CS 0445 – Data Structures (COE 0445)

Sherif Khattab

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
- BagTest.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.

Directed Lab Work

Step 0. Read the PPT file that comes with the lab instructions.

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 string 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 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 PPT file 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 subsequence. 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 an algorithm 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 PPT file.

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. 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.)

String 1	**String 2**	**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

- 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 arbitrarily large input string.)
- 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.
- Modify the LongestCommonSubsequence program so that it will determine the longest common subsequence of three input strings.
- 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 palindromic subsequence of length one.