Lab 2 Bag Client

Goal

In this lab you will complete an application that uses the Abstract Data Type (ADT) bag.

Resources

- Chapter 1: Bags

In `javadoc` directory

- *`BagInterface.html`*—Interface documentation for the interface `BagInterface`

Java Files

- *`ArrayBag.java`*
- *`BagInterface.java`*
- *`LongestCommonSubsequence.java`*

Introduction

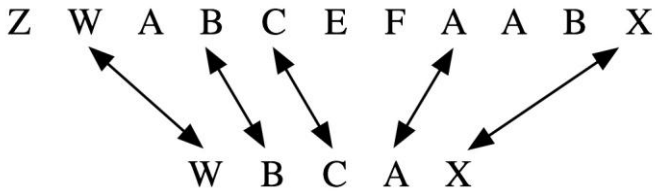
The ADT bag is one of the primary collection structures defined in mathematics. It is a generalization of a set that is allowed to hold multiple copies of an item. Like a set, the items contained within the set have no particular ordering. Before continuing the lab you should review the material in Chapter 1. In particular, review the documentation of the interface *`BagInterface.java`*. While not all of the methods will be used in our application, most of them will.

In the application, we will take as input two strings of characters and determine the longest subsequence of characters that is common to both strings. This kind of computation has uses in determining how similar two genes are.

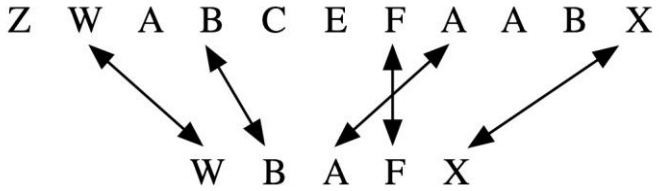
Pre-Lab Visualization

The Pre-lab Visualization gives a number of exercises intended to help you think about the problem and prepare you to write the code. You should do these exercises before doing any work at a computer. This kind of preparatory work is a good habit to get into and will increase the quality of your code.

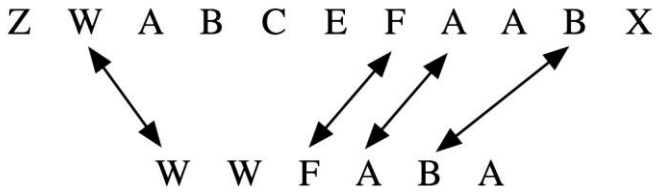
We want to find the longest sequence of letters that is common between two strings. For one string to be a subsequence of the other, all letters in the first string must match up uniquely with a letter in the second string. The matches have to be the same order, but they need not be consecutive. For 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.



 Write an algorithm for a method that given two strings (test, against) will return true if the first is a subsequence of the second. (Hint: We will accept an empty string as a subsequence of any other string.)

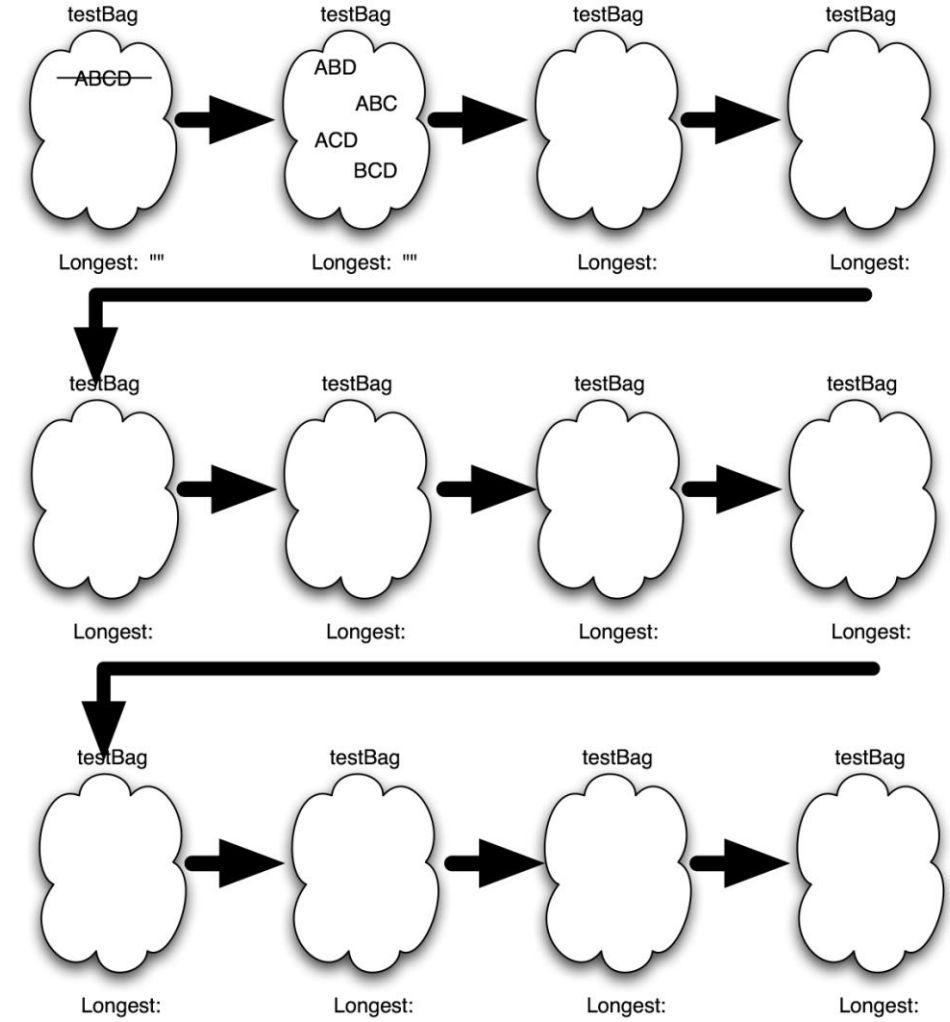
Finding A Longest Common Subsequence

Now that we have an algorithm that determines if one string is a subsequence of another, we will examine an algorithm that will find the longest string that is a substring of two input strings. Our algorithm will take a brute force approach of generating all possible subsequences and checking them. While our algorithm will work, there are much more efficient algorithms that should be used in a production setting.

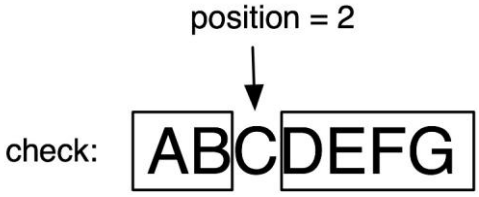
Longest Common Subsequence (first, second)
Create an empty bag
Put the first string into the bag
Set the longest match (subsequence) to the empty string
While the bag is not empty
Remove a test string from the bag
If the longest match is shorter than the test string
If the test string is a subsequence of the second string
Set the longest match to the test string
Otherwise if the test string is at least two longer than the longest match
Generate new strings from test by removing each single character
Put the new strings into the bag
Print the bag of strings to check.

Report the longest match

Complete the trace of this algorithm on the input first = "ABCD" and second = "FAC". Every time a string is removed from the bag, cross it out. Use a new section when a group of new strings are added. The first iteration of the while loop has been done for you. The number of iterations of the while loop that are required for this trace depends on the order that the strings are removed from the bag. It could be as few as 9 or as many as 41.



One detail of this algorithm that we want to explore further before creating code is the step that generates all strings that are one smaller than the test string. Given a position in the string, we can use the substring() method to get the characters before and after that position.



 Given a string check and an integer position, write down an expression that concatenates the two substrings from before and after the position.

Directed Lab Work

The skeleton of the `LongestCommonSubsequence` class already exists and is in `LongestCommonSubsequence.java`.

Step 1. Look at the skeleton in `LongestCommonSubsequence.java`. Compile and run the `main` method.

Checkpoint: If all has gone well, the program will run and accept input. It will report that the strings to check is null and give the empty string as the longest common subsequence. The goal now is to create the bag of strings to check.

Step 2. In `main` create a new bag and assign it to `toCheckContainer`. Add in the string `first`.

Step 3. Print out the bag `toCheckContainer`.

Checkpoint: Compile and run the program. Enter ABD and BCD for the two strings. You should see Bag [ABD] The next goal is to do take a string from the bag and see if it is a subsequence of the input string second. This check will be encapsulated in the method isSubsequence().

Step 4. Refer to the pre-lab exercises and complete the method `isSubsequence()`.

Step 5. Remove an item from `toCheckContainer` and store it in an `String` variable.

Step 6. Call the method `isSubsequence`() with the string you just removed from the bag and the second input string. If the method returned true, report that it was a subsequence

Checkpoint: Compile and run the program. Enter A and ABC for the two input strings. The code should report that it was a subseqence. The following are some pairs of strings and the expected result.

D	*ABC*	*no*
AA	*ABA*	*yes*
AA	*AAA*	*yes*
AABC	*ABBC*	*no*
ABBC	*ABCC*	*no*
ABCC	*CABAC*	*no*
ABA	*AA*	*no*
ABC	*CBACBA*	*no*
ABC	*CBACBACBA*	*yes*
ABC	*BCABCA*	*yes*
ABCD	*DCBADCBA*	*no*
ABFCD	*ADBAFDCBA*	*no*
ABFCD	*ADBADCBA*	*no*
ABCDF	*ADBADCBA*	*no*
ABADA	*BADABAABDBA*	*yes*

The next goal is to complete the body of the while loop in our main method.

Step 7. The following is the algorithm we saw in the pre-lab for computing the longest common subsequence. We have already completed parts of it, but now you should finish everything ***but the while loop***.

1. ***Put the first string into the bag***
2. ***Set the longest match to the empty string***
3. ***While the bag is not empty***
    - ***3.1 Remove a test string from the bag***
    - ***3.2 If the longest match is shorter than the test string***
        - ***3.2.1 If the test string is a subsequence of the second string***
            - ***3.2.1.1 Set the longest match to the test string***
        - ***3.2.2 Otherwise if the test string is at least two longer than the longest match***
            - ***3.2.2.1 Generate new strings and put them into the bag***
    - ***3.3 Print the bag of strings to check***
4. ***Report the longest match***

The only tricky part of this is the generation step 3.2.2.1. You will need to create a `for` loop that loops over the positions of characters in the test string and creates the smaller test string as discussed in the pre-lab exercises.

Checkpoint: Compile and run the program. Enter ABCD and FAC for the two input strings. The code should report that the new bag is Bag[BCD ACD ABD ABC] and the longest subsequence should remain empty.

Run the code again and enter BA and ABCA for the two input strings. The code should report that the new bag is Bag[ ] and the longest subsequence is BA.

Run the code again and enter ABA and ABCB for the two input strings. The code should report that the new bag is Bag[BA AA AB] and the longest subsequence should remain empty

Run the code again and enter D and ABCA for the two input strings. The code should report that the new bag is Bag[ ] and the longest subsequence should remain empty

For our final goal, we will add in the code that loops over the items in the bag

Step 8. Wrap the code from steps 3.1 to 3.3 in the algorithm in a `while` loop that continues as long as the `toCheckContainer` bag is not empty.

Final checkpoint: Compile and run the program. Enter ABCD and FAC for input strings. Compare the results with the pre-lab exercises. Reconcile any differences.

Run the program with the following input strings and check the results. (Note that the longest common subsequence is not unique. As long as you find one of the right length, it's ok.)

		Longest Common Subsequence
D	*ABC*	
AA	*ABA*	*AA*
AA	*AAA*	*AA*
AABC	*ABBC*	*ABC*
ABBC	*ABCC*	*ABC*
ABCC	*CABAC*	*ABC*
ABA	*AA*	*AA*
ABC	*CBACBA*	*AB or BC or AC*
ABC	*CBACBACBA*	*ABC*
ABC	*BCABCA*	*ABC*
ABCD	*DCBADCBA*	*AB or AC or AD or BC or BD or CD*
ABFCD	*ADBAFDCBA*	*ABFC or ABFD*
ABFCD	*ADBADCBA*	*ABC or ABD*
ABCDF	*ADBADCBA*	*ABC or ABD*
ABADA	*BADABAABDBA*	*ABADA*

Post-Lab Follow-Ups

1. Modify the LongestCommonSubsequence program so that it will report an error if there is a bag overflow. Run tests to determine how big a string we can accommodate before overflow errors start to occur. Given the size of the bag, come up with a mathematical formula to determine the largest size string that is guaranteed not to cause an overflow. (Note that if the first string is a subsequence of the second string, we only need to be able to place a single string in the bag. Therefore, as long as the bag can hold one item, we can find cases that will work for an arbitrarly large input string.)

2. We know that the longest common subsequence of two strings can never be longer than the shorter string. Modify the LongestCommonSubsequence program so that it will use the shorter string to generate the test strings.

3. Modify the LongestCommonSubsequence program so that it will determine the longest common subsequence of three input strings.

4. A palindrome is a string that reads the same forwards and backwards. Write a program that reads in a string and determines the longest subsequence that is a palindrome. Note that since any string with just a single character is a palindrome, every non-empty string will have a palidromic subsequence of length one.