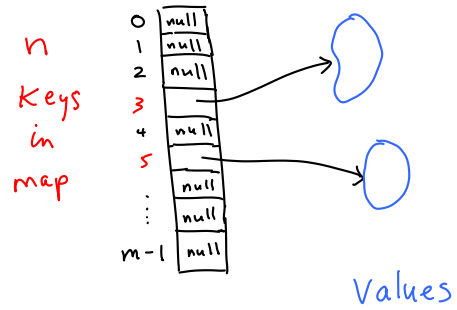


lecture 27

hashing

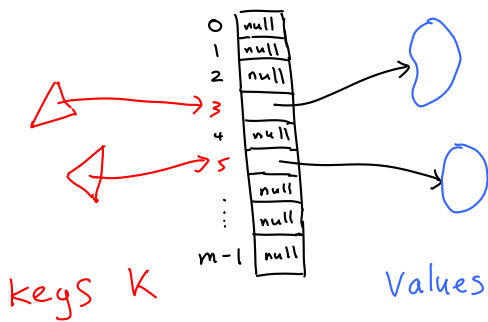
Direct Addressing

possible keys $K = \{0, 1, \dots, m-1\}$



$O(1)$ access

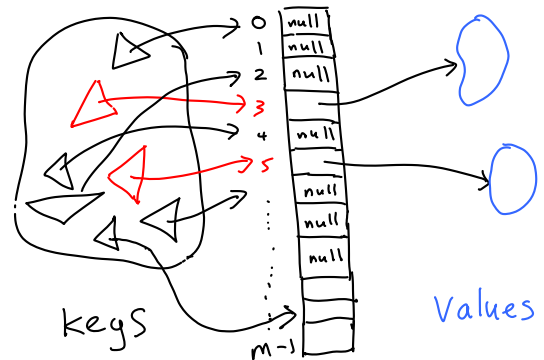
What if keys are not ints?
Let's map them to ints.



Need to map: $K \rightarrow \{0, 1, \dots, m-1\}$

Hash Function

$$h: K \rightarrow \{0, 1, \dots, m-1\}$$



Hash function, $h: K \rightarrow \{0, \dots, m-1\}$

$$h: K \xrightarrow{\text{hash coding}} \{\text{integers}\} \xrightarrow{\text{compression}} \{0, 1, \dots, m-1\}$$

hash coding compression
hash codes hash values



$$\text{hash function} = \text{compression} \circ \text{hash coding}$$

↑
composition of functions

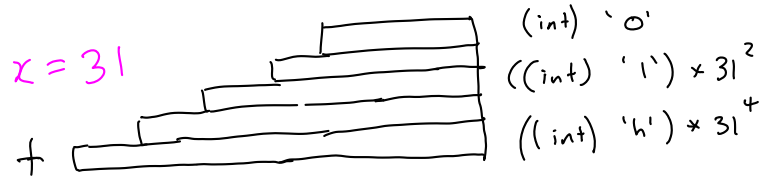
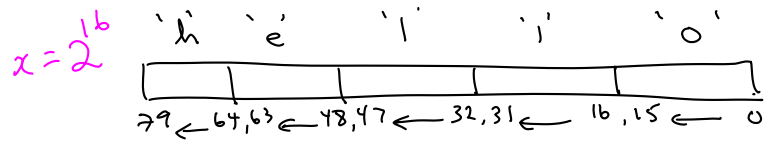
Example (hash code): String

$$h: K \rightarrow \{\text{integers}\} \rightarrow \{0, 1, \dots, m-1\}$$

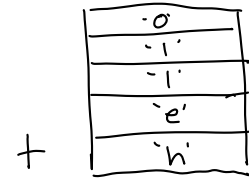
Let s be a string $s[0] s[1] \dots s[s.length-1]$

$$s.hashCode() = \sum_{i=0}^{s.length-1} s[i] x^{length-i-1}$$

e.g. $x = 2^{16}, 31, 1$



$x = 1$



Note: for $x = 1, 31$,
two strings can
have the
same hash code

ASIDE: Horner's algorithm

Efficient Evaluation of Polynomial

$$s.hashCode() = \sum_{i=0}^{s.length-1} s[i] x^{length-i-1}$$

Avoid computing $x, x^2, x^3, \dots, x^{length-1}$

Horner's Algorithm

$$\begin{aligned} & a_n x^n + \dots + a_2 x^2 + a_1 x + a_0 \\ &= (a_n x^{n-1} + \dots + a_2 x + a_1) x + a_0 \\ &= ((\dots + a_2) x + a_1) x + a_0 \\ &= (((\dots (a_n x + a_{n-1}) x + a_{n-2}) \\ &\quad \text{etc} \dots + a_1) x + a_0) \end{aligned}$$

Horner's algorithm

$$\text{sum} \leftarrow a_n$$

for $i = n-1$ to 0

$$\text{sum} \leftarrow \text{sum} * x + a_i$$

// only n multiplies and
 n adds are needed.

Example (compression)

$$h: K \rightarrow \{\text{integers}\} \rightarrow \{0, 1, \dots, m-1\}$$

hash Code Compression

$$x \bmod m$$

(often take m prime)

hash code.

41
16
25
21
36
35
53

$m = 7$

hashcode % 7

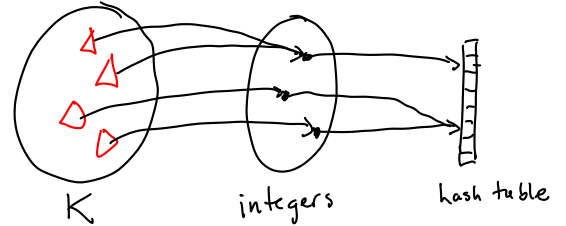
6
2
4
0
1
0
4

Collisions

Two keys map to the same slot in the hash table:

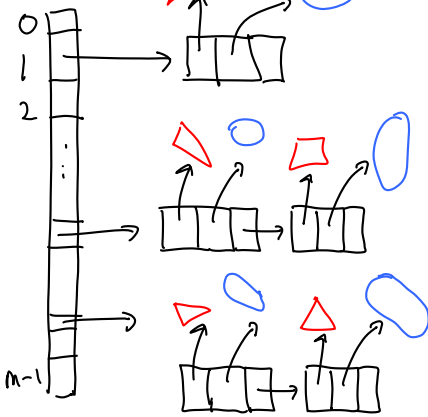
either {

- two keys have same hash code
- two keys have different hash codes but get compressed to same hash value.



Hash Table Implementation

m slots in table



Using an array of linked lists is called

"separate chaining".

Each linked list is called a "bucket".

A "perfect hash function" puts at most one entry (k, v) in each bucket.

A good hash function puts at most a few entries in each bucket. $\Rightarrow O(1)$ access

(Note: you can change the hash code or the compression function)

Load factor

$$\equiv \frac{\text{number of } (k, v) \text{ pairs in map}}{\text{number of buckets } (m)}$$

Rule of thumb:

keep load factor $< \frac{3}{4}$

Java API

```
interface Map <K, V> {  
    V put (K, V)  
    V get (K)  
    ...  
}
```

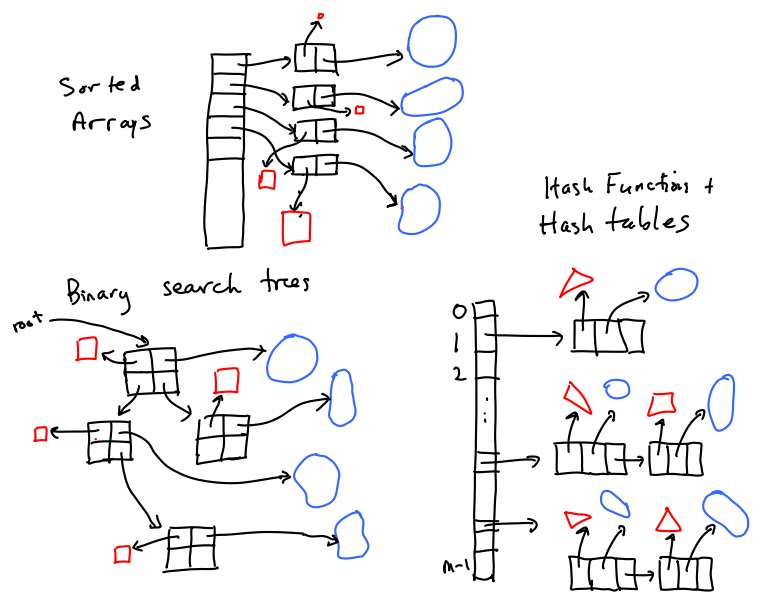
In fact, the parameter is "Object".

Java API

```

class HashMap <K, V> {
    V put (K, V)
    V get (K)
    ...
}
    
```

↑
In fact, the parameter is "Object".



find

Sorted Arrays	$O(\log n)$
Binary Search Trees	$O(n)$
average case	$\rightarrow \{O(\log n)\}$
Hash maps / Hash tables	$O(1)$