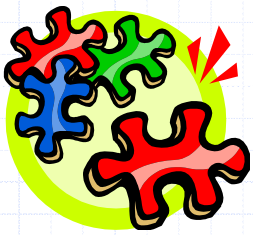
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014

Sets and Multimaps



Definitions

- A **set** is an unordered collection of elements, without duplicates that typically supports efficient membership tests.
  - Elements of a set are like keys of a map, but without any auxiliary values.
- A **multiset** (also known as a **bag**) is a set-like container that allows duplicates.
- A **multimap** is similar to a traditional map, in that it associates values with keys; however, in a multimap the same key can be mapped to multiple values.
  - For example, the index of a book maps a given term to one or more locations at which the term occurs.

Set ADT

add(e): Adds the element e to S (if not already present).

remove(e): Removes the element e from S (if it is present).

contains(e): Returns whether e is an element of S.

iterator( ): Returns an iterator of the elements of S.

There is also support for the traditional mathematical set operations of ***union***, ***intersection***, and ***subtraction*** of two sets S and T:

$$S \cup T = \{e: e \text{ is in } S \text{ or } e \text{ is in } T\},$$
$$S \cap T = \{e: e \text{ is in } S \text{ and } e \text{ is in } T\},$$
$$S - T = \{e: e \text{ is in } S \text{ and } e \text{ is not in } T\}.$$

addAll(T): Updates S to also include all elements of set T, effectively replacing S by $S \cup T$.

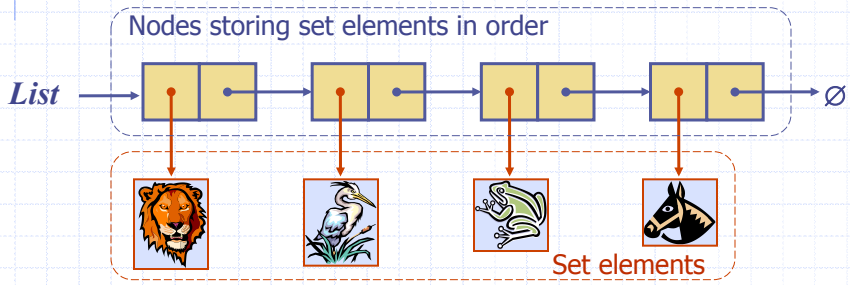
retainAll(T): Updates S so that it only keeps those elements that are also elements of set T, effectively replacing S by $S \cap T$.

removeAll(T): Updates S by removing any of its elements that also occur in set T, effectively replacing S by $S - T$.

 Sets and Multimaps 3

Storing a Set in a List

- We can implement a set with a list
- Elements are stored sorted according to some canonical ordering
- The space used is $O(n)$



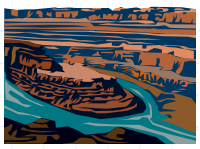
 Sets and Multimaps 4

Generic Merging

- Generalized merge of two sorted lists A and B
- Template method genericMerge
- Auxiliary methods
  - aIsLess
  - bIsLess
  - bothAreEqual
- Runs in $O(n_A + n_B)$ time provided the auxiliary methods run in $O(1)$ time

```
Algorithm genericMerge(A, B)
    S ← empty sequence
    while ¬A.isEmpty() ∧ ¬B.isEmpty()
        a ← A.first().element(); b ← B.first().element()
        if a < b
            aIsLess(a, S); A.remove(A.first())
        else if b < a
            bIsLess(b, S); B.remove(B.first())
        else { b = a }
            bothAreEqual(a, b, S)
            A.remove(A.first()); B.remove(B.first())
    while ¬A.isEmpty()
        aIsLess(a, S); A.remove(A.first())
    while ¬B.isEmpty()
        bIsLess(b, S); B.remove(B.first())
    return S
```


Using Generic Merge for Set Operations



- Any of the set operations can be implemented using a generic merge
- For example:
  - For intersection: only copy elements that are duplicated in both list
  - For union: copy every element from both lists except for the duplicates
- All methods run in linear time

Multimap

- A **multimap** is similar to a map, except that it can store multiple entries with the same key
- We can implement a multimap M by means of a map M'
  - For every key k in M, let E(k) be the list of entries of M with key k
  - The entries of M' are the pairs (k, E(k))

Mulitmaps

get(k):	Returns a collection of all values associated with key k in the multimap.
put(k, v):	Adds a new entry to the multimap associating key k with value v, without overwriting any existing mappings for key k.
remove(k, v):	Removes an entry mapping key k to value v from the multimap (if one exists).
removeAll(k):	Removes all entries having key equal to k from the multimap.
size( ):	Returns the number of entries of the multiset (including multiple associations).
entries( ):	Returns a collection of all entries in the multimap.
keys( ):	Returns a collection of keys for all entries in the multimap (including duplicates for keys with multiple bindings).
keySet( ):	Returns a nonduplicative collection of keys in the multimap.
values( ):	Returns a collection of values for all entries in the multimap.

Java Implementation

```
1  public class HashMultimap<K,V> {
2    Map<K,List<V>> map = new HashMap<>();   // the primary map
3    int total = 0;                          // total number of entries in the multimap
4    /** Constructs an empty multimap. */
5    public HashMultimap( ) { }
6    /** Returns the total number of entries in the multimap. */
7    public int size( ) { return total; }
8    /** Returns whether the multimap is empty. */
9    public boolean isEmpty( ) { return (total == 0); }
10   /** Returns a (possibly empty) iteration of all values associated with the key. */
11   Iterable<V> get(K key) {
12     List<V> secondary = map.get(key);
13     if (secondary != null)
14       return secondary;
15     return new ArrayList<>( );            // return an empty list of values
16   }
```


Java Implementation, 2

```
17   /** Adds a new entry associating key with value. */
18   void put(K key, V value) {
19     List<V> secondary = map.get(key);
20     if (secondary == null) {
21       secondary = new ArrayList<>( );
22       map.put(key, secondary);       // begin using new list as secondary structure
23     }
24     secondary.add(value);
25     total++;
26   }
27   /** Removes the (key,value) entry, if it exists. */
28   boolean remove(K key, V value) {
29     boolean wasRemoved = false;
30     List<V> secondary = map.get(key);
31     if (secondary != null) {
32       wasRemoved = secondary.remove(value);
33       if (wasRemoved) {
34         total--;
35         if (secondary.isEmpty( ))
36           map.remove(key);          // remove secondary structure from primary map
37       }
38     }
39     return wasRemoved;
40   }
```

Java Implementation, 3

```
41    /** Removes all entries with the given key. */
42    Iterable<V> removeAll(K key) {
43      List<V> secondary = map.get(key);
44      if (secondary != null) {
45        total -= secondary.size( );
46        map.remove(key);
47      } else
48        secondary = new ArrayList<>( );       // return empty list of removed values
49      return secondary;
50    }
51    /** Returns an iteration of all entries in the multimap. */
52    Iterable<Map.Entry<K,V>> entries( ) {
53      List<Map.Entry<K,V>> result = new ArrayList<>( );
54      for (Map.Entry<K,List<V>> secondary : map.entrySet( )) {
55        K key = secondary.getKey( );
56        for (V value : secondary.getValue( ))
57          result.add(new AbstractMap.SimpleEntry<K,V>(key,value));
58      }
59      return result;
60    }
61  }
```

