## CS 10: Problem solving via Object Oriented Programming

Hashing

# Java provides us faster Sets and Maps using hashing instead of Trees

- Sets hold unique objects, Maps hold Key/Value pairs
- Map Keys are unique, but Values may be duplicated
- As we saw last class, using a Tree is a natural fit for implementing Sets and Maps
- Performance with a Tree is *generally* better than a List
- We can do better than Tree performance by using today's topic of discussion hashing
- Java provides the HashSet and HashMap out-of-the-box that do a lot of the hard work for us

## Agenda

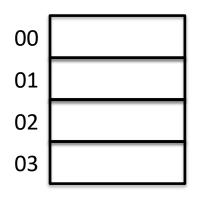
## 📫 1. Hashing

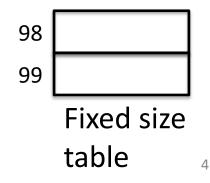
- 2. Computing Hash functions
- 3. Implementing Maps/Sets with hashing
- 4. Handling collisions
  - 1. Chaining
  - 2. Open Addressing

### Sears store implementation of hash table

- Used to have 100 slots behind order desk, 0...99
- Shipments arrive, details of where item stored in warehouse put in slot by last two digits of customer phone number (e.g., 03)

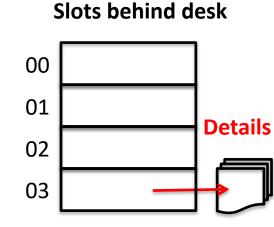


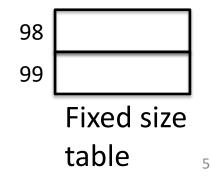




### Sears store implementation of hash table

- Used to have 100 slots behind order desk, 0...99
- Shipments arrive, details of where item stored in warehouse put in slot by last two digits of customer phone number (e.g., 03)

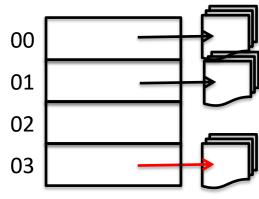


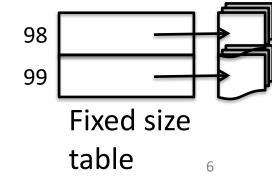


### Sears store implementation of hash table

- Used to have 100 slots behind order desk, 0...99
- Shipments arrive, details of where item stored in warehouse put in slot by last two digits of customer phone number (e.g., 03)
- Customer arrives, gives last two digits of phone

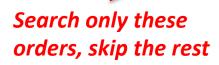




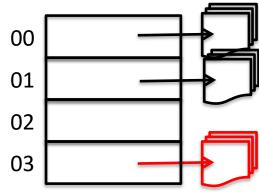


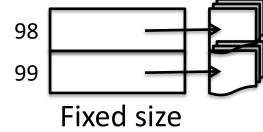
### Sears store implementation of hash table

- Used to have 100 slots behind order desk, 0...99
- Shipments arrive, details of where item stored in warehouse put in slot by last two digits of customer phone number (e.g., 03)
- Customer arrives, gives last two digits of phone
- Clerk finds slot with that two-digit number
- Clerk searches contents of that slot only
- Could be multiple orders, but can find the order quickly because only a few orders in slot







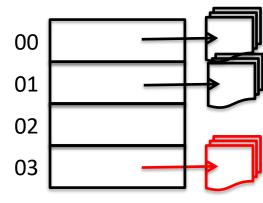


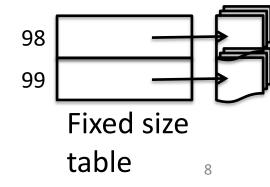
table

### Sears store implementation of hash table

- Used to have 100 slots behind order desk, 0...99
- Shipments arrive, details of where item stored in warehouse put in slot by last two digits of customer phone number (e.g., 03)
- Customer arrives, gives last two digits of phone
- Clerk finds slot with that two-digit number
- Clerk searches contents of that slot only
- Could be multiple orders, but can find the order quickly because only a few orders in slot
- Splits set of (possibly) hundreds or thousands of orders into 100 slots of a few items each

#### **Slots behind desk**

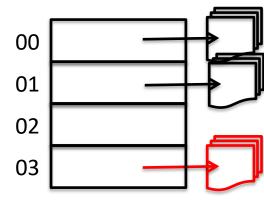


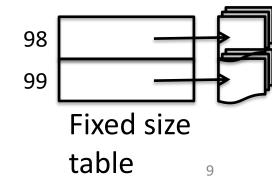


### Sears store implementation of hash table

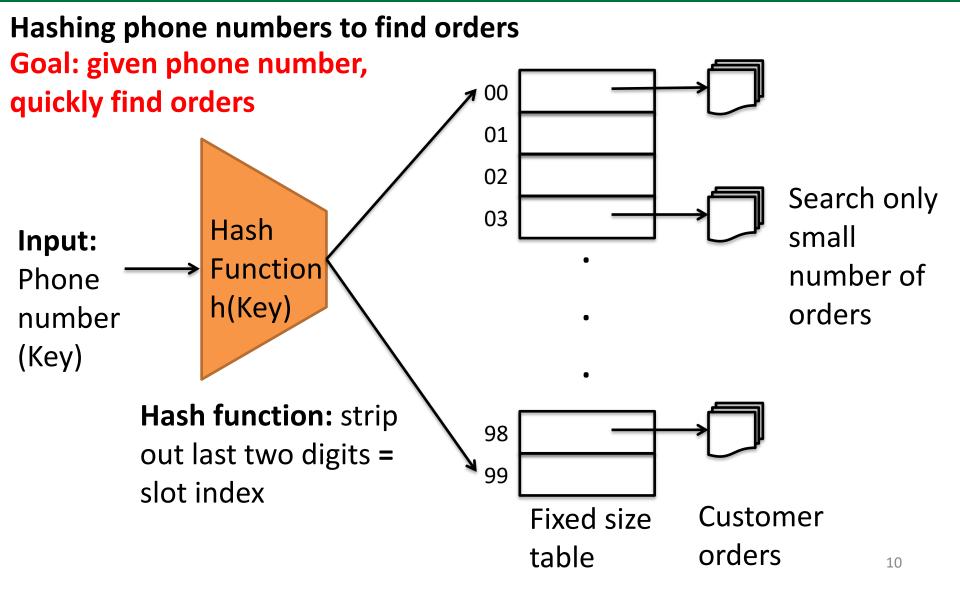
- Used to have 100 slots behind order desk, 0...99
- Shipments arrive, details of where item stored in warehouse put in slot by last two digits of customer phone number (e.g., 03)
- Customer arrives, gives last two digits of phone
- Clerk finds slot with that two-digit number
- Clerk searches contents of that slot only
- Could be multiple orders, but can find the order quickly because only a few orders in slot
- Splits set of (possibly) hundreds or thousands of orders into 100 slots of a few items each
- Trick: find a hash function that spreads customers evenly
- Last two digits work, why not first two?

#### **Slots behind desk**

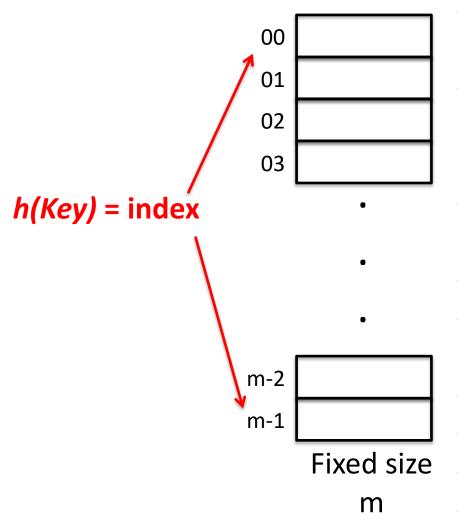




## The store is using a form of hashing based on customer's phone number



# Hashing's big idea: map a Key to an array index, then access is fast



### Map hash table implementation

- Begin with array of fixed size m (called a hash table)
- Each array index holds item we want to find (e.g., warehouse location of customer's order)
- Use hash function h on Key to give index into hash table
- *h(Key)* = table index *i* = 0..*m*-1
- Get item from hash table at index given by hash function
- <u>Fast</u> to get/set/add/remove items
- What about a HashSet?
- Use object itself as Key
- How to hash Key or object? <sup>11</sup>

## Agenda

### 1. Hashing

- 2. Computing Hash functions
  - 3. Implementing Maps/Sets with hashing
  - 4. Handling collisions
    - 1. Chaining
    - 2. Open Addressing

## Good hash functions map keys to indexes in table with three desirable properties

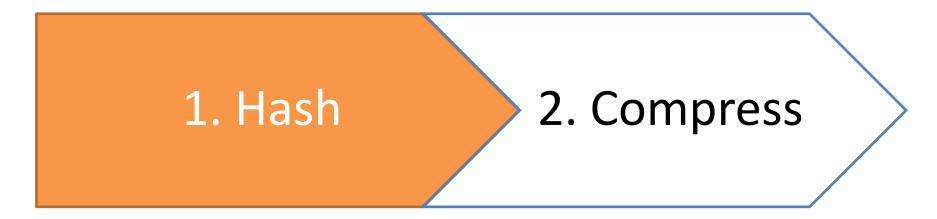
### Desirable properties of a hash function

- 1. Hash can be computed quickly and consistently
- 2. Hash spreads the universe of keys evenly over the table (simple uniform hashing)
- 3. Small changes in the key (e.g., changing a character in a string or order of letters) should result in different hash value

### **Cryptographic hash function also:**

- Difficult to determine key given the result of hash
- Unlikely that different keys will result in same hash
- We will not focus on crypto requirements

# Hashing is often done in two steps: hash then compress



- Get an integer representation of Key
- Integer could be in range

   –infinity to +infinity

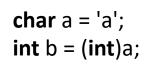
Constrain integer to table index [0..m)

# First step in hashing is to get an integer representation of the key

Goal: given key compute an index into hash table array

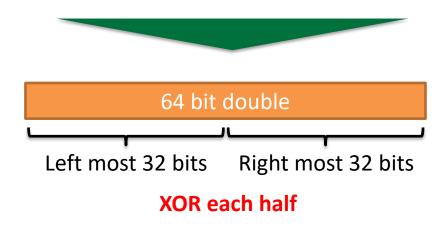
## Some Java objects can be directly cast to integers

- byte
- short
- int
- char



## Some items too long cast to integers

- double (64 bits)
- long (64 bits)
- Too long to make 32 bit integers



# Complex objects such as Strings can also be hashed to a single integer

### Hashing complex objects

- Consider String x of length n where  $x = x_0 x_1 \dots x_{n-2} x_{n-1}$
- Pick prime number a (book recommends 31, 37, or 41)
- Cast each character in  $\boldsymbol{x}$  to an integer
- Calculate polynomial hashcode as  $x_0a^{n-1} + x_1a^{n-2} + \dots + x_{n-2}a + x_{n-1}$
- Use Horner's rule to efficiently compute hash code

```
public int hashCode() {
    final int a=37;
    int sum = x[0]; //first item in array
    for (int j=1;j<n;j++) {
        sum = a*sum + x[j]; //array element j
    }
    return sum;
}</pre>
```

 Experiments show that when using a as above, 50,000 English words had fewer than 7 collisions

# Good news: Java provides a *hashCode()* method to compute hashes for us!

### hashCode()

Java does the hashing for us for Strings and autoboxed types with *hashCode()* method

Character a = 'a'; a.hashCode() returns 97

String b = "Hello"; b.hashCode() returns 69609650

# Bad news: We need to override *hashCode()* and *equals()* for our own Objects

- By default Java uses memory address of objects as a *hashCode*
- But we typically want to hash based on properties of object, not whatever memory location an object happened to be assigned
- This way two objects with same instance variables will hash to the same table location (those objects are considered equal)
- Java says that two equal objects must return same hashCode()

```
public class BlobHash extends Blob{
```

```
@Override
public boolean equals(Object otherBlob) {
    Blob b = (Blob)otherBlob; //cast as Blob
    if (x == b.x && y == b.y && r == b.r)
        return true;
    return false;
}
@Override
public int hashCode() {
    final int a=37;
    int sum = a * a * (int)x;
    sum += a * (int)y;
    sum += a * (int)r;
    return sum;
}
```

Here we consider two Blobs *equal* if they have the same *x*, *y* and *r* values <u>*equals()* IS THE RIGHT WAY TO</u> <u>COMPARE OBJECT EQUALITY (not ==)</u>

Override hashCode() to provide the same hash if two Blobs are equal

If don't override *hashCode()* then even though two objects are considered equal, Java will look in the wrong slot 18

#### hashCode()

r <b>cs10</b> ~/Documents/Idea	P 1 2 ► 3	<pre>public class HashTest {</pre>	Some types can be
🕨 🖿 data	4 ►	public static void main(String[] args) {	
🕨 🖿 day1	5	char a = 'a';	directly cast to an integer
🕨 🖿 day2	6	<pre>int b = (int)a; System.out.println("Casting 'a' to int is: "+ b);</pre>	
🕨 🖿 day3	8	Character z = 'a';	
🕨 🖿 day4	9	<pre>System.out.println("hashCode for 'a' is: " + z.hashCode());</pre>	
🕨 🚞 day5	10	<pre>String y = "Hello"; System.out.println("hashCode for 'hello' is: " + y.hashCode</pre>	()).
🕨 🖿 day6	12	System.out.println();	());
🕨 🖿 day7	13		
🕨 🖿 day8	14	<pre>//create new Blob with overridden equals and hashCode funct.</pre>	ions
🕨 🖿 day9	15 16	<pre>BlobHash b1 = new BlobHash(); b1.x = 5; b1.y = 5; b1.r = 5; //update b1's location</pre>	
🕨 🖿 day10	17	BlobHash b2 = new BlobHash(); //create new HashBlob	
🔻 🖿 day11	18	System.out.println("bl is at (x,y,r): " + bl.x + ", " + bl.	
© BlobHash	19 20	<pre>System.out.println("b2 is at (x,y,r): " + b2.x + ", " + b2. System.out.println("hashCode b1: " + b1.hashCode() + " b2:"</pre>	
😅 HashTest	21	System.out.println("bl is equal to b2: " + b1.equals(b2));	
🕨 🖿 day12	22	b2.x = 5; b2.y = 5; b2.r = 5; //set b2 to same location as location as b2 and	
🕨 🖿 day13	23 24	<pre>System.out.println("after update b1 now equals b2: " + b1.e System.out.println("hashCode b1: " + b1.hashCode() + " b2:"</pre>	
▶ 🖿 day14	25	}	
🕨 🖿 day15	26	}	
day16	27		
0.5 day 10		HashTest > main()	
	shTest ×		
<ul> <li>         ↑ /Library/Java/Jav Casting 'a' to in         ↓         ↓ ↓</li></ul>		lachines/jdk-12.0.1.jdk/Contents/Home/bin/java "–javaagent:/Applicati	ions/IntelliJ IDEA.app/

#### hashCode()

\*

	cs10 ~/Documents/IdeaP	1 2 ►	<pre>public class HashTest {</pre>	
	.idea	3		moutos bash for
	🖿 data	4 🕨	<pre>public static void main(String[] args) {</pre>	nputes hash for
•	ay1	5	char a = 'a';	ed types with
•	ay2	6		types with
•	ay3	7	System.out.println("Casting 'a' to int is: "+ b); Character z = 'a'; Anather of the state of the	101)
•	ay4	9	System.out.println("hashCode for 'a' is: " + z.hashCode());	
	ay5	10	<pre>String y = "Hello";</pre>	
		11	<pre>System.out.println("hashCode for 'hello' is: " + y.hashCode());</pre>	
	ay6	12	System.out.println();	
•	ay7	13	//create new Blob with overridden equals and hashCode functions	
•	ay8	14 15	BlobHash b1 = new BlobHash();	
•	🖿 day9	16	b1.x = 5; $b1.y = 5$ ; $b1.r = 5$ ; $//update b1's location$	
•	ay10	17	BlobHash b2 = new BlobHash(); //create new HashBlob	
	ay11	18	<pre>System.out.println("bl is at (x,y,r): " + b1.x + ", " + b1.y + ", " + b1.r);</pre>	
	© BlobHash	19 20	<pre>System.out.println("b2 is at (x,y,r): " + b2.x + ", " + b2.y + ", " + b2.r); System.out.println("hashCode b1: " + b1.hashCode() + " b2:" + b2.hashCode());</pre>	
	d HashTest	20	System.out.println("b1 is equal to b2: " + b1.equals(b2));	
1	ay12	22	b2.x = 5; b2.y = 5; b2.r = 5; //set b2 to same location as b1	
		23	System.out.println("after update b1 now equals b2: " + b1.equals(b2));	
- P	ay13	24	<pre>System.out.println("hashCode b1: " + b1.hashCode() + " b2:" + b2.hashCode());</pre>	
	ay14	25		
►	ay15	26 27	3	
•	ay16	21	HashTest > main()	
-	Physical Company			
Run		hTest		
			lMachines/jdk-12.0.1.jdk/Contents/Home/bin/java "-javaagent:/Applications/IntelliJ IDEA.app,	
	Casting 'a' to int hashCode for 'a' i			
	hashCode for 'a' 1	.5. 57		
Ō				
€	<b>±</b> ±			
	÷			
	1			20
	-			20

#### hashCode()

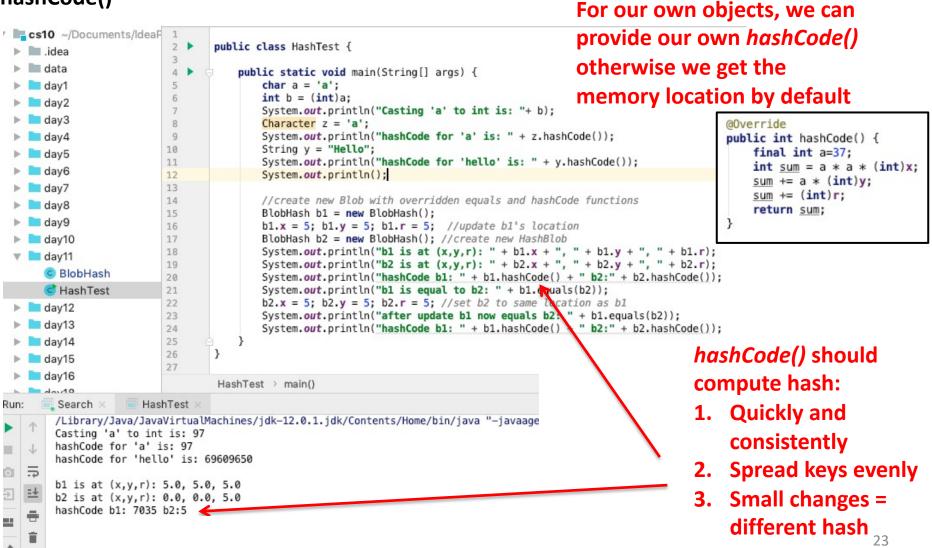
\*

cs	10 ~/Documents/IdeaP	1			
- Film	.idea	2		<pre>public class HashTest {</pre>	
- F 🖿	data	3		<pre>public static void main(String[] args) {</pre>	
Image:	dav1	5		char a = 'a';	
- F	day2	6		<pre>int b = (int)a;</pre>	
	day3	7		<pre>System.out.println("Casting 'a' to int is: "+ b);</pre>	
	day4	8 9		<pre>Character z = 'a'; System.out.println("hashCode for 'a' is: " + z.hashCode());</pre>	
	day5	10		String v = "Hello":	
		11		System.out.println("hashCode for 'hello' is: " + y.hashCode()); hashCode	e() also works
	day6	12		System.out.println();	
	day7	13 14		//create new Blob with overridden equals and hashCode functions	complex built-
	day8	15		BlobHash b1 = new BlobHash(); b1 x = 5: b1 x = 5: b1 x = 5: //update b1's location in types	
	day9	16		bink = 5, bing = 5, bin = 5, //apdate bi 5 totation	
	day10	17		BlobHash b2 = new BlobHash(); //create new HashBlob	
<b>V</b>	day11	18 19		<pre>System.out.println("bl is at (x,y,r): " + b1.x + ", " + b1.y + ", " + b1.r); System.out.println("b2 is at (x,y,r): " + b2.x + ", " + b2.y + ", " + b2.r);</pre>	
1000	C BlobHash	20		<pre>System.out.println("hashCode b1: " + b1.hashCode() + " b2:" + b2.hashCode());</pre>	
	C HashTest	21		<pre>System.out.println("bl is equal to b2: " + b1.equals(b2));</pre>	
- F 🖿	day12	22 23		<pre>b2.x = 5; b2.y = 5; b2.r = 5; //set b2 to same location as b1 System.out.println("after update b1 now equals b2: " + b1.equals(b2));</pre>	
- 🕨 🖿	day13	23		System.out.println("hashCode b1: " + b1.hashCode() + " b2:" + b2.hashCode());	
- F 🖿	day14	25		A }	
- F 🖿	day15	26		}	
- F	day16	27			
	dau10			HashTest > main()	
Run:	📑 Search 🛛 📄 Hasl	hTest	×		
				Machines/jdk-12.0.1.jdk/Contents/Home/bin/java "-javaagent:/Applications/IntelliJ IDEA.app/	
	Casting 'a' to int				
■ ↓	hashCode for 'a' i hashCode for 'hell			69609650	
o .⇒	hashcode for hert	0 1	5. 0	03003030	
Ð 🖬					
- ÷					
- ÷					21
					$\angle \bot$

#### hashCode()

hashCode()		, For our own objects, we can
🛚 🔚 cs10 ~/Documents/IdeaP	1	public class HashTast ( provide our own hashCode()
🕨 🖿 .idea	2 >	public class hashest i
data	4	<pre>public static void main(String[] args) {</pre>
🕨 🖿 day1	5	char a = 'a':
day2	6	int b = (int)a; memory location by default
day3	7	<pre>System.out.println("Casting 'a' to int is: "+ b);</pre>
	8	<pre>Character z = 'a'; System.out.println("hashCode for 'a' is: " z.hashCode());</pre>
day4	10	String y = "Hello";
🕨 🖿 day5	11	System.out.println("hashCode for 'hello' is: " + y.hashCode());
🕨 🖿 day6	12	System.out.println();
🕨 🖿 day7	13	
🕨 🖿 day8	14	<pre>//create new Blob with overridden equal and hashCode functions</pre>
🕨 🖿 day9	15 16	BlobHash b1 = new BlobHash(); b1.x = 5; b1.y = 5; b1.r = 5; //update b1's location
day10	17	BlobHash b2 = new BlobHash(); //create new HashBlob
v 🖿 day11	18	<pre>System.out.println("bl is at (x,y,r): " + bl.x + ", " + bl.y + ", " + bl.r);</pre>
C BlobHash	19	System.out.println("b2 is at (x,y,r): " + b2.x + ", " + b2.y + ", " + b2.r);
G HashTest	20	<pre>System.out.println("hashCode b1: " + b1.hashCode() + " b2:" + b2.hashCode()); System.out.println("b1 is equal to b2: " + b1.equals(b2));</pre>
	21	$b_{2,x} = 5$ ; $b_{2,y} = 5$ ; $b_{2,r} = 5$ ; $//set b_2$ to same location as bl
🕨 🖿 day12	23	System.out.println("after update bl now equals b2: " + b1.equals(b2));
🕨 🖿 day13	24	System.out.println("hashCode b1: " + b1.hashCode() + " b2:" + b2.hashCode());
day14	25	International and the second se
🕨 🖿 day15	26	}
🕨 🖿 day16	27	
h dau10		HashTest > main()
Run: 📃 Search 🛛 🗐 Has	hTest $ imes$	
<pre>/Library/Java/Java Casting 'a' to int hashCode for 'a' i hashCode for 'hell is at (x,y,r): b1 is at (x,y,r): b2 is at (x,y,r): hashCode b1: 7035</pre>	t is: 97 is: 97 lo' is: 6 5.0, 5.0 0.0, 0.0	, 5.0

#### hashCode()



#### equals()

Designed Designed and a D

🕐 🖿 cs10 ~/Documents/IdeaP		Otherwise equals() checks it same memory location
🕨 🖿 .idea	2 ► 3	<pre>public class HashTest {</pre>
🕨 🖿 data	4	<pre>public static void main(String[] args) {     @Override</pre>
🕨 🖿 day1	5	char a = 'a':
day2	6	<pre>int b = (int)a; public boolean equals(Object otherBlob) {</pre>
🕨 🖿 day3	7 8	System.out.println("Casting 'a' to int is: "+ b); Character z = 'a'; Blob b = (Blob)otherBlob; //cast as Blob
day4	9	System.out.println("hashCode for 'a' is: " + z.ha if $(x == b.x \&\& y == b.y \&\& r == b.r)$
day5	10	String y = "Hello"; return true;
day6	11	System.out.println("hashCode for 'hello' is: " +
day0	12 13	System.out.println(); return Talse;
day8	14	//create new Blob with overridden equals and hash
day8	15	BlobHash b1 = new BlobHash();
day9	16 17	<pre>b1.x = 5; b1.y = 5; b1.r = 5; //update b1's location BlobHash b2 = new BlobHash(); //create new HashBlob</pre>
	18	System.out.println("bl is at (x,y,r): " + b1.x + ", " + b1.y + ", " + b1.r);
V ay11	19	System.out.println("b2 is at (x,y,r): " + b2.x + ", " + b2.y + ", " + b2.r);
© BlobHash	20	<pre>System.out.println("hashCode b1: " + b1.hashCode() + " b2:" + b2.hashCode());</pre>
C HashTest	21 22	<pre>System.out.println("b1 is equal to b2: " + b1.equals(b2)); b2.x = 5; b2.y = 5; b2.r = 5; //set b2 to same location as b1</pre>
▶ 🖿 day12	23	System.out.println("after update bl now equals b2: " + b1.equals(b2));
🕨 🖿 day13	24	<pre>System.out.println("hashCode b1: " + b1.hashCode() + " b2:" + b2.hashCode());</pre>
🕨 🖿 day14	25 26	
🕨 🖿 day15	20	ſ
🕨 🖿 day16		HashTest > main()
Run: 📃 Search 🗙 🗐 Has	shTest $ imes$	
		Techings/idk 12.0.1 idk/Contents/Wars/bis/invall investment/Appliestions/Intallil IDEA app
Casting 'a' to int hashCode for 'a' i hashCode for 'hell is at (x,y,r): b1 is at (x,y,r): b2 is at (x,y,r): hashCode b1: 7035 b1 is equal to b2:	t is: 97 is: 97 lo' is: 0 5.0, 5.0 0.0, 0.0 b2:5	, 5.0
- <b>=</b>		24

#### equals()

<ul> <li>▶ ■ data</li> <li>▶ ■ day1</li> <li>3</li> <li>3</li> <li>4</li> <li>5</li> </ul>	pub	Otherwise equals() checks if same	
▶ ■ data 4 ▶ ■ day1 5	i 🕨 👳		
▶       day2       6         7       ▶       day3       8         ▶       day4       9         ▶       day5       10         ▶       day6       12         ▶       day7       13         ▶       day8       14	5 7 8 9 9 1 2 9 8 8 8	<pre>Character z = 'a'; System.out.println("hashCode for 'a' is: " + z.ha String y = "Hello"; System.out.println("hashCode for 'hello' is: " + System.out.println(); //create new Blob with overridden equals and hash</pre> if (x == b.x && y return true; return false; }	G(Object otherBlob) { CherBlob; //cast as Blob A == b.y && r == b.r)
▶       day9       15         ▶       day10       17         ▼       day11       18         ©       BlobHash       20         ©       HashTest       21         ▶       day12       22         ▶       day13       24         ▶       day14       25         ▶       day15       27		<pre>System.out.println("b2 is at (x,y,r): " + b2.x + ", " + b2.y + ", " + b2.r): System.out.println("hashCode b1: " + b1.hashCode() + " b2:" + b2 hashCode()); System.out.println("b1 is equal to b2: " + b1.equals(b2)); b2.x = 5; b2.y = 5; b2.r = 5; //set b2 to same location as b1 System.out.println("after update b1 now equals b2: " + b1.equals(b2)); System.out.println("hashCode b1: " + b1.hashCode() + " b2:" + b2.hashCode()); </pre>	This is the right way to compare if two objects are equivalent (not b1 == b2)
<pre>&gt; day16 Run: Search × HashTe /Library/Java/JavaVir Casting 'a' to int is hashCode for 'a' is: hashCode for 'hello'  &gt; ↓ hashCode for 'hello' b1 is at (x,y,r): 5.0 b2 is at (x,y,r): 0.0 hashCode b1: 7035 b2: b1 is equal to b2: fa</pre>	est × rtualMachi s: 97 97 is: 69609 0, 5.0, 5. 0, 0.0, 5. :5	shTest → main() nes/jdk-12.0.1.jdk/Contents/Home#bin/java "-javaagent:/Applications/IntelliJ IDEA.ap 1650 0	

#### equals()

/ Cs10 ~/Documents/IdeaP	1	Otherwise <i>equals()</i> checks	if same memory location
idea	2 🕨 pu	olic class HashTest {	-
	3 4 5 6 7 8 9 10 11 12	System.out.println("Casting 'a' to int is: "+ b); Character z = 'a'; System.out.println("hashCode for 'a' is: " + z.haBlob b = if (x ==	<pre>n equals(Object otherBlob) { (Blob)otherBlob; //cast as Blob b.x &amp;&amp; y == b.y &amp;&amp; r == b.r) n true; lse;</pre>
▶ ■ day8 ▶ ■ day9	13 14 15 16 17	<pre>//create new Blob with overridden equals and hash BlobHash b1 = new BlobHash(); b1.x = 5; b1.y = 5; b1.r = 5; //update b1's location BlobHash b2 = new BlobHash(); //create new HashBlob</pre>	
▼ <b>a</b> day11 © BlobHash	17 18 19 20 21	<pre>System.out.println("bl is at (x,y,r): " + bl.x + ", " + bl.y + ", " - System.out.println("b2 is at (x,y,r): " + b2.x + ", " + b2.y + ", " - System.out.println("hashCode b1: " + b1.hashCode() + " b2:" + b2.hash System.out.println("bl is equal to b2: " + b1.equals(b2);</pre>	+ b2.r);
<ul> <li>▶ ■ day12</li> <li>▶ ■ day13</li> <li>▶ ■ day14</li> <li>▶ ■ day15</li> </ul>	22 23 24 25 26 }	<pre>b2.x = 5; b2.y = 5; b2.r = 5; //set b2 to same location as b1 System.out.println("after update b1 now equals b2: " + b1.equals(b2) System.out.println("hashCode b1: " + b1.hashCode() + " b2:" + b2.hash }</pre>	
► ay16 Run: Search × ■ Hash	HanTest ×	lishTest → main()	After updating x, y, and r
<ul> <li>              \Library/Java/Java/ Casting 'a' to int hashCode for 'a' is hashCode for 'hello      </li> <li>             is at (x,y,r): 5         </li> <li>             is at (x,y,r): 6         </li> <li>             is equal to b2:         </li> <li>             after update b1 now         </li> </ul>	is: 97 s: 97 o' is: 6960 5.0, 5.0, 5 0.0, 0.0, 5 b2:5 false	9650 .0 .0	two Blobs are now equal

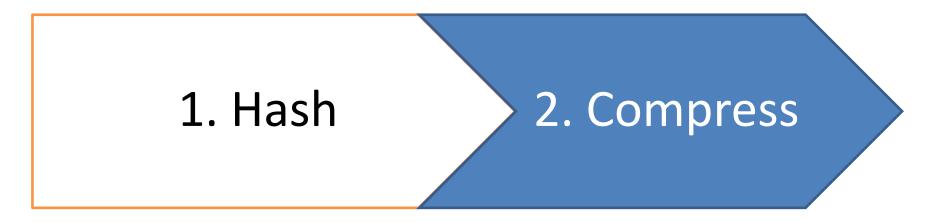
#### equals()

cs10 ~/Documents/IdeaP	1		Otherwise <i>equ</i>	<i>als()</i> checks if sam	ne memory location
▶ idea	2 🕨	<pre>public class HashTest {</pre>	•		-
data	3	public static void main(St	tring[] args) {		
dav1	5	char a = 'a';	(Ting[] args/ t	@Override	
day2	6	<pre>int b = (int)a;</pre>			.s(Object otherBlob) {
day3	7		asting 'a' to int is: "+ b);	Blob $b = (Blob)c$	otherBlob; //cast as Blob
day3	8	Character z = 'a'; System.out.println("ba	ashCode for 'a' is: " + z.ha	if (x == b.x &&	y == b.y && r == b.r)
day5	10	String y = "Hello";		return true;	
day6	11		ashCode for 'hello' is: " +	return false;	
	12 13	System.out.println();		recurn false,	
day7	13	//create new Blob with	n overridden equals and hash		
day8	15	BlobHash b1 = new Blob		n de la constante de la constan La constante	
🕨 🖿 day9	16		<pre>l.r = 5; //update b1's loca</pre>		
▶ 🖿 day10	17 18		<pre>bHash(); //create new HashBl L is at (x,y,r): " + b1.x +</pre>		This is the right
🔻 🖿 day11	19	System.out.println("b2	2 is at (x,y,r): " + b1.x +	", " + $b_{2,v}$ + ", " + $b_{2,r}$ );	This is the fight
C BlobHash	20	System.out.println("ha	ashCode b1: " + b1.hashCode(	) + " b2:" + b2.hashCode());	way to compare
😅 HashTest	21		L is equal to b2: " + b1.equ		
day12	22 23		<pre>2.r = 5; //set b2 to same lo fter update b1 now equals b2</pre>		if two objects
▶ 🖿 day13	24			) + " <b>b2</b> :" + b2.hashCode());	ana anuivalant
	25	↓ }	•		are equivalent
🕨 🖿 day15	26 27	}	7		(not b1 == b2)
🕨 🖿 day16	21	HechTest > main()			(1101 01 02)
01.00 A		HashTest → main()		After	updating <i>x,y,</i> and <i>r</i>
	nTest ×				upuating x, y, and r
		Machines/jdk-12.0.1.jdk/Content	s/Home/bin/java "–javaagent:	Applications/ two B	lobs are now equal
Casting 'a' to int hashCode for 'a' i					
hashCode for 'hell		69609650			
₫ ☴			· · · · · · · · · · · · · · · · · · ·	hashCode() also re	eturns the same
∃ ➡ b1 is at (x,y,r): b2 is at (x,y,r):				••	
hashCode b1: 7035		0, 510		value for equivale	nt objects
bi is equal to b2:					
after update b1 not					27
hashCode b1: 7035	02:7035				

#### equals()

r cs10 ~/Docu	uments/IdeaF	1		Otherwise <i>eq</i>	<i>uals()</i> che	cks if sam	ne memory loca	tion
idea		2 🕨	public	class HashTest {				
data		3		<pre>blic static void main(String[] args) {</pre>				
day1		5	o pu	char a = 'a';	@Override			
day2		6		<pre>int b = (int)a;</pre>		<mark>olea</mark> n equal	s(Object otherBlok	) {
day2		7		System.out.println("Casting 'a' to int is: "+ b	Blob	b = (Blob)c	therBlob; //cast a	as Blob
day3		8 9		<pre>Character z = 'a'; System.out.println("hashCode for 'a' is: " + z.</pre>			y == b.y && r == b	
day4		10		String y = "Hello";		eturn true;		
day5		11		System.out.println("hashCode for 'hello' is: " ·	+	n false;		
		12 13		System.out.println();	necur	n luise,		
day7		14		//create new Blob with overridden equals and has	sh }			
▶ aday8		15		BlobHash b1 = new BlobHash();				
▶ aday9		16		b1.x = 5; b1.y = 5; b1.r = 5; //update b1's loc				
▶ 📄 day10		17 18		<pre>BlobHash b2 = new BlobHash(); //create new Hash System.out.println("b1 is at (x,y,r): " + b1.x -</pre>		", " + b1,r);	This is the righ	t
🔻 🖿 day11		19		System.out.println("b2 is at (x,y,r): " + b2.x -			•	
© BlobHa		20		System.out.println("hashCode b1: " + b1.hashCode		2.hashCode());	way to compa	re
C HashTe	est	21 22		<pre>System.out.println("b1 is equal to b2: " + b1.ee b2.x = 5; b2.y = 5; b2.r = 5; //set b2 to same</pre>			if the state of the state	
day12		23		System.out.println("after update b1 now equals I		s(b2));	if two objects	
day13		24		System.out.println("hashCode b1: " + b1.hashCode			are equivalent	
🕨 🖿 day14		25 26	₽ <b>,</b> }				•	
🕨 🖿 day15		27	1				(not b1 == b2)	
🕨 🖿 day16			HashT	est > main()				
Run: Runte		nTest ×				After	updating x,y, an	d r
			Machines	s/jdk-12.0.1.jdk/Contents/Home/bin/java "-javaagen	t:/Applications		•	
	'a' to int			s/ juk=12.0.1. juk/ contents/home/ bin/ java = javaagen	c./Appcicacions/	two B	lobs are now ec	lual
hashCod	e for 'a' i	s: 97						-
hashCod	le for 'hell	o'is:	69609650	<sup>e</sup> HashMap and HashSet will	hash Cod			
bl 15 a	t (x,y,r):	5.0, 5.	0. 5.0		nasncoa	e() also re	turns the same	
🔁 📑 b2 is a	t (x,y,r):	0.0, 0.		now put equivalent objects	value for	equivale	nt objects	
hashCod	le b1: 7035	b2:5		in the series slot in the		cquivale		
after u	indate b1 no	w equal	s b2: tr	in the same slot in the			2	0
* hashCod	le b1: 7035	b2:7035		table (after compression)			2	ŏ

# Hashing is often done in two steps: hash then compress

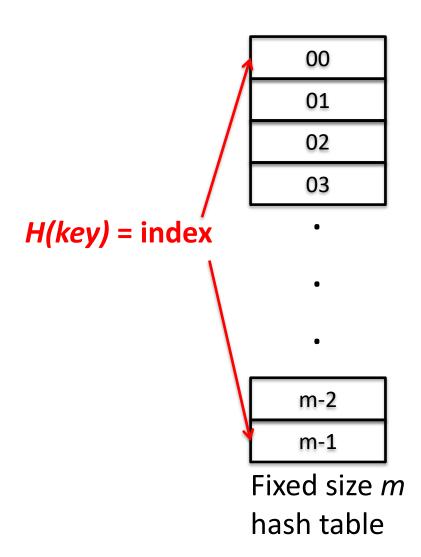


- Get an integer representation of Key
- Integer could be in range

   –infinity to +infinity

Constrain integer to table index [0..m)

# May have to compress hash value to table index [0..m)



### Compressing

- hashCode() value may be larger than the table (or negative!)
- Need to constrain value to one of the table slots [0..*m*)
- "Division method" is simple:
   h(key) = key.hashCode() % m
- Works well if *m* is prime
- Book gives a more advanced version called Multiply-Add-And-Divide (MAD)
- Java takes care of this for us  $\textcircled{\odot}$
- Eventually will encounter collisions where multiple keys map to the same slot <sup>30</sup>



- 1. Hashing
- 2. Computing Hash functions
- 3. Implementing Maps/Sets with hashing
- 4. Handling collisions
  - 1. Chaining
  - 2. Open Addressing

# Map methods can be easily implemented with hashing

#### put(key, value)

- Hash key to get table index
  - Get i=key.hashCode()
  - Compress i to 0..m-1
- Store key/value

### get(key)

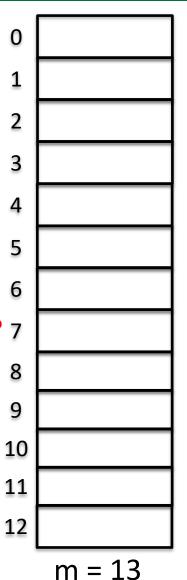
- Hash key to get table index
  - Get i=key.hashCode()
  - Compress i to 0..m-1
- Return stored value

#### remove(key)

- Hash key to get table index
  - Get i=key.hashCode()
  - Compress i to 0..m-1
- Remove stored key/value



- What if multiple items
   hash to the same index? 7
- What if table fills up?



## Agenda

- 1. Hashing
- 2. Computing Hash functions
- 3. Implementing Maps/Sets with hashing
- 4. Handling collisions
  - 1. Chaining
  - 2. Open Addressing

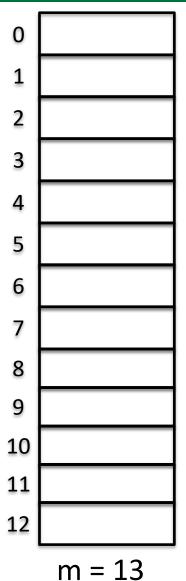
# Collisions happen when multiple keys map to the same table index

### **Integer keys**

Given table size m = 13 put(key,value)

- Hash & constrain key
- Store value at index

index = key.hashCode() % m



# Collisions happen when multiple keys map to the same table index

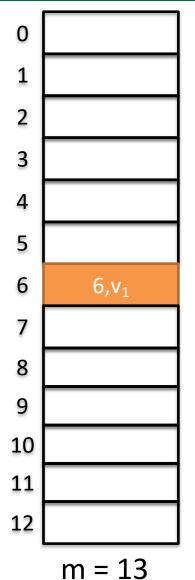
### **Integer keys**

Given table size m = 13 put(key,value)

- Hash & constrain key
- Store value at index

index = key.*hashCode()* % *m* Example

• put(6,v<sub>1</sub>) = 6 % 13 = 6



# Collisions happen when multiple keys map to the same table index

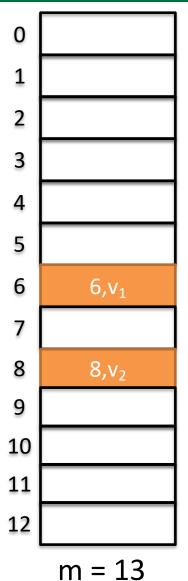
### **Integer keys**

Given table size m = 13 put(key,value)

- Hash & constrain key
- Store value at index index = key.hashCode() % m

Example

- $put(6,v_1) = 6 \% 13 = 6$
- put(8,v<sub>2</sub>) = 8 % 13 = 8

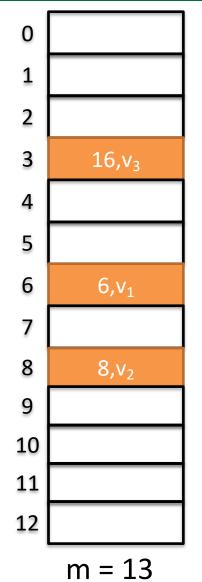


# Collisions happen when multiple keys map to the same table index

#### **Integer keys**

Given table size m = 13 put(key,value)

- Hash & constrain key
- Store value at index
   index = key.hashCode() % m
   Example
- $put(6,v_1) = 6 \% 13 = 6$
- put(8,v<sub>2</sub>) = 8 % 13 = 8
- put(16,v<sub>3</sub>) = 16 % 13 = 3

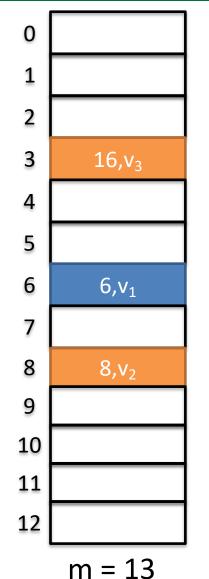


# Collisions happen when multiple keys map to the same table index

#### **Integer keys**

Given table size m = 13 put(key,value)

- Hash & constrain key
- Store value at index
   index = key.hashCode() % m
   Example
- $put(6,v_1) = 6 \% 13 = 6$
- put(8,v<sub>2</sub>) = 8 % 13 = 8
- put(16,v<sub>3</sub>) = 16 % 13 = 3
- put(19,v<sub>4</sub>) = 19 % 13 = 6



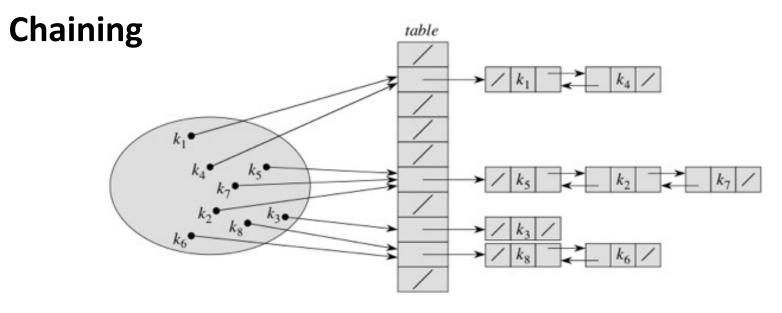
Collision! 6 and 19 mapped to the same index

h(6)=h(19)

### Agenda

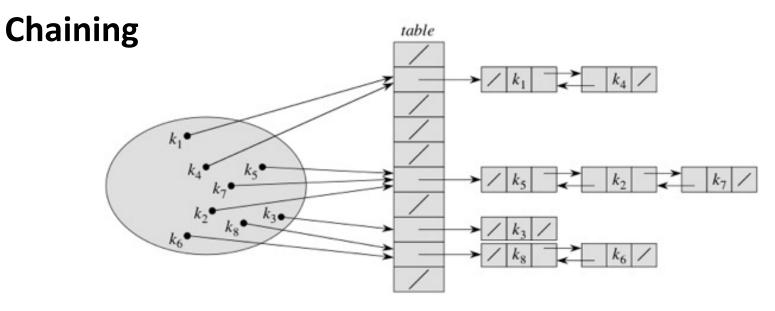
- 1. Hashing
- 2. Computing Hash functions
- 3. Implementing Maps/Sets with hashing
- 1. Handling collisions
- 1. Chaining2. Open Addressing

# Chaining handles collisions by creating a linked list for each table entry



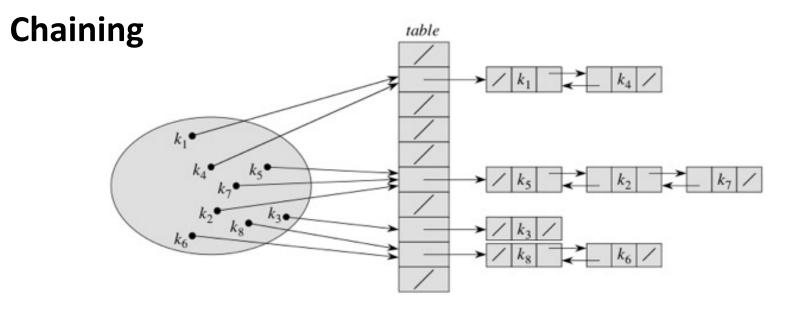
- Create a table pointing to linked list of items that hash to the same index (similar to last class word positions)
- Slot *i* holds all keys *k* for which *h(k) = i*
- Splice in new elements at head
- NOTE: Values associated with Keys are not shown, here just showing Keys

### Load factor measures number of items in the list that must be searched on average



- Assume table with *m* slots and *n* keys are stored in it
- On average, we expect *n/m* elements per collision list
- This is called the *load factor* ( $\lambda = n/m$ )
- <u>Expected</u> search time is  $\Theta(1+\lambda)$ , assuming **simple uniform hashing** (each possible key equally likely to hash into a particular slot), worst case  $\Theta(n)$  if bad hash function <sup>41</sup>

### If the load factor gets too high, then we should increase the table size



- If n (# elements) becomes larger than m (table size), then collisions are inevitable and search time goes up
- Java increases <u>table size</u> by 2X and *rehashes* into new table when  $\lambda > 0.75$  to combat this problem
- Problem: memory fragmentation with link lists spread out all over, might not be good for embedded systems

### Agenda

- 1. Hashing
- 2. Computing Hash functions
- 3. Implementing Maps/Sets with hashing
- 1. Handling collisions
  - 1. Chaining
  - 2. Open Addressing

### Open addressing is different solution, everything is stored in the table itself

### **Open addressing using linear probing**

- Insert item at hashed index (no linked list)
- For key k compute h(k)=i, insert at index i
- If collision, a simple solution is called *linear probing* 
  - Try inserting at *i*+1
  - If slot *i*+1 full, try *i*+2... until find empty slot
  - Wrap around to slot 0 if hit end of table at *m-1*
  - If  $\lambda < 1$  will find empty slot
  - If  $\lambda \approx 1$ , increase table size ( $m^*2$ ) and rehash
- Search analogous to insertion, compute key and probe until find item or empty slot (key not in table)

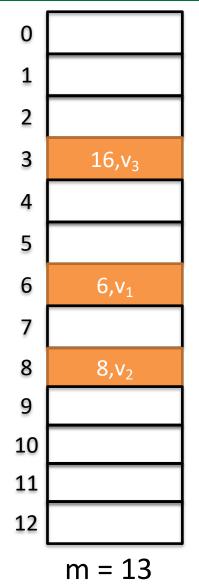
### Linear probing is one way of handling collisions under open addressing

**Integer keys** 

Given table size m = 13

index = key.hashCode() % m

- $put(6,v_1) = 6 \% 13 = 6$
- $put(8,v_2) = 8 \% 13 = 8$
- put(16,v<sub>3</sub>) = 16 % 13 = 3



### Linear probing is one way of handling collisions under open addressing

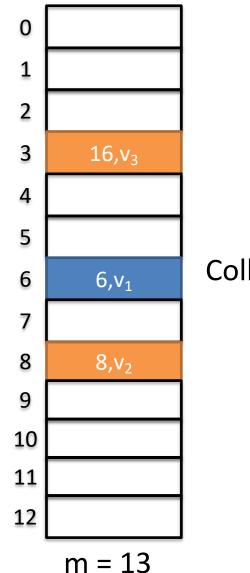
**Integer keys** 

Given table size m = 13

index = key.hashCode() % m

#### Example

- $put(6,v_1) = 6 \% 13 = 6$
- put(8,v<sub>2</sub>) = 8 % 13 = 8
- put(16,v<sub>3</sub>) = 16 % 13 = 3
- put(19,v<sub>4</sub>) = 19 % 13 = 6



Collision!

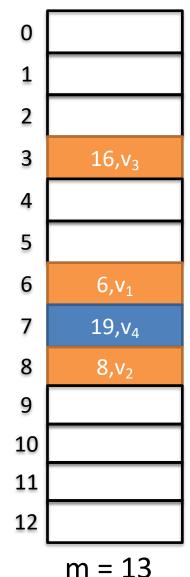
### Try next index if hashed index is full, repeat if next index is also full

#### **Integer keys**

Given table size m = 13

index = key.hashCode() % m

- $put(6,v_1) = 6 \% 13 = 6$
- put(8,v<sub>2</sub>) = 8 % 13 = 8
- put(16,v<sub>3</sub>) = 16 % 13 = 3
- put(19,v<sub>4</sub>) = 19 % 13 = 6



### To find items, probe until find Key or hit an empty space

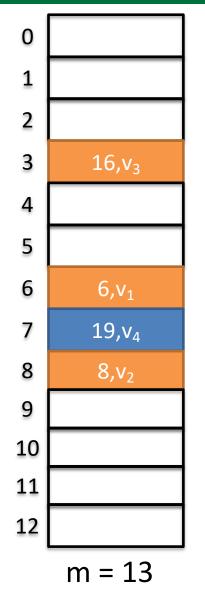
#### **Integer keys**

Given table size m = 13

index = key.hashCode() % m

#### Example

- $put(6,v_1) = 6 \% 13 = 6$
- $put(8,v_2) = 8 \% 13 = 8$
- $put(16,v_3) = 16\% 13 = 3$
- $put(19,v_4) = 19\% 13 = 6$
- get(19)



Insert at i+1 = 7

To find items later, hash to table index, then probe until find item or hit empty slot 48

# Deleting items is tricky, need to mark deleted spot as available but not empty

### **Problems deleting items under linear probing**

- Insert  $k_1$ ,  $k_2$ , and  $k_3$  where  $h(k_1)=h(k_2)=h(k_3)$
- All three keys hash to the same slot in this example
- $k_1$  in slot *i*,  $k_2$  in slot *i*+1,  $k_3$  in slot *i*+2
- Remove k<sub>2</sub>, creates hole at *i*+1
- Search for k<sub>3</sub>
  - Hash k<sub>3</sub> to *i*, slot *i* holds k<sub>1</sub>≠k<sub>3</sub>, advance to slot *i*+1
  - Find hole at *i+1*, assume k<sub>3</sub> not in hash table
- Can mark deleted spaces as available for insertion, and search skips over marked spaces
- This can be a problem if many deletes create many marked slots, search approaches linear time

### Clustering of keys can build up and reduce performance

### **Clustering problem**

- Long runs of occupied slots (clusters) can build up increasing search and insert time
- Clusters happen because empty slot preceded by t full slots gets filled with probability (t+1)/m, instead of 1/m (e.g., t keys can now fill open slot instead of just 1 key)
- Clusters can bump into each other exacerbating the problem

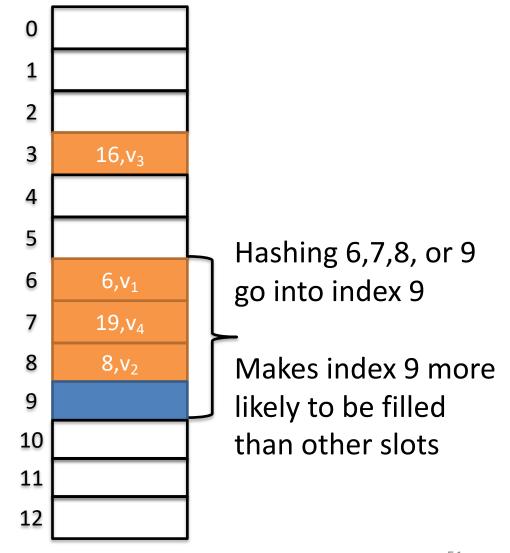
### Clustering of keys can build up and reduce performance

#### **Integer keys**

Given table size m = 13

index = key.hashCode() % m

- $put(6,v_1) = 6 \% 13 = 6$
- put(8,v<sub>2</sub>) = 8 % 13 = 8
- put(16,v<sub>3</sub>) = 16 % 13 = 3
- put(19,v<sub>4</sub>) = 19 % 13 = 6



### **Double hashing**

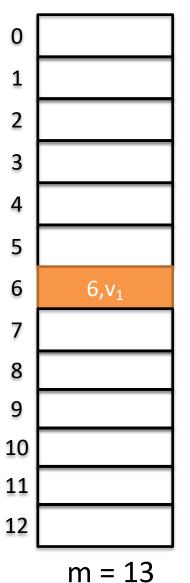
- <u>Big idea</u>: instead of stepping by 1 at each collision like linear probing, step by a different amount where the step size depends on the key
- Use two hash functions  $h_1$  and  $h_2$  to make a third h'
- $h'(k,p)=(h_1(k) + ph_2(k)) \mod m$ , where p number of probes
- First probe h<sub>1</sub>(k), p=0, then p incremented by 1 on each collision until space is found
- Result is a step by h<sub>2</sub>(k) on each collision (then mod m to stay inside table size), instead of 1
- Need to design hashes so that if  $h_1(k_1)=h_1(k_2)$ , then **unlikely**  $h_2(k_1)=h_2(k_2)$

h<sub>1</sub> same as before

#### **Integer keys**

Given table size m = 13 Compute  $h_2$  new hash function p = probe number (initially 0)  $h_1(key) = (key \%m)$   $h_2(key) = 1 + (key \% (m-1))$  $h'(k,p)=(h_1(k) + ph_2(k)) \% m$ 

Кеу	р	$h_1$	h <sub>2</sub>	h'
6	0	6	7	(6+0*7)%13 = 6

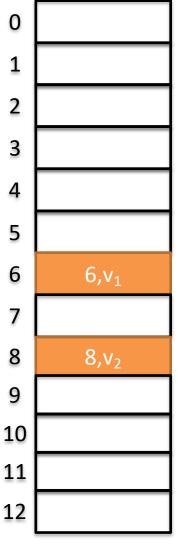


h<sub>1</sub> same as before

#### **Integer keys**

Given table size m = 13 Compute  $h_2$  new hash function p = probe number (initially 0)  $h_1(key) = (key \%m)$   $h_2(key) = 1 + (key \% (m-1))$  $h'(k,p)=(h_1(k) + ph_2(k)) \% m$ 

Кеу	р	$h_1$	h <sub>2</sub>	h'
6	0	6	7	(6+0*7)%13 = 6
8	0	8	9	(8+0*9)%13 = 8

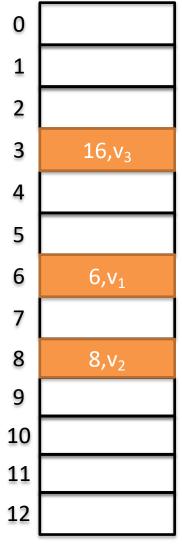


h<sub>1</sub> same as before

#### **Integer keys**

Given table size m = 13 Compute  $h_2$  new hash function p = probe number (initially 0)  $h_1(key) = (key \%m)$   $h_2(key) = 1 + (key \% (m-1))$  $h'(k,p)=(h_1(k) + ph_2(k)) \% m$ 

Кеу	р	$h_1$	h <sub>2</sub>	h'
6	0	6	7	(6+0*7)%13 = 6
8	0	8	9	(8+0*9)%13 = 8
16	0	3	5	(3+0*5)%13 = 3



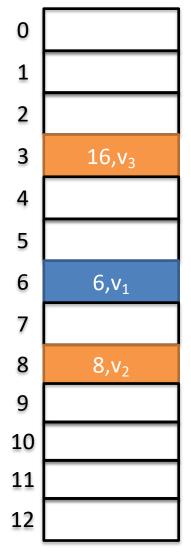
h<sub>1</sub> same as before

#### **Integer keys**

Given table size m = 13 Compute  $h_2$  new hash function p = probe number (initially 0)  $h_1(key) = (key \%m)$   $h_2(key) = 1 + (key \% (m-1))$  $h'(k,p)=(h_1(k) + ph_2(k)) \% m$ 

#### Example

Кеу	р	$h_1$	h <sub>2</sub>	h'
6	0	6	7	(6+0*7)%13 = <mark>6</mark>
8	0	8	9	(8+0*9)%13 = 8
16	0	3	5	(3+0*5)%13 = 3
19	0	6	8	(6+0*8)%13 = <mark>6</mark>



#### **Collision!**

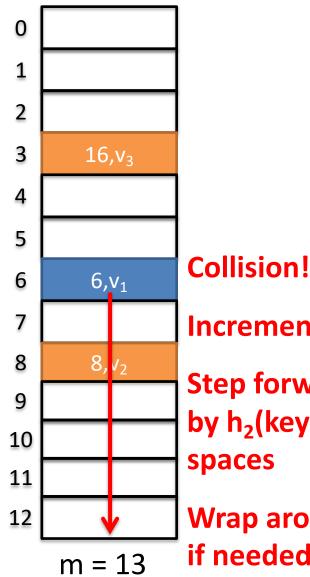
h<sub>1</sub> same as before

#### **Integer keys**

h<sub>2</sub> new hash function Given table size m = 13p = probe number (initially 0) Compute  $h_1(key) = (key \%m)$  $h_2(key) = 1 + (key \% (m-1))$  $h'(k,p)=(h_1(k) + ph_2(k)) \% m$ 

#### Example

Кеу	р	$h_1$	h <sub>2</sub>	h'
6	0	6	7	(6+0*7)%13 = 6
8	0	8	9	(8+0*9)%13 = 8
16	0	3	5	(3+0*5)%13 = 3
19	0	6	8	(6+0*8)%13 = 6
19	1	6	8	(6+ <mark>1*8</mark> )%13 = <b>1</b>



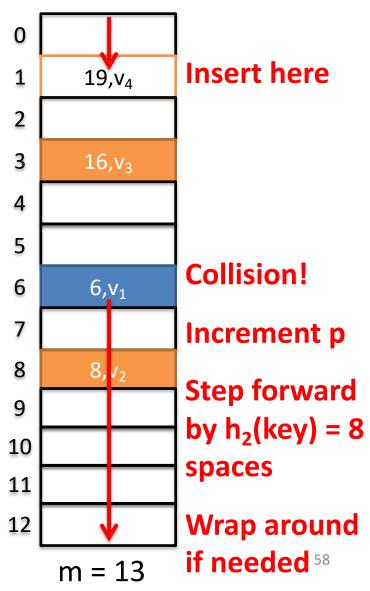
**Increment p Step forward by**  $h_2(key) = 8$ spaces Wrap around if needed 57

h<sub>1</sub> same as before

#### **Integer keys**

Given table size m = 13 Compute  $h_2$  new hash function p = probe number (initially 0)  $h_1(key) = (key \%m)$   $h_2(key) = 1 + (key \% (m-1))$  $h'(k,p)=(h_1(k) + ph_2(k)) \% m$ 

Кеу	р	$h_1$	h <sub>2</sub>	h'
6	0	6	7	(6+0*7)%13 = 6
8	0	8	9	(8+0*9)%13 = 8
15	0	2	4	(2+0*4)%13 = 2
19	0	6	8	(6+0*8)%13 = 6
19	1	6	8	(6+ <mark>1*8</mark> )%13 = <b>1</b>



# Run time degrades as $\lambda$ gets large, so keep $\lambda$ small by growing hash table

### **Expected insert and search time**

- Average number of probes is approximately  $1/(1-\lambda)$
- As λ ->1, expected number of probes becomes large, when λ small, number of probes approaches 1
- If table 90% full, then expect about 10 probes for unsuccessful search
- Successful search generally a little faster, about 2.5 probes (math on course web page and in book)
- Must grow table and <u>rehash</u> when copying to new table to keep the table sparsely populated or performance suffers

Sparsely populated table trades memory for speed

Operation	Expected run time	Notes
hash(k)	O(1)	<ul> <li>Math operations on key k to hash and compress, outputs 0m-1</li> <li>Constant time, does not depend on number of items in Set or Map</li> </ul>

Operation	Expected run time	Notes
hash(k)	O(1)	<ul> <li>Math operations on key k to hash and compress, outputs 0m-1</li> <li>Constant time, does not depend on number of items in Set or Map</li> </ul>
find(k)	O(1)	<ul> <li>Once have index of table due to hash:</li> <li>Chaining: traverse linked list O(λ) = O(1)</li> <li>Probing: probe until find O(1/(1-λ)) = O(1)</li> </ul>

Operation	Expected run time	Notes
hash(k)	O(1)	<ul> <li>Math operations on key k to hash and compress, outputs 0m-1</li> <li>Constant time, does not depend on number of items in Set or Map</li> </ul>
find(k)	O(1)	<ul> <li>Once have index of table due to hash:</li> <li>Chaining: traverse linked list O(λ) = O(1)</li> <li>Probing: probe until find O(1/(1-λ)) = O(1)</li> </ul>
get(k)	O(1+1) = O(1)	<ul> <li>Hash + find:</li> <li>chaining = O(1+λ) = O(1), probing = O(1+(1/(1-λ))) = O(1)</li> </ul>

Operation	Expected run time	Notes
hash(k)	O(1)	<ul> <li>Math operations on key k to hash and compress, outputs 0m-1</li> <li>Constant time, does not depend on number of items in Set or Map</li> </ul>
find(k)	O(1)	<ul> <li>Once have index of table due to hash:</li> <li>Chaining: traverse linked list O(λ) = O(1)</li> <li>Probing: probe until find O(1/(1-λ)) = O(1)</li> </ul>
get(k)	O(1+1) = O(1)	<ul> <li>Hash + find:</li> <li>chaining = O(1+λ) = O(1), probing = O(1+(1/(1-λ))) = O(1)</li> </ul>
put(k,v)	O(1) <u>+O(1)</u> O(1)	<ul> <li>Hash + find = O(1)</li> <li>Plus update or add element: <ul> <li>Chaining: update value or add at head O(1)</li> <li>Probing: store value in array O(1)</li> </ul> </li> </ul>

Operation	Expected run time	Notes
hash(k)	O(1)	<ul> <li>Math operations on key k to hash and compress, outputs 0m-1</li> <li>Constant time, does not depend on number of items in Set or Map</li> </ul>
find(k)	O(1)	<ul> <li>Once have index of table due to hash:</li> <li>Chaining: traverse linked list O(λ) = O(1)</li> <li>Probing: probe until find O(1/(1-λ)) = O(1)</li> </ul>
get(k)	O(1+1) = O(1)	<ul> <li>Hash + find:</li> <li>chaining = O(1+λ) = O(1), probing = O(1+(1/(1-λ))) = O(1)</li> </ul>
put(k,v)	O(1) <u>+O(1)</u> O(1)	<ul> <li>Hash + find = O(1)</li> <li>Plus update or add element: <ul> <li>Chaining: update value or add at head O(1)</li> <li>Probing: store value in array O(1)</li> </ul> </li> </ul> Assuming a small load
remove(k)	O(1) <u>+O(1)</u> O(1)	<ul> <li>Hash + find = O(1)</li> <li>Plus remove element:         <ul> <li>Chaining: update one pointer O(1)</li> <li>Probing: mark space empty O(1)</li> </ul> </li> <li>factor and uniform hashing, the core operations of HashSets and HashMaps are</li> </ul>
		constant time!