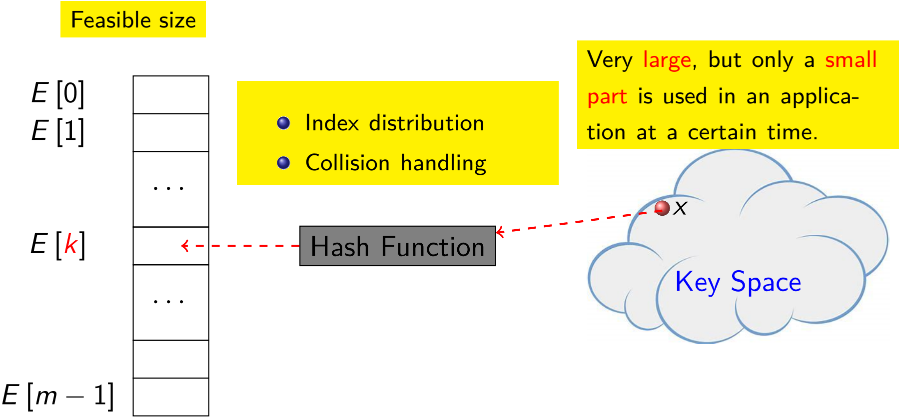
Hashing: the idea

Q : When should we use hash table?
What situations is the hash table suitable for?



Hashing: the idea

Q : When should we use hash table?
What situations is the hash table suitable for?

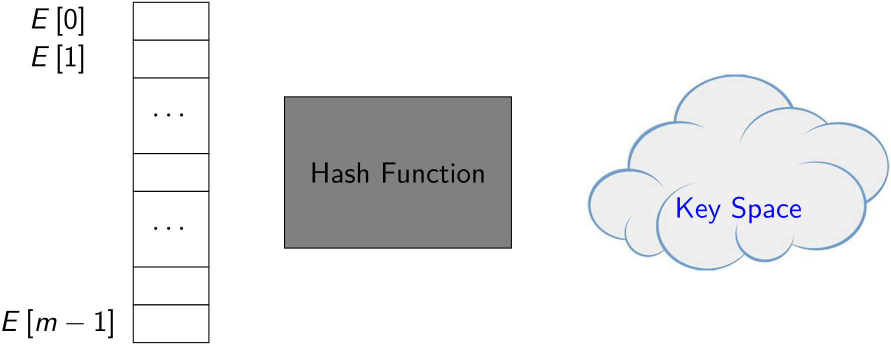


Hashing: the idea

Q : What is a Collision? When does it take place?

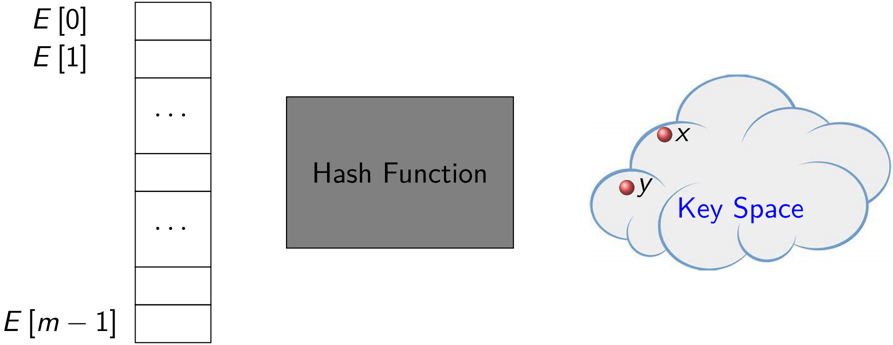
Hashing: the idea

Q : What is a Collision? When does it take place?



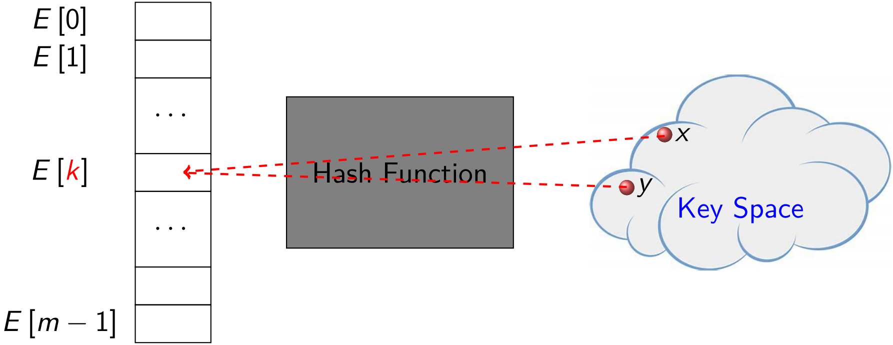
Hashing: the idea

Q : What is a Collision? When does it take place?



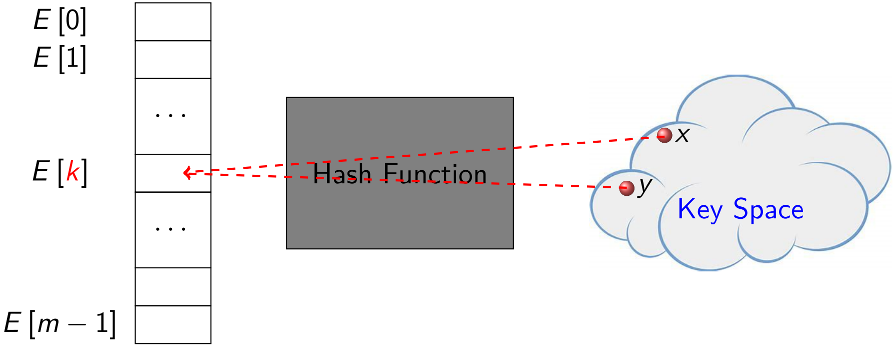
Hashing: the idea

Q : What is a Collision? When does it take place?



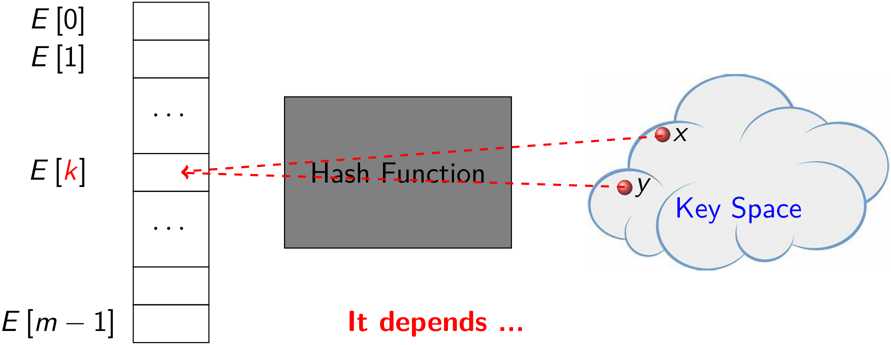
Collision

Q : Given n keys and m slots, what is the expected number of keys hashed into the same slot?



Collision

Q : Given n keys and m slots, what is the expected number of keys hashed into the same slot?



It depends ...

Model & Assumption

Model

- Inserting n keys: sequence of n s independent trails.
- Each insertion corresponds to a value from $\{0, 1, \cdots, m-1\}$.

Model & Assumption

Model

- Inserting n keys: sequence of n s independent trails.
- Each insertion corresponds to a value from $\{0, 1, \cdots, m-1\}$.

Assumption

- For each insertion, the result is Uniformly Distributed, i.e. each value from $\{0, 1, \cdots, m-1\}$ is equally picked.

Model & Assumption

Model

- Inserting n keys: sequence of n s independent trails.
- Each insertion corresponds to a value from $\{0, 1, \cdots, m-1\}$.

Assumption

- For each insertion, the result is Uniformly Distributed, i.e. each value from $\{0, 1, \cdots, m-1\}$ is equally picked.

In hashing n items into a hash table of size m, the expected number of items that hash to any one location is $\alpha = n/m$ (α: loading factor)

Empty location

After inserting n items into m locations

Empty location

After inserting n items into m locations

Q : What is the probability of a given location is empty?

Empty location

After inserting n items into m locations

Q : What is the probability of a given location is empty?

$$\left(1 - \frac{1}{m}\right)^n$$

Empty location

After inserting n items into m locations

Q : What is the probability of a given location is empty?

$$(1 - \frac{1}{m})^n$$

Q : What is the expected number of empty locations?

Empty location

After inserting n items into m locations

Q : What is the probability of a given location is empty?

$$(1 - \frac{1}{m})^n$$

Q : What is the expected number of empty locations?

- $X_i = \begin{cases} 1 & \text{, location i is empty} \\ 0 & \text{, otherwise} \end{cases}$

Empty location

After inserting n items into m locations

Q : What is the probability of a given location is empty?

$$(1 - \frac{1}{m})^n$$

Q : What is the expected number of empty locations?

- $X_i = \begin{cases} 1 & \text{, location i is empty} \\ 0 & \text{, otherwise} \end{cases}$
- $X = \sum\limits_{1 \leq i \leq m} X_i$

Empty location

After inserting n items into m locations

Q : What is the probability of a given location is empty?

$$(1 - \frac{1}{m})^n$$

Q : What is the expected number of empty locations?

- $X_i = \begin{cases} 1 & \text{, location i is empty} \\ 0 & \text{, otherwise} \end{cases}$
- $X = \sum_{1 \le i \le m} X_i$
- $E(X) = E(\sum_{1 \le i \le m} X_i) = \sum_{1 \le i \le m} E(X_i) = \sum_{1 \le i \le m} (1 - 1/m)^n = m(1 - 1/m)^n$

After inserting n items into a hashtable with n locations

After inserting n items into a hashtable with n locations

Q : What is the expected number of items in a given location?

After inserting n items into a hashtable with n locations

Q : What is the expected number of items in a given location?

- $Y_i = \begin{cases} 1 & \text{, the } i\text{th item in the given location} \\ 0 & \text{, otherwise} \end{cases}$

After inserting n items into a hashtable with n locations

Q : What is the expected number of items in a given location?

- $Y_i = \begin{cases} 1 & \text{, the } i\text{th item in the given location} \\ 0 & \text{, otherwise} \end{cases}$
- $Y = \sum_{1 \le i \le n} Y_i$

After inserting n items into a hashtable with n locations

Q : What is the expected number of items in a given location?

- $Y_i = \begin{cases} 1 & \text{, the } i\text{th item in the given location} \\ 0 & \text{, otherwise} \end{cases}$
- $Y = \sum\limits_{1 \le i \le n} Y_i$
- $E(Y) = E(\sum\limits_{1 \le i \le n} Y_i) = \sum\limits_{1 \le i \le n} E(Y_i) = \sum\limits_{1 \le i \le n} 1/n = 1$

After inserting n items into a hashtable with n locations

Q : What is the expected number of items in a given location?

- $Y_i = \begin{cases} 1 & \text{, the } i\text{th item in the given location} \\ 0 & \text{, otherwise} \end{cases}$
- $Y = \sum_{1 \le i \le n} Y_i$
- $E(Y) = E(\sum_{1 \le i \le n} Y_i) = \sum_{1 \le i \le n} E(Y_i) = \sum_{1 \le i \le n} 1/n = 1$

Expected number of empty locations: $n(1 - 1/n)^n = n/e \approx 0.368n$

After inserting n items into a hashtable with n locations

Q : What is the expected number of items in a given location?

- $Y_i = \begin{cases} 1 & \text{, the } i\text{th item in the given location} \\ 0 & \text{, otherwise} \end{cases}$
- $Y = \sum_{1 \le i \le n} Y_i$
- $E(Y) = E(\sum_{1 \le i \le n} Y_i) = \sum_{1 \le i \le n} E(Y_i) = \sum_{1 \le i \le n} 1/n = 1$

Expected number of empty locations: $n(1 - 1/n)^n = n/e \approx 0.368n$

PARADOX?

After inserting n items into a hashtable with n locations

Q : What is the expected number of items in a given location?

- $Y_i = \begin{cases} 1 & \text{, the } i\text{th item in the given location} \\ 0 & \text{, otherwise} \end{cases}$
- $Y = \sum_{1 \le i \le n} Y_i$
- $E(Y) = E(\sum_{1 \le i \le n} Y_i) = \sum_{1 \le i \le n} E(Y_i) = \sum_{1 \le i \le n} 1/n = 1$

Expected number of empty locations: $n(1 - 1/n)^n = n/e \approx 0.368n$

PARADOX?

collisions!

Expected number of collisions

After inserting n items into m locations

Q : What is the expected number of collisions?

Expected number of collisions

$\boxed{\text{After inserting } n \text{ items into } m \text{ locations}}$

Q : What is the expected number of collisions?

$$E(\text{collisions}) = n - E(\text{occupied locations})$$

Expected number of collisions

$\boxed{\text{After inserting } n \text{ items into } m \text{ locations}}$

Q : What is the expected number of collisions?

$$
\begin{aligned}
E(\text{collisions}) &= n - E(\text{occupied locations}) \\
&= n - (m - E(\text{empty locations}))
\end{aligned}
$$

Expected number of collisions

After inserting n items into m locations

Q : What is the expected number of collisions?

$$
\begin{aligned}
E(\text{collisions}) &= n - E(\text{occupied locations}) \\
&= n - (m - E(\text{empty locations})) \\
&= n - (m - m(1 - 1/m)^n) \\
&= n - m + m(1 - 1/m)^n
\end{aligned}
$$

Expected number of collisions

$\boxed{\text{After inserting } n \text{ items into } m \text{ locations}}$

Q : What is the expected number of collisions?

$$
\begin{aligned}
E(\text{collisions}) &= n - E(\text{occupied locations}) \\
&= n - (m - E(\text{empty locations})) \\
&= n - (m - m(1 - 1/m)^n) \\
&= n - m + m(1 - 1/m)^n
\end{aligned}
$$

Example:

When $n = 100, m = 100$, then $E(\text{collision}) \approx 37$

After inserting n items into m locations

Q : What is the expected number of items needed to fullfill all m locations?

After inserting n items into m locations

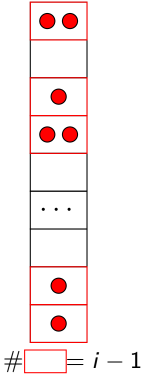
Q : What is the expected number of items needed to fullfill all m locations?

- X_i: the number of items to be added to increase the number of occupied locations from $i-1$ to i.

After inserting n items into m locations

Q : What is the expected number of items needed to fullfill all m locations?

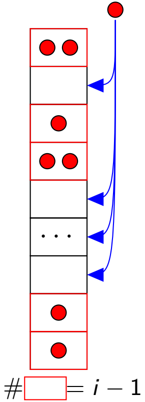
- X_i: the number of items to be added to increase the number of occupied locations from $i-1$ to i.
- Given $i-1$ occupied locations,



After inserting n items into m locations

Q : What is the expected number of items needed to fullfill all m locations?

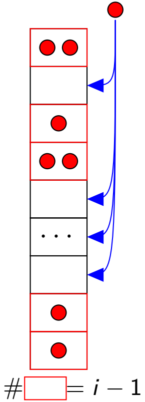
- X_i: the number of items to be added to increase the number of occupied locations from $i-1$ to i.
- Given $i-1$ occupied locations,



After inserting n items into m locations

Q : What is the expected number of items needed to fullfill all m locations?

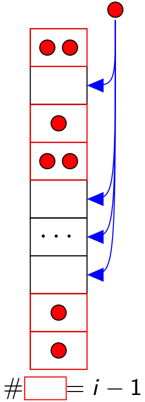
- X_i: the number of items to be added to increase the number of occupied locations from $i-1$ to i.
- Given $i-1$ occupied locations, the success probability for each trial is $(m-i+1)/m$



After inserting n items into m locations

Q : What is the expected number of items needed to fullfill all m locations?

- X_i: the number of items to be added to increase the number of occupied locations from $i-1$ to i.
- Given $i-1$ occupied locations, the success probability for each trial is $(m-i+1)/m$
- Thus, $E(X_i) = m/(m-i+1)$

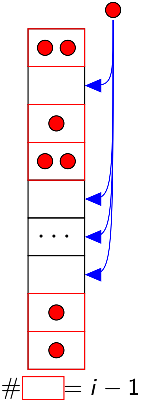


After inserting n items into m locations

Q : What is the expected number of items needed to fullfill all m locations?

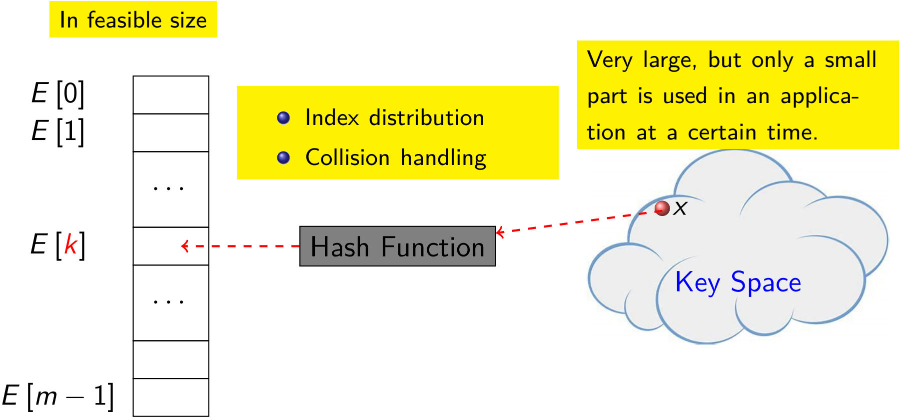
- X_i: the number of items to be added to increase the number of occupied locations from $i-1$ to i.
- Given $i-1$ occupied locations, the success probability for each trial is $(m-i+1)/m$
- Thus, $E(X_i) = m/(m-i+1)$

$$\begin{aligned} E(X) &= \sum_{i=1}^{m} E(X_i) = \sum_{i=1}^{m} m/(m-i+1) \\ &= m \sum_{i=1}^{m} 1/(m-i+1) \\ &= m \sum_{j=1}^{m} 1/j = \Theta(m \lg m) \end{aligned}$$

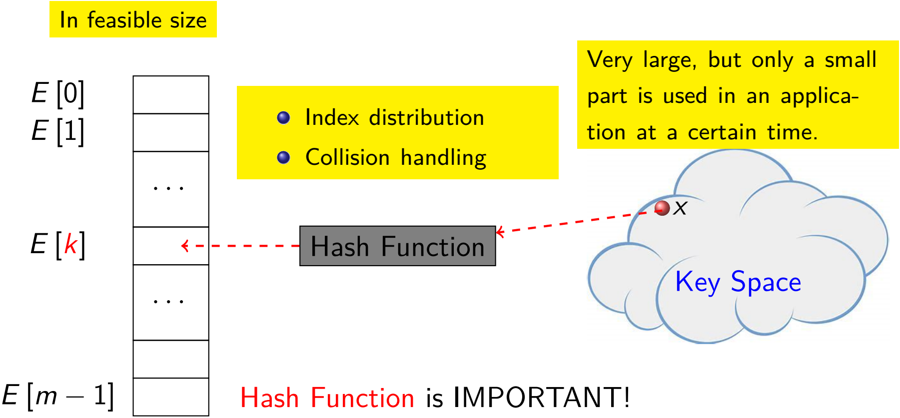


Contents

Hashing: the idea



Hashing: the idea



Hash Function is IMPORTANT!

Two typical hash functions

Division Method: map a key k into one of m slots by taking the remainder of k divided by m.

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

Multiplication Method:

- **Step-1**: multiply the k by a constant A in the range $0 < A < 1$ and extract the **fractional part** of kA.
- **Step-2**: multiply the **fractional part** by m.

Two typical hash functions

Division Method: map a key k into one of m slots by taking the remainder of k divided by m:

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

Two typical hash functions

Division Method: map a key k into one of m slots by taking the remainder of k divided by m:

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

Q : Why should we avoid $m = 2^p$?

Two typical hash functions

Division Method: map a key k into one of m slots by taking the remainder of k divided by m:

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

Q : Why should we avoid $m = 2^p$?

- Since if $m = 2^p$, then $h(k)$ is just the p lowest-order bits of k.

Two typical hash functions

Division Method: map a key k into one of m slots by taking the remainder of k divided by m:

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

Q : Why should we avoid $m = 2^p$?

- Since if $m = 2^p$, then $h(k)$ is just the p lowest-order bits of k.
- Unless we know that all low-order p-bit patterns are equally likely, we are better off designing the hash function to depend on all the bits of the key.

Two typical hash functions

Division Method: map a key k into one of m slots by taking the remainder of k divided by m:

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

Q : Why should we avoid $m = 2^p$?

- Since if $m = 2^p$, then $h(k)$ is just the p lowest-order bits of k.
- Unless we know that all low-order p-bit patterns are equally likely, we are better off designing the hash function to depend on all the bits of the key.

A **prime** not too close to an exact power of 2 is often a good choice for m.

Exercise 11.3-3

Two typical hash functions

Multiplication Method:

- **Step-1**: multiply the k by a constant A in the range $0 < A < 1$ and extract the **fractional part** of kA.
- **Step-2**: multiply the **fractional part** by m

$$h(k) = \lfloor m(kA \mod 1) \rfloor$$

Two typical hash functions

Multiplication Method:

- **Step-1**: multiply the k by a constant A in the range $0 < A < 1$ and extract the **fractional part** of kA.
- **Step-2**: multiply the **fractional part** by m

$$h(k) = \lfloor m(kA \mod 1) \rfloor$$

Q : Why do we usually choose $m = 2^p$?

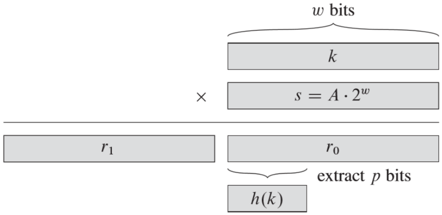
Two typical hash functions

Multiplication Method:

- **Step-1**: multiply the k by a constant A in the range $0 < A < 1$ and extract the **fractional part** of kA.
- **Step-2**: multiply the **fractional part** by m

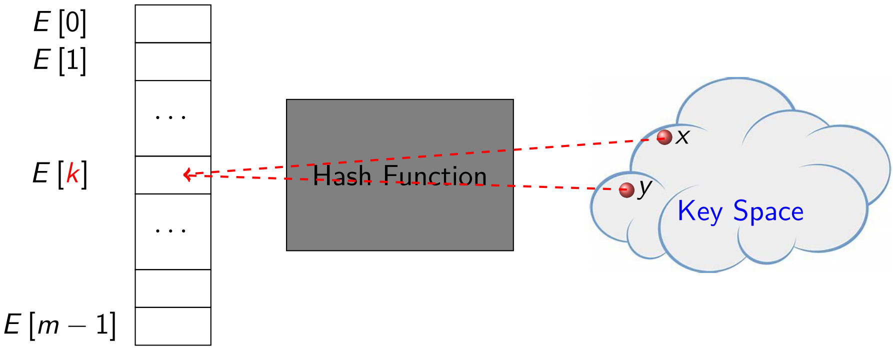
$$h(k) = \lfloor m(kA \mod 1) \rfloor$$

Q : Why do we usually choose $m = 2^p$?



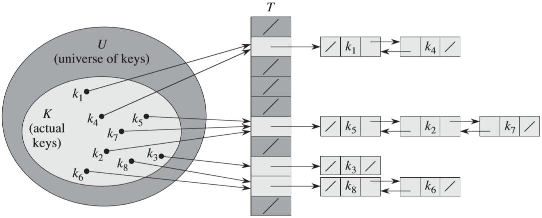
Contents

Collision

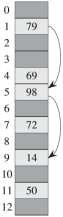


Collision Resolution

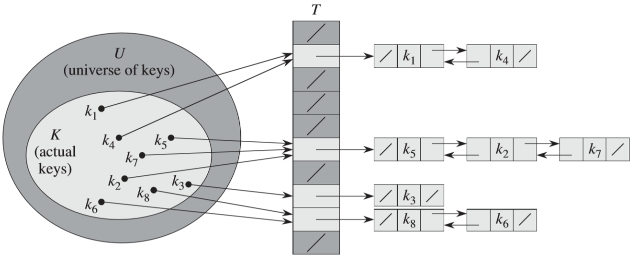
Chaining (Closed Addressing)



Open Addressing



Collision Resolution by Chaining



CHAINED-HASH-INSERT(T, x)

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

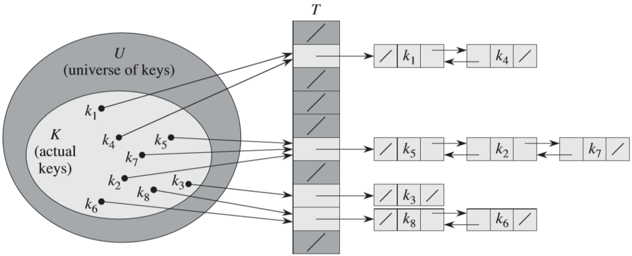
CHAINED-HASH-SEARCH(T, k)

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

CHAINED-HASH-DELETE(T, x)

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

Collision Resolution by Chaining



CHAINED-HASH-INSERT(T, x)

1 insert x at the $\boxed{\text{head}}$ of list $T[h(x.key)]$

CHAINED-HASH-SEARCH(T, k)

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

CHAINED-HASH-DELETE(T, x)

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

Collision Resolution by Chaining: Unsuccessful Search

Q : What is the average cost of an **unsuccessful** search?

Collision Resolution by Chaining: Unsuccessful Search

Q : What is the average cost of an **unsuccessful** search?

> ***Theorem 11.1***
> In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

Collision Resolution by Chaining: Unsuccessful Search

Q : What is the average cost of an **unsuccessful** search?

Theorem 11.1
In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

For $j = 0, 1, 2, ..., m-1$, the average length of the list at $E[j]$ is $n/m = \alpha$.

Collision Resolution by Chaining: Unsuccessful Search

Q : What is the average cost of an **unsuccessful** search?

> ***Theorem 11.1***
> In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

> For $j = 0, 1, 2, ..., m-1$, the average length of the list at $E[j]$ is $n/m = \alpha$.

- Any key that is not in the table is **equally** likely to hash to any of the m addresses.

Collision Resolution by Chaining: Unsuccessful Search

Q : What is the average cost of an **unsuccessful** search?

> ***Theorem 11.1***
> In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

> For $j = 0, 1, 2, \ldots, m-1$, the average length of the list at $E[j]$ is $n/m = \alpha$.

- Any key that is not in the table is **equally** likely to hash to any of the m addresses.
- The **average cost** to determine that the key is not in the list $E[h(k)]$ is the cost to search to the end of the list, which is α.

Collision Resolution by Chaining: Unsuccessful Search

Q : What is the average cost of an **unsuccessful** search?

Theorem 11.1
In a hash table in which collisions are resolved by chaining, an unsuccessful search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

For $j = 0, 1, 2, ..., m-1$, the average length of the list at $E[j]$ is $n/m = \alpha$.

- Any key that is not in the table is **equally** likely to hash to any of the m addresses.
- The **average cost** to determine that the key is not in the list $E[h(k)]$ is the cost to search to the end of the list, which is α.
- Total cost is $\Theta(1 + \alpha)$

Collision Resolution by Chaining: Successful Search

Q : What is the average cost of an **successful** search?

Collision Resolution by Chaining: Successful Search

Q : What is the average cost of an **successful** search?

> ***Theorem 11.2***
> In a hash table in which collisions are resolved by chaining, a successful search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

Collision Resolution by Chaining: Successful Search

Q : What is the average cost of an **successful** search?

> ***Theorem 11.2***
> In a hash table in which collisions are resolved by chaining, a successful search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

- x_i: the ith element inserted into the table.

Collision Resolution by Chaining: Successful Search

Q : What is the average cost of an **successful** search?

> ***Theorem 11.2***
> In a hash table in which collisions are resolved by chaining, a successful search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

- x_i: the ith element inserted into the table.
- The probability of that x_i is searched is $1/n$.

Collision Resolution by Chaining: Successful Search

Q : What is the average cost of an **successful** search?

> ***Theorem 11.2***
> In a hash table in which collisions are resolved by chaining, a successful search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

- x_i: the ith element inserted into the table.
- The probability of that x_i is searched is $1/n$.
- For a specific x_i, the number of elements examined in a successful search is $t + 1$

Collision Resolution by Chaining: Successful Search

Q : What is the average cost of an **successful** search?

> ***Theorem 11.2***
> In a hash table in which collisions are resolved by chaining, a successful search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

- x_i: the ith element inserted into the table.
- The probability of that x_i is searched is $1/n$.
- For a specific x_i, the number of elements examined in a successful search is $t+1$
  - t: the number of elements inserted into the same list as x_i, **after** x_i has been inserted.

Collision Resolution by Chaining: Successful Search

Q : What is the average cost of an **successful** search?

> ***Theorem 11.2***
> In a hash table in which collisions are resolved by chaining, a successful search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

- x_i: the ith element inserted into the table.
- The probability of that x_i is searched is $1/n$.
- For a specific x_i, the number of elements examined in a successful search is $t + 1$
  - t: the number of elements inserted into the same list as x_i, **after** x_i has been inserted.
  - The probability of that x_j is inserted into the **same** list of x_i is $1/m$.

Collision Resolution by Chaining: Successful Search

Q : What is the average cost of an **successful** search?

> ***Theorem 11.2***
> In a hash table in which collisions are resolved by chaining, a successful search takes average-case time $\Theta(1+\alpha)$, under the assumption of simple uniform hashing.

- x_i: the ith element inserted into the table.
- The probability of that x_i is searched is $1/n$.
- For a specific x_i, the number of elements examined in a successful search is $t + 1$
  - t: the number of elements inserted into the same list as x_i, **after** x_i has been inserted.
  - The probability of that x_j is inserted into the **same** list of x_i is $1/m$.
  - The **expected number of elements examined** for a successful search of x_i is $(1 + \sum_{j=i+1}^{n} \frac{1}{m})$

Collision Resolution by Chaining: Successful Search

Q : What is the average cost of a **successful** search?

- Average **number of elements examined** for an successful search is

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

Collision Resolution by Chaining: Successful Search

Q : What is the average cost of a **successful** search?

- Average **number of elements examined** for an successful search is

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

Collision Resolution by Chaining: Successful Search

Q : What is the average cost of a **successful** search?

- Average **number of elements examined** for an successful search is

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

Collision Resolution by Chaining: Successful Search

Q : What is the average cost of a **successful** search?

- Average **number of elements examined** for an successful search is

$$\begin{aligned}\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\frac{1}{m}\right) &= 1+\frac{1}{nm}\left(\sum_{j=i+1}^{n}1\right)\\ &= 1+\frac{1}{nm}\left(\sum_{i=1}^{n}n-i\right)\\ &= 1+\frac{1}{nm}\left(n^2-\sum_{i=1}^{n}i\right)\end{aligned}$$

Collision Resolution by Chaining: Successful Search

Q : What is the average cost of a **successful** search?

- Average **number of elements examined** for an successful search is

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

Collision Resolution by Chaining: Successful Search

Q : What is the average cost of a **successful** search?

- Average **number of elements examined** for an successful search is

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

Average **cost** for a successful search is $\mathbf{2}+\alpha-\frac{\alpha}{2n}=\Theta(1+\alpha)$

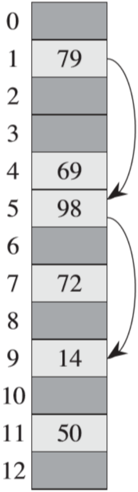
Collision Resolution by Chaining: Example

JAVA 7 HashMap

```java
static int hash(int h) {
    h^= (h>>>20)^(h>>>12);
    return h^(h>>>7)^(h>>>4);
}
```

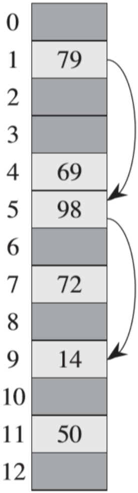
- In Java 7, after calculating hash from hash function, if more then one element has same hash than they are searched by linear search, so it's complexity is $O(n)$.
- In Java 8, that search is performed by **binary search** so the complexity will become $\log n$.

Open Addressing



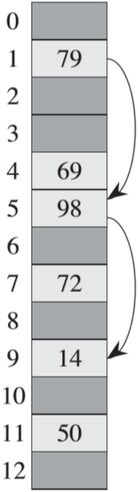
- All elements are stored in the hash table, **no linked list is used**. So, $\alpha \leq 1$.

Open Addressing



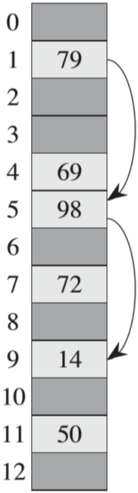
- All elements are stored in the hash table, **no linked list is used**. So, $\alpha \leq 1$.
- Collision is settled by "**rehashing**":

Open Addressing



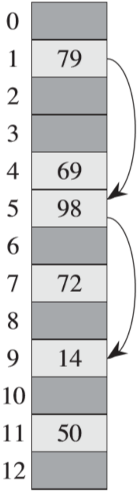
- All elements are stored in the hash table, **no linked list is used**. So, $\alpha \leq 1$.
- Collision is settled by "**rehashing**":
  - A function used to get **a new hashing address for each collided address**

Open Addressing



- All elements are stored in the hash table, **no linked list is used**. So, $\alpha \leq 1$.
- Collision is settled by "**rehashing**":
  - A function used to get **a new hashing address for each collided address**
  - The hash table slots are probed successively, until a valid location is found.

Open Addressing



- All elements are stored in the hash table, **no linked list is used**. So, $\alpha \leq 1$.
- Collision is settled by "**rehashing**":
  - A function used to get **a new hashing address for each collided address**
  - The hash table slots are probed successively, until a valid location is found.
- The **probing sequence** can be seen as a **permutation** of $(0, 1, 2, \cdots, m-1) \rightarrow \langle h(k, 0), h(k, 1), \cdots, h(k, m-1) \rangle$

Open Addressing: Commonly Used Probings

Linear Probing: (**primary clustering** may occur)

Given an auxiliary hash function h', the hash function is:

$$h(k, i) = (h'(k) + i) \mod m, (i = 0, 1, ..., m - 1)$$

Open Addressing: Commonly Used Probings

Linear Probing: (**primary clustering** may occur)

Given an auxiliary hash function h', the hash function is:

$$h(k,i) = (h'(k) + i) \mod m, (i = 0, 1, ..., m-1)$$

Quadratic Probing: (**secondary clustering** may occur)

Given auxiliary function h' and nonzero auxiliary constant c_1 and c_2, the hash function is:

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \mod m, (i = 0, 1, ..., m-1)$$

Open Addressing: Commonly Used Probings

Linear Probing: (**primary clustering** may occur)

Given an auxiliary hash function h', the hash function is:

$$h(k,i) = (h'(k) + i) \mod m, (i = 0, 1, ..., m-1)$$

Quadratic Probing: (**secondary clustering** may occur)

Given auxiliary function h' and nonzero auxiliary constant c_1 and c_2, the hash function is:

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \mod m, (i = 0, 1, ..., m-1)$$

Double Hashing:

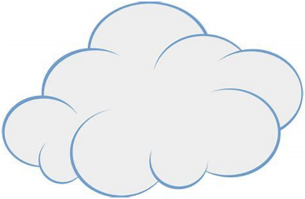
Given auxiliary functions h_1 and h_2, the hash function is:

$$h(k,i) = (h_1(k) + ih_2(k)) \mod m, (i = 0, 1, ..., m-1)$$

Linear Probing: an Example

0	1776
1	
2	
3	1055
4	1492
5	
6	1918
7	

Hasing function: $h(x) = 5x \mod 8$

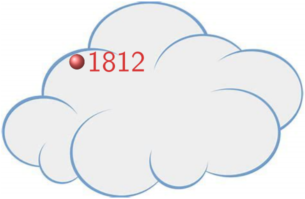


Rehasing function: $rh(j) = (j + 1) \mod 8$

Linear Probing: an Example

Hasing function: $h(x) = 5x \mod 8$

0	1776
1	
2	
3	1055
4	1492
5	
6	1918
7	

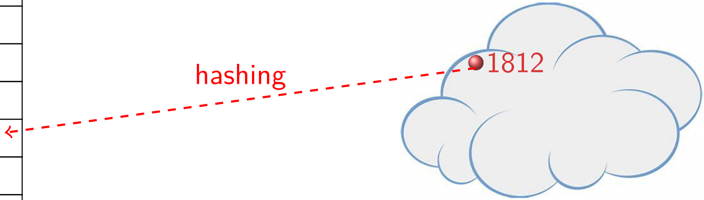


Rehasing function: $rh(j) = (j + 1) \mod 8$

Linear Probing: an Example

Hasing function: $h(x) = 5x \mod 8$

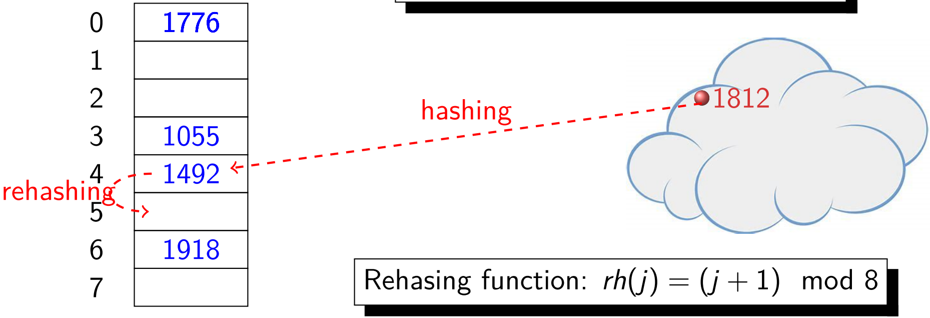
0	1776
1	
2	
3	1055
4	1492
5	
6	1918
7	



Rehasing function: $rh(j) = (j + 1) \mod 8$

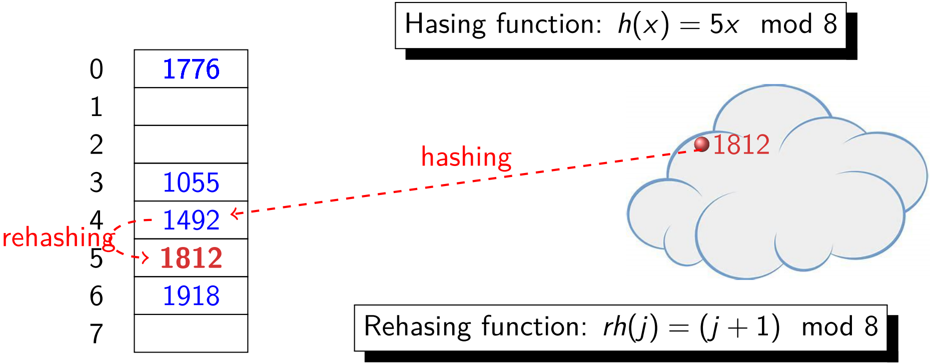
Linear Probing: an Example

Hasing function: $h(x) = 5x \mod 8$



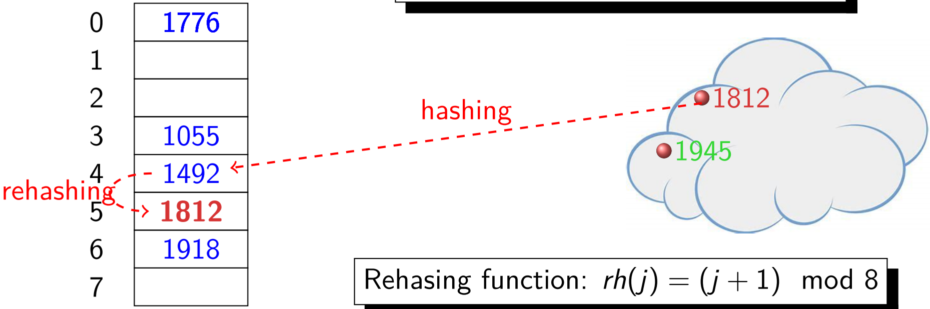
Rehasing function: $rh(j) = (j+1) \mod 8$

Linear Probing: an Example



Linear Probing: an Example

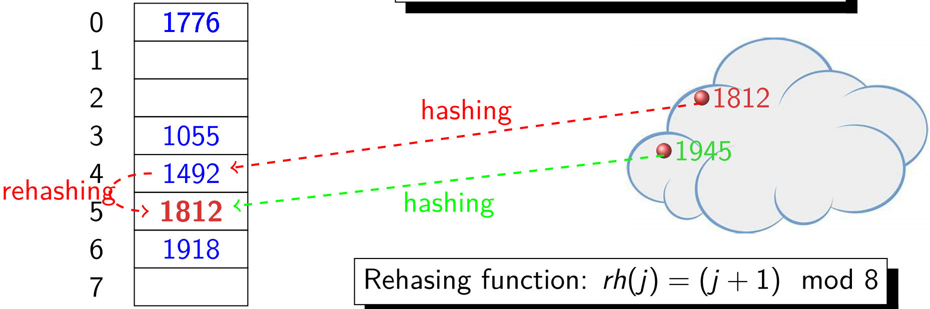
Hasing function: $h(x) = 5x \mod 8$



Rehasing function: $rh(j) = (j+1) \mod 8$

Linear Probing: an Example

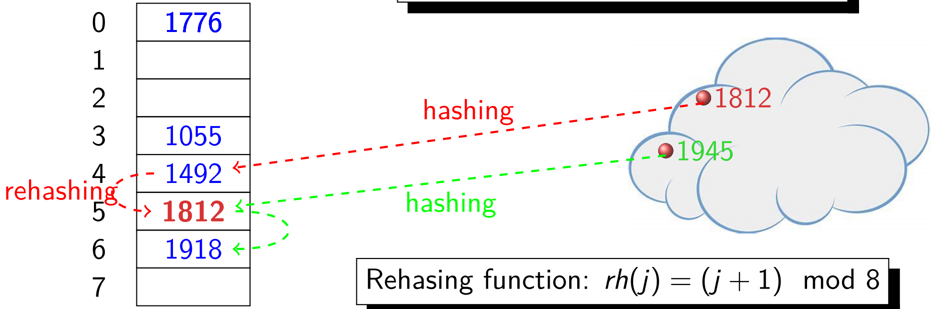
Hasing function: $h(x) = 5x \mod 8$



Rehasing function: $rh(j) = (j + 1) \mod 8$

Linear Probing: an Example

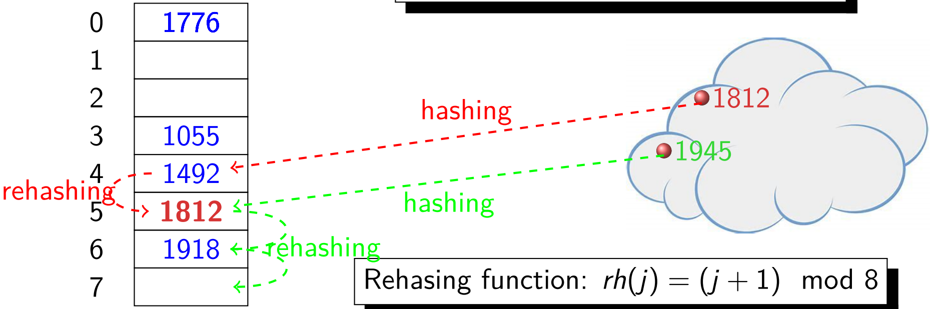
Hasing function: $h(x) = 5x \mod 8$



Rehasing function: $rh(j) = (j + 1) \mod 8$

Linear Probing: an Example

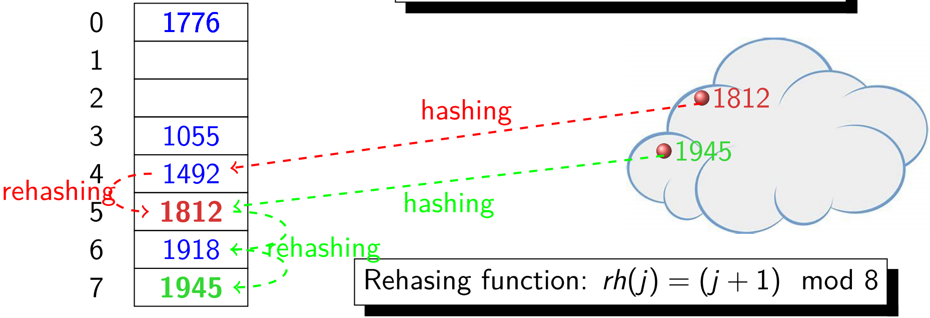
Hasing function: $h(x) = 5x \mod 8$



Rehasing function: $rh(j) = (j + 1) \mod 8$

Linear Probing: an Example

Hasing function: $h(x) = 5x \mod 8$



Rehasing function: $rh(j) = (j + 1) \mod 8$

Open Addressing: Commonly Used Probings

Q : How to evaluate the **goodness** of a probing?

Open Addressing: Commonly Used Probings

Q : How to evaluate the **goodness** of a probing?

Assumption

Each key is **equally** likely to have any of the $m!$ permutations of $(1, 2, \cdots, m-1)$ as its probe sequence.

Open Addressing: Commonly Used Probings

Q : How to evaluate the **goodness** of a probing?

Assumption

Each key is **equally** likely to have any of the $m!$ permutations of $(1, 2, \cdots, m-1)$ as its probe sequence.

- Both linear and quadratic probing have only m distinct probe sequences, as determined by the first probe.

$$h(k, i) = (h'(k) + i) \mod m, (i = 0, 1, ..., m-1)$$
$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \mod m, (i = 0, 1, ..., m-1)$$

Open Addressing: Commonly Used Probings

Q : How to evaluate the **goodness** of a probing?

Assumption

Each key is **equally** likely to have any of the $m!$ permutations of $(1, 2, \cdots, m-1)$ as its probe sequence.

- Both linear and quadratic probing have only m distinct probe sequences, as determined by the first probe.

$$h(k, i) = (h'(k) + i) \mod m, (i = 0, 1, ..., m-1)$$
$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \mod m, (i = 0, 1, ..., m-1)$$

- Double hashing improves over linear or quadratic probing in that $\Theta(m^2)$ probe sequences are used.

$$h(k, i) = (h_1(k) + ih_2(k)) \mod m, (i = 0, 1, ..., m-1)$$

Open Addressing: Deleting Element

Q : Why Open Addressing is not suitable for situations where items might be deleted?

Open Addressing: Deleting Element

Q : Why Open Addressing is not suitable for situations where items might be deleted?

- The probing chain might be broken if an item is deleted.

Open Addressing: Deleting Element

Q : Why Open Addressing is not suitable for situations where items might be deleted?

- The probing chain might be broken if an item is deleted.
- How to deal with it?

Open Addressing: Unsuccessful Search

Q : Assuming uniform hashing, what is the average number of probes in an **unsuccessful** search?

Open Addressing: Unsuccessful Search

Q : Assuming uniform hashing, what is the average number of probes in an **unsuccessful** search?

Theorem 11.6
Given an open-address hash table with load factor $\alpha = n/m < 1$, the expected number of probes in an unsuccessful search is at most $1/(1-\alpha)$, assuming uniform hashing.

Open Addressing: Unsuccessful Search

Q : Assuming uniform hashing, what is the average number of probes in an **unsuccessful** search?

> ***Theorem 11.6***
> Given an open-address hash table with load factor $\alpha = n/m < 1$, the expected number of probes in an unsuccessful search is at most $1/(1-\alpha)$, assuming uniform hashing.

- X: the number of probes made in an **unsuccessful** search

Open Addressing: Unsuccessful Search

Q : Assuming uniform hashing, what is the average number of probes in an **unsuccessful** search?

> ***Theorem 11.6***
> Given an open-address hash table with load factor $\alpha = n/m < 1$, the expected number of probes in an unsuccessful search is at most $1/(1-\alpha)$, assuming uniform hashing.

- X: the number of probes made in an **unsuccessful** search

$$
\begin{aligned}
E(X) &= \sum_{i=0}^{\infty} i \cdot Pr(X = i) \\
&= \sum_{i=0}^{\infty} i \cdot (Pr(X \geq i) - Pr(X \geq i+1)) \\
&= \sum_{i=1}^{\infty} Pr(X \geq i)
\end{aligned}
$$

Open Addressing: Unsuccessful Search

Q : How to compute $Pr(X \geq i)$?

Open Addressing: Unsuccessful Search

Q : How to compute $Pr(X \geq i)$?

- A_i: the ith probe occurs at an occupied slot

Open Addressing: Unsuccessful Search

Q : How to compute $Pr(X \geq i)$?

- A_i: the ith probe occurs at an occupied slot
- $X \geq i = A_1 \cap A_2 \cap \cdots \cap A_{i-1}$

Open Addressing: Unsuccessful Search

Q : How to compute $Pr(X \geq i)$?

- A_i: the ith probe occurs at an occupied slot
- $X \geq i = A_1 \cap A_2 \cap \cdots \cap A_{i-1}$

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

Open Addressing: Unsuccessful Search

Q : How to compute $Pr(X \geq i)$?

- A_i: the ith probe occurs at an occupied slot
- $X \geq i = A_1 \cap A_2 \cap \cdots \cap A_{i-1}$

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

Then, $\quad E(x) = \sum_{i=1}^{\infty} Pr(X \geq i) \leq \sum_{i=1}^{\infty} \alpha^{i-1} =\leq \sum_{i=0}^{\infty} \alpha^{i} = \frac{1}{1-\alpha}$

Open Addressing: Inserting an Element

Q : What is the average cost for inserting an element into a table with load factor α?

Open Addressing: Inserting an Element

Q : What is the average cost for inserting an element into a table with load factor α?

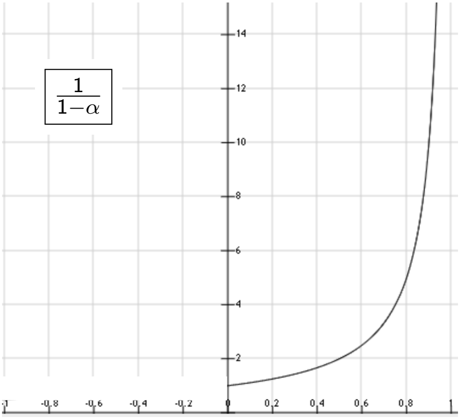
> ***Corollary 11.7***
> Inserting an element into an open-address hash table with load factor α requires at most $1/(1-\alpha)$ probes on average, assuming uniform hashing.

Proof.

- An element is inserted only if there is room in the table, and thus $\alpha < 1$.
- Inserting a key requires an unsuccessful search followed by placing the key into the first empty slot found.
- Thus, the expected number of probes is at most $1/(1-\alpha)$

□

Open Addressing: Insertion/Unsuccessful Search



Open Addressing: Successful Search

Q : What is the average number of probes in a **successful** search of an element in a table with load factor α?

Open Addressing: Successful Search

Q : What is the average number of probes in a **successful** search of an element in a table with load factor α?

Theorem 11.8
Given an open-address hash table with load factor $\alpha < 1$, the expected number of probes in a successful search is at most

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

assuming uniform hashing and assuming that each key in the table is equally likely to be searched for.

Open Addressing: Successful Search

Q : what is the average number of probes in a successful search for an element in a table with load factor α?

Proof.

Open Addressing: Successful Search

Q : what is the average number of probes in a successful search for an element in a table with load factor α?

Proof.

- A **search** for a key k reproduces the **same probe sequence** as when the element with key k was **inserted**.

Open Addressing: Successful Search

Q : what is the average number of probes in a successful search for an element in a table with load factor α?

Proof.

- A **search** for a key k reproduces the **same probe sequence** as when the element with key k was **inserted**.
- If k is the (i+1)st key inserted into the table, the expected number of probes in a search for k is at most $1/(1 - i/m) = m/(m - i)$

Open Addressing: Successful Search

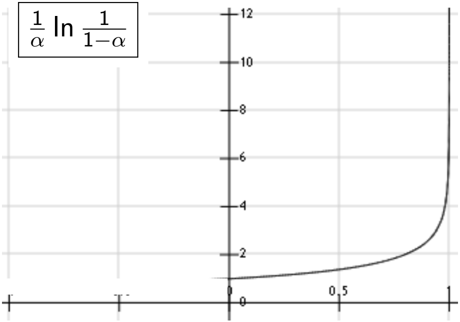
Q : what is the average number of probes in a successful search for an element in a table with load factor α?

Proof.

- A **search** for a key k reproduces the **same probe sequence** as when the element with key k was **inserted**.
- If k is the (i+1)st key inserted into the table, the expected number of probes in a search for k is at most $1/(1 - i/m) = m/(m - i)$
- The expected number of probes is:

$$\begin{aligned} \frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i} &= \frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i} = \frac{1}{\alpha}\sum_{k=m-n+1}^{m}\frac{1}{k} \\ &\leq \frac{1}{\alpha}\int_{m-n}^{m}\frac{1}{x}dx = \frac{1}{\alpha}\ln\frac{m}{m-n} \\ &= \frac{1}{\alpha}\ln\frac{1}{1-\alpha} \end{aligned}$$

Open Addressing: Successful Search



For your reference:

- $\alpha = 0.5$: 1.387
- $\alpha = 0.9$: 2.559

Thank You!
Questions?

Office 819
majun@nju.edu.cn