Junior Knights Competitive Programming Notes: 1/20/2024

Topic 1: Sets

In Java, there are two types of built in sets:

1. TreeSet
2. HashSet

The former keeps objects in sorted order while the latter does not.

A set stores objects of a designated class without regards to duplicates. So, if you try to add 3 to the set {2, 3, 7} no change will be made to the set.

Here are key functions you can perform on all sets (ordered and unordered):

1. add an item
2. remove an item
3. search for an item
4. loop through the items
5. Find the size of the set (# of unique items in it)

Complete description is here: https://docs.oracle.com/javase/8/docs/api/java/util/HashSet.html

Here are the relevant methods in the HashSet class:

`boolean`	`add(E e)` Adds the specified element to this set if it is not already present.
`void`	`clear()` Removes all of the elements from this set.
`boolean`	`contains(Object o)` Returns `true` if this set contains the specified element.
`boolean`	`isEmpty()` Returns `true` if this set contains no elements.
`boolean`	`remove(Object o)` Removes the specified element from this set if it is present.
`int`	`size()` Returns the number of elements in this set (its cardinality).

Here is the syntax for creating a HashSet of Integer and calling some of these methods:

```
HashSet<Integer> set = new HashSet<Integer>();
set.add(5);
set.add(12);
set.add(5);
System.out.println(set.size());

if (set.contains(12))
    System.out.println(“we have 12”);

if (!set.contains(3))
    System.out.println(“we don’t have 3”);

set.remove(5);
set.add(11);
set.add(2);
```

To loop through each item of a set, do this:

```
int sum = 0;
for (Integer x: set)
    sum += x;
System.out.println(“Sum of items in the set is “+sum);
```

In an unordered set, the run time of add, remove, contains and size is O(1), so basically constant time. In an orderes set, the run time of add, remove and contains is O(lg n), where n is the number of items in the set. So an unordered set is faster by a little bit, but most times it won’t matter. ($\log_2 1000000$ is about 20…) The ordered set gives you more capabilities which we will discuss next.

In a TreeSet, which is ordered, we can access and remove the smallest or largest item, we can find the smallest item greater than some item or the largest item smaller than some item. All of these functions can be done in O(lg n) time, where n is the number of items in the set. Here is the full listing of the TreeSet class:

https://docs.oracle.com/javase/8/docs/api/java/util/TreeSet.html

Here are some of the most useful methods that aren’t in HashSet:

E	**ceiling(E e)** **Returns the least element in this set greater**

	than or equal to the given element, or null if there is no such element.
E	first() Returns the first (lowest) element currently in this set.
E	floor(E e) Returns the greatest element in this set less than or equal to the given element, or null if there is no such element.
E	higher(E e) Returns the least element in this set strictly greater than the given element, or null if there is no such element.
E	last() Returns the last (highest) element currently in this set.
E	lower(E e) Returns the greatest element in this set strictly less than the given element, or null if there is no such element.
E	pollFirst() Retrieves and removes the first (lowest) element, or returns null if this set is empty.
E	pollLast() Retrieves and removes the last (highest) element, or returns null if this set is empty.

The posted example, testds.java has some examples calling these methods.

Topic 2: Maps

In Java, there are two types of built in maps:

1. TreeMap
2. HashMap

A map is similar to aa set, but instead of just storing items, which are called keys, for each item a map stores, it associates it with a value. For example, here is a set of names:

Aryna
Bella
Carl

Imagine if we asked each person how many pets they own and stored that information along with each key:

Aryna → 2
Bella → 0
Carl → 5

Now, in addition to adding and item, removing an item and searching for an item, we could look up the associated value to an item (key). Thus, a map maintains (key, value) pairs. Each key must be unique, like a set, but the values need not be. In the previous example, it's pretty obvious that two different people could own the same number of pets. Maps can be pretty useful. They can store things like how many votes each person got in an election (map each candidate to number of votes), or they can map each city to a unique identification code (say an integer from 0 to 99,999, if there were 100,000 cities.)

In a HashMap, the keys are unordered, but in a TreeMap, the keys are ordered. If you don't need the keys to be ordered, a HashMap is a little faster. Also, when you add an item to a map, you have to add both the key and its initial corresponding value. At any time you can change the mapping for a key in the map.

The whole listing of the HashMap class is here:

https://docs.oracle.com/javase/8/docs/api/java/util/HashMap.html

Here are some of the relevant methods:

`void`	`clear()` Removes all of the mappings from this map.
`boolean`	`containsKey(Object key)`

	Returns true if this map contains a mapping for the specified key.
V	get(Object key) Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
V	getOrDefault(Object key, V defaultValue) Returns the value to which the specified key is mapped, or defaultValue if this map contains no mapping for the key.
Set<K>	keySet() Returns a Set view of the keys contained in this map.
V	put(K key, V value) Associates the specified value with the specified key in this map.
V	remove(Object key) Removes the mapping for the specified key from this map if present.
int	size() Returns the number of key-value mappings in this map.

The most important functions are put, get, and containsKey. These allow you to add to the map and safely search for a mapping. To loop through each key, we can do the following:

```
for (KeyType obj : map.keySet()) {
    // use obj

}
```

This will loop through each key in O(n) time total, where n is the number of items in the map. KeyType is the type of the key. Here is how to declare a HashMap from String to Integer:

```
HashMap<String,Integer> map = new HashMap<String,Integer>();
```

Imagine we are reading through n names, not necessarily unique, and we wanted to create a code, mapping each unique name to an integer, starting at 0. Here is a code segment to do it:

```
int id = 0;
for (int i=0; i<n; i++) {
    String name = stdin.next();

    if (!map.containsKey(name)) {
        map.put(name, id);
        id++;
     }
}
```

This can be shortened to:

```
int id = 0;
for (int i=0; i<n; i++) {
    String name = stdin.next();

    if (!map.containsKey(name))
        map.put(name, id++);
}
```

The id++ is post increment, so after it does the put, it will add 1 to the current value of id.

In a TreeMap, the keys are ordered, so just like with a TreeSet, for a key, we can find the largest key less than it or the smallest key greater than it. Also, we can loop through all the keys in sorted order via the keySet loop shown previously.

Here is the full listing of methods for TreeMap:

https://docs.oracle.com/javase/8/docs/api/java/util/TreeMap.html

Here are some relevant methods not in the HashMap:

K	`firstKey()` Returns the first (lowest) key currently in this map.
K	`floorKey(K key)` Returns the greatest key less than or equal to the given key, or null if there is no such key.
K	`higherKey(K key)`

	Returns the least key strictly greater than the given key, or null if there is no such key.
K	higherKey(K key) Returns the least key strictly greater than the given key, or null if there is no such key.
K	lowerKey(K key) Returns the greatest key strictly less than the given key, or null if there is no such key.

In class, we used TreeMap to output the number of votes candidates got in alphabetical order by candidate.

The exercises given were:

https://open.kattis.com/problems/cd (for sets)

https://open.kattis.com/problems/zoo (for maps)

Two issues we ran into with CD are:

1) A TreeSet is too slow due to the size of the input and the tight time limit. A HashSet must be used. (There are solutions that avoid sets all together, but the HashSet solution is the fastest to type, probably…)

2) A regular Scanner is too slow for the time limit.

To alleviate the second issue, a FastScanner class was given to the students. It’s in its entirety below:

```
class FastScanner {
    BufferedReader br;
    StringTokenizer st;

    public FastScanner(InputStream i) {
        br = new BufferedReader(new InputStreamReader(i));
        st = new StringTokenizer("");
    }

    public String next() throws IOException {
        if(st.hasMoreTokens())
            return st.nextToken();
        else
            st = new StringTokenizer(br.readLine());
        return next();
    }
    public int nextInt() throws IOException {
        return Integer.parseInt(next());
    }
    //#
    public long nextLong() throws IOException {
        return Long.parseLong(next());
    }
    public double nextDouble() throws IOException {
        return Double.parseDouble(next());
    }
    //$
}
```

To incorporate it, do the following:

1. Place this class after your public class that is solving the problem.

2. Add this import at the top of your code:

```
import java.io.*;
```

3. In your main method in your public class do

```
public static void main(String[] args) throws Exception {
...
}
```

4. When creating your Scanner object do:

```
FastScanner stdin = new FastScanner(System.in);
```

Everything else should be the same!

Introduction to USACO

Website: https://usaco.org/

USACO hosts four monthly contests a year in December, January, February and March.

There are four levels: Bronze, Silver, Gold and Platinum

Everyone starts in Bronze.

All contests have 3 problems with a time limit of 4 hours and give partial credit. Each question is worth 333 points (or 334). If a problem has 15 test cases, then each test case is worth roughly 333/15 points. You can submit during the contest as many times as you want. You get scored on your last submission in the contest. If you score 750 or higher (usually that is the cut off), then you get promoted to the next division.

This is used to determine a group of about 24 high school students across the nation who get invited to a training camp in the summer, free of charge. From that camp, 4 students are selected to represent the United States at the International Olympiad of Informatics.

For each contest, there is a 4-day contest window. For 2023-2024 season, the three remaining contest windows are

Jan 26 – Jan 29
Feb 16 – Feb 19
Mar 15 – Mar 18

If you want to compete, pick any 4 hour window within the 4 day contest window, and log on, You'll click on the "January 2024 Contest Page" (Or whichever month it is) and click on it. Then it'll ask you if you want your contest to begin and you click that button to start.

In class I illustrated the problem on this link:

https://usaco.org/index.php?page=viewproblem2&cpid=1180

Non-Transitive Dice from Bronze January 2022

I lived coded this and submitted it to show how to use USACO.