Data Structures

Implementing Hash Tables

CS284

Implementing Hash Tables

Open Addressing

Chaining

Class HashTableOpen

Data fields

```
// The table itself
private Entry<K,V>[] table;
private static final int START_CAPACITY = 101;
// max load factor
private double LOAD_THRESHOLD = 0.75;
// no. of keys in table excluding those that were deleted
private int numKeys;
// no. of deleted keys
private int numDeletes;
// Special object to indicate deletion
private final Entry<K, V> DELETED =
                        new Entry<K, V>(null, null);
```

Class HashTableOpen

```java
/** Hash table implementation using open addressing. */
public class HashtableOpen<K, V> implements KWHashMap<K, V> {
        // Data Fields
        private Entry<K, V>[] table;
        private static final int START_CAPACITY = 101;
        private double LOAD_THRESHOLD = 0.75;
        private int numKeys;
        private int numDeletes;
        private final Entry<K, V> DELETED =
                new Entry<K, V>(null, null);

        // Constructor
        public HashTableOpen() {
                table = new Entry[START_CAPACITY];
        }

        // Insert inner class Entry<K, V> here.
        ...
```

Class HashTableOpen (cont.)

Algorithm for `HashtableOpen.find(Object key)`

```
index = key.hashCode() % table.length.
if (index<0)  add table.length.
while table[index] is not empty and the key is not at table[in
          index++.
          if (index>=table.length) { index = 0. }
Return the index.
```

Class HashTableOpen (cont.)

```java
private int find(Object key) {
  // Calculate the starting index.
  int index = key.hashCode() % table.length;
  if (index < 0)
    index += table.length; // Make it positive.

    // Increment index until an empty slot is reached
    // or the key is found.
    while ( (table[index] != null)
         && (!key.equals(table[index].key))) {
    index++;
    // Check for wraparound.
    if (index >= table.length)
      index = 0; // Wrap around.
  }
  return index;
}
```

Class HashTableOpen (cont.)

Algorithm for `get(Object key)`

```
Find the first table element that is empty or the table element
if (the table element found contains the key)
    return the value at this table element.
else
    return null.
```

Class HashTableOpen (cont.)

```java
public V get(Object key) {
  // Find the first table element that is empty
  // or the table element that contains the key.
  int index = find(key);

  // If the search is successful, return the value.
  if (table[index] != null)
    return table[index].value;
  else
    return null; // key not found.
}
```

Class HashTableOpen (cont.)

Algorithm for `HashtableOpen.put(K key, V value)`

```
Find the first table element that is empty or the table element
if (an empty element was found)
    insert the new item and increment numKeys.
    check for need to rehash.
    return null.
The key was found. Replace the value associated with this tabl
```

Class HashTableOpen (cont.)

```
public V put(K key, V value) {
   int index = find(key);
   // If an empty element was found, insert new entry.
   if (table[index] == null) {
     table[index] = new Entry <K,V> (key, value);
     numKeys++;
        // Check whether rehash is needed.
     double loadFactor =
         (double) (numKeys + numDeletes) / table.length;
     if (loadFactor > LOAD_THRESHOLD)
       rehash();
     return null;
   }
   // assert: table element that contains the key was found.
   // Replace value for this key.
   V oldVal = table[index].value;
   table[index].value = value;
   return oldVal;
}
```

Class HashTableOpen (cont.)

Algorithm for `remove(Object key)`

```
Find the first table element that is empty or the table elemen
if (an empty element was found)
    return null.
Key was found. Remove this table element by setting it to refe
Return the value associated with this key.
```

Class HashTableOpen (cont.)

```
@Override
 public V remove(Object key) {
     int index = find(key);
     if (table[index] == null) {
         return null;
     }
     V oldValue = table[index].value;
     table[index] = DELETED;
     numKeys--;
     return oldValue;
 }}
```

Class HashTableOpen (cont.)

Algorithm for `HashtableOpen.rehash`

```
Allocate a new hash table that is at least double the size and
Reset the number of keys and number of deletions to 0.
Reinsert each table entry that has not been deleted in the new
```

Class HashTableOpen (cont.)

```java
private void rehash() {
    // Save a reference to oldTable.
    Entry <K,V> [] oldTable = table;
    // Double capacity of this table.
    table = new Entry[2 * oldTable.length + 1];

    // Reinsert all items in oldTable into expanded table.
    numKeys = 0;
    numDeletes = 0;
    for (int i = 0; i < oldTable.length; i++) {
      if ( (oldTable[i] != null) && (oldTable[i] != DELETED))
        // Insert entry in expanded table
        put(oldTable[i].key, oldTable[i].value);
      }
    }
```

Implementing Hash Tables

Open Addressing

Chaining

Class HashTableChain

```
public class HashtableChain <K,V>
    implements KWHashMap <K,V> {
  /** The table */
  private LinkedList <Entry <K,V>> [] table;

  /** The number of keys */
  private int numKeys;

  /** The capacity */
  private static final int CAPACITY = 101;

  /** The maximum load factor */
  private static final double LOAD_THRESHOLD = 3.0;
```

Class HashTableChain (cont.)

```
/** Insert inner class Entry <K,V> here */

 // Constructor
 public HashtableChain() {
   table = new LinkedList[CAPACITY];
 }

 public int size() {
  return numKeys;
 }

 public boolean isEmpty() {
  return numKeys == 0;
 }
```

Class HashTableChain (cont.)

Algorithm for `HashtableChain.get(Object key)`

```
index = key.hashCode() % table.length.
if (index<0)
  add table.length.
if (table[index]==null)
    key is not in the table; return null.
For each element in the list at table[index]
    if that element's key matches the search key
             return that element's value.
key is not in the table; return null.
```

Class HashTableChain (cont.)

```
public V get(Object key) {
  int index = key.hashCode() % table.length;
  if (index < 0)
    index += table.length;
  if (table[index] == null)
    return null; // key is not in the table.
 // Search the list at table[index] to find the key.
  for (Entry <K,V> nextItem : table[index]) {
    if (nextItem.key.equals(key))
      return nextItem.value;
  }

  // assert: key is not in the table.
  return null;
}
```

Class HashTableChain (cont.)

Algorithm for `HashtableChain.put(K key, V value)`

```
Set index to key.hashCode() % table.length.
if (index<0) add table.length.
if (table[index]=null)
    create a new linked list at table[index].
Search the list at table[index] to find the key.
if (search is successful)
    replace the value associated with this key.
    return the old value.
else
    insert the new key-value pair in the linked list located a
    increment numKeys.
    if the load factor exceeds the LOAD_THRESHOLD
            Rehash.
    return null.
```

Class HashTableChain (cont.)

```
public V put(K key, V value) {
  int index = key.hashCode() % table.length;
  if (index < 0)
    index += table.length;
  if (table[index] == null) {
    // Create a new linked list at table[index].
    table[index] = new LinkedList < Entry < K, V >> ();
  }

  // Search the list at table[index] to find the key.
  for (Entry < K, V > nextItem : table[index]) {
    // If the search is successful, replace the old value.
    if (nextItem.key.equals(key)) {
      // Replace value for this key.
      V oldVal = nextItem.value;
      nextItem.setValue(value);
      return oldVal;
    }
  }
  ... continued ...
```

Class HashTableChain (cont.)

```
   ... continued from above ...
  // assert: key is not in the table, add new item.
  table[index].addFirst(new Entry <K,V> (key, value));
  numKeys++;
  if (numKeys > (LOAD_THRESHOLD * table.length))
    rehash();
  return null;
}
```

Class HashTableChain (cont.)

Algorithm for `HashtableChain.remove(Object key)`

```
index = key.hashCode() % table.length.
if (index<0) add table.length.
if table[index] is null
    key is not in the table; return null.
Search the list at table[index] to find the key.
if the search is successful
    remove the entry with this key and decrement numKeys.
    if the list at table[index] is empty
            Set table[index] to null.
    return the value associated with this key.
The key is not in the table; return null.
```

Class HashTableChain (cont.)

```java
public V remove(Object key) {
int index = key.hashCode() % table.length;
if (index < 0) {
    index += table.length;
}
if (table[index] == null) {
    return null; // key is not in table
}
for (Entry<K, V> entry : table[index]) {
    if (entry.getKey().equals(key)) {
        V value = entry.getValue();
        table[index].remove(entry);
        numKeys--;
        if (table[index].isEmpty()) {
            table[index] = null;
        }
        return value;
    }
}
return null;
}
```

HashMap of HashMap, etc.

Grouping names by the first character (GroupName.java):

```
public void insert_keyval() {
    HashMap<Character, ArrayList<String>> dict = new
    HashMap<Character, ArrayList<String>>();
    String[] all_strings = new String[] {"marry",
    "matt", "nancy", "nelson", "pete", "patrick"};

    for (int i = 0; i < all_strings.length; i ++) {
           String this_name = all_strings[i];
           Character first_char = this_name.charAt(0);
```

HashMap of HashMap, etc.

```
  if (dict.containsKey(first_char) == true) {
     ArrayList<String> updated_list = dict.get(first_char);
     updated_list.add(this_name);

     dict.put(first_char, updated_list);
  }

  else {
      ArrayList<String> new_list = new ArrayList<String>();
      new_list.add(this_name);

      dict.put(first_char, new_list);
  }
}
```

HashMap of HashMap, etc.

```
    ArrayList<Character> sorted_keys = new ArrayList<Character
    (dict.keySet());
    Collections.sort(sorted_keys);

    for (Character this_char: sorted_keys) {
        System.out.println("starting from " + this_char + ":")
        for (String each_name: dict.get(this_char)){
            System.out.print(each_name + ",");
        }
        System.out.println("\n");
    }
}
```

Demo: insert_keyval and insert_keyval2