

# Hashing

Chapter 5 in Weiss

CSE 373  
Data Structures and Algorithms  
Ruth Anderson

10/28/2011

1

## Today's Outline

- **Announcements**
  - **Homework #4 coming:**
    - Java programming: disjoint sets and mazes
    - due Thurs, Nov 10<sup>th</sup>
    - partners allowed
- **Today's Topics:**
  - Disjoint Sets & Dynamic Equivalence
  - Hashing

10/28/2011

2

## The Dictionary ADT

- **Data:**
  - a set of (key, value) pairs
- **Operations:**
  - Insert (key, value)
  - Find (key)
  - Remove (key)



The Dictionary ADT is sometimes called the "Map ADT"

10/28/2011

3

## Dictionary Implementations

For dictionary with  $n$  key/value pairs

	insert	find	delete
• Unsorted linked-list	$O(1) *$	$O(n)$	$O(n)$
• Unsorted array	$O(1) *$	$O(n)$	$O(n)$
• Sorted linked list	$O(n)$	$O(n)$	$O(n)$
• Sorted array	$O(n)$	$O(\log n)$	$O(n)$
• BST			
• AVL Tree			

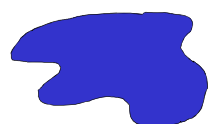
10/28/2011

\*Note: If we do not allow duplicates values to be inserted, we would need to do  $O(n)$  work (a find operation) to check for a key's existence before insertion

4

## Hash Tables

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



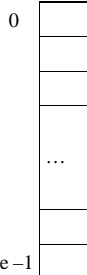
key space (e.g., integers, strings)

hash function:  
 $h(K)$



TableSize - 1

hash table



10/28/2011

5

## Hash Tables

Key space of size  $M$ , but we only want to store subset of size  $N$ , where  $N \ll M$ .

- Keys are identifiers in programs. Compiler keeps track of them in a symbol table.
- Keys are student names. We want to look up student records quickly by name.
- Keys are chess configurations in a chess playing program.
- Keys are URLs in a database of web pages.

10/28/2011

6

## Example

- key space = integers
- TableSize = 10
- $h(K) = K \bmod 10$
- **Insert:** 7, 18, 41, 94

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

10/28/2011

7

## Another Example

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

0	
1	
2	
3	
4	
5	

10/28/2011

8

Student Activity

## Hash Functions

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

Perfect Hash function:

10/28/2011

9

## Sample Hash Functions:

- key space = strings
  - $s = s_0 s_1 s_2 \dots s_{k-1}$
1.  $h(s) = s_0 \bmod \text{TableSize}$
  2.  $h(s) = \left( \sum_{i=0}^{k-1} s_i \right) \bmod \text{TableSize}$
  3.  $h(s) = \left( \sum_{i=0}^{k-1} s_i \cdot 37^i \right) \bmod \text{TableSize}$

10/28/2011

10

## Designing a Hash Function for web URLs

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

Issues to take into account:

$h(s) =$

Student Activity

11

## 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)

10/28/2011

12

## Separate Chaining

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Insert:

10  
22  
107  
12  
42

- **Separate chaining:** All keys that map to the same hash value are kept in a list ("bucket").

10/28/2011

13

## Analysis of find

- The **load factor**,  $\lambda$ , of a hash table is the ratio:

$$\frac{N}{M} \quad \leftarrow \text{no. of elements}$$

$$\quad \quad \quad \leftarrow \text{table size}$$

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

- unsuccessful:
- successful:

10/28/2011

14

## How big should the hash table be?

- For Separate Chaining:

10/28/2011

15

## tableSize: Why Prime?

- 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

Real-life data tends to have a pattern

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

10/28/2011

16

## 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.

10/28/2011

17

## Terminology Alert!

“Open Hashing”

equals

“Separate Chaining”

“Closed Hashing”

equals

“Open Addressing”

Weiss

10/28/2011

18

## Linear Probing

$$f(i) = i$$

- Probe sequence:
  - 0<sup>th</sup> probe =  $h(k) \bmod \text{TableSize}$
  - 1<sup>th</sup> probe =  $(h(k) + 1) \bmod \text{TableSize}$
  - 2<sup>th</sup> probe =  $(h(k) + 2) \bmod \text{TableSize}$
  - ...
  - i<sup>th</sup> probe =  $(h(k) + i) \bmod \text{TableSize}$

10/28/2011

19

## Linear Probing – Clustering



[R. Sedgwick]

10/28/2011

20

## 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  $\lambda > 1/2$

10/28/2011

21

## Quadratic Probing

$$f(i) = i^2$$

Less likely to encounter Primary Clustering

- Probe sequence:
  - 0<sup>th</sup> probe =  $h(k) \bmod \text{TableSize}$
  - 1<sup>th</sup> probe =  $(h(k) + 1) \bmod \text{TableSize}$
  - 2<sup>th</sup> probe =  $(h(k) + 4) \bmod \text{TableSize}$
  - 3<sup>th</sup> probe =  $(h(k) + 9) \bmod \text{TableSize}$
  - ...
  - i<sup>th</sup> probe =  $(h(k) + i^2) \bmod \text{TableSize}$

10/28/2011

22

## Quadratic Probing

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

Insert:  
89  
18  
49  
58  
79

10/28/2011

23

## Quadratic Probing:

- $h(k) = k \bmod 7$
- Perform these inserts:
  - Insert(65)
  - Insert(10)
  - Insert(47)

0	
1	
2	93
3	
4	
5	40
6	76

10/28/2011

24

## Quadratic Probing Example

insert(76)    insert(40)    insert(48)    insert(5)    insert(55)  
 $76\%7 = 6$      $40\%7 = 5$      $48\%7 = 6$      $5\%7 = 5$      $55\%7 = 6$

0	
1	
2	
3	
4	
5	
6	76

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

10/28/2011

25

## 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  $(i + j)$

10/28/2011

26

## 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!*

10/28/2011

27

## Double Hashing

$$f(i) = i * g(k)$$

where g is a second hash function

- Probe sequence:
  - 0<sup>th</sup> probe =  $h(k) \bmod \text{TableSize}$
  - 1<sup>th</sup> probe =  $(h(k) + g(k)) \bmod \text{TableSize}$
  - 2<sup>th</sup> probe =  $(h(k) + 2 * g(k)) \bmod \text{TableSize}$
  - 3<sup>th</sup> probe =  $(h(k) + 3 * g(k)) \bmod \text{TableSize}$
  - ...
  - i<sup>th</sup> probe =  $(h(k) + i * g(k)) \bmod \text{TableSize}$

10/28/2011

28

## Double Hashing Example

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

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

	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

10/28/2011

29

## 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

10/28/2011

30

## 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?

10/28/2011

31

## 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.

10/28/2011

32