

# CS 240

## Data Structures and Algorithms I

Alex Vondrak

`ajvondrak@csupomona.edu`

November 30, 2011

# Hash Tables

- Take the idea of a hash function storing objects in an array...
- ... But use two distinct parameters (the **key** and **value**)

Before

```
data[hash(i)] = i;
```

After

```
data[hash(k)] = v;
```

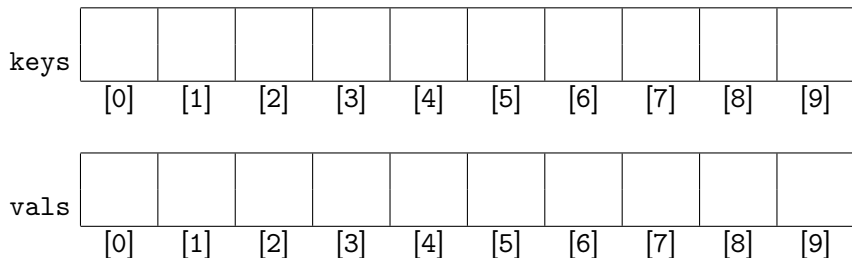
(It gets a little trickier than this, though.)

# Hash Tables

## Example

Suppose we have the following key/value pairs:

38  $\mapsto$  31   16  $\mapsto$  14   47  $\mapsto$  15   15  $\mapsto$  92   53  $\mapsto$  65   90  $\mapsto$  35   29  $\mapsto$  89



# Problems

## Definition

A **perfect** hash function maps every key to a unique index.

## Definition

A hash **collision** occurs when two keys get hashed to the same index.

- Designing a hash function takes a lot of consideration
  - Uniformity
  - Efficiency
  - Predictability of results
- What about non-integer keys?
  - Each Object comes with a hashCode method.
  - Roughly: `data[hash(k.hashCode())] = v`

# Open Address Hashing

**Idea:** when there's a collision, search ahead for a vacant spot.

## Example

Suppose we have the following key/value pairs:

38  $\mapsto$  31   98  $\mapsto$  14   48  $\mapsto$  15   15  $\mapsto$  92   53  $\mapsto$  65   90  $\mapsto$  35   29  $\mapsto$  89

keys										
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
vals										
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

# Chained Hashing

**Idea:** handle hash collisions by storing linked lists in the array

## Example

Suppose we have the following key/value pairs:

38  $\mapsto$  31    98  $\mapsto$  14    48  $\mapsto$  15    15  $\mapsto$  92    53  $\mapsto$  65    90  $\mapsto$  35    29  $\mapsto$  89

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
vals										

# Time Analysis of Hashing

- **Worst-case:** everything gets put at the same index (always a collision)
  - Searching for the proper key/value requires linear probing— $O(n)$
- **Average case:** with a proper hash function, collisions are reduced
  - ... But, it's difficult to analyze

## Analysis (Open Addressing)

In open-address hashing with linear probing, a nonfull hash table, and no removals, the average number of table elements examined in a successful search is approximately

$$\frac{1}{2} \left( 1 + \frac{1}{1 - \alpha} \right)$$

where the **load factor**  $\alpha = \frac{\# \text{ elements stored in table}}{\text{size of array}}$

# Time Analysis of Hashing

- **Worst-case:** everything gets put at the same index (always a collision)
  - Searching for the proper key/value requires linear probing— $O(n)$
- **Average case:** with a proper hash function, collisions are reduced
  - ... But, it's difficult to analyze

## Analysis (Chained)

In chained hashing, the average number of table elements examined in a successful search is approximately

$$1 + \frac{\alpha}{2}$$

Note here, though, that  $\alpha$  can be  $> 1$ .