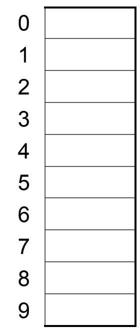
## Collision Resolution

**Collision**: when two keys map to the same location in the hash table.

Two ways to resolve collisions:

1. Separate Chaining
2. Open Addressing (linear probing, quadratic probing, double hashing)

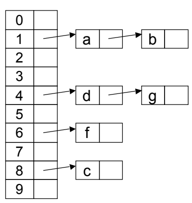
## Separate Chaining




**Insert**:
10
22
107
12
42

- **<u>Separate chaining</u>**: All keys that map to the same hash value are kept in a list (or “bucket”).

## Open Hashing (Chaining)




- h(a) = h(b) and h(d) = h(g)
- Chains may be ordered or unordered. Little advantage to ordering.

## Analysis of find

- Defn: The load factor, $\lambda$, of a hash table is the ratio: $\frac{N}{M}$ $\leftarrow$ no. of elements, $\leftarrow$ table size

For separate chaining, $\lambda$ = average # of elements in a bucket

- Unsuccessful find cost:
- Successful find cost:

## How big should the hash table be?

- For Separate Chaining:

## Closed Hashing (Open Addressing)

- No chaining, every key fits in the hash table.
- Probe sequence
  - h(k)
  - (h(k) + f(1)) mod HSize
  - (h(k) + f(2)) mod HSize , …
- Insertion: Find the first probe with an empty slot.
- Find: Find the first probe that equals the query or is empty. Stop at HSize probe, in any case.
- Deletion: lazy deletion is needed. That is, mark locations as deleted, if a deleted key resides there.

## Open Addressing

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

**Insert**:
38
19
8
109
10

- **Linear Probing**: after checking spot h(k), try spot h(k)+1, if that is full, try h(k)+2, then h(k)+3, etc.

## Linear Probing

f(i) = i

- Probe sequence:
  - 0th probe = h(k) mod TableSize
  - 1th probe = (h(k) + 1) mod TableSize
  - 2th probe = (h(k) + 2) mod TableSize
  - . . .
  - ith probe = (h(k) + i) mod TableSize

## Terminology Alert!

"**Open** Hashing" equals "Separate Chaining"

"Closed Hashing" equals "**Open** Addressing"

Weiss

## Linear Probing Example

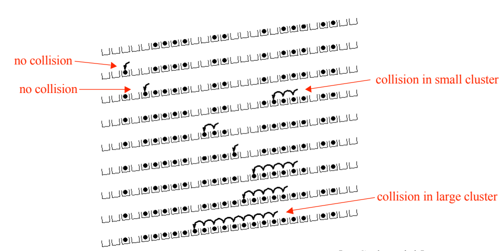
| | 76 | 93 | 40 | 47 | 10 | 55 |
|---|---|---|---|---|---|---|
| 0 | | | | 47 | 47 | 47 |
| 1 | | | | | | 55 |
| 2 | | 93 | 93 | 93 | 93 | 93 |
| 3 | | | | | 10 | 10 |
| 4 | | | | | | |
| 5 | | | 40 | 40 | 40 | 40 |
| 6 | 76 | 76 | 76 | 76 | 76 | 76 |
| Probes | 1 | 1 | 1 | 3 | 1 | 3 |

## Write pseudocode for find(k) for Open Addressing with linear probing

- Find(k) returns i where T(i) = k

Student Activity

## Linear Probing – Clustering




[R. Sedgewick]

## Load Factor in Linear Probing

- For *any* $\lambda < 1$, linear probing *will* find an empty slot
- Expected # of probes (for large table sizes)
  - successful search:

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

  - unsuccessful search:

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

- Linear probing suffers from ***primary clustering***
- Performance quickly degrades for $\lambda > 1/2$

## Quadratic Probing

Less likely to encounter Primary Clustering

$f(i) = i^2$

- Probe sequence:

  $0^{th}$ probe = h(k) mod TableSize

  $1^{th}$ probe = (h(k) + 1) mod TableSize

  $2^{th}$ probe = (h(k) + 4) mod TableSize

  $3^{th}$ probe = (h(k) + 9) mod TableSize

  . . .

  $i^{th}$ probe = (h(k) + $i^2$) mod TableSize

## Exercise: Quadratic Probing

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

Insert:
89
18
49
58
79

## Quadratic Probing Example

insert(76) 76%7 = 6

insert(40) 40%7 = 5

insert(48) 48%7 = 6

insert(5) 5%7 = 5

insert(55) 55%7 = 6

But… insert(47) 47%7 = 5

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | 76 |

## Quadratic Probing: Success guarantee for $\lambda < 1/2$

- If size is prime and $\lambda < 1/2$, then quadratic probing will find an empty slot in size/2 probes or fewer.
  - show for all `0 ≤ i,j < size/2` and `i ≠ j`

    `(h(x) + i²) mod size ≠ (h(x) + j²) mod size`
  - by contradiction: suppose that for some i ≠ j:

```
(h(x) + i²) mod size = (h(x) + j²) mod size
⇒ i² mod size = j² mod size
⇒ (i² - j²) mod size = 0
⇒ [(i + j)(i - j)] mod size = 0
```

BUT size does not divide `(i-j)` or `(i+j)`

## Quadratic Probing may Fail if $\lambda \geq 1/2$

51

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 16 |
| 3 | 45 |
| 4 | 59 |
| 5 | |
| 6 | 76 |

51 mod 7 = 2 ; i = 0

(2 + 1) mod 7 = 3; i = 1

(3 + 3) mod 7 = 6; i = 2

(6 + 5) mod 7 = 4; i = 3

(4 + 7) mod 7 = 4; i = 4

(4 + 9) mod 7 = 6; i = 5

(6 + 11) mod 7 = 3; i = 6

(3 + 13) mod 7 = 2, i = 7

…

# Quadratic Probing: Properties

- For *any* $\lambda$ < 1/2, quadratic probing will find an empty slot; for bigger $\lambda$, quadratic probing *may* find a slot
- Quadratic probing does not suffer from *primary* clustering: keys hashing to the same *area* are not bad
- But what about keys that hash to the same *spot*?
  - ***Secondary Clustering!***

# Double Hashing

f(i) = i * g(k)
where g is a second hash function

- Probe sequence:
  - $0^{th}$ probe = h(k) mod TableSize
  - $1^{th}$ probe = (h(k) + g(k)) mod TableSize
  - $2^{th}$ probe = (h(k) + 2*g(k)) mod TableSize
  - $3^{th}$ probe = (h(k) + 3*g(k)) mod TableSize
  - . . .
  - $i^{th}$ probe = (h(k) + i*g(k)) mod TableSize

# Double Hashing Example

h(k) = k mod 7 and g(k) = 5 – (k mod 5)

| | 76 | 93 | 40 | 47 | 10 | 55 |
|---|---|---|---|---|---|---|
| 0 | | | | | | |
| 1 | | | | 47 | 47 | 47 |
| 2 | | 93 | 93 | 93 | 93 | 93 |
| 3 | | | | | 10 | 10 |
| 4 | | | | | | 55 |
| 5 | | | 40 | 40 | 40 | 40 |
| 6 | 76 | 76 | 76 | 76 | 76 | 76 |
| Probes | 1 | 1 | 1 | 2 | 1 | 2 |

# Resolving Collisions with Double Hashing

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |
| 7 | |
| 8 | |
| 9 | |

<u>Hash Functions</u>:
H(K) = K mod M
$H_2$(K) = 1 + ((K/M) mod (M-1))
M =

**Insert these values into the hash table in this order. Resolve any collisions with double hashing:**
13
28
33
147
43

# Double Hashing is Safe for $\lambda$ < 1

- Let h(k) = k mod p and g(k) = q – (k mod q) where 2 < q < p and p and q are primes. The probe sequence h(k) + ig(k) mod p probes every entry of the hash table.

  Let 0 $\leq$ m < p, h = h(k), and g = g(k). We show that h+ig mod p = m for some i. 0 < g < p, so g and p are relatively prime. By extended Euclid's algorithm that are s and t such that
  sg + tp = 1. Choose i = (m-h)s mod p
  (h + ig) mod p =
  (h + (m-h)sg) mod p =
  (h + (m-h)sg + (m-h)tp) mod p =
  (h + (m-h)(sg + tp) mod p =
  (h + (m-h)) mod p = m mod p = m

# Deletion in Hashing

- Open hashing (chaining) – no problem
- Closed hashing – must do lazy deletion. Deleted keys are marked as deleted.
  - Find: done normally
  - Insert: treat marked slot as an empty slot and fill it.

h(k) = k mod 7
Linear probing

Find 59

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 16 |
| 3 | 23 |
| 4 | 59 |
| 5 | |
| 6 | 76 |

Insert 30

| | |
|---|---|
| 0 | |
| 1 | |
| 2 | 16 |
| 3 | 30 |
| 4 | 59 |
| 5 | |
| 6 | 76 |

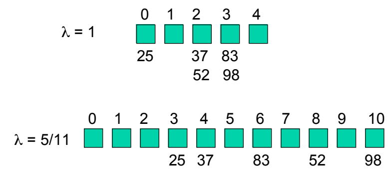
## Rehashing

**Idea:** When the table gets too full, create a bigger table (usually 2x as large) and hash all the items from the original table into the new table.

- When to rehash?
  - › half full ($\lambda$ = 0.5)
  - › when an insertion fails
  - › some other threshold
- Cost of rehashing?

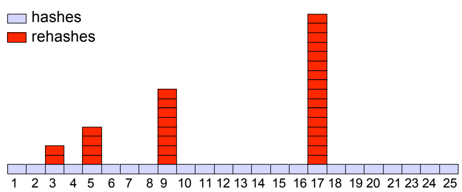
## Rehashing Example

- Open hashing – $h_1(x)$ = x mod 5 rehashes to $h_2(x)$ = x mod 11.




## Rehashing Picture

- Starting with table of size 2, double when load factor > 1.




## Amortized Analysis of Rehashing

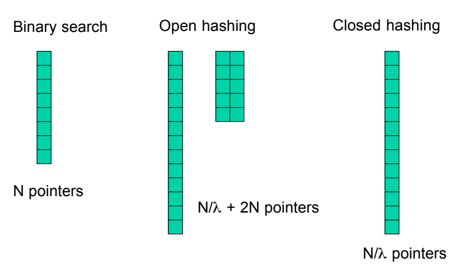
- Cost of inserting n keys is < 3n
- $2^k + 1 \leq n \leq 2^{k+1}$
  - › Hashes = n
  - › Rehashes = $2 + 2^2 + \ldots + 2^k = 2^{k+1} - 2$
  - › Total = $n + 2^{k+1} - 2 < 3n$
- Example
  - › n = 33, Total = 33 + 64 –2 = 95 < 99

## Case Study

- Spelling Dictionary - 30,000 words
- Goals
  - › Fast spell checking
  - › Minimal storage
- Possible solutions
  - › Sorted array and binary search
  - › Open hashing (chaining)
  - › Closed hashing with linear probing
- Notes
  - › Almost all searches are successful
  - › 30,000 word average 8 bytes per word, 240,000 bytes
  - › Pointers are 4 bytes

## Storage

- Assume word are stored as strings and entries in the arrays are pointers to the strings.

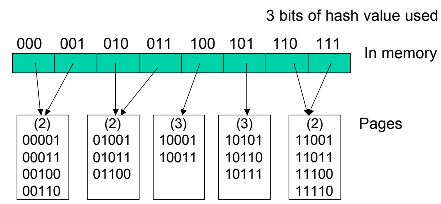


# Analysis

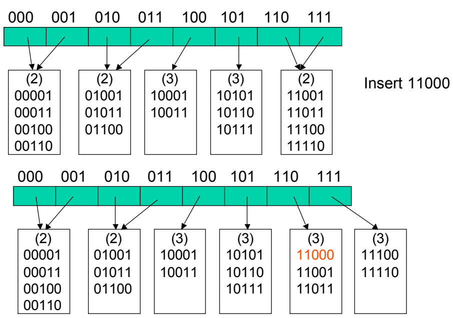
- Binary Search
  - Storage = N pointers + words = 360,000 bytes
  - Time = $\log_2 N \leq 15$ probes in worst case
- Open hashing
  - Storage = 2N + N/ λ pointers + words
    λ = 1 implies 600,000 bytes
  - Time = 1 + λ/2 probes per access
    λ = 1 implies 1.5 probes per access
- Closed hashing
  - Storage = N/ λ pointers + words
    λ = 1/2 implies 480,000 bytes
  - Time = (1/2)(1+1/(1-λ)) probes
    λ = 1/2 implies 1.5 probes per access

# Extendible Hashing

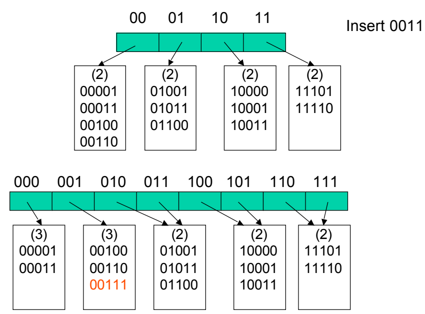
- Extendible hashing is a technique for storing large data sets that do not fit in memory.
- An alternative to B-trees




# Splitting




# Rehashing




# Fingerprints

- Given a string x we want a fingerprint x' with the properties.
  - x' is short, say 128 bits
  - Given x ≠ y the probability that x' = y' is infintesimal (almost zero)
  - Computing x' is very fast
- MD5 - Message Digest Algorithm 5 is a recognized standard
- Applications in databases and cryptography

# Fingerprint Math

Given 128 bits and N strings what is the probability that the fingerprints of two strings coincide?

$$1-\frac{2^{128}(2^{128}-1)\cdots(2^{128}-N+1)}{(2^{128})^N}$$

This is essentially zero for $N < 2^{40}$.

## Hashing Summary

- Hashing is one of the most important data structures.
- Hashing has many applications where operations are limited to find, insert, and delete.
- Dynamic hash tables have good amortized complexity.