```java
package edu.caltech.cs2.datastructures;

import edu.caltech.cs2.interfaces.ICollection;
import edu.caltech.cs2.interfaces.IDictionary;

import java.util.Iterator;
import java.util.function.Supplier;

public class ChainingHashDictionary<K, V> implements IDictionary<K, V> {
    private Supplier<IDictionary<K, V>> chain;

    private int[] primes = {2, 5, 7, 17, 31, 67, 127, 257, 509, 1021, 2053, 4099, 8191, 16381, 32771, 65537, 131071, 262147, 399989};
    private int size;
    private int capacity;
    private int prime_index;
    private IDictionary<K,V>[] back = new IDictionary[size];
    public ChainingHashDictionary(Supplier<IDictionary<K, V>> chain) {
        this.size = 0;
        this.prime_index = 4;
        this.capacity = primes[prime_index];
        this.chain = chain;
        this.back = new IDictionary[capacity];
    }

    /**
     * @param key
     * @return value corresponding to key
     */
    @Override
    public V get(K key) {
        int index = (key.hashCode() % this.capacity + this.capacity) % this.capacity;
        if (this.back[index] != null) {
            return this.back[index].get(key);
        }
        return null;
    }

    @Override
    public V remove(K key) {
        int index = (key.hashCode() % this.capacity + this.capacity) % this.capacity;
        if (this.back[index] == null){
            return null;
        }
        V value = this.back[index].remove(key);
        if (value != null){
            this.size--;
        }
        return value;
    }

    @Override
    public V put(K key, V value) {
        int index = (key.hashCode() % this.capacity + this.capacity) % this.capacity;
        if (this.back[index] == null){
            this.back[index] = this.chain.get();
        }
        if (this.back[index].containsKey(key)){
            V return_val = this.back[index].get(key);
            this.back[index].put(key, value);
            if (this.size >= this.capacity){
                resize();
            }
            if (value.equals(" ")){
                return null;
            }
            return return_val;
        }
        this.back[index].put(key, value);
        this.size++;
        if (this.size >= this.capacity){
            resize();
        }
        if (value.equals("")){
            return null;
        }
        return value;
    }

    private void resize(){
        int old_capacity = this.capacity;
        this.prime_index = this.prime_index + 1;
        if (this.prime_index >= this.primes.length){
            return;
        }
        this.capacity = primes[this.prime_index];
        IDictionary<K,V>[] copy = new IDictionary[this.capacity];
        for (int i=0; i < this.capacity; i++){
```

```
            copy[i] = chain.get();
        }
        for (int x=0; x<old_capacity; x++){
            if (this.back[x] != null) {
                for (K k : this.back[x].keys()) {
                    copy[(k.hashCode() % this.capacity + this.capacity) % this.capacity].put(k, this.back[x].get(k));
                }
            }
        }
        this.back = copy;
    }

    @Override
    public boolean containsKey(K key) {
        return keys().contains(key);
    }

    /**
     * @param value
     * @return true if the HashDictionary contains a key-value pair with
     * this value, and false otherwise
     */
    @Override
    public boolean containsValue(V value) {
        return values().contains(value);
    }

    /**
     * @return number of key-value pairs in the HashDictionary
     */
    @Override
    public int size() {
        return this.size;
    }

    @Override
    public ICollection<K> keys() {
        ArrayDeque<K> keys = new ArrayDeque<>();
        for (int i=0; i<this.size; i++) {
            for (K k : this.back[i].keys()){
                keys.add(k);
            }
        }
        System.out.println(keys);
        return keys;
    }

    @Override
    public ICollection<V> values() {
        ArrayDeque<V> values = new ArrayDeque<>();
        for (int i=0; i<this.size; i++){
            if (this.back[i] != null){
                for (V v : this.back[i].values()){
                    values.add(v);
                }
            }
        }
        return values;
    }

    /**
     * @return An iterator for all entries in the HashDictionary
     */
    @Override
    public Iterator<K> iterator() {
        return null;
    }
    private class ChainingHashIterator implements Iterator<K> {
        private int index;
        public ChainingHashIterator(){
            this.index = 0;
        }
        @Override
        public boolean hasNext() {
            return index < size();
        }
        @Override
        public K next() {
            return null;
        }
    }
}
```