Lab: Iterators

In this lab you will implement client applications that use lists with iterators.

Java File

- *Subsequence.java*

Introduction

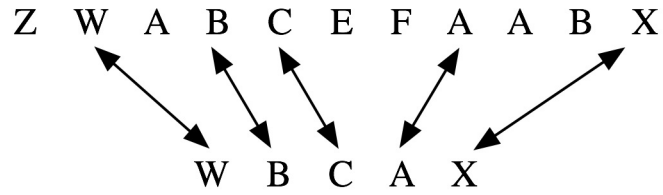
An iterator is an object that allows you to access the items stored in a data structure sequentially. This has two major advantages. The first advantage is that because many very different kinds of data structures have iterators defined for them, code can be written that will work independent of the choice of data structure. That code is protected against changes in data structure. For example, using Java's `Iterator`, here is code that will remove all items from a data structure holding items of type X.

```
Iterator<X> toClear = someDataStructure.iterator();
while(toClear.hasNext()) {
   toClear.next();
   toClear.remove();
}
```

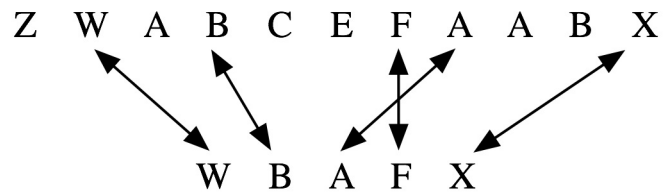
As long as the object `someDataStructure` has implemented the `Iterable` interface, the rest of the code is insulated from change. Since sequential access to a collection of items is very common, iterators are fairly useful.

The second advantage is that the iterator may be specialized to provide fast sequential access to the items in the collection. For example, consider a list that uses a linked chain of nodes. The items in a simple singly linked chain could be accessed one at a time using a get-entry method. The only problem with this is that each time a get-entry executes, the chain must be traversed from the front. An iterator would be able to keep a reference to the nodes in the linked chain and would not have to restart from the beginning for each access.

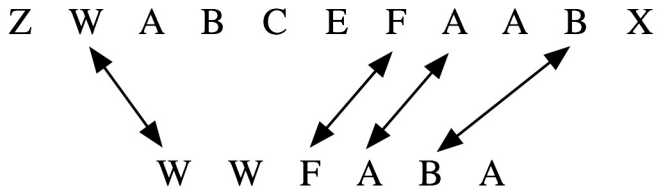
In this lab, you will develop a method that will determine if one sequence is a subsequence of another. For one string to be a subsequence of the other, all letters in the first string must match up uniquely with a letter in the second string. The matches have to be the same order, but they need not be consecutive. For example, WBCAX is a subsequence of ZWABCEFAABX as we can see from the matching.



On the other hand, WBAFX is not a subsequence of ZWABCEFAABX since there is no way to match up the letters in the correct order.



As another example, WWFABA is not a subsequence of ZWABCEFAABX. There are a couple issues that we run into with this third example. First, we can only match up one character with one character so the subsequence check fails due to an excess of W's. Second, while ABA is a subsequence of ZWABCEFAABX, FABA is not.



Here is a formal definition of a sequence:

> Given $X = < x_1, x_2, ..., x_n >$ a sequence of length n greater than or equal to zero and $Y = < y_1, y_2, ..., y_m >$ a sequence of length m greater than or equal to zero, the function subsequence(X, Y) is true if and only if there exists a strictly increasing sequence of indices $K = < k_1, k_2, ..., k_n >$ such that every element x_i is equal to y_j where $j = k_i$.

Consider the following sequences.

> X = < a b a > and Y = < b c a c b a>

X is a subsequence of Y because you can find a sequence of indices that satisfies the requirements. Keep in mind that K represents the sequence of index values from Y that correspond to the element from X.

> K = < 3 5 6>
>
> x_1 = a is equal to y_3 = a
>
> x_2 = b is equal to y_5 = b
>
> x_3 = a is equal to y_6 = a

The requirement that the sequence of indices, K, is strictly increasing means that the items we pick out of Y must be in the same order as the ones from X. Since there is no requirement that the indices in K are consecutive, there is no requirement that the items we pick from Y are consecutive. In other words, there can be other elements between each selected element in Y. (A substring test is similar to the subsequence test but it requires consecutive indices.)

Finally, because the indices in K must be strictly increasing, we cannot reuse a value in Y. Here are some examples of pairs that are <u>not</u> subsequences.

> X = < a b a a> and Y = < b c a c b a>
>
> X = < a b a > and Y = < b c a c b >
>
> X = < a b c> and Y = < b c a c b a>

You might be surprised to learn that the following pair is a subsequence.

> X = < > and Y = < b c a c b a>

In this, case X is a subsequence of Y because you can find a sequence of indices, K = < >, that satisfies the requirements. Notice that every element in X is equal to some element of Y trivially. If this were not the case, you would be able to demonstrate an element from X that is not equal any element of Y.

Prepare for Lab - Subsequence

Come up with an example of two sequences X and Y where you can guarantee that X is not a subsequence of Y without having to look at the values of the items in the sequences.

Lets look at a matching algorithm in detail. Consider the sequences

X = < a b c a >

Y = < b c d a e c a e a f b a b b d e c d e a b d >

Match 1:

The first “a” in X must be matched with an “a” in Y. Can it be safely matched with the first “a” in Y?

(Safe in the sense that it does not prevent us from matching a later item from X.)

Yes

Can any items before the first “a” in Y be matched with an item from X?

Yes

What positions in Y must be checked to find the “a” ?

1,2,3,4

To visualize, you can cross out the items from both lists that have been checked to make the first match.

X = < ~~a~~ b c a >

Y = < ~~b c d a~~ e c a e a f b a b b d e c d e a b d >

Match 2:

The "b" in X must be matched with a “b” in Y. What positions in Y must be checked to find the “b” ?

5,6,7,8,9,10,11

To visualize, you can cross out the items from both lists that have been checked to make the first and second matches.

X = < ~~a b~~ c a >

Y = < b ~~c d a e c a e a f b~~ a b b d e c d e a b d >

Match 3:

The “c” in X must be matched with a “c” in Y. What positions in Y must be checked to find the “c”?

12,13,14,15,16,17

To visualize, you can cross out the items from both lists that have been checked to make the first three matches.

X = < ~~a b c~~ a >

Y = < ~~b c d a e c a e a f b a b b d e c~~ d e a b d >

Match 4:

The second “a” in X must be matched with an “a” in Y. What positions in Y must be checked to find the “a”?

18,19,20

To visualize, you can cross out the items from both lists that have been checked to make the first four matches.

X = < a b c a >

Y = < b c d a e c a e a f b a b b d e c d e a b d >

An iterator is appropriate if you checked the values sequentially. Did you check the values in both X and Y sequentially?

Yes

Develop an algorithm to detect a subsequence that uses two iterators (one `Iterator` for each sequence). .

If X and Y are empty, return true
If Y has values and X is empty, return true
If Y is empty and X has values, return false
Iterator one will go through X's values while two will go through Y's values
While Iterator 2 hasNext && Iterator1 hasNext{
if iterator 1 == iterator 2 {
Iterator1.next and Iterator2.next} else {
Iterator2.next}}
if Iterator 1 == iterator 2, return true
if iterator 1 does not have next return true
return false

Does the algorithm work if

i. X is an empty sequence?

ii. X is a sequence of length one?

iii. Y is an empty sequence?

iv. Y is a sequence of length one?

Directed Lab Work

Subsequence

Download the started code for *`Subsequence.java`*.

Step 1. Compile the class `Subsequence`. Run the `main` method in `Subsequence`.

Checkpoint: If all has gone well, the program will run and accept two lines of words for input. It will fail most test cases.

Step 2. Refer to your algorithm from the pre-lab exercises and complete the `subSequence()` method.

Final checkpoint: Compile and run the program. All tests should pass. If not, debug the code so that it works correctly, then submit your Subsequence.java file to Gradescope,