CS 61B Fall 2023

Iterators, Polymorphism

Exam-Level 5: September 18, 2023

1 Take Us to Your "Yrnqre"

You're a traveler who just landed on another planet. Luckily, the aliens there use the same alphabet as the English language, but in a different order.

Given the `AlienAlphabet` class below, fill in `AlienComparator` class so that it compares strings lexicographically, based on the order passed into the `AlienAlphabet` constructor. For simplicity, you may assume all words passed into `AlienComparator` have letters present in `order`.

For example, if the alien alphabet has the order `"dba..."`, which means that `d` is the first letter, `b` is the second letter, etc., then `AlienComparator.compare("dab", "bad")` should return a negative value, since `dab` comes before `bad`.

If one word is an exact prefix of another, the longer word comes later. For example, `"bad"` comes before `"badly"`. *Hint:* `indexOf` *might be helpful.*

```
1  public class AlienAlphabet {
2      private String order;
3      public AlienAlphabet(String alphabetOrder) {
4          order = alphabetOrder;
5      }
6      public class AlienComparator implements Comparator<____________> {
7          public int compare(String word1, String word2) {
8
9              int minLength = Math.min(_________________________, _________________________);
10
11             for (_____________________________________________________________________) {
12
13                 int char1Rank = _____________________________________________;
14
15                 int char2Rank = _____________________________________________;
16
17                 if (_____________________________________________) {
18                     return -1;
19
20                 } else if (_____________________________________________) {
21                     return 1;
22                 }
23             }
24
25             return _________________________________ - _________________________________;
26         }
27     }
28 }
```

2 Iterator of Iterators

Implement an `IteratorOfIterators` which takes in a `List` of `Iterators` of `Integers` as an argument . The first call to `next()` should return the first item from the first iterator in the list. The second call should return the first item from the second iterator in the list. If the list contained `n` iterators, the `n+1`th time that we call `next()`, we would return the second item of the first iterator in the list.

Note that if an iterator is empty in this process, we continue to the next iterator. Then, once all the iterators are empty, `hasNext` should return **`false`**. For example, if we had 3 `Iterators` A, B, and C such that A contained the values `[1, 3, 4, 5]`, B was empty, and C contained the values `[2]`, calls to `next()` for our `IteratorOfIterators` would return `[1, 2, 3, 4, 5]`.

```
import java.util.*;
public class IteratorOfIterators  _____________________________ {


    public IteratorOfIterators(List<Iterator<Integer>> a) {









    }

    @Override
    public boolean hasNext() {









    }

    @Override
    public Integer next() {









    }
}
```