CS206

Maps

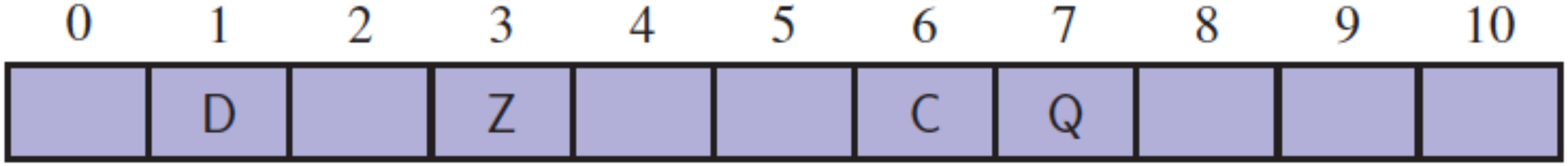
Intro to Hashtables

Map

- A searchable collection of key-value pairs
- Multiple entries with the same key are not allowed
- Also known as dictionary (python), associative array (perl)

Notion of a Map

- Intuitively, a map `M` supports the abstraction of using keys as indices with a syntax such as `M[k]`.
- Simplest setting is a map with n items using keys that are known to be integers from 0 to $N - 1$, for some $N \geq n$.



Map206 Interface

```
public interface MEntry<K,V> {
    public K getKey(); public V getValue();
}

public interface Map206<K, V> {
    public int size();
    public boolean isEmpty();
    public void clear();

    public boolean containsKey(K key);
    public boolean containsValue(V value);

    public V get(K key);
    /**Associates the specified value with the specified key in this map
      *If the map previously contained a mapping for the key,
      *the old value is replaced by the specified value **/
    public V put(K key, V value);
    public V remove(K key);

    public Iterable<K> keySet();
    public Iterable<V> values();
    public Iterable<MEntry<K, V>> entrySet();
}
```

Example

```
public void doExample()    {
HashMap<Integer, String> m = new HashMap<>();
System.out.println(m.isEmpty() + " " + m.toString());    true {}
System.out.println(m.put(5,"A") + " " + m.toString());   null {5=A}
System.out.println(m.put(7,"B") + " " + m.toString());   null {5=A, 7=B}
System.out.println(m.put(2,"E") + " " + m.toString());   null {2=E, 5=A, 7=B}
System.out.println(m.put(8,"A") + " " + m.toString());   null {2=E, 5=A, 7=B, 8=A}
System.out.println(m.put(2,"Q") + " " + m.toString());   E {2=Q, 5=A, 7=B, 8=A}

System.out.println(m.get(7) + " " + m.toString());       B {2=Q, 5=A, 7=B, 8=A}
System.out.println(m.get(4) + " " + m.toString());       null {2=Q, 5=A, 7=B, 8=A}
System.out.println(m.remove(2) + " " + m.toString());    Q {5=A, 7=B, 8=A}
System.out.println(m.remove(5) + " " + m.toString());    A {7=B, 8=A}
System.out.println(m.isEmpty() + " " + m.toString());    false {7=B, 8=A}
System.out.println(m.entrySet() + " " + m.toString());   [7=B, 8=A] {7=B, 8=A}
System.out.println(m.values() + " " + m.toString());     [B, A] {7=B, 8=A}
}
```

Abstract Class

- A class between a (concrete) class and an interface
  - abstract methods – method signatures without implementation (like interface)
  - concrete methods – regular methods
  - instance variables
- An abstract class may not be instantiated

AbstractMap

```
public abstract class AbstractMap<K,V> implements Map206<K,V> {
    private class Entry<L,W> implements MEntry<L,W> {
       private final L key;
       private final W value;
   public L getKey() {
       return key;
   }
   public W getValue() {
       return value;
   }
   public Entry(L k, W v) {
       this.key=k;
       this.value=v;
   }}
    @Override
    public boolean isEmpty() { return size()==0; }

  // More??
  }
```

keySet/values

```
private class KeyIterator<L> implements Iterator<K> {
   private Iterator<MEntry<K,V>> entries = entrySet().iterator();
   public boolean hasNext() { return entries.hasNext(); }
   public K next() { return entries.next().getKey(); }
   public void remove(L key) {
     throw new UnsupportedOperationException(); }
}

private class KeyIterable<L> implements Iterable<K>  {
   public Iterator<K> iterator()  {
       return new KeyIterator<K>(); }
}

public Iterable<K> keySet() { return new KeyIterable<K>(); }
```

Why not just return an iterator?? … because Map206 follows the interface definition in java.util.Map and it returns a java.util.Set which implements Iterable

UnsortedMap

```
public class UnsortedMap<K,V> extends AbstractMap<K,V> {
    private ArrayList<Entry<K,V>> entryList;

    public UnsortedMap()
    {
   entryList = new ArrayList<>();
    }

    @Override
    public int size() {
   return entryList.size();
    }

    @Override
    public void clear() {
   entryList.clear();
    }
```

write containsKey

UnsortedMap (contd)

```
 private int findKeyIndex(K key)
 {
int idx=0;
for (Entry<K,V> e : entryList)
{
    if (e.getKey().equals(key))
   return idx;
    idx++;
}
return -1;
 }

 @Override
 public boolean containsKey(K key)
 {
return findKeyIndex(key) >= 0;
 }
```

UnsortedMap (contd)

```
@Override
public V get(K key) {
    int idx = findKeyIndex(key);
    return (idx<0) ? null : entryList.get(idx).getValue();
    }

@Override
public V remove(K key) {
    int idx = findKeyIndex(key);
    V v=null;
    if (idx>=0)
    {
        v = entryList.get(idx).getValue();
        entryList.remove(idx);
    }
    return v;
    }

@Override
public V put(K key, V value) {
    V v = remove(key);
    entryList.add(new Entry<K,V>(key, value));
    return v;
    }
```

UnsortedMap (contd)

```
private class EntryIterator implements Iterator<MEntry<K,V>> {
    int idx=0;
    public boolean hasNext() { return idx<entryList.size(); }
    public MEntry<K,V> next() {
        if (idx>=entryList.size())
        throw new NoSuchElementException();
        return entryList.get(idx++);
    }
    public void remove(K key) {
        throw new UnsupportedOperationException();
    }}

private class EntryIterable implements Iterable<MEntry<K,V>> {
      public Iterator<MEntry<K,V>> iterator()  {
         return new EntryIterator();
    }}

@Override
public Iterable<MEntry<K,V>> entrySet() {
       return new EntryIterable();
    }
```

Performance Analysis

	Unsorted array	Sorted array	Unsorted list	Sorted list
search				
insert				
remove				
min/max				

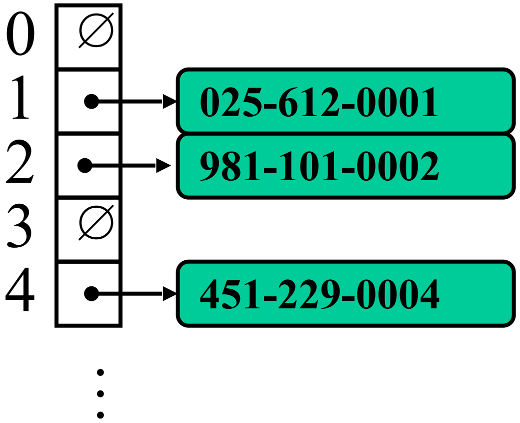
- What does this analysis suggest about usage scenarios?

Improving Maps

- Can we tradeoff time and space
  - UnsortedMap implementation
    - efficient spacewise
    - not great timewise
    - So if storing lots of info but accessing rarely, OK
  - But what if storing less and access often?
  - Can we get O(1) time for get/set/remove at a cost of space?

More General Keys

- Earlier: motivated Maps with discussion of keys as integers. What if our keys are not integers in range 0 to $N-1$?
- Use a function to map keys to integers into the right range
- Example: Rather than entire SSN, use only last 4 digits

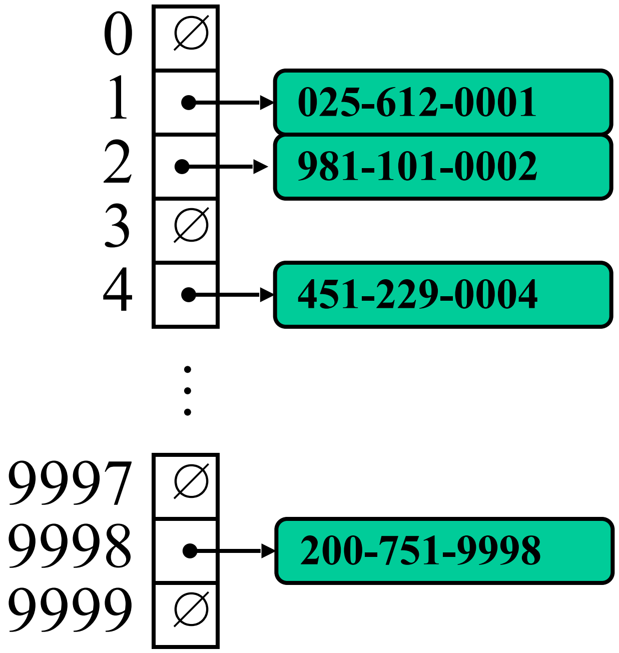


Hash Functions and Tables

- A hash function h maps a key to integers in a fixed interval $[0, N - 1]$
- $h(x) = x\%N$ is such a function for integers
- A hash table is an array of size N
    - associated hash function h
    - item (k, v) is stored at index $h(k)$

Example

- A hash table storing entries as (SSN, Name), where SSN is a nine-digit positive integer
- Use an array of size $N = 10000$ and the hash function $h(x) = \text{last 4 digits of } x$
- Issues?


0
1
2
3
4
9997
9998
9999
∅
025-612-0001
981-101-0002
451-229-0004
200-751-9998