Lecture 13: Hard sets/maps

- Brief review of sets/maps
  - A set is a collection of elements - how different from an ArrayList or LinkedList?
    - Unordered, no duplicates
    - add/remove, but no indices
  - And how does a map work
    - Key-->value pairs
    - put/get/remove
  - Both can be Hash or Tree
- The Friends program
  - We're going to be modeling friendships
  - Think of visualizing the Facebook graph
  - People are nodes, and friendships between them are edges
  - First of all: are friendships bidirectional (i.e. if I'm friends with you, are you friends with me?)
    - For our purposes YES (a la Faceook)
  - I'm using a program called GraphViz to visualize the friendship graph
    - (show the input file - .dot)
  - We'll write a program that takes two names and figures out how far apart they are - how many friends-of-friends away are they?
    - E.g. Melissa/Ashley
    - Melissa/Bart
    - Bart/Tom
    - (run the complete version of the program)
- We need a structure to keep track of these friendships
  - What kind of structure
    - A map
  - Why a map? What are the keys? What are the values? (in English)
    - Names are the keys, values are the people that that person is friends with (edges)
  - What are the types of the keys/values?
    - Map<String, Set<String>>
    - Not Map<String, String> because then each person could have only 1 friend!
  - (show the starter code - implement readFile)
  - First we need to create our structure

```
Map<String, Set<String>> friends = new TreeMap<String,
Set<String>>();
```

  - Given two names, how do we update our structure?
    - Remember, friendships are bidirectional
    - So we have to update BOTH people's friends
    - We're going to do the same thing twice - great place for a helper method

```
addTo(friends, name1, name2);
addTo(friends, name2, name1);
```

- In pseudocode:
  - Get the set for the first name
  - Add the second name to that set
  - (start with the second two lines of code) - (could write as one line)
- But what's the problem?
  - Originally, there's nothing in the map! What if name1 isn't in the map?
  - We'd get a null pointer exception
  - So the very first time, we have to add the name to the map
- The method:

```
public static void addTo(Map<String, Set<String>> friends, String
name1,
            String name2) {
      if (!friends.containsKey(name1)) {
            friends.put(name1, new TreeSet<String>());
      }
      Set<String> name1Friends = friends.get(name1);
      name1Friends.add(name2);
}
```

- Now that we've constructed the map, review what we want to do
  - (run the program again, Jessica to Melissa)
  - (use highlighters to color the friendship graph)
  - How does the program find the people who are 1 away?
    - Jessica's friends
  - How does the program find the people who are 2 away?
    - The friends of Ashley/Michael
  - etc.
  - So we need to do the same thing repeatedly (find the friends of all the current friends) - a loop!
  - We also need a way to store the current friends who are the current distance away
    - A set
    - And who is the very first group of people to consider?
  - So preliminary code:

```
Set<String> currentGroup = new TreeSet<String>();
currentGroup.addAll(friends.get(name1));
int distance = 1;
while (we haven't found the person) {
    distance++;
    // update group
}
System.out.println("found at a distance of " + distance);
```

  - How do we know when to stop? What does it mean to "find" the target person?

```
!currentGroup.containsKey(name2)
```

- How do we update the group?
  - We said that we use the friends of everyone in the current group
  - So we create a new set for the next group, loop through the current group, and add all their friends
  - Also, we had a println

```
Set<String> nextGroup = new TreeSet<String>();
for (String friend : currentGroup) {
    nextGroup.addAll(friends.get(friend));
}
currentGroup = nextGroup;
System.out.println("    " + distance + " away: " +
currentGroup);
```

  - (try program: Jessica/Melissa)
    - Does find friends at the right distances, but finds them more than once
    - Does find Melissa at the right distance
  - (try program: Jessica/Jessica)
    - But Jessica shouldn't be a friend of a friend - distance should probably be 0
    - How can we fix that?
    - Start with distance 0, which contains just Jessica

```
Set<String> currentGroup = new TreeSet<String>();
currentGroup.add(name1);
int distance = 0;
```

    - Rerun with Jessica/Jessica, Jessica/Melissa
    - Now we also print out distance 1, which is more complete
- Another problem: Ashley appears at distance 1, 3, 4...
  - Why is this?
    - She's friends with a friend of a friend
  - But we don't want this - we just want the first time we see the friend
  - How can we do this?
    - Naive: ignore the previous group's names in the next group

```
nextGroup.removeAll(currentGroup);
```

    - But this doesn't actually work because names like Ashley might not appear for a level, but then reappear b/c friend-of-a-friend
    - Keep track of another set of people who have been seen before

```
Set<String> alreadySeen = new TreeSet<String>();
...
while (!currentGroup.contains(name2)) {
    distance++;
    alreadySeen.addAll(currentGroup);
    Set<String> nextGroup = new TreeSet<String>();
    for (String friend : currentGroup) {
        nextGroup.addAll(friends.get(friend));
    }
    nextGroup.removeAll(alreadySeen);
```

```
        ...
```

- Another problem: run Melissa/Bart
  - We never stop!
  - Solution:

```
while (!currentGroup.contains(name2) && !currentGroup.isEmpty()) {
            ...
}
if (currentGroup.contains(name2)) {
    System.out.println("found at a distance of "+distance);
} else {
    System.out.println("not found");
}
```

- The goal of this program
  - Review of sets/maps
  - Demonstration of mapping with complicated values
- Midterm info
  - What types of problems
  - Sample midterms online by this evening
  - I'll try to keep it the same length as a midterm during the year - but you get 1 hour instead of 50 minutes!