CS 61B Fall 2024

Comparators, Iterators

Exam-Level 04: September 23, 2024

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. Fill in `AlienComparator` class so that it compares strings lexicographically, based on the order passed into the constructor. 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"`. The following are some functions from the `String` class that you might find useful:

- `public int indexOf(char c)`: Returns the index of the first occurrence of the specified character in this string, or -1 if the character does not occur.
- `public char charAt(int index)`: Returns the character at the specified index. For instance, `"hello".charAt(1)` returns `'e'`.
- `public int length()`: Returns the length of this string.

```
public class AlienComparator implements Comparator<____________> {
    private String order;
    public AlienAlphabet(String alphabetOrder) { order = alphabetOrder; }

    public int compare(String word1, String word2) {

        int minLength = Math.min(_____________________, _____________________);

        for (_____________________________________________________________) {

            int char1Rank = _____________________________________________;

            int char2Rank = _____________________________________________;

            if (_____________________________________________) {
                return -1;

            } else if (_____________________________________________) {
                return 1;
            }
        }

        return _____________________________ - _____________________________;
    }
}
```

Solution:

```
public class AlienAlphabet {
    private String order;

    public AlienAlphabet(String alphabetOrder) {
        order = alphabetOrder;
    }

    public class AlienComparator implements Comparator<String> {
        public int compare(String word1, String word2) {
            int minLength = Math.min(word1.length(), word2.length());
            for (int i = 0; i < minLength; i++) {
                int char1Rank = order.indexOf(word1.charAt(i));
                int char2Rank = order.indexOf(word2.charAt(i));
                if (char1Rank < char2Rank) {
                    return -1;
                } else if (char1Rank > char2Rank) {
                    return 1;
                }
            }

            return word1.length() - word2.length();
        }
    }
}
```

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]`.

```
public class IteratorOfIterators ____________________________________________ {
    private List<Iterator<Integer>> iterators;
    private int curr;
    public IteratorOfIterators(List<Iterator<Integer>> a) {
        iterators = new LinkedList<>();
        for (____________________________________________) {
            if (____________________________________________) {
                ____________________________________________;
            }
        }
        curr = 0;
    }
    @Override
    public boolean hasNext() {
        return ____________________________________________;
    }

    @Override
    public Integer next() {
        if (!hasNext()) { throw new NoSuchElementException(); }

        Iterator<Integer> currIterator = ____________________________________________;

        int result = ____________________________________________;

        if (____________________________________________) {

            ____________________________________________;

        } else {

            curr = ____________________________________________;
        }
        return result;
    }
}
```

Solution:

```
public class IteratorOfIterators implements Iterator<Integer> {

    private List<Iterator<Integer>> iterators;
    private int curr;

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

        iterators = new LinkedList<>();
        for (Iterator<Integer> iterator : a) {
            if (iterator.hasNext()) {
                iterators.add(iterator);
            }
        }

        curr = 0;
    }

    @Override
    public boolean hasNext() {
        return !iterators.isEmpty();
    }

    @Override
    public Integer next() {
        if (!hasNext()) {
            throw new NoSuchElementException();
        }

        Iterator<Integer> currIterator = iterators.get(curr);
        int result = iterators.get(curr).next();
        if (!currIterator.hasNext()) {
            iterators.remove(curr);
        } else {
            curr = (curr + 1) % iterators.size();
        }

        return result;
    }
}
```

For this problem, we use the instance variable `iterators` to store all the iterators that still has elements. We use `curr` to indicate the next iterator to get the next element from. Therefore, in the constructor, we initialize `iterators` by iterating through the input list of iterators and adding the iterators that are not empty. We then initialize `curr` to 0. For the `hasNext()` method, we can test whether our list `iterators` is empty. For the `next()` method, we first check if there are any elements left to iterate through (throwing an error if we do not have). If there are, we get the current iterator and the next element from that iterator.

If the iterator is empty, we remove it from the list of iterators. Otherwise, we increment `curr` to the next iterator. Notice that we do not increment `curr` if we remove an iterator from the list, as all the indices of the following iterators will shift by 1, and next iterator will take its place.