CS201 Discussion 6

DIRSORT AND MEMBERCHECK
Dirsort Problem Statement

Problem Statement

Sean can't remember where he put an important file on his computer. He desperately needs the file tonight to finish his homework, since he procrastinated all day by playing computer games. Unfortunately, his computer has gotten fried from all the gaming, and can no longer perform automatic searches for files. He therefore has to look for the file manually.

Since he tends not to use subdirectories too much, he knows his best bet is to look for the file first in the root directory ("/"), and next in directories only one level deep (for example, "/Games/"), and so forth, checking the "deepest" directories last. His computer still has enough fire power to run some simple programs, so he has asked you to code for him a simple algorithm to sort all of the directories on his computer.

The algorithm should take a String[] dirs as an input and should sort dirs first by directory depth, and then lexicographically for each depth. So "/d/e/" comes before "/a/b/c/", but not before "/c/d/". Also, "/a/bc/", comes before "/ab/c/", since "a" comes before "ab" in lexicographical order.
Example

{"/","/usr/","/usr/local/","/usr/local/bin/","/games/","/games/snake/","/homework/","/temp/downloads/"} becomes:

{"/",
"/games/", "/homework/", "/usr/",
"/games/snake/", "/temp/downloads/", "/usr/local/",
"/usr/local/bin/" }
Sorting Criteria for our Comparator

1. Sort by number of levels in the path/the path depth
   ◦ Given directory names a and b, how can we find the number of levels easily?

2. Sort lexicographically, by directory name.
   ◦ i.e. The path whose first directory comes earlier in the alphabet should come first (e.g. “/games/snake/” before “/temp/downloads/” because “games” comes before “temp”)
   ◦ If both the first directory names are the same, sort by the second directory name. If both of those are the same, sort by the third, etc. e.g. “/games/pacman/” would come before “/games/snake/”
Sorting Lexicographically

One way we could determine which of two paths comes first lexicographically is:

- Split them into a list of directory names using .split()
- Iterate over each level. Once we find a level where the names are different, return a value based on their difference
Problem Statement

A health club chain allows its members to visit any of its many health club locations an unlimited number of times per day. The only constraining rule is, a customer can only visit one health club location per day, even though he or she may return to that location an unlimited number of times for the rest of that day.

Although the honor system has always worked quite well, the club wants to run some tests to see how many people really follow the rules. You are to write a program that takes the entrance log files from three different clubs (all logging the same day) and return a sorted list of the people who are not honest and went to more than one health club location in the same day.

The log files are represented as String[]'s where each element is the member name of a customer who entered that day. For example, if a customer showed up three times to one of the club locations that day, the member's name would appear three times in the corresponding String[].

```java
public class MemberCheck {
    public String[] whosDishonest(String[] club1,
                                  String[] club2,
                                  String[] club3) {
        // TODO: fill in code here
    }
}
```
Things to think about

Given that we don’t care how many times a member attends the same club, what data structure might be useful in this APT?

Let’s say $a_1$ is a boolean which is true if a member went to club 1, $a_2$ is the same for club 2, $a_3$ is the same for club 3. How could we write in terms of $a_1$, $a_2$, and $a_3$ if that member has been going to multiple clubs?
Things to think about

Given that we don’t care how many times a member attends the same club, what data structure might be useful in this APT?

Sets, since they don’t contain duplicates.

Let’s say \( a_1 \) is a boolean which is true if a member went to club 1, \( a_2 \) is the same for club 2, \( a_3 \) is the same for club 3. How could we write in terms of \( a_1, a_2, \) and \( a_3 \) if that member has been going to multiple clubs?

\[(a_1 \land a_2) \lor (a_1 \land a_3) \lor (a_2 \land a_3)\]

\[(a_1 \land a_2)\] is true if a member went to both club 1 and club 2. Then, this expression is true if there is any pair of clubs they went to.
Set intersection and union

We could iterate through every member and use the boolean expression from the previous slide to determine if we should report them. However, we can take advantage of set union and intersection to avoid having to write any loops at all!

If for sets A and B, set C = A ∪ B, (C is the union of A and B), then C contains any element which was in A or which was in B. For two Sets of Strings a and b in Java, we can make their union as follows:

```java
HashSet<String> c = new HashSet<String>(a); c.addAll(b);
```

If for sets A and B, set C = A ∩ B, (C is the intersection of A and B), then C contains any element which was in A and which was in B. For two Sets of Strings a and b in Java, we can make their union using

```java
HashSet<String> c = new HashSet<String>(a); c retainAll(b);
```
Rethinking the problem

If A1 is the set of people who went to club 1, A2 the same for club 2, A3 the same for club 3, then in terms of unions and intersections, the set of people to report is:

\[(A \cap B) \cup (A \cap C) \cup (B \cap C)\]

This is just the boolean expression from earlier, but we replaced && and || with set notation!

(When you actually solve the APT, it’ll probably be easier and cleaner to compute this one step at a time rather than all at once. i.e. compute \((A \cap B)\) in one step, then \((A \cap C)\) in another step, etc.)
Solve the APTs!

At this point, you know some useful tricks to solve Dirsort and MemberCheck efficiently. Spend the rest of discussion solving them if you haven’t already! These are collaborative APTs, so feel free to solve them with a partner(s), and as always, your TAs are here to help.