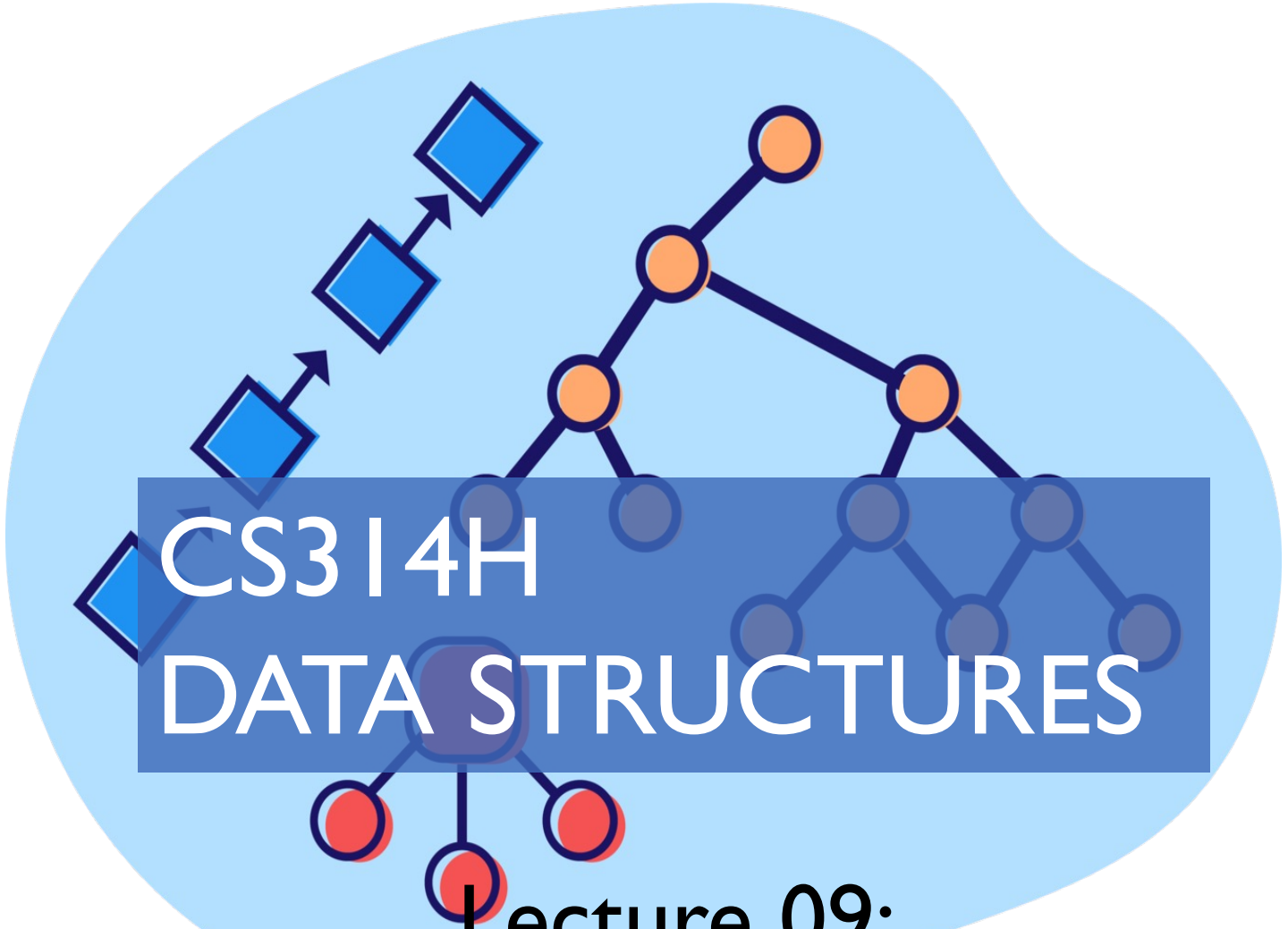


CS314H
DATA STRUCTURES

Lecture 09:
Hashing – Open Addressing

Mikyung Han

Please, interrupt and ask questions AT ANY TIME !

Outline

1. Administrative
2. Open addressing

Open Addressing resolves collisions by choosing a different location when the natural choice is full

Sure, different location OC! But what should be the criteria?

Open Addressing In General

Choose a new function $f(x)$ and then probe with

$$(h(\text{key}) + f(i)) \bmod |T|$$

- We should be able to reproduce the path
  - Cannot be random. Still deterministic
- We want to utilize most of the spaces in the table
- We should avoid putting keys too close together. Why?
  - More collisions means even more collisions!

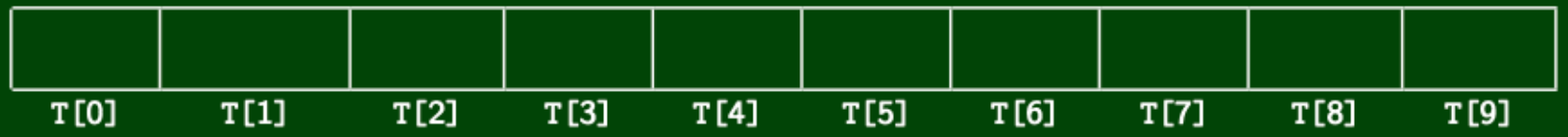
1st attempt: Linear Probing

Strategy #1: Linear Probing

```
1  i = 0;
2  while (index in use) {
3      try (h(key) + i) % |T|
4  }
```

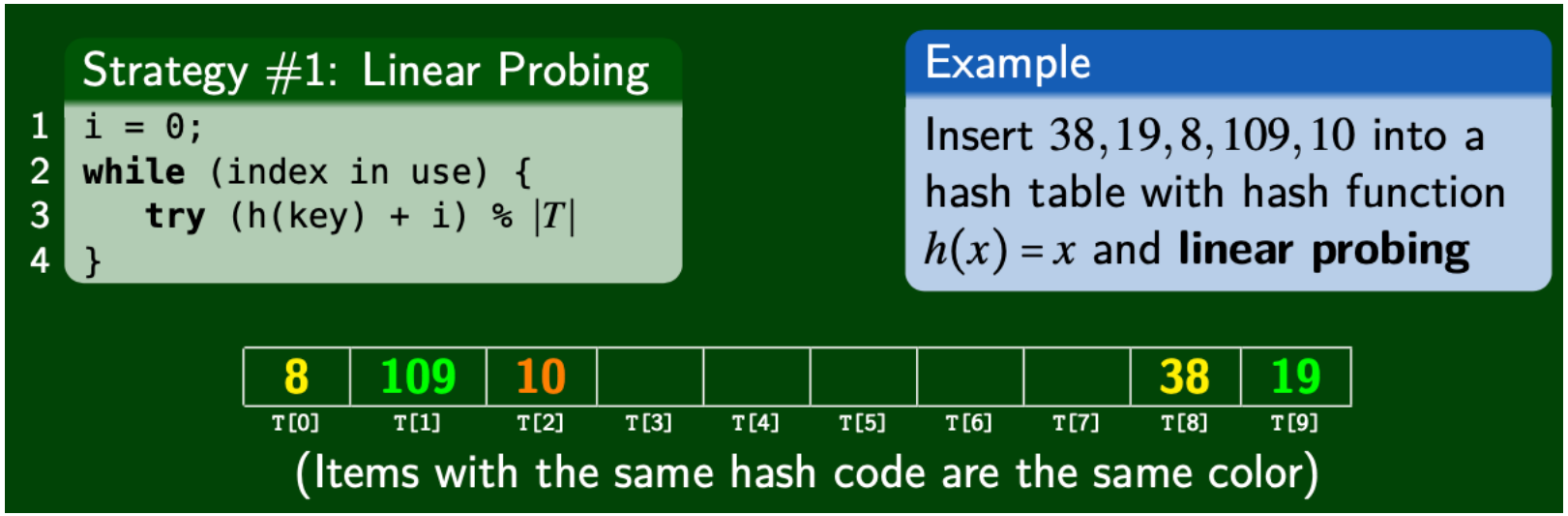
Example

Insert $38, 19, 8, 109, 10$ into a hash table with hash function $h(x) = x$ and **linear probing**



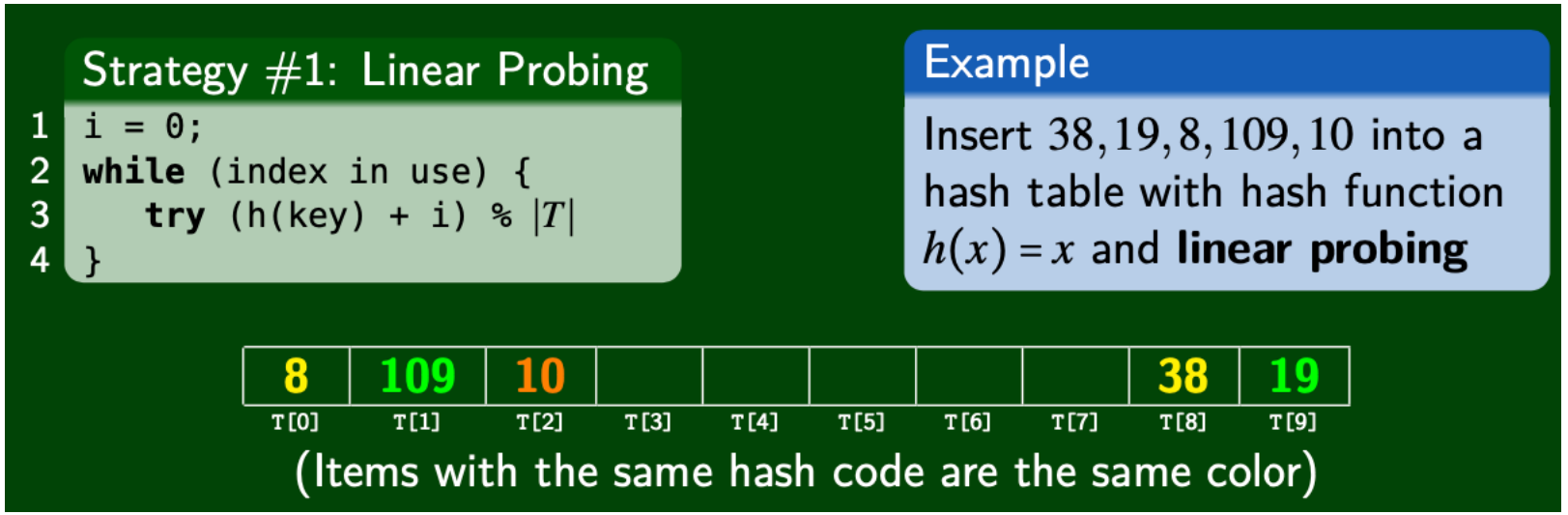
(Items with the same hash code are the same color)

Time complexity



- Insert? What is the worst case of finding the next slot?
- Find?
- Delete?

Time complexity



- Insert? What is the worst case of finding the next slot?
- Find?

Time complexity

Strategy #1: Linear Probing

```
1  i = 0;
2  while (index in use) {
3     try (h(key) + i) % |T|
4  }
```

Example

Insert $38, 19, 8, 109, 10$ into a hash table with hash function $h(x) = x$ and **linear probing**

T[0]	T[1]	T[2]	T[3]	T[4]	T[5]	T[6]	T[7]	T[8]	T[9]
8	109	10						38	19

(Items with the same hash code are the same color)

- Insert? What is the worst case of finding the next slot?
- Find?
- Wait, can you delete? How?
  - What happens when you just delete 19 and try to find 109?

One way: Lazy deletion

- Hint: Each entry now has an active bit
- Think about the modified actions for
  - find(x):
  - insert(x):
  - delete(x):
  - rehash():

- When can you actually (physically) remove the item from the table?

See Example Code: ProbingHashSet.java

- **Instead, the slot at which x is located is marked inactive**

- **Modified actions (Show code!)**
  - Delete: find(x). If found, mark the slot as in active
  - find(x): locate element x. Only if the slot is active then it is found otherwise not found
  - insert(x): Go to hashed location. If collision, do linear search to find an available slot including inactive slot and insert mark active
  - rehash: double the hash table size and only insert active elements from the old table

Let's revisit the criteria and analyze Linear Probing

Thumbs up or down

- We should be able to reproduce the path
- We want to use most of the spaces in the table
- We should avoid putting keys close together

Linear probing leads to primary clustering

Why primary clustering is bad?

- Odd filled + Even empty vs first half filled + second half not filled

Primary clustering is when different keys collide to form one big group

Which would have worse performance? (i.e., more # of probes?)

- Successful find
- Unsuccessful find

High-level intuition: Why unsuccessful search is far worse than successful search?

- In fact, exponentially worse

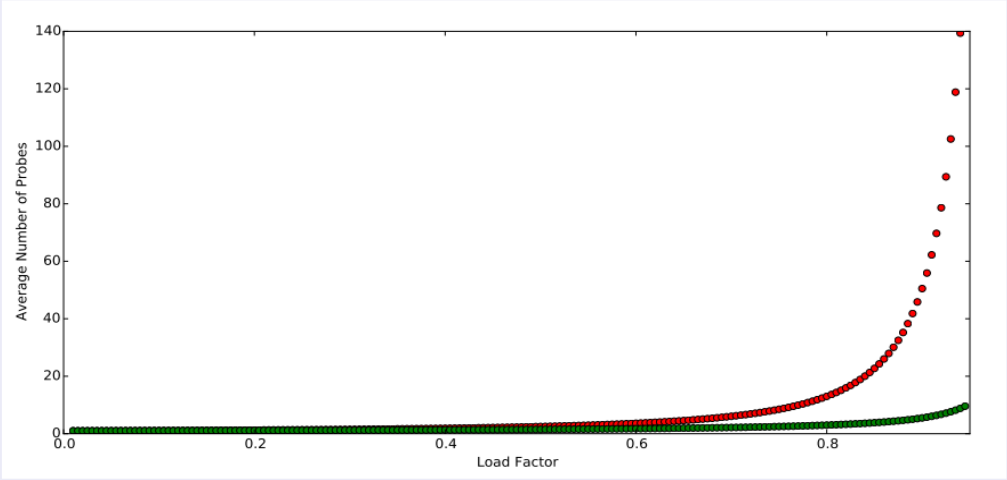
What is a good load factor?

Unsuccessful Search

$$\frac{1}{2}\left(1+\frac{1}{(1-\lambda)^2}\right)$$

Successful Search

$$\frac{1}{2}\left(1+\frac{1}{(1-\lambda)}\right)$$



- **Load factor = 0.5** (table half full)
  - Successful: $0.5 \times (1 + 2) = 1.5$ probes
  - Unsuccessful: $0.5 \times (1 + 4) = 2.5$ probes
- **Load factor = 0.9** (table almost full)
  - Successful: $0.5 \times (1 + 10) = 5.5$ probes
  - Unsuccessful: $0.5 \times (1 + 100) = 50.5$ probes (!!)

About half full!

Mathematically….

- P_s is the average number of probes for successful find

$$P_s = \frac{1}{2}\left(1 + \frac{1}{1-\lambda}\right)$$

Why 1 +

Why 1 / (1 - λ)

Why 1 / 2

Mathematically….

- P_u is the average number of probes for unsuccessful find

$$P_u = \frac{1}{2}\left(1 + \frac{1}{(1-\lambda)^2}\right)$$

Why square in $1 / (1 - \lambda)^2$?

Beyond the scope of this class ☺

Outline

1. Administrative
2. Open addressing – Linear Probing
3. Open addressing – Quadratic Probing

Quadratic probing tries to reduce primary clustering by using a quadratic function

Open Addressing In General

Choose a new function $f(x)$ and then probe with

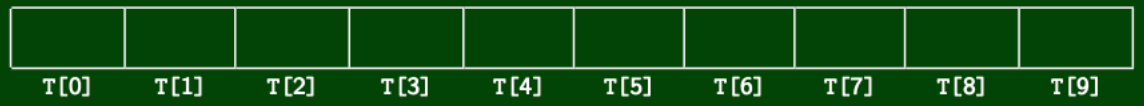
$$(h(\text{key}) + f(i)) \bmod |T|$$

Strategy #2: Quadratic Probing

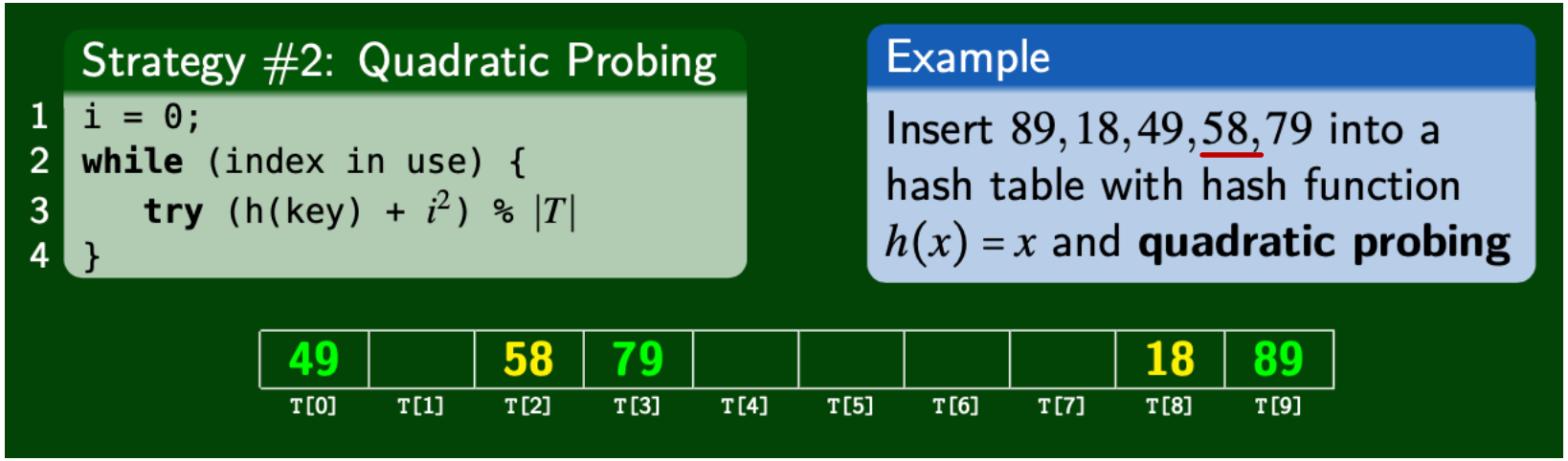
```
1 i = 0;
2 while (index in use) {
3     try (h(key) + i^2) % |T|
4 }
```

Example

Insert $89, 18, 49, 58, 79$ into a hash table with hash function $h(x) = x$ and **quadratic probing**



Using quadratic probing
primary clusters are spread out more effectively



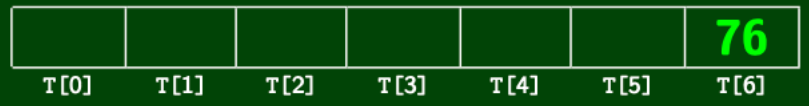
Quadratic probing example

Strategy #2: Quadratic Probing

```
1 i = 0;
2 while (index in use) {
3     try (h(key) + i^2) % |T|
4 }
```

Example

Insert $76, 40, 48, 5, 55, 47$ into a hash table with hash function $h(x) = x$ and **quadratic probing**



$$h(76) \xrightarrow{i=0} 76 + 0^2 \equiv_7 6$$

Can you insert 47?

Strategy #2: Quadratic Probing

```
1 i = 0;
2 while (index in use) {
3     try (h(key) + i^2) % |T|
4 }
```

Example

Insert $76, 40, 48, 5, 55, 47$ into a hash table with hash function $h(x) = x$ and **quadratic probing**

T[0]	T[1]	T[2]	T[3]	T[4]	T[5]	T[6]
48		5	55		40	76

$$h(47) \xrightarrow{i=0} 47 + 0^2 \equiv_7 5$$
$$\xrightarrow{i=1} 47 + 1^2 \equiv_7 6$$
$$\xrightarrow{i=2} 47 + 2^2 \equiv_7 2$$
$$\xrightarrow{i=3} 47 + 3^2 \equiv_7 0$$
$$\xrightarrow{i=4} 47 + 4^2 \equiv_7 0$$
$$\xrightarrow{i=5} 47 + 5^2 \equiv_7 2$$

We will never get a 1 or a 4!

Even with empty slots insertion can fail due to cycle!

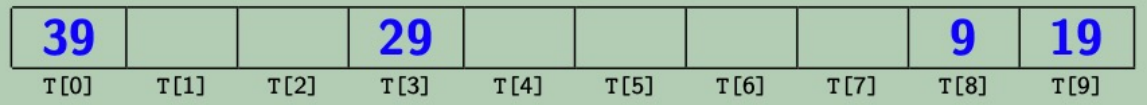
Good news!
If m is a prime number and load factor < 0.5

- Quadratic probing will find an empty slot in at most m/2 probes.
- Proof will be posted in Ed Discussion
  - Not required to do the formal proof in our course
  - But you do for discrete mathematics

Quadratic probing: What could go wrong?

Secondary Clustering

Secondary Clustering is when different keys hash to the same place and follow the same probing sequence.



- Consider keys 9, 19, 39, 29 and their probing sequence
  - 9 -> 0 -> 3 -> 8 -> 5 -> 4

Can we avoid secondary clustering? What is the problem here?

This is the motivation for Double Hashing

Outline

1. Administrative
2. Open addressing – Linear Probing
3. Open addressing – Quadratic Probing
4. Open addressing – Double Hashing

Double hashing idea

- **Each key should "jump" in their own way (each key has a different delta)**
  - Key1 jumps multiples of 3 (+3, +6, +9, etc)
  - Key2 jumps multiples of 11 (+11, +22, +33, etc)

Double hashing example

Strategy #3: Double Hashing

```
1 i = 0;
2 while (index in use) {
3     try (h(key) + i*g(key)) % |T|
4 }
```

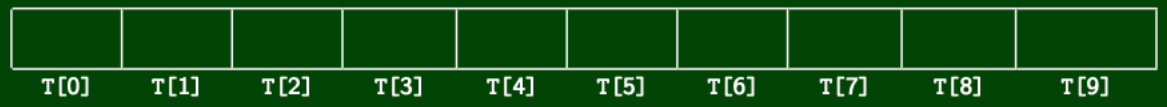
We insist $g(x) \neq 0$.

Example

Insert $13, 28, 33, 147, 43$ into a hash table with:

- $h(x) = x$
- $g(x) = 1 + \left(\frac{x}{|T|}\right) \bmod (|T| - 1)$

using **double hashing**



Can you insert 43?

Strategy #3: Double Hashing

```
1  i = 0;
2  while (index in use) {
3     try (h(key) + i*g(key)) % |T|
4  }
```

We insist $g(x) \neq 0$.

Example

Insert $13, 28, 33, 147, 43$ into a hash table with:

- $h(x) = x$
- $g(x) = 1 + \left(\frac{x}{|T|}\right) \bmod (|T| - 1)$

using **double hashing**

T[0]	T[1]	T[2]	T[3]	T[4]	T[5]	T[6]	T[7]	T[8]	T[9]
			13				33	28	147

$$h(43) \xrightarrow{i=0} 43 + 0 \equiv 3$$

$$\xrightarrow{i=1} 43 + 1(1 + 4 \bmod 9) \equiv 8$$

$$\xrightarrow{i=1} 43 + 2(1 + 4 \bmod 9) \equiv 3$$

$$\xrightarrow{i=1} 43 + 3(1 + 4 \bmod 9) \equiv 8$$

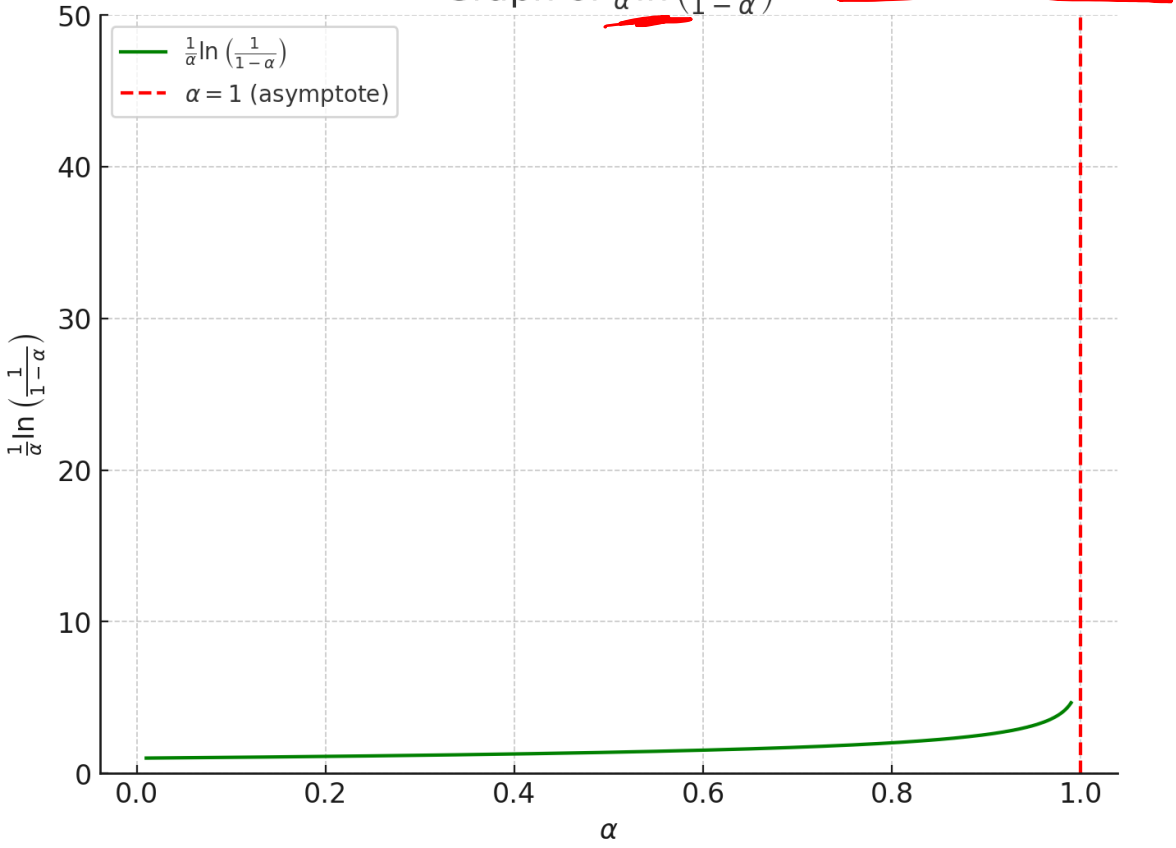
We will not get an infinite loop when…

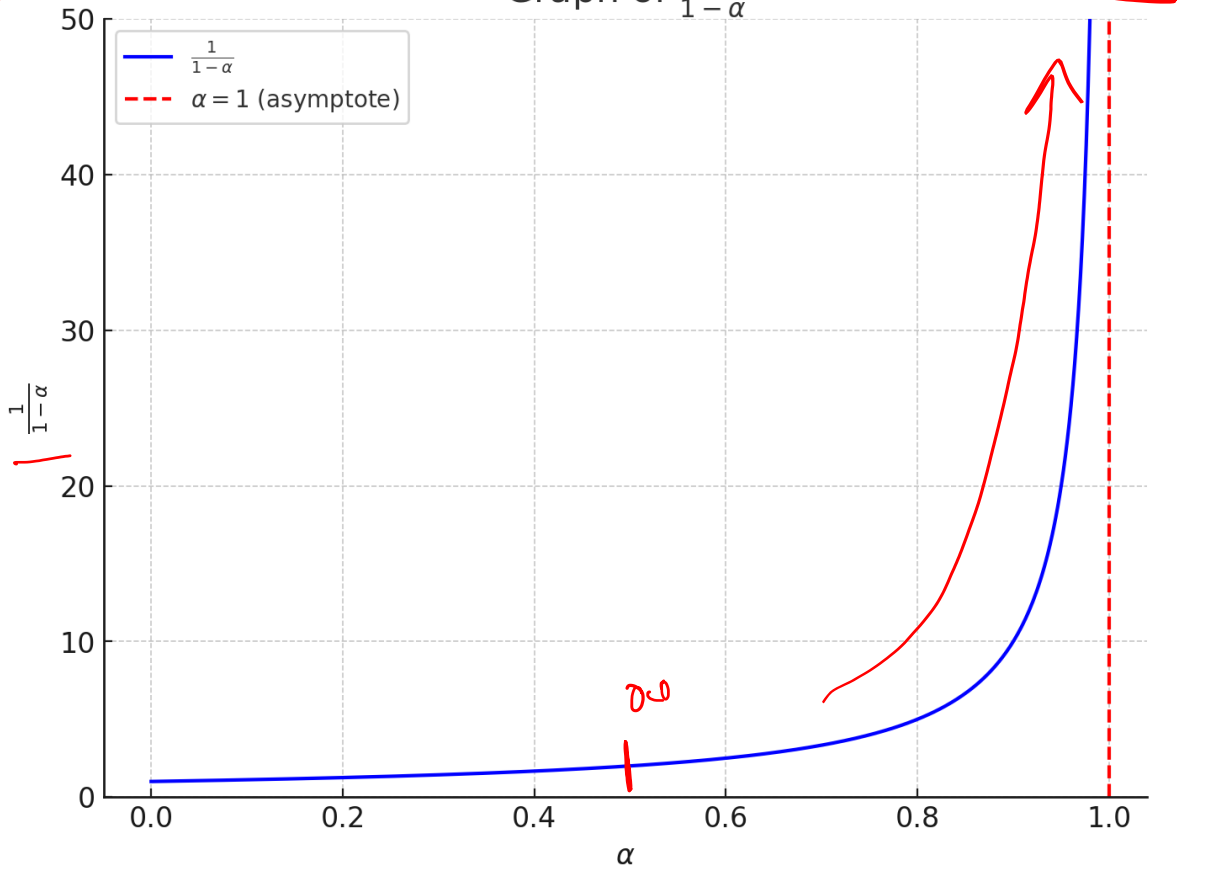
With primes p,q such that 2<q< p:

- h(key) = key mod p
- g(key)=q−(key mod q)

How much better is double hashing than linear probing?

Successful search vs unsuccessful search





Double hashing is much better than linear probing

Outline

1. Administrative
2. Open addressing – Linear Probing
3. Open addressing – Quadratic Probing
4. Open addressing – Double Hashing
5. Mini math lecture

Analysis of linear probing

- P_s is the average number of probes for successful find

$$P_s = \frac{1}{2}\left(1 + \frac{1}{1-\lambda}\right)$$

Why 1 +

Why 1 / (1 - λ)

Why 1 / 2

Probability 101

- What is the expected number of coin toss to see a tail?
- Coin is biased: 70% chance for head, 30% chance for tail

Geometric series…

This series converges only when $|\alpha| < 1$, meaning that for values $0 \leq \alpha < 1$, you can represent $\frac{1}{1-\alpha}$ as:

$$\frac{1}{1-\alpha} = 1 + \alpha + \alpha^2 + \alpha^3 + \dots$$

Outline

1. Administrative
2. Open addressing – Linear Probing
3. Open addressing – Quadratic Probing
4. Open addressing – Double Hashing
5. Mini math lecture
6. Rehashing

Rehashing

- With separate chaining, we decide when to resize (good if $\lambda \leq 1$)
- With open addressing, we need to keep $\lambda < 0.5$
- New table size should be a prime number
  - Roughly twice as big

- What is the time complexity of rehashing?

Outline

1. Administrative
2. Open addressing – Linear Probing
3. Open addressing – Quadratic Probing
4. Open addressing – Double Hashing
5. Mini math lecture
6. Rehashing
7. Other hash functions

Other hash functions

- **Two-probe hashing (separate-chaining variant)**
  - Hash to two positions, insert key in shorter of the two chains.
  - Reduces expected length of the longest chain to ~ lg ln N
- **Cuckoo hashing (linear-probing variant)**
  - Use two hash table (or one longer table) with 2 different hash functions
  - Try to insert to either table
  - If occupied, evict existing entry and try to insert to the other table
  - Guarantees lookup time is O(1)
- **Perfect hashing**
  - Useful for static data set
  - Construct a perfect secondary hash

Backup Slides