USA  Tutor

Set and Maps Structures

Slides by Adam Zweiger

Sets and Maps

- In Java, there are two versions of sets and maps.
- One uses sorting and the other uses hashing.

Sets

- A set is a collection of objects with no duplicates.
- An ordered set maintains the objects in sorted order.
- Insertions, deletions, and searches are all O(log N), where N is the number of elements in the set.
- In Java, the `TreeSet` class is an ordered set.
- The `add` method inserts the element into the set if it is not already there.
- The `remove` method deletes the element if it is there.
- The `contains` method checks if the element is in the set.

More Operations

- first(): returns the lowest element in the set
- last(): returns the greatest element in the set
- lower(a): returns the greatest element less than a
- floor(a): returns the greatest element less than or equal to a
- higher(a): returns the least element greater than a
- ceiling(a): returns the least element greater than or equal to a

Sets Example

```
5   class Main {
6     public static void main(String[] args) {
7       TreeSet<Integer> set = new TreeSet<Integer>();
8       set.add(3); // [3]
9       set.add(22); // [3, 22]
10      set.add(3); // [3, 22]
11      set.add(5); // [3, 5, 22]
12      set.add(8); // [3, 5, 8, 22]
13      set.remove(set.higher(6)); // [3, 5, 22]
14      System.out.println(set.contains(8)); // false
15      System.out.println(set.first()); // 3
16      System.out.println(set.last()); // 22
17      System.out.println(set.higher(7)); // 22
18      System.out.println(set.ceiling(3)); // 3
19      System.out.println(set.lower(5)); // 3
20
21      System.out.println();
22      for (int i : set){
23        System.out.print(i + "\n");
24      }
25    }
26  }
```

Maps

- A map is a collection of ordered pairs, each containing a key and a value.
- Keys must be distinct, but values can be repeated.
- Insertions, deletions, and searches are all O(log N), where N is the number of elements in the map.
- In Java, the `TreeMap` class is an ordered map.
- The `put(key, value)` method inserts the key and value pair into the map. If the key is already present, the value is still changed.
- The `remove(key)` method deletes the map entry associated with the key.
- The `containsKey(key)` method checks if the key is in the map.
- The `get(key)` method returns the value associated with the key.

Maps Example

```
5   class Main {
6     public static void main(String[] args) {
7       TreeMap<Integer, Integer> map = new TreeMap<Integer, Integer>();
8       map.put(3,10);
9       map.put(22,60);
10      map.put(2,9); // [(2,9);(3,10);(22,60)]
11      System.out.println(map.firstKey()); // 2
12      System.out.println(map.firstEntry()); // (2,9)
13      System.out.println(map.higherEntry(5)); // (22,60)
14      System.out.println();
15
16      for (Map.Entry<Integer, Integer> i : map.entrySet()){
17        System.out.print(i.getKey() + " " + i.getValue() + "\n");
18      }
19
20    }
21  }
```

Multisets

- A multiset is a sorted set that allows for multiple copies of the same element
- In Java, there is no built in Multiset data structure, but we can implement one with TreeMap.
- The first, last, higher, and lower operations still function as intended; just use firstKey, lastKey, higherKey, and lowerKey respectively.

Multiset Example

```
1  import java.io.*;
2  import java.util.*;
3
4  class Main {
5    static TreeMap<Integer, Integer> multiset = new TreeMap<Integer, Integer>();
6
7    public static void main(String[] args){
8      add(3);
9      add(13);
10     add(11);
11     add(5);
12     add(5);
13     add(13);
14     remove(13); // [3, 5, 5, 11, 13]
15     System.out.println(multiset.containsKey(3)); // true
16     System.out.println(multiset.get(5)); // 2
17   }
18
19   static void add(int x){
20     if(multiset.containsKey(x)){
21       multiset.put(x, multiset.get(x) + 1);
22     } else {
23       multiset.put(x, 1);
24     }
25   }
26
27   static void remove(int x){
28     multiset.put(x, multiset.get(x) - 1);
29     if(multiset.get(x) == 0){
30       multiset.remove(x);
31     }
32   }
33 }
```

Priority Queue(Heap)

- Can insert elements, find element of highest “priority,” and delete element of highest priority, all in O(log N)
- In Java, it’s actually elements of the lowest priority rather than highest
- The `add` method inserts the element into the priority queue
- The `poll` method deletes the element of lowest priority (the smallest element)
- The `peek` method returns the element of lowest priority

Priority Queue Example

```
5   class Main {
6     public static void main(String[] args) {
7       PriorityQueue<Integer> pq = new PriorityQueue<Integer>();
8       pq.add(7); // [7]
9       pq.add(2); // [7, 2]
10      pq.add(1); // [7, 2, 1]
11      pq.add(5); // [7, 5, 2, 1]
12      System.out.println(pq.peek()); // 1
13      pq.poll(); // [7, 5, 2]
14      pq.poll(); // [7, 5]
15      pq.add(6); // [7, 6, 5]
16    }
17  }
```

Problems

- https://cses.fi/problemset/task/1091
- http://www.usaco.org/index.php?page=viewproblem2&cpid=667
- http://www.usaco.org/index.php?page=viewproblem2&cpid=763