Sets and Hashing

CSC212—Fall 2014

Dominique Thiébaut
dthiebaut@smith.edu
The Problem:

- Large amount of live tweets
- Want the list of all tweeters
- Each only listed once
What data structure?

Assumption: all tweeters have a unique Id number (integer)
Option 1: BST
Complexity?
O(1)?
Yes! Hashing!
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>40100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D. Thiebaut, Computer Science, Smith College</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>23445</th>
<th>100</th>
<th>5</th>
<th>1</th>
<th>20000</th>
<th>50000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>40100</td>
<td></td>
<td></td>
<td>49000</td>
<td>60000</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>50000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>50000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>40100</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>49000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>60000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>20000</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

23445
100
5
<table>
<thead>
<tr>
<th>0</th>
<th>50000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>40100</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>49000</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>60000</td>
</tr>
<tr>
<td>9</td>
<td>20000</td>
</tr>
<tr>
<td></td>
<td>50000</td>
</tr>
<tr>
<td>---</td>
<td>-------</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>40100</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>49000</td>
</tr>
<tr>
<td>5</td>
<td>23445</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>60000</td>
</tr>
<tr>
<td>8</td>
<td>20000</td>
</tr>
</tbody>
</table>
Have we seen 100 yet?

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>50000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>40100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>100</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>49000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>23445</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>60000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>20000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

D. Thiebaut, Computer Science, Smith College
Have we seen 100 yet?
Have we seen 100 yet?
Have we seen 100 yet?

<table>
<thead>
<tr>
<th></th>
<th>50000</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>40100</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>49000</td>
</tr>
<tr>
<td>5</td>
<td>23445</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>60000</td>
</tr>
<tr>
<td>8</td>
<td>20000</td>
</tr>
</tbody>
</table>
YES!

O(1)

50000
40100
49000
23445
60000
20000
How?
What Properties?
How: a hash function

```java
int[] table = new int[ DIM ];

int index = hash( key );
table[ index ] = key;
```
Hash Properties

• Fast
• Avoid clustering
• Avoid collisions
• Must work with all types of objects
• Fast

• Avoid clustering

• Avoid collisions

• Must work with all types of objects

arithmetic operations, integers
• Fast
• Avoid clustering
• Avoid collisions
• Must work with all types of objects
• Fast
• Avoid clustering
• Avoid collisions
• Must work with all types of objects
• Fast

• Avoid clustering

• Avoid collisions

• Must work with all types of objects

Don't want two different keys with same hash index
• Fast
• Avoid clustering
• Avoid collisions

• Must work with all types of objects

Fortunately, everything in memory is a number!
Fast

Avoid clustering

Avoid collisions

Must work with all types of objects

"Alpha" = 'A' 'l' 'p' 'h' 'a' = 65 108 112 104 97

65 + 108 + 112 + 104 + 97 = 486

Fortunately, everything in memory is a number!
Implementation
Division/Modulo

- index = key % Dim

Example:
- hash 10, 90, 40, 70...
- change Dim to 13
  - 10%13=10, 90%13=12
  - 40%13=1, 70%13=5

If Dim is not prime, then take modulo by a prime first:

index = (key % p) % Dim
Division/Modulo

- index = key % Dim

\[
\begin{align*}
2012 \mod 10 & \rightarrow 2 \\
49 \mod 10 & \rightarrow 9 \\
\end{align*}
\]

Dim = 10
Division/Modulo

- `index = key % Dim`

2012%10 → 2
49%10 → 9
35691%10 → 1

Dim = 10
Division/Modulo

- index = key % Dim
- Works best if Dim is prime

Example:
- hash 10, 90, 40, 70 when Dim=10...
- change Dim to 13
- 10%13=10, 90%13=12
  40%13=1, 70%13=5
- If Dim is not prime, then take modulo by a prime first:
  index = (key % p) % Dim
Folding-Based Hashing
Shift-Folding, Boundary Folding

• **Shift Folding**
  
  • **key** = 123 45 6789 —> 123 | 456 | 789
  —> 123 + 456 + 789 = 1368 % Dim = index

• **Boundary Folding**
  
  • **key** = 123 45 6789 —> 123 | 654 | 789
  —> 123 + 654 + 789 = 1566 % Dim = index
Extraction Hashing

- Extract some digits from the key

  - \textbf{key} = 123 45 6789 \rightarrow 12 | 34567 | 89
  \rightarrow 1289 \% \text{Dim} = \text{index}
Avoiding Collisions 1
Open Addressing

```
Dim = 10
```

```
0
1  34
2
3  106
4  14
5
6
7
8
9
```
Open Addressing

key = 44
hash( 44 ) → 7

Dim = 10
Open Addressing

key = 44
hash( 44 ) → 7

Dim = 10

```
|   0  |
|   1  | 34 |
|   2  |
|   3  | 106|
|   4  | 14 |
|   5  |
|   6  |
|   7  | 44 |
|   8  |
|   9  |
```
Open Addressing

key = 1034
hash(1034) -> 1

Dim = 10
Open Addressing

key = 1034
hash(1034) \rightarrow 1

Dim = 10

0
1
2
3
4
5
6
7
8
9

34
106
14
44
Open Addressing

key = 1034
hash( 1034 ) \rightarrow 1
Open Addressing

key = 3
hash( 3 ) → 2

Dim = 10
Open Addressing

key = 3
hash( 3 ) -> 2
Open Addressing

key = 3
hash(3) → 2

Dim = 10
Open Addressing

key = 3
hash(3) → 2
Open Addressing

key = 3
hash(3) → 2

Dim = 10

linear probing
Open Addressing

key = 3
hash(3) → 2

What if we want to delete keys?

Dim = 10

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th>34</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1034</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>106</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>44</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Avoiding Collisions 2
Chaining

key = 34
hash(key) = 4

Dim = 10
Chaining

key = 34
hash(key) = 4
Chaining

key = 10034
hash(key) = 1
Chaining

key = 10034
hash(key)=1

Dim = 10
Chaining

key = 99
hash(key) = 1

Dim = 10
Chaining

key = 99
hash(key) = 1

Dim = 10

10034 → 99
34 →
Complexity
Open Addressing: It depends on the **Load Factor**

\[
LF = \frac{6}{9} = 66.6\%
\]

<table>
<thead>
<tr>
<th># probes</th>
<th>Linear Probing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Successful Search (LF=50%)</strong></td>
<td>(\frac{1}{2}(1+1/(1-LF))) ((1.5))</td>
</tr>
<tr>
<td><strong>Unsuccessful Search (LV=50%)</strong></td>
<td>(\frac{1}{2}(1+1/(1-LF)^2)) ((2.5))</td>
</tr>
</tbody>
</table>
Chaining: It depends on the **Average List Length**

- $N$ items
- Table of dimension $Dim$
- Average list length = $N/Dim$
- Complexity = $O( N/Dim )$
$O(1)$
Hashing in Java
Hashing in Java

- Hashing integers
- Hashing floats
- Hashing strings
- Hashing custom objects
public class HashingExamples {

    public static void main(String[] args) {
        String s = "Smith College";
        System.out.println( "s.hashCode() = " + s.hashCode() );

        int n = 12345;
        System.out.println( "n's hashCode() = " + Integer.valueOf( n ).hashCode() );

        double y = 6.02E23;
        System.out.println( "y's hashCode() = " + Double.valueOf( y ).hashCode() );

        float x = 3.14159f;
        System.out.println( "x's hashCode() = " + Float.valueOf( x ).hashCode() );
    }
}
Java Sets: 
HashSet
HashSet

• HashSet = collection of **unique objects**

*Remember the example of unique tweeters*
HashSet<Integer> set = new HashSet<Integer>(2000000);
int count = 0;
for (int i=0; i<1000000; i++) {
    count++;
    set.add(i);
    // replicate some elements
    if (i%100 == 13) {
        count++;
        set.add(i);
    }
}
System.out.println(count + " ints added to set. Set contains " + set.size() + " unique items");

1010000 ints added to set. Set contains 1000000 unique items
HashSet<Integer> set = new HashSet<Integer>();
for ( int i=0; i<10; i++ )
    set.add( i*101 );

System.out.print( "set = " );
for ( int x: set )
    System.out.print( x + " " );
System.out.println();

set = 0 909 101 505 202 707 808 404 606 303
HashSet<Object> set = new HashSet<Object>();
HashSet<Object> mySet = new HashSet<Object>();

mySet.add( "Hello" );
mySet.add( 3 );

set.add( 3.14159 );
set.add( 2014 );
set.add( "Smith College" );
set.add( mySet );

System.out.println("Set: " + set);

Set: [[3, Hello], Smith College, 2014, 3.14159]
Default Hash Function
class MyPair {
    public int x; public int y;
    MyPair( int xx, int yy ) { x = xx; y = yy; }
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}

private void example5() {
    HashSet<MyPair> set = new HashSet<MyPair>();
    set.add( new MyPair( 1, 2 ) );
    set.add( new MyPair( 3, 30 ) );
    set.add( new MyPair( 1, 2 ) );
    System.out.println( "Set: " + set );
}
class MyPair {
    public int x; public int y;
    MyPair( int xx, int yy ) { x = xx; y = yy; }
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}

private void example5() {
    HashSet<MyPair> set = new HashSet<MyPair>();
    set.add( new MyPair( 1, 2 ) );
    set.add( new MyPair( 3, 30 ) );
    set.add( new MyPair( 1, 2 ) );
    System.out.println( "Set: " + set );
}
```java
class MyPair {
    public int x; public int y;
    MyPair( int xx, int yy ) { x = xx; y = yy; }
    public String toString() { return "(" + x + ", " + y + ")"; }
    public int hashCode() { return (x+97) * (y+101) ; }
    public boolean equals( Object o ) { return true; }
}
```
class MyPair {
    public int x; public int y;
    MyPair( int xx, int yy ) { x = xx; y = yy; }
    public String toString() {
        return "(" + x + ", " + y + ")";
    }
    public int hashCode(){ return (x+97) * (y+101) ; }
    public boolean equals( Object o ) {
        if ( o==null ) return false;
        MyPair p = (MyPair) o;
        if ( p.x != x || p.y != y ) return false;
        return true;
    }
}

Set: [(1, 2), (3, 30)]
Always implement `equals()` if you implement `hashCode()`!
Java Trick

• Instead of computing the hash code integer every time hashCode() is called, the hash code can be pre-computed once, when the object is created (or modified), and stored in a special field.

• In this case `hashCode()` is definitely O(1)
HashMaps
HashMaps

• Associate a **value** to a **key**.

• A HashMap is a set of (key:value) pairs where the hashing is done on the key.

• **Chaining** is used to resolve collision
<table>
<thead>
<tr>
<th>Colleges</th>
<th>Towns</th>
</tr>
</thead>
<tbody>
<tr>
<td>0  “Smith”</td>
<td>0  “Northampton”</td>
</tr>
<tr>
<td>1  “Amherst”</td>
<td>1  “Amherst”</td>
</tr>
<tr>
<td>2  “Hampshire”</td>
<td>2  “Amherst”</td>
</tr>
<tr>
<td>3  “MtHolyoke”</td>
<td>3  “South Hadley”</td>
</tr>
</tbody>
</table>
Colleges

<table>
<thead>
<tr>
<th>0</th>
<th>“Smith”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“Amherst”</td>
</tr>
<tr>
<td>2</td>
<td>“Hampshire”</td>
</tr>
<tr>
<td>3</td>
<td>“MtHolyoke”</td>
</tr>
</tbody>
</table>

Towns

<table>
<thead>
<tr>
<th>0</th>
<th>“Northampton”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>“Amherst”</td>
</tr>
<tr>
<td>2</td>
<td>“Amherst”</td>
</tr>
<tr>
<td>3</td>
<td>“South Hadley”</td>
</tr>
</tbody>
</table>

CollegeTowns

<table>
<thead>
<tr>
<th>“Smith”</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Northampton”</td>
</tr>
<tr>
<td>“Amherst”</td>
</tr>
</tbody>
</table>

| “Hampshire” |
| “Amherst” |

| “MtHolyoke” |
| “South Hadley” |
```java
import java.util.HashMap;

HashMap<String, String> colTown = new HashMap<String, String>();
colTown.put( "Amherst", "Amhest" );
colTown.put( "Smith", "Northampton" );
colTown.put( "Hampshire", "Amherst" );
colTown.put( "MtHoyoke", "South Hadley" );

for ( String college: colTown.keySet() )
    System.out.println( college + " College is in " + colTown.get( college ) );

Smith College is in Northampton
MtHoyoke College is in South Hadley
Hampshire College is in Amherst
Amherst College is in Amhest
```
Can use user-defined key objects, but must implement `equals()` and `hashCode()`!
import java.util.HashMap;

HashMap< MyPair, String > map = new HashMap<MyPair, String>();
map.put( new MyPair( 1, 2), "George" );
map.put( new MyPair( 10, 20 ), "Smith College" );
map.put( new MyPair( 1, 3 ), null );

System.out.println( "map = " + map );

MyPair x = new MyPair( 10, 20 );
if ( map.containsKey( x ) )
    System.out.println( x + " --> " + map.get( x ) );

map = {(1, 3)=null, (1, 2)=George, (10, 20)=Smith College}
(10, 20) --> Smith College
HashMaps/HashSets in Final Project?

• Where could a **HashMap** or a **HashSet** be useful in the final project?

• What would the key be? What would the value be?