

Hash Table ADT

Prof Bill - Apr 2018

A hash table provides $O(1)$ access to storing and searching name-value pairs.

Hash table operations are: create, put(v, k), v get(k), v remove(k)

Pseudocode is provided for 2 hash table implementations: chaining and linear probing.

Chaining

Easy one first... chaining maintains a linked list of key-value pairs with the same hash code.

```
// create array for table data and save table settings
create( size=16, load_factor=0.75) // these are Java defaults
array = new item[size]
init each array[i].list = null
num_items = 0
load_factor = load_factor // save load factor

// insert key-value pair into the hash table
put( value, key)
if num_items/array.length >= load_factor then resize_rehash()
hash = hashcode( key) % array.length
if ( array[hash].list == null) then array[hash].list = new list
array[hash].list.add( value, key); num_items++

// get (and return) value associated with this key; if not present, return null
value get( key)
hash = hashcode( key) % array.length
v = null
if array[hash].list != null
  for each item in array[hash].list
    if item.key == key
      v = item.value; break loop
return v

// remove (key, value) pair from hash table if present
value remove( key)
hash = hashcode( key) % array.length
v = null
if array[hash].list != null
  for each item in array[hash].list
    if item.key == key
      remove item from list; num_items--; break loop
return v
```

Linear probing

In linear probing, when a collision happens, we look to the next slot in the array for an opening. This makes get and remove a little tricky.

```
// create array for table data and save table settings
create( size=16, load_factor=0.75) // these are Java defaults
array = new item[size]
init each array[i].rm_flag = null
num_items = 0
load_factor = load_factor // save load factor

// insert key-value pair into the hash table
put( value, key)
if num_items/array.length >= load_factor then resize_rehash()
hash = hashCode( key) % array.length
while( array[hash] != null)
    hash++ % array.length // circular increment
array[hash] = (value, key); num_items++

// get (and return) value associated with this key; if not present, return null
value get( key)
hash = hashCode( key) % array.length
while( array[hash] != null || array[hash].rm_flag)
    if array[hash] != null && array[hash].key == key
        return array[hash].value
    hash = hash % array.length
return null

// remove (key, value) pair from hash table if present
value remove( key)
hash = hashCode( key) % array.length
while( array[hash] != null || array[hash].rm_flag)
    if array[hash] != null && array[hash].key == key
        array[hash] = null; num_items-- // remove item
        return array[hash].value
    hash = hash % array.length
return null

// increase array size, then rehash all items into new array
resize_rehash()
array2 = new item[2*array.length] // double or 10x, whatever
for each item in array
    put( key, value) into array2
array = array2
```