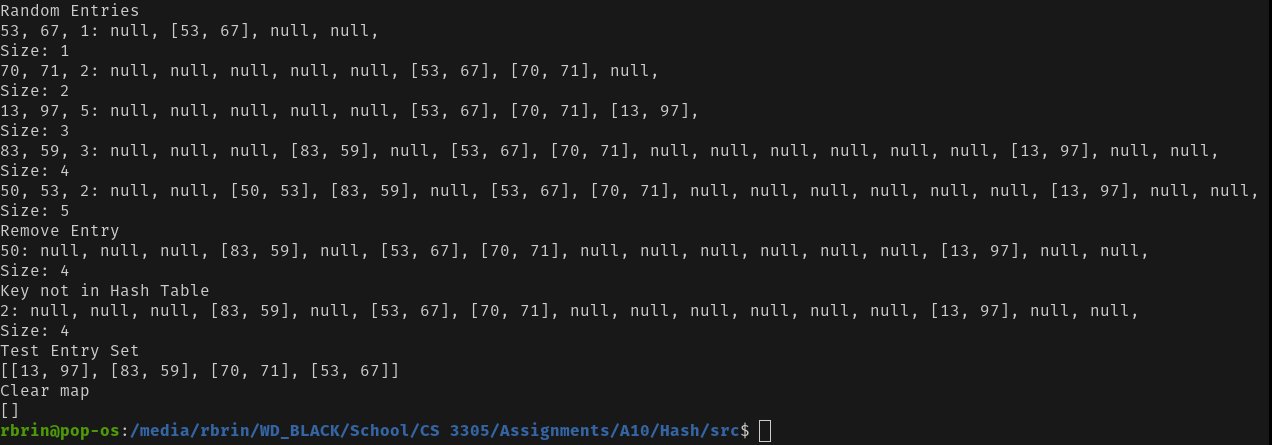
# Assignment 10 - Hashing

## Ryan Brinson

## 11/27/2023

### Output:



### Code:

// Name: Ryan Brinson

// Class: CS 3305 W04

// Term: Spring 2023

// Instructor: Carla McManus

// Assignment: 10-Part-1-Hashing

import java.lang.reflect.Array;

import java.util.\*;

public class Hash {

public static void main(String[] args) {

MyLinearMap<Integer,Integer> map = new MyLinearMap<>();

Integer randomKey = 0;

Integer randomValue = 0;

// Test Put

randomKey = testPut(map, randomKey, randomValue);

// Test Remove

testRemove(map, randomKey);

// Test Entry Set

testEntrySet(map);

//Test Clear

testClear(map);

}

public static Integer testPut(MyLinearMap<Integer, Integer> map, Integer randomKey, Integer randomValue){

System.out.println("Random Entries");

for (int i = 0; i < 5; i++){

randomKey = (int)(Math.random() \* 100);

randomValue = (int)(Math.random() \* 100);

System.out.print(randomKey + ", " + randomValue);

map.put(randomKey, randomValue);

map.printHashTable();

System.out.println("Size: " + map.size());

}

return randomKey;

}

public static void testRemove(MyLinearMap<Integer, Integer> map, Integer randomKey){

System.out.println("Remove Entry");

map.remove(randomKey);

System.out.print(randomKey + ": ");

map.printHashTable();

System.out.println("Size: " + map.size());

map.remove(2);

System.out.print(2 + ": ");

map.printHashTable();

System.out.println("Size: " + map.size());

}

public static void testEntrySet(MyLinearMap<Integer, Integer> map){

System.out.println("Test Entry Set");

System.out.println(map.entrySet());

}

public static void testClear(MyLinearMap<Integer, Integer> map){

System.out.println("Clear map");

map.clear();

System.out.println(map.entrySet());

}

}

// ----- My Linear Map ----- //

class MyLinearMap<K, V> implements MyMap<K, V> {

// ----- Class Constants ----- //

private static final Integer DEFAULT\_CAPACITY = 4;

private static final Float LOAD\_FACTOR = 0.5f;

// ----- Class Variables ----- //

private Integer capacity;

private Float loadFactor;

private Integer size = 0;

Entry<K, V> genericEntry;

Entry<K, V>[] hashTable;

// ----- Constructors ----- //

MyLinearMap(){

capacity = DEFAULT\_CAPACITY;

loadFactor = LOAD\_FACTOR;

hashTable = new Entry[DEFAULT\_CAPACITY];

}

// ----- Class PRIVATE Methods ----- //

private Integer hash(Integer hashCode){

return hashCode % capacity;

}

private void rehash(){

capacity = capacity << 1;

Entry<K, V>[] list = new Entry[capacity];

Integer index = 0;

// Remap each entry in the original hashTable to the new list

for (Entry<K, V> entry : hashTable){

if (entry != null){

index = hash(entry.key.hashCode());

// Linearly find a home

while (list[index] != null){

if (index > (capacity - 1))

index = 0;

index++;

}

// Place the entry into it's new home

list[index] = entry;

}

}

// Write over the old table with the new one

hashTable = list;

}

// ----- Class PUBLIC Methods ----- //

// Print Hashtable

public void printHashTable(){

for (Entry<K,V> entry : hashTable)

System.out.print(entry + ", ");

System.out.println();

}

/\*\* Remove all entries from this map \*/

@Override

public void clear(){

size = 0;

// Let the garbage collector handle the old list and make a new empty one

hashTable = new Entry[DEFAULT\_CAPACITY];

}

/\*\* Return true if the specified key is in the map \*/

@Override

public boolean containsKey(K key){

return get(key) != null;

}

/\*\* Return true if this map contains the specified value \*/

@Override

public boolean containsValue(V value){

// I guess search the entire hash table for the one value

// Doesn't seem effecient but I have no key for reference

for (Entry<K, V> entry : hashTable){

if (entry != null)

if (entry.value == value)

return true;

}

return false;

}

/\*\* Return a set of entries in the map \*/

@Override

public Set<Entry<K, V>> entrySet(){

Set<Entry<K, V>> set = new HashSet<>();

// Add each non-null entry into the set

for (Entry<K, V> entry : hashTable)

if (entry != null)

set.add(entry);

return set;

}

/\*\* Return the first value that matches the specified key \*/

@Override

public V get(K key){

Integer index = hash(key.hashCode());

// Check if the hash location is null

if (hashTable[index] == null){

System.out.println("Hey is not in the hash table");

return null;

}

// If it's not null, check if the key matches

// If it doesn't match, do a linear search for it

while (hashTable[index].key != key){

if (index >= (capacity - 1))

index = 0;

else index++;

// If while doing the linear search, you find a null space instead,

// exit becase the key isn't there

if (hashTable[index] == null){

System.out.println("Hey is not in the hash table");

return null;

}

}

return hashTable[index].value;

}

/\*\* Return true if this map contains no entries \*/

@Override

public boolean isEmpty(){return size == 0;}

/\*\* Return a set consisting of the keys in this map \*/

@Override

public Set<K> keySet(){

Set<K> set = new HashSet<>();

for (Entry<K, V> entry : hashTable){

if (entry != null)

set.add(entry.key);

}

return set;

}

/\*\* Add an entry (key, value) into the map \*/

@Override

public V put(K key, V value){

Integer index = hash(key.hashCode());

System.out.print(", " + index + ": ");

Entry<K, V> e = new Entry<>(key, value);

while (hashTable[index] != null){

if (index >= (capacity - 1))

index = 0;

else

index++;

}

// Add entry to hashtable

hashTable[index] = e;

// Increase size

size++;

// Check if we have gone past our load factor

if (size >= capacity \* LOAD\_FACTOR) {

rehash();

}

return value;

}

/\*\* Remove the entries for the specified key \*/

@Override

public void remove(K key){

Integer index = hash(key.hashCode());

Integer initialIndex = index;

// Chech if the index is populated

// If it's not, the it's not in the table

if (hashTable[index] == null) {

System.out.println("Key not in Hash Table");

return;

}

// Find where the key is located

while (hashTable[index].key != key){

if (index >= (capacity - 1))

index = 0;

else index++;

// Check to see if you've reached a null entry

if (hashTable[index] == null) {

System.out.println("Key not in Hash Table");

return;

}

}

// Remove the key and value from the hashtable

hashTable[index] = null;

// Reduce size by one

size--;

}

/\*\* Return the number of mappings in this map \*/

@Override

public int size(){return size;}

/\*\* Return a set consisting of the values in this map \*/

@Override

public Set<V> values(){

Set<V> set = new HashSet<>();

for (Entry<K, V> entry : hashTable){

set.add(entry.value);

}

return set;

}

}

// ----- Listing ----- //

interface MyMap<K, V> {

/\*\* Remove all of the entries from this map \*/

public void clear();

/\*\* Return true if the specified key is in the map \*/

public boolean containsKey(K key);

/\*\* Return true if this map contains the specified value \*/

public boolean containsValue(V value);

/\*\* Return a set of entries in the map \*/

public java.util.Set<Entry<K, V>> entrySet();

/\*\* Return the first value that matches the specified key \*/

public V get(K key);

/\*\* Return true if this map contains no entries \*/

public boolean isEmpty();

/\*\* Return a set consisting of the keys in this map \*/

public java.util.Set<K> keySet();

/\*\* Add an entry (key, value) into the map \*/

public V put(K key, V value);

/\*\* Remove the entries for the specified key \*/

public void remove(K key);

/\*\* Return the number of mappings in this map \*/

public int size();

/\*\* Return a set consisting of the values in this map \*/

public java.util.Set<V> values();

/\*\* Define inner class for Entry \*/

public static class Entry<K, V> {

K key;

V value;

public Entry(K key, V value) {

this.key = key;

this.value = value;

}

public K getKey() {

return key;

}

public V getValue() {

return value;

}

@Override

public String toString() {

return "[" + key + ", " + value + "]";

}

}

}