**Computer Science 2**   **Lab # 10**



**Dr. Hanh Pham**

**Student Last Name: Martinez Student First Name: Adriel**

**CS2 Section # 01**

**Due:** Problem A by the **end of the lab** and Problems B by the end of **Saturday** of the same week.

**TOPIC: ADT**

**Problem A:**

Please fill-in the field following this symbol images:bighand.jpg with your own words and in **BLUE font color**

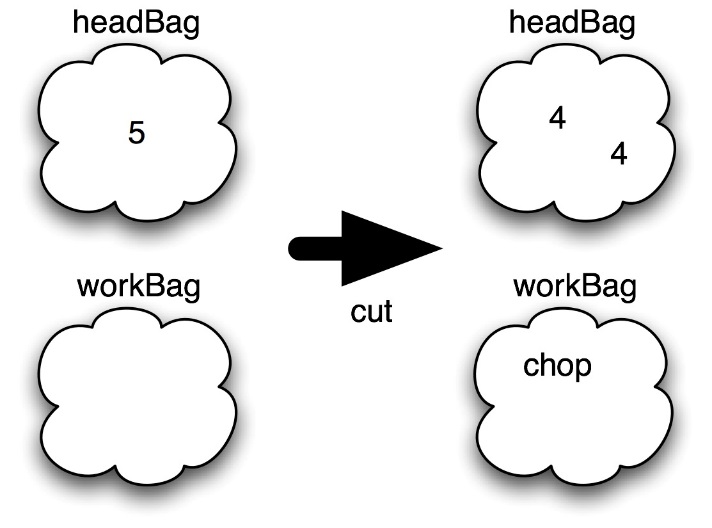
## Pre-Lab Visualization

### Hydra

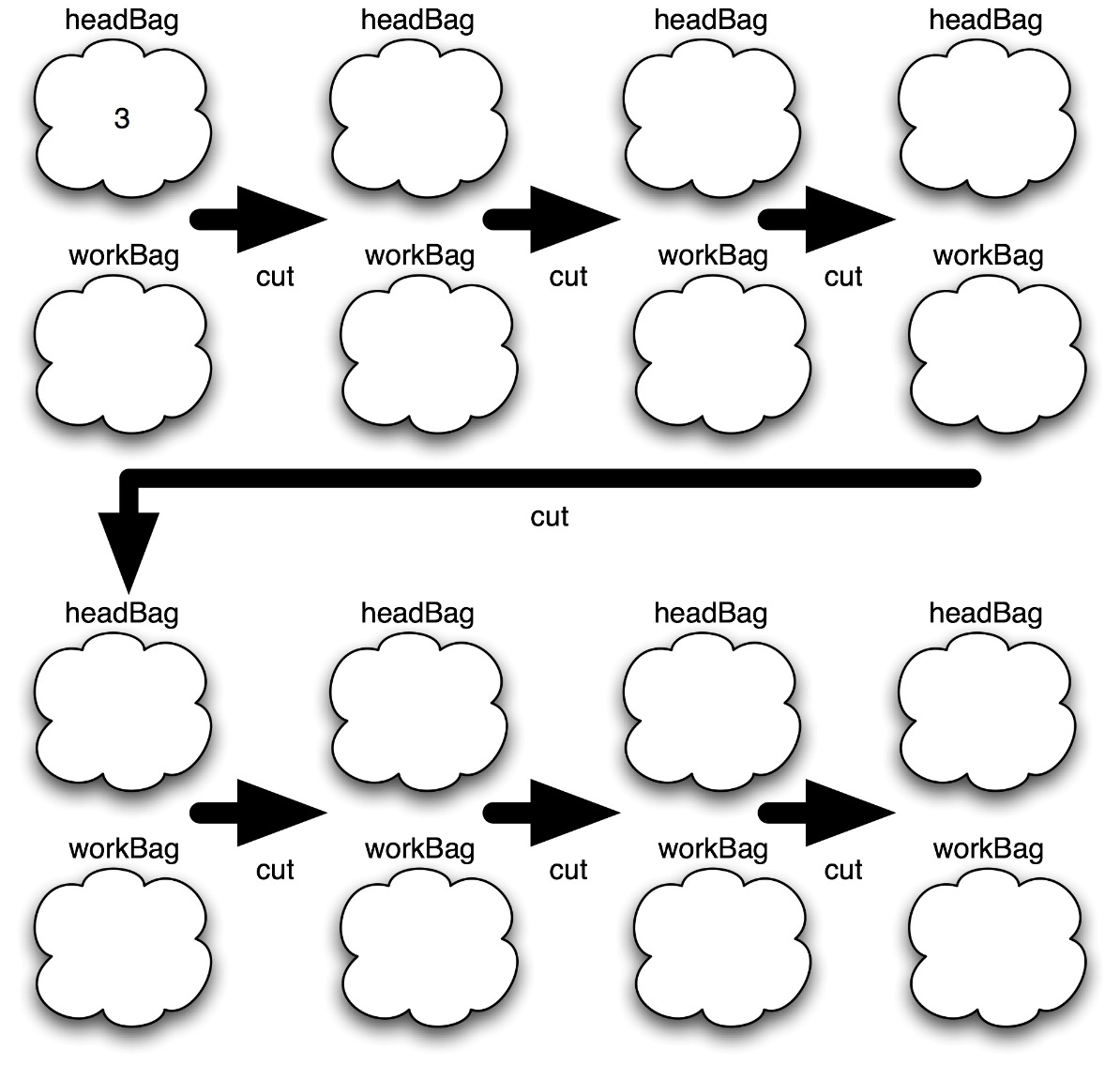
We can view our hydra as a collection of heads, each of which has a size. To indicate the size, we will use an integer value. Each time we cut off a head it is replaced by two smaller heads that are one size smaller. For example, if we chop off a head of size 5, two heads of size 4 spring up in its place. The exception to this rule is that a size 1 head does not grow back. (Fortunately for us, otherwise we would never finish.) A bag is perfect to represent the state of the hydra as the fight continues. We need to know what heads the hydra currently has and what the size of each of the heads is, but they are in no particular order. In addition, there can be multiple heads of the same size.

We will use a second bag to accumulate the answer to “How many cuts did it take to kill the hydra?”. Each time we cut off a head, we will put the string “chop” into the bag. Again, a bag will work well. We don’t care about the order of the strings in the bag and we will certainly have duplicates. At the end of the simulation, the number of strings in the bag will give us the answer to the question.

We want to visualize the process of the simulation as a series of steps and from that determine an algorithm. For example, if we start with one head of size 5, one cut results in the following transition.



images:bighand.jpg Using the above as a model, complete the seven steps in the simulation for a hydra starting with a single head of size 3.



Chop chop

Chop chop

chop chop

chop

Chop chop

Chop chop

chop chop

Chop chop

Chop chop

chop

Chop chop

chop

chop

1

1 1

1 1

1

1 1

2

chop

chop

chop

chop

chop

chop

1 1

1 1

2

2

images:bighand.jpg Examine your sample simulation and give an algorithm for what to do during a single step.

Single Step: A number is chopped into two numbers that are one less than the original number. A chop is also added to the workbag.

images:bighand.jpg Given your previous algorithm, come up with an algorithm that performs the simulation. Don’t forget to do initialization and report the result.

Hydra: Two bags are initialized, a headbag and a workbag. The program will go through the headbag and chop the integer in the first headbag into two separate integers that are one less than the number of the first headbag in the next headbag. When a chop is made, a chop is added to the workbag.

There is one issue that we need to be aware of with the bag ADT. The add() method may not always succeed. If there is not enough space in the bag to add the item, the add method will return false and the item will not be added into the bag. Obviously, this will have an effect on our simulation. Every time we add an item into a bag, we need to examine the returned value. If we ever get false, we can immediately stop the simulation and report that there was a problem.

images:bighand.jpg Modify your single step algorithm from before so that it is a method which returns true if the step was successful and false otherwise.

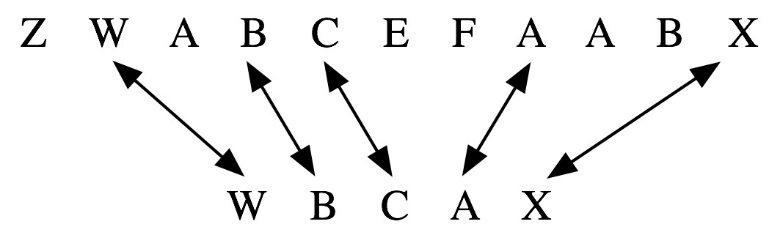
Single Step: The program will check after you chopped an item if there is enough space left in the bag to add another item, if there is no space remaining then the method will return false and stop the simulation and report the problem.

images:bighand.jpg Modify your program algorithm from before so that it will end early if there is a bag overflow.

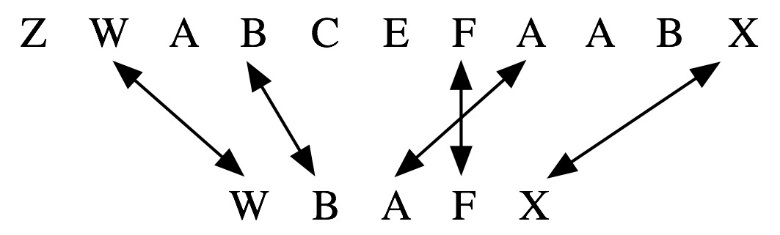
Hydra: Two bags are initialized, a headbag and a workbag. The program will go through the headbag and chop the integer in the first headbag into two separate integers that are one less than the number of the first headbag in the next headbag. When a chop is made, a chop is added to the workbag. Before the headbag goes on to chop again, a method will check if it’s possible for more items to be added to the bag, preventing the bag from overflowing.

### Longest Common Subsequence

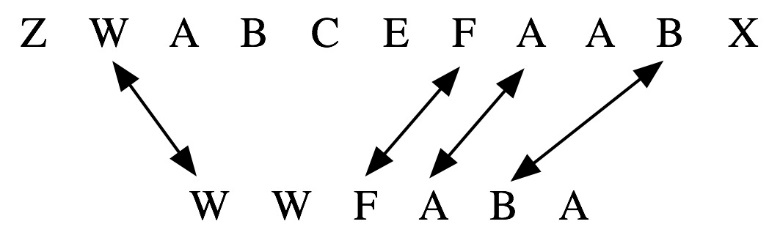
We want to find the longest sequence of letters that is common between two strings. For one string to be a subsequence of the other, all letters in the first string must match up uniquely with a letter in the second string. The matches have to be the same order, but they need not be consecutive. For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



images:bighand.jpg Write an algorthim 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.)

Before the method splits the test string, it is checked to see if it is longer than the largest match, and if the test string is a subsequence of the second string, the longest match is the test string. If the test string is not a subsequence, then the test string is split into new strings that removing a single character from the test string.

* Test string is checked to see if it is longer than the largest match, and if so, the test string is checked to see if it is a subsequence of the second string.
  + If the test string is a subsequence of the second, then the longest match is the test string.
* Check the subsequence string length and the test string length to see if they match.
* If the test is not a subsequence of the second, then the test is split into new strings that removes a single character from the test 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*

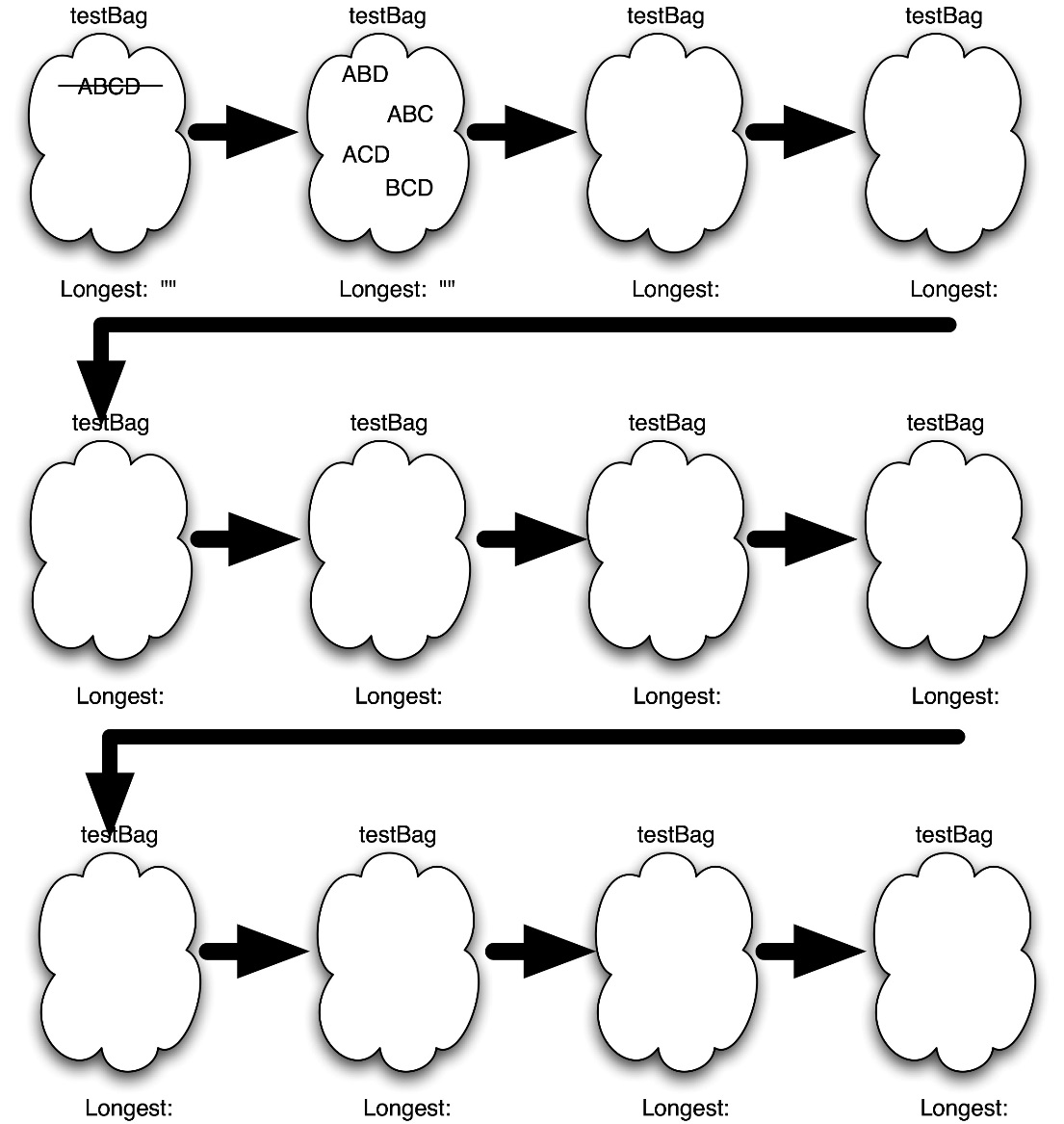
*Generate new strings from test by removing each single character*

*Put the new strings into the bag*

*Print the bag of strings to check.*

*Report the longest match*

images:bighand.jpg 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.



AC

AC

~~AB~~

AC

AB

AC

AB ~~BC~~

AC

AB BC

AC

AB BC

~~ACD~~

AC

AB BC

ACD

~~BCD~~

AC

AB BC

ACD

BCD

~~ABD~~

AB BC

~~AC~~

ACD

ABD

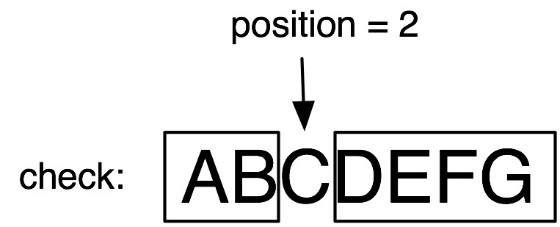
BCD

AC

AB BC AC ABD ACD BCD

~~ABC~~

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.



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

String1.substring(0,2)+String2.substring(3);