public class LinearProbingHashST<Key, Value> {

private static final int INIT\_CAPACITY = 4;

private int n;

private int m;

private Key[] keys;

private Value[] vals;

public LinearProbingHashST() {

this(INIT\_CAPACITY);

}

public LinearProbingHashST(int capacity) {

m = capacity;

n = 0;

keys = (Key[]) new Object[m];

vals = (Value[]) new Object[m];

}

public int size() {

return n;

}

public boolean isEmpty() {

return size() == 0;

}

public boolean contains(Key key) {

return get(key) != null;

}

private int hash(Key key) {

return (key.hashCode() & 0x7fffffff) % m;

}

private void resize(int capacity) {

LinearProbingHashST<Key, Value> temp = new LinearProbingHashST<Key, Value>(capacity);

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

if (keys[i] != null) {

temp.put(keys[i], vals[i]);

}

}

keys = temp.keys;

vals = temp.vals;

m = temp.m;

}

public void put(Key key, Value val) {

if (val == null) {

delete(key);

return;

}

if (n >= m/2) {

resize(2\*m);

}

int i;

for (i = hash(key); keys[i] != null; i = (i + 1) % m) {

if (keys[i].equals(key)) {

vals[i] = val;

return;

}

}

keys[i] = key;

vals[i] = val;

n++;

}

public Value get(Key key) {

for (int i = hash(key); keys[i] != null; i = (i + 1) % m) {

if (keys[i].equals(key)) {

return vals[i];

}

}

return null;

}

public void delete(Key key) {

if (!contains(key)) {

return;

}

int i = hash(key);

while (!key.equals(keys[i])) {

i = (i + 1) % m;

}

keys[i] = null;

vals[i] = null;

i = (i + 1) % m;

while (keys[i] != null) {

Key keyToRehash = keys[i];

Value valToRehash = vals[i];

keys[i] = null;

vals[i] = null;

n--;

put(keyToRehash, valToRehash);

i = (i + 1) % m;

}

n--;

if (n > 0 && n <= m/8) {

resize(m/2);

}

}

public Iterable<Key> keys() {

Queue<Key> queue = new Queue<Key>();

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

if (keys[i] != null) queue.enqueue(keys[i]);

}

return queue;

}

}