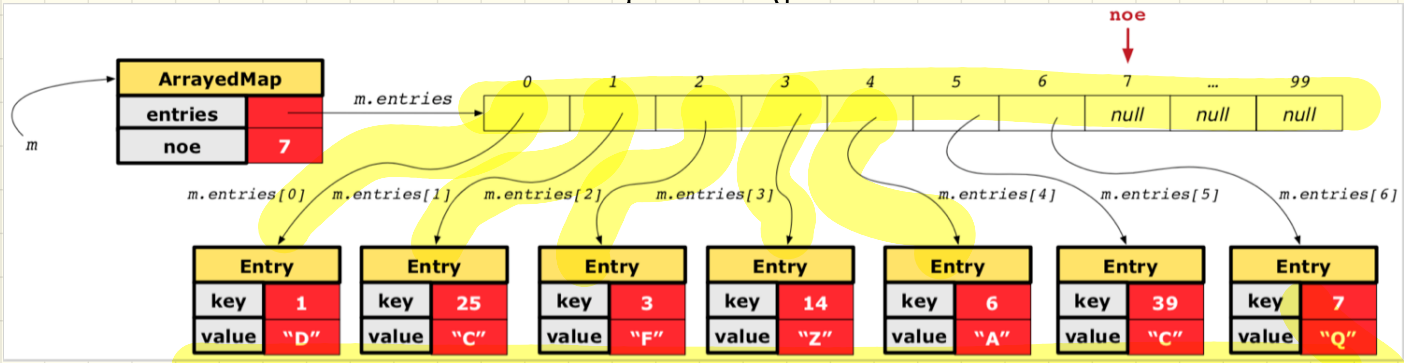
Wednesday Oct. 3

Lecture 9

Inefficient Implementation of Map: Array

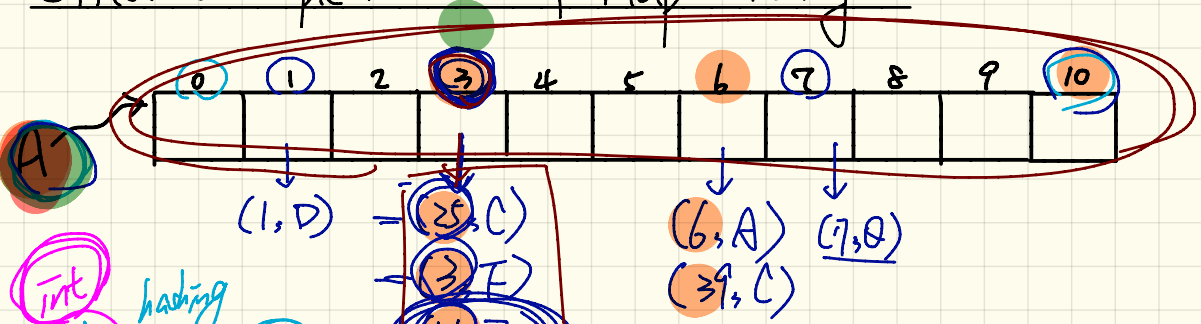


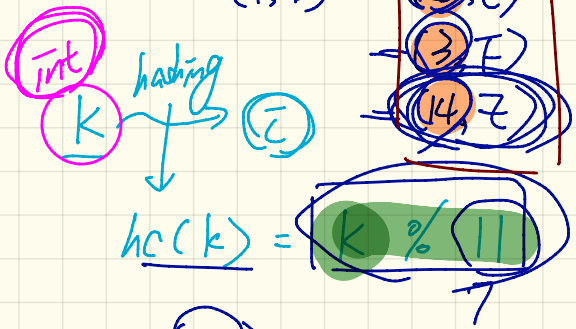
Running Time of Searching

≈ # of iterations for search

ENTRY	
(SEARCH) KEY	VALUE
1	D
25	C
3	F
14	Z
6	A
39	C
7	Q

Efficient Implementation of Map: Hashing





m.get(7)

m.get(14) → 14 % 11 ==> 3

Running Time of Searching:

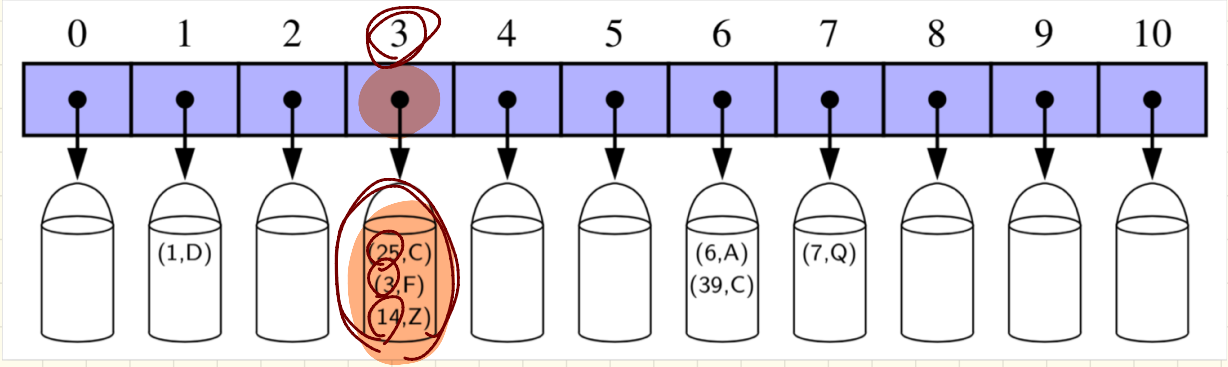
1. calculate hc(k)
2. indexing A[hc(k)]

hc(k)	(SEARCH) KEY	VALUE
1	1	D
3	25	C
3	3	F
3	14	Z
6	6	A
6	39	C
7	7	Q

ENTRY

Bucket Array

Entry	
(Search) Key	Value
1	D
25	C
3	F
14	Z
6	A
39	C
7	Q



How do you search through a bucket where entry keys have same hash code?

Implementing hashCode() for IntegerKey

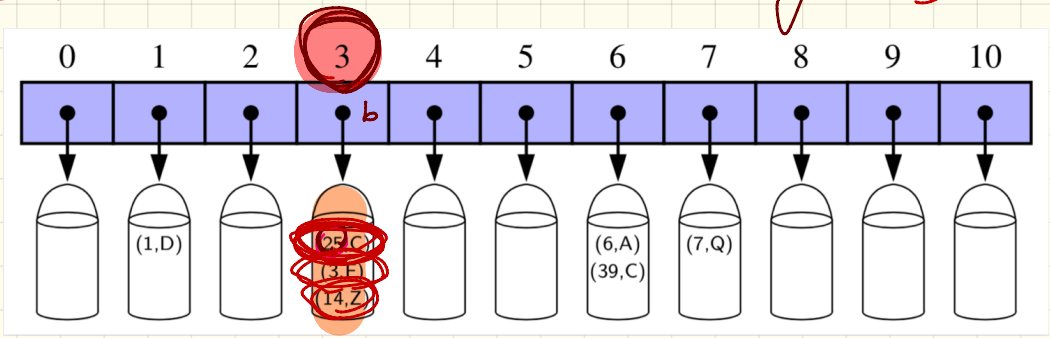
IK ik_25 = new IK(25);

```
1  public class IntegerKey {
2    private int k;
3    public IntegerKey(int k) { this.k = k; }
4    @Override
5    public int hashCode() { return k % 11; }
6    @Override
7    public boolean equals(Object obj) {
8      if(this == obj) { return true; }
9      if(obj == null) { return false; }
10     if(this.getClass() != obj.getClass()) { return false; }
11     IntegerKey other = (IntegerKey) obj;
12     return this.hashCode() == other.hashCode()
13   } }
```

for each entry in bucket 'b' {
  if(entry.key.equals(25)){
    return entry;
  }
}

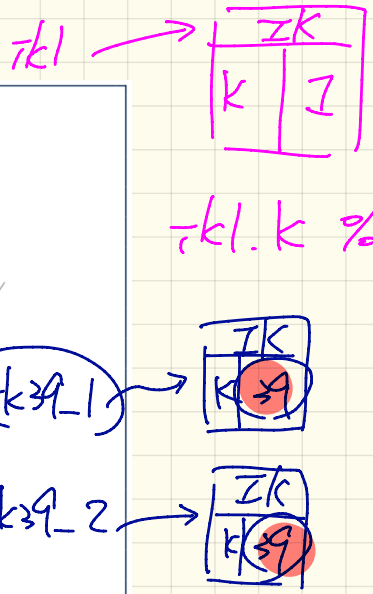
- m.get(25)
- m.get(3)

Q. Change L12 to this.hashCode() == other.hashCode() ?



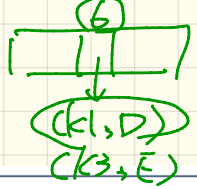
Testing Overridden Hash Function

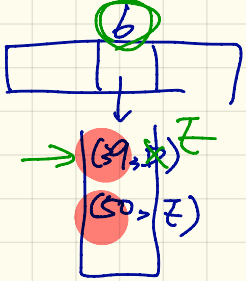
```java
@Test
public void testCustomizedHashFunction() {
  IntegerKey ik1 = new IntegerKey(1);
  /* 1 % 11 == 1 */
  assertTrue(ik1.hashCode() == 1);

  IntegerKey ik39_1 = new IntegerKey(39); /* 39 % 11 == 6 */
  IntegerKey ik39_2 = new IntegerKey(39);
  IntegerKey ik6 = new IntegerKey(6); /* 6 % 11 == 6 */

  assertTrue(ik39_1.hashCode() == 6);
  assertTrue(ik39_2.hashCode() == 6);
  assertTrue(ik6.hashCode() == 6);

  assertTrue(ik39_1.hashCode() == ik39_2.hashCode());
  assertTrue(ik39_1.equals(ik39_2));

  assertTrue(ik39_1.hashCode() == ik6.hashCode());
  assertFalse(ik39_1.equals(ik6));
}
```




```java
1  public class IntegerKey {
2    private int k;
3    public IntegerKey(int k) { this.k = k; }
4    @Override
5    public int hashCode() { return k % 11; }
6    @Override
7    public boolean equals(Object obj) {
8      if(this == obj) { return true; }
9      if(obj == null) { return false; }
10     if(this.getClass() != obj.getClass()) { return false; }
11     IntegerKey other = (IntegerKey) obj;
12     return this.k == other.k;
13   } }
```

Using hashCodes for HashTable






```java
@Test
public void testHashTable() {
Hashtable<IntegerKey, String> table = new Hashtable<>();
IntegerKey k1 = new IntegerKey(39);
IntegerKey k2 = new IntegerKey(39);
assertTrue(k1.equals(k2));
assertTrue(k1.hashCode() == k2.hashCode());
table.put(k1, "D");
assertTrue(table.get(k2).equals("D"));
}
```

Ik k3 = new Ik(50); hc 6

table.put(k3, "E");

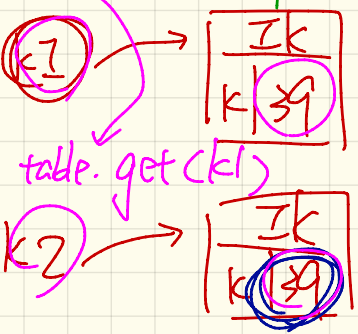
k1

k2. hashCode() 6

table.put(k2, "E");

k1. equals(k3) F

k1. hc() == k3. hc() T




```java
1  public class IntegerKey {
2    private int k;
3    public IntegerKey(int k) { this.k = k; }
4    @Override
5    public int hashCode() { return k % 11; }
6    @Override
7    public boolean equals(Object obj) {
8      if(this == obj) { return true; }
9      if(obj == null) { return false; }
10     if(this.getClass() != obj.getClass()) { return false; }
11     IntegerKey other = (IntegerKey) obj;
12     return this.k == other.k;
13   } }
```

Using Default Hash Function for HashTable

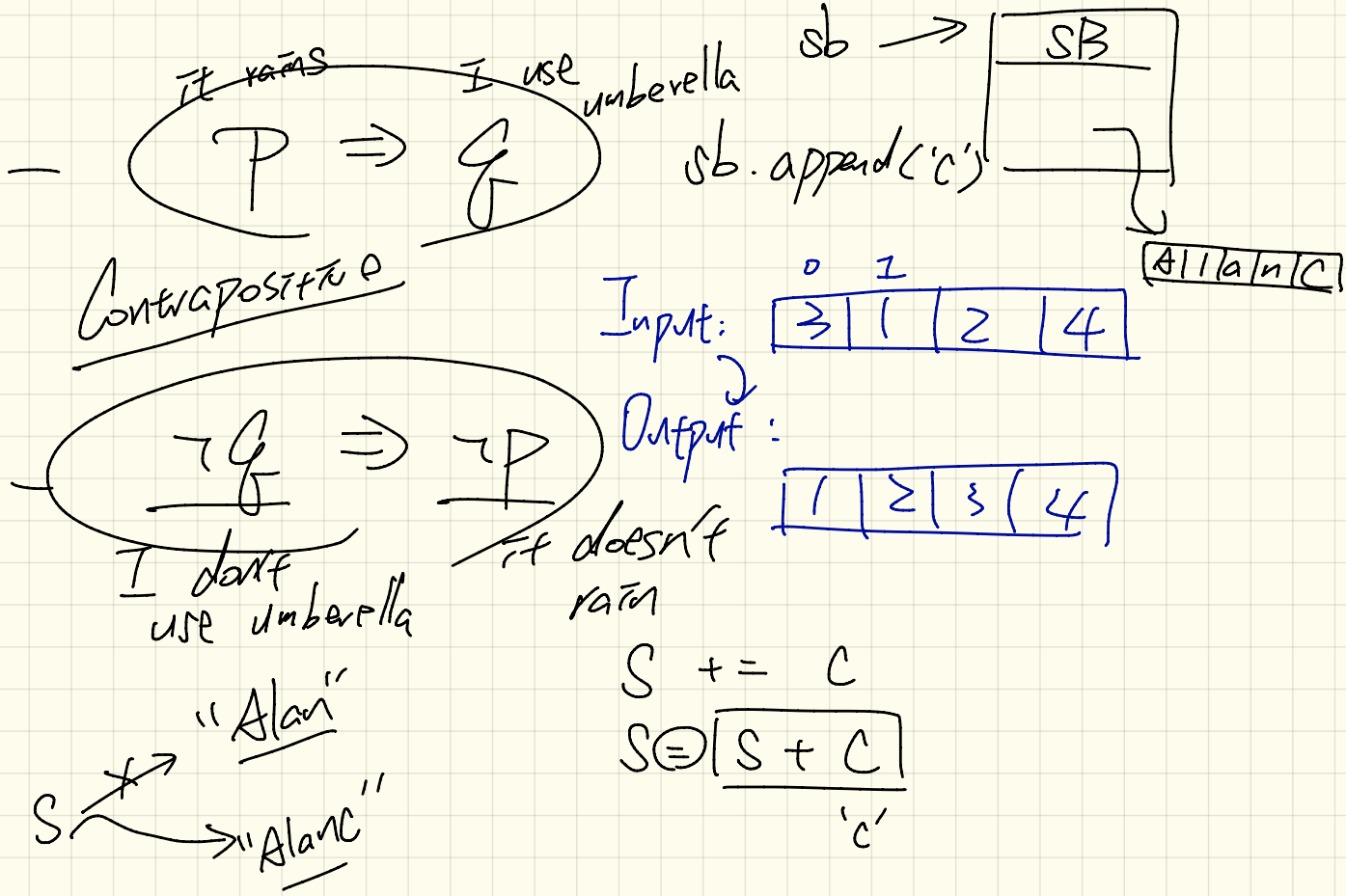
```
@Test
public void testDefaultHashFunction() {
  IntegerKey ik39_1 = new IntegerKey(39);
  IntegerKey ik39_2 = new IntegerKey(39);
  assertTrue(ik39_1.equals(ik39_2));
  assertTrue(ik39_1.hashCode() != ik39_2.hashCode()); }
@Test
public void testHashTable() {
  Hashtable<IntegerKey, String> table = new Hashtable<>();
  IntegerKey k1 = new IntegerKey(39);
  IntegerKey k2 = new IntegerKey(39);
  assertTrue(k1.equals(k2));
  assertTrue(k1.hashCode() != k2.hashCode());
  table.put(k1, "D");
  assertTrue(table.get(k2) == null); }
```

Contract for hashing:

$hc(k_1) \neq hc(k_2) \Rightarrow \neg\, k_1.equals(k_2)$

Say: k1 ≠ k2

```
public class IntegerKey {
  private int k;
  public IntegerKey(int k) { this.k = k; }
  /* hashCode() inherited from Object NOT overridden. */
  @Override
  public boolean equals(Object obj) {
    if(this == obj) { return true; }
    if(obj == null) { return false; }
    if(this.getClass() != obj.getClass()) { return false; }
    IntegerKey other = (IntegerKey) obj;
    return this.k == other.k;
  } }
```


It rains
I use umberella
P ⇒ Q
Contrapositive
¬Q ⇒ ¬P
I don't use umberella
It doesn't rain
"Alan"
"Alanc"
sb
SB
sb.append('c')
Allanc
Input:
Output:
0 1
3 1 2 4
1 2 3 4
S += C
S = S + C
'c'