Hashing

Linda Deng

Agenda

- Motivation
- Hashing and its application
- Runtime
  - Resize
- 2 key warnings for hashing
- Hashcode

Motivation

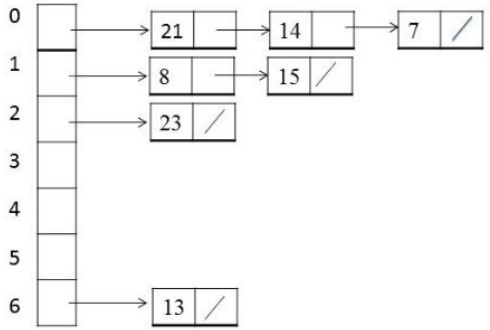
- Implementation of sets and maps
  - Array: quick access but inflexible size
  - Linked List: slow search & insertion
  - Search tree: items need to be comparable
- Hashing
  - Combine the advantages of linked lists and arrays

Hashing

Trash-ing

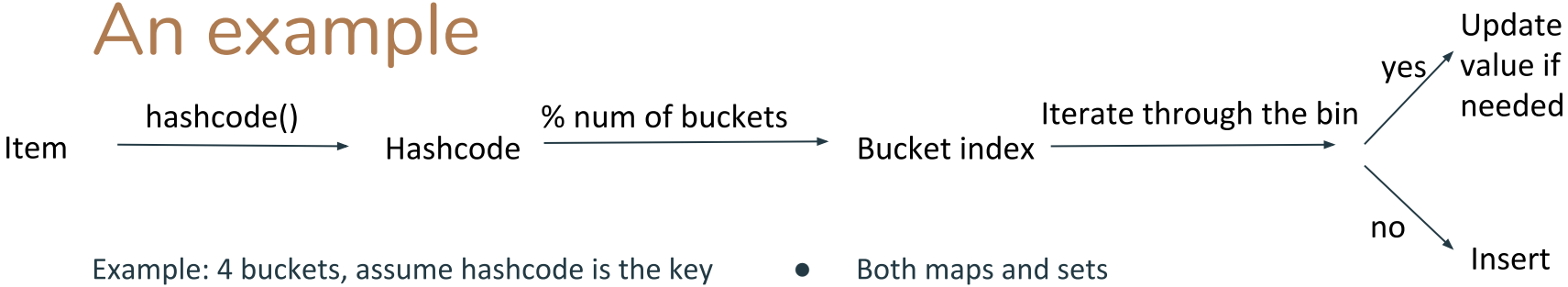
1. Identify the type of trash
2. Throw it in the correct trash can

Hash-ing



1. Identify the label of the item (hashcode)
2. Place it in the correct bucket based on hashcode & number of bins
    a. If doesn't exist, insert
    b. If exists, update the value or no action

An example



Example: 4 buckets, assume hashcode is the key

put(1, A)

put(3, B)

put(5, C)

put(3,D)

- Both maps and sets
- Maps
  - Use “key” to compute hashcode
- Set
  - Use the data itself to compute hashcode

Runtime

Key observation: only care about the linked list in the bin that the item maps to

	add(x)	contains(x)
Worst case	Θ(length of longest list)	Θ(length of longest list)

Shortest length: Θ(N/M) = Θ(N) —> M is a fixed constant

Longest length: Θ(N)

Resize

- Fixed M buckets —> increasing # of buckets
- Load factor (N/M)
  - N/M >= 1.5, double M
- Θ(length of longest list) = Θ(n/m) = Θ(1)
  - Assume even distribution
- Careful: must remap ALL elements when increasing M
  - # buckets changes = # mod changes = bucket index changes

	add(x)	contains(x)
Runtime (Avg.)	Θ(1)	Θ(1)

2 important warnings

1. Never store objects that can change state in HashSet or HashMap
2. Never override equals() without overriding hashCode()

* An important assumption of hash-based collection is that an object’s hash value won’t change while it was used as a key in the collection.

Hashcode

Valid Hashcode	Good Hashcode
1. Consistency a. hashCode() on the same object remains the same	1. Minimizes or completely eliminates collisions
2. Equality constraint a. Equal objects same hashCode()	