Hashing

Operations: Insert(x), Delete(x), Find(x)
(x: Key for Delete; x: Key for Find)

Applies to a class with a "key" field.
No two objects have the same key. No ordering is assumed.

Idea: Hash function $h(x) \longrightarrow \mathbb{Z}^+ : [0, n-1]$

(x: Key; $[0, n-1]$: Table size = n)

Choose n to be proportional to the number of elements in hash table.

Preferably, ~~each element~~ for any two keys x, y:

$$Pr\{h(x) = h(y)\} = 1/n .$$

In ideal situation: given n elements,

$$\#\text{ of elements with } \{h(x) = i\} = O(1).$$

$h(x) = h(y)$ —— Collision.

~~What does~~ How do you select $h()$?

Use a pseudorandom function —

$h(1), h(2), h(3), \ldots$ ← indistinguishable statistically from a random sequence

Yet deterministic — derived from math related to cryptography.

Ideally, easy to calculate.

Insert(Key, x)
  $i \leftarrow h(key)$
  // Key is a field of x.
  Place x in index i of hash table

Find(key)
  $i \leftarrow h(key)$
  Search for object with "key" in its key field. in index i of hash table

Delete(key)
  $i \leftarrow h(key)$
  Find x in index i of hash table with x.key = key, delete it.

Collision Resolution

(1) Open hashing — Separate chaining — used by Java's HashSet, HashMap

Hash table — array of linked lists.

Insert(x): // x has "key" field.

```
i ← h(x.key)
~~Table[i].add(x)~~. if (!Table[i].contains(x))
                        {Table[i].add(x);}
```

Find(x): `i ← h(x.key)`
`Table[i].get(x.key)`

Delete(x): `i ← h(x.key)`
`Table[i].remove(x.key)`.

Java's hash tables

HashSet<T> — Set of elements stored in a hash table

For user defined classes, hashCode(), equals() should be defined:

x.equals(y) = true ⇒ x.hashCode() == y.hashCode()

After x is inserted into a set, if x.key is changed, set may become inconsistent.

HashMap<K,V> : K - Key, V - Value — choice when "V" is not inherent property of "K".

Implementation: Separate chaining.

Separate Chaining: Each entry in Hash Table is a linked list.
All objects with equal hashCode() will be stored in the same list.