

CSE 326 Data Structures

Dave Bacon

Hashing

Logistics

- Project 26 due at 11:59pm.
 - Turn in everything.
 - Max 20 → use zip (no rar)
office 4-5 David 002
- Homework 4 due Friday
- Midterm in section Thur.
given back

Implementations So Far

Skipped



	Unsorted linked list	Sorted Array	BST	AVL	Splay (amortized)
Insert	$O(1)$	$O(N)$	$O(N)$	$O(\log N)$	$O(\log N)$
Find	$O(N)$	$O(\log N)$	$O(N)$	$O(\log N)$	$O(\log N)$
Delete	$O(N)$	$O(N)$	$O(N)$	$O(\log N)$	$O(\log N)$

Worst Case

Hash Tables

- Constant time accesses!
- A **hash table** is an array of some 0 fixed size, usually a prime number.
- General idea:

hash table



hash function:

$h(K)$



key space (e.g., integers, strings)

TableSize - 1

Example

- key space = integers

positive

- TableSize = 10

- $h(K) = K \bmod 10$

- Insert: 7, 18, 41, 94

$$7 \bmod 10 = 7$$

$$18 \bmod 10 = 8$$

$$41 \bmod 10 = 1$$

insert 38?

$$38 \bmod 10$$

= 8

Find (28) → 8

0	
1	41
2	
3	
4	94
5	
6	
7	7
8	18
9	

Another Example

- key space = integers
- TableSize = 6
- $h(K) = K \bmod 6$
- Insert: 7, 18, 41, 34

0	18
1	7
2	
3	
4	34
5	41

$$7 \bmod 6 = 1$$

$$18 \bmod 6 = 0$$

$$41 \bmod 6 = 5$$

$$34 \bmod 6 = 4$$

What do we want
from Hash function

Hash Functions

1. simple/fast to compute, hash function
2. Avoid collisions
3. have keys distributed evenly among cells.

evenly uniform



Perfect Hash function:

No collisions
No blank locations.

"Ignorant"
about
data



26 letters 10 digits + 1 "space" 37

Sample Hash Functions:



• key space = strings length K

different combinations

• $s = s_0 s_1 s_2 \dots s_{k-1}$

s_0
 s_1

Table Size = 10^6

1. $h(s) = s_0 \bmod \text{TableSize}$

Spread: 37 0, ..., 36

DAVE \rightarrow 4 collisions
DUMB

2. $h(s) = \left(\sum_{i=0}^{k-1} s_i \right) \bmod \text{TableSize}$ spread

SPOT, POST, h \rightarrow X
STOP \rightarrow X
37K

3. $h(s) = \left(\sum_{i=0}^{k-1} s_i \cdot 37^i \right) \bmod \text{TableSize}$

Herner's

mod TableSize

spread: 37K

s_0 s_1 s_2 ...



$s_0 + s_1 \cdot 37 + s_2 \cdot 37^2 + \dots$

anagram problem is gone.

Designing a Hash Function for web URLs *www.hash.com*

$$S = \underline{s_0 s_1 s_2 \dots s_{k-1}}$$

Issues to take into account:

→ huge urls.

→ "www" - "com" - "edu"

→ "." "?"

simple + fast ↑ as much as possible

$h(s) =$

spread large
common keys → same location

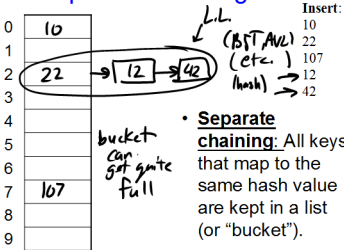
Collision Resolution

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



Analysis of find

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

For separate chaining, $\lambda = \underline{\text{average \# of elements in a bucket}}$



- unsuccessful: $\lambda \leftarrow \text{"Avg"}$
- successful: LL , not sorted
 $1 + \frac{\lambda}{2}$

How big should the hash table be?

- For Separate Chaining:

$$\left[\begin{array}{l} \lambda = 1 \leftarrow \text{want} \\ \lambda = \frac{1}{2} \text{ wasting space.} \end{array} \right]$$

tableSize: Why Prime? ←

= prime number

- Suppose

- data stored in hash table: 7160, 493, 60, 55, 321, 900, 810

- tableSize = 10

data hashes to 0, 3, 0, 5, 1, 0, 0

- tableSize = 11

data hashes to 10, 9, 5, 0, 2, 9, 7

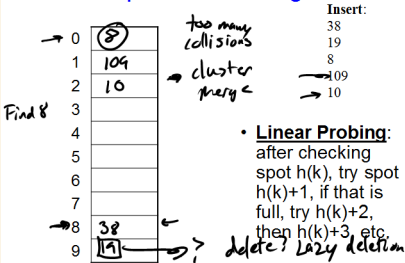


↑
collision

Real-life data tends to have a pattern

Being a multiple of 11 is usually *not* the pattern ☺

Open Addressing



Terminology Alert!



“Open Hashing”

equals

“Separate Chaining”

Weiss

Weiss ↗



“Closed Hashing”

equals

“Open Addressing”



$$\underline{h(k) + f(i)}$$

\uparrow
one f

Linear Probing

$$\underline{f(i) = i}$$

- Probe sequence:

$$0^{\text{th}} \text{ probe} = \underline{h(k) \bmod \text{TableSize}}$$

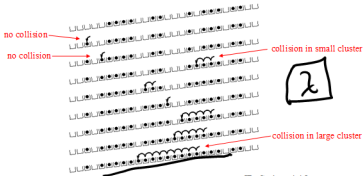
$$1^{\text{th}} \text{ probe} = \underline{(h(k) + 1) \bmod \text{TableSize}}$$

$$2^{\text{th}} \text{ probe} = \underline{(h(k) + 2) \bmod \text{TableSize}}$$

...

$$i^{\text{th}} \text{ probe} = (h(k) + i) \bmod \text{TableSize}$$

Linear Probing – Clustering



[R. Sedgwick]

When are we OK?

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:

$$0^{\text{th}} \text{ probe} = h(k) \bmod \text{TableSize}$$

$$1^{\text{th}} \text{ probe} = (h(k) + 1) \bmod \text{TableSize}$$

$$2^{\text{th}} \text{ probe} = (h(k) + 4) \bmod \text{TableSize}$$

$$3^{\text{th}} \text{ probe} = (h(k) + 9) \bmod \text{TableSize}$$

...

$$i^{\text{th}} \text{ probe} = (h(k) + i^2) \bmod \text{TableSize}$$

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)

$$55 \% 7 = 6$$

0	
1	
2	
3	
4	
5	
6	76

But... insert(47)
 $47 \% 7 = 5$

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 $0 \leq i, j \leq \text{size}/2$ and $i \neq j$
$$(h(x) + i^2) \bmod \text{size} \neq (h(x) + j^2) \bmod \text{size}$$
 - by contradiction: suppose that for some $i \neq j$:
$$(h(x) + i^2) \bmod \text{size} = (h(x) + j^2) \bmod \text{size}$$
$$\Rightarrow i^2 \bmod \text{size} = j^2 \bmod \text{size}$$
$$\Rightarrow (i^2 - j^2) \bmod \text{size} = 0$$
$$\Rightarrow [(i + j)(i - j)] \bmod \text{size} = 0$$
- BUT size does not divide $(i-j)$ or $(i+j)$

Quadratic Probing: Properties

- For *any* $\lambda < \frac{1}{2}$, quadratic probing will find an empty slot; for bigger λ , 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:

$$0^{\text{th}} \text{ probe} = h(k) \bmod \text{TableSize}$$

$$1^{\text{th}} \text{ probe} = (h(k) + g(k)) \bmod \text{TableSize}$$

$$2^{\text{th}} \text{ probe} = (h(k) + 2 * g(k)) \bmod \text{TableSize}$$

$$3^{\text{th}} \text{ probe} = (h(k) + 3 * g(k)) \bmod \text{TableSize}$$

...

$$i^{\text{th}} \text{ probe} = (h(k) + i * g(k)) \bmod \text{TableSize}$$

Double Hashing Example

$$h(k) = k \bmod 7 \text{ and } g(k) = 5 - (k \bmod 5)$$

76

93

40

47

10

55

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

Resolving Collisions with Double Hashing

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Hash Functions:

$$H(K) = K \bmod M$$

$$H_2(K) = 1 + ((K/M) \bmod (M-1))$$

$$M =$$

Insert these values into the hash table in this order. Resolve any collisions with double hashing:

13

28

33

147

43

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?

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.

