95-771 Data Structures and Algorithms Project 1

Due: Tuesday, January 28, 2025 at 11:59:00 PM

This project has three objectives. First, the student will be introduced to the linked list data structure using Michael Main's ObjectNode.java. Second, the student will use the linked list class created in Part 1 to implement Merkle-Hellman Knapsack cryptography. Third, the student will use the same linked list to build a Merkle Tree.

The Java collection classes may not be used for this assignment.

Part 1.Building a LinkedList class 30 points

Create a Java application in IntelliJ with the name ObjectNode-Project. Download Michael Main's ObjectNode.java class.

Documentation for the ObjectNode class may be found here:
http://www.cs.colorado.edu/~main/docs/

The source code (which you need to get running in IntelliJ) is found on our course schedule at this location:
http://www.andrew.cmu.edu/user/mm6/95-771/Homeworks/ObjectNode.java

Make the following modifications to ObjectNode.java.

0. We will be using packages. Leave ObjectNode.java in the package within which it is defined (edu.colorado.nodes). Later, you will place your own code in a package named edu.cmu.andrew.<your ID>. For example, my SinglyLinkedList class is found within the package edu.cmu.andrew.mm6.
1. Add a recursive listCopy() called listCopy_rec(). This new method will behave the same way as Main's listCopy() method but will be recursive.
2. Add a recursive listLength() called listLength_rec(). This new method will behave the same way as Main's listLength() method but will be recursive.
3. Add a method called displayEveryThird() that displays every third node on the list. If the list contains 1-2-3-4-5-6

then your routine will display 3 and 6. If the list contains 1-2-3-4-5 then your routine will display only 3.

4. Add a comment to each method in ObjectNode.java. Your comment will describe its runtime in big theta notation.
5. Add pre-conditions comments just prior to ListCopy() and ListCopyWithTail(). What precondition makes sense?
6. Add an instance method String toString() to the ObjectNode class. This method returns a string holding the data of each node on the list beginning with head.
7. Provide a main routine (a driver) for this class. The main routine will do the following activities.
   a. Create a list containing 26 nodes. Each node will contain the letters of the English language in order in lower case. The list will hold "a" → "b" → "c" → … -> "z"--||
   b. Call toString() on the front node of the list to display the list data. The output will be abcd…z.
   c. Call displayEveryThird(). The output will be cfi etc.
   d. Print the size of this list twice. Once with listLength() and again with listLength_rec().
   e. Make a copy of the list into a new list, use listCopy(), with the front node of the new list being pointed to by an ObjectNode named k.
   f. On the ObjectNode named k, call its toString() method. The output will be abcd…z.
   g. Print the size of the list k twice. Once with listLength() and again with listLength_rec.
   h. Make a copy of the list into a new list, use listCopy_rec(), with the front node of the new list being pointed to by an ObjectNode named k2.
   i. On k2, call its toString() method. The output will be abcd..z.
   j. Print the size of the list k2 twice. Once with listLength() and again with listLength_rec.
   k. Leave this main driver in your submission for part 1. The grader will be able to run the main routine of the ObjectNode class.
   l. Your output from this section will look like the following:

```
abcdefghijklmnopqrstuvwxyz
cfil etc.
Number of nodes = 26
Number of nodes = 26
abcdefghijklmnopqrstuvwxyz
Number of nodes in k = 26
```

```
Number of nodes in k = 26
abcdefghijklmnopqrstuvwxyz
Number of nodes in k2 = 26
Number of nodes in k2 = 26
```

8. Write a class named SinglyLinkedList.java that uses a head and tail pointer. The head reference always points to the head of the list and the tail reference always points to the last node on the list. Javadoc for this class is provided here. Also, included is Javadoc for the ObjectNode class. It is found under http://www.andrew.cmu.edu/~mm6/95-771/ObjectNodeProject/dist/javadoc/index.html
9. Write code in the main method of SinglyLinkedList.java. This code should test each method of the class. In particular, it must include testing of list iteration using reset(), hasNext() and next(). You need to add these three methods to the linked list class. If s is a list, this code will display its contents from the main routine:

```
s.reset();
while(s.hasNext()) {
    System.out.println(s.next());
}
```

   Also, see the Javadoc on the schedule for a description of these three methods.

10. Write a class called OrderedLinkedListOfIntegers. This class will be of your own design. You are free to use any code that you have written or worked on above. It must allow integers to be added to the list by making calls to a instance method named sortedAdd(). The sortedAdd() method always maintain the list of integers in increasing order.

    It will also provide for list iteration (as in 9).

    Provide an efficient static method named merge() that returns a new OrderedLinkedListOfIntegers that holds the merged contents of its two OrderedLinkedListOfIntegers parameters. Write a main routine that demonstrates adding 20 random values to two OrderedLinkedListOfIntegers and merging them into a third list of size 40. In order for the main routine to merge the contents, it must call the merge()

method that you have written. Duplicate integers may be present in your lists. Of course, the number 20 is just for testing and your solution must accommodate a larger or smaller number. If two empty lists are passed to merge() then merge() will return an empty list.

An inefficient way to write merge() would be to simply call sortedAdd() during a traversal of the first list, and then the second. Your solution should be better than that and run in worst case time of $\Theta(m + n)$ where m is the size of the first list and n is the size of the second. Be sure to state an appropriate pre-condition on your merge() method.

Part 1 Grading:

Comment with your name, course, and assignment number
Working code with three working main routines: 70%
Comments describing big theta of each routine and preconditions and postconditions where appropriate 20%
Screenshots demonstrating program execution: 10%
Be sure to site the source of any code that you copy.

Part 2. Use Part 1 to implement a Merkle-Hellman Knapsack Cryptosystem 30 Points

Create a Java application in IntelliJ with the name Merkle-Hellman-Knapsack-Crypto-Project

In this part you will implement key generation, encryption and decryption using the Merkle-Hellman Knapsack Cryptosystem. A very clear and well-written description of this algorithm can be found at the following link. This is a required reading for the course and should be understood prior to writing code:

http://en.wikipedia.org/wiki/Merkle–Hellman_knapsack_cryptosystem

Note that the example provided on the wiki is an example using small integers with w = {2, 7, 11, 21, 42, 89, 180, 354}. In this project, w will consist of 640 huge integers.

You will use your singly linked list class of objects (from part 1) to hold two lists of Java BigIntegers. One list, w, will be used to hold the superincreasing sequence of integers that make up part of the private key and used for decryption. You are required to use powers of 7 to make up your superincreasing sequence. The second list, b, will be used to hold the public key material used for encryption. Your list class should encapsulate all of the work associated with lists and should not know anything about Merkle-Hellman.

Using a singly linked list for this problem is appropriate but not ideal. It is very appropriate for a first course in data structures. Hence, use it for this project but you should be aware that there are alternatives.

Use the built in methods of the BigInteger class provided by Java. These methods make it fairly easy to implement some of the tedious but essential parts of Merkle-Hellman.

To do a practice example, you might like to use Wolfram Alpha at http://www.wolframalpha.com. It accepts operations such as (3 * 5) mod 2, gcd(32,5) and PowerMod(15,-1,64) to find the multiplicative inverse of 15 modulo 64.

Your program will be interactive and will behave in a similar manner as mine. An example run of my program appears below. You will need to submit several screenshots showing example executions of your code. Note that you may assume that the user will enter a string of less than 80 characters in length. If the user enters a longer string, inform them that the string entered is too long and ask the user to try again.

Hint: Since your program must handle 80 characters of input and since each character can be represented in 8 bits, your lists will have (80 * 8) 640 nodes. Note that key generation is typically done once. Then, the key is used. It is not typical that the key size would depend on the size of the input (which may vary).

Example execution:

```
Enter a string and I will encrypt it as single large integer.
Welcome to Data Structures and Algorithms
Clear text:
Welcome to Data Structures and Algorithms
Number of clear text bytes = 41
Welcome to Data Structures and Algorithms is encrypted as
317817076350145785260656997739621373931469119807217110529280649332942724728174120224095587842484635884305367159163265896397856056390477056525611299805189287161133817602808317806202994211855425964737022224210974561644533817599450919039345942975178915818105632933959978787221138943336909734004773052722627400695
Result of decryption: Welcome to Data Structures and Algorithms
```

Part 2 Grading:

You need to submit:

Comments with your name, course, and assignment number
Working code performing key generation, encryption and decryption and using the singly linked lists from Part 1 to hold BigIntegers: 70%

Comments in your code describing how key generation and the Merkle Hellman Knapsack cryptography is being performed: 20%

Screenshots demonstrating program execution: 10%
Be sure to site the source of any code that you copy.

Part 3. Using Part 1 to implement a Merkle tree. 40 Points

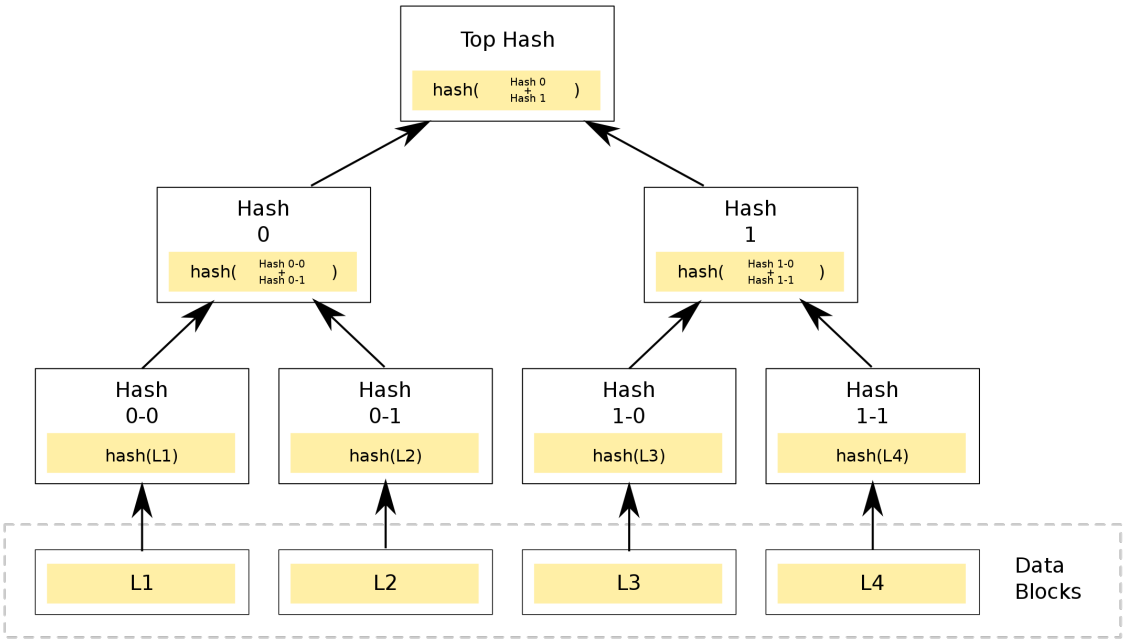
Create a Java application in IntelliJ with the name MerkleTree-Project.

In this part, you will use the classes from Part 1 to build a Merkle tree from a text file and compute the Merkle root of the Merkle tree.

You will read a UTF-8 file of text lines – delimited by line breaks. Each line will be stored in a node on a list.

Once each line is stored in a node – forming a list of lines, a second list will be created containing the cryptographic hashes of these nodes. If, after completing a list, it is found that the list has an odd number of nodes, your program will copy the last node and then add it to the end of the list – forcing all lists to be even in size – except, of course, for the Merkle root.

Your program will create a new list for each level in the Merkle tree. In the example below, your program will store these data in four distinct lists. L1, L2, L3, and L4 would be four nodes of the first list. Hash(l1), hash(l2), hash(l3) and hash(l4) will live in nodes on the second list, and so on.



Your program will implement this tree within a list of lists. The first list in the list of lists will contain the list: L1, L2, L3,L4.

A very nice description of Merkle Trees is found here: https://en.wikipedia.org/wiki/Merkle_tree

Your program will prompt the user for a file name. Your program will then read the file name, read the file, build the Merkle tree as a list of lists and display the Merkle root.

For all hashing, you are required to use the following method:

```
public static String h(String text) throws
                                        NoSuchAlgorithmException {
    MessageDigest digest = MessageDigest.getInstance("SHA-256");
    byte[] hash =
           digest.digest(text.getBytes(StandardCharsets.UTF_8));
    StringBuffer sb = new StringBuffer();
    for (int i = 0; i <= 31; i++) {
        byte b = hash[i];
        sb.append(String.format("%02X", b));
     }
     return sb.toString();
}
```

For example, suppose we have a single letter 'A' in a file. The line in the file has length 1. We are not including newlines at the end of the line.

The SHA 256 hash of 'A' is
559AEAD08264D5795D3909718CDD05ABD49572E84FE55590EEF31A88A08FDFFD

To force the number of leaves in the initial Merkle tree to be even, we create a new leaf
with 'A'. So far, our initial list looks like L0 --> 'A' --> 'A' ---||

We construct the first list of hashes from the initial list of values.

Initial list of hashes is L1 --> h('A') ---> h('A') ---->||

The actual hashes appear as follows:
559AEAD08264D5795D3909718CDD05ABD49572E84FE55590EEF31A88A08FDFFD
559AEAD08264D5795D3909718CDD05ABD49572E84FE55590EEF31A88A08FDFFD

We concatenate these two hashes and hash the concatenation to compute a new hash.

BE263C0044B95044951327B0D9ABBD7E4E3719CC1AE59B57DF059945616219C1

Since we have only a single line in the file, we are done.

The Merkel root is
BE263C0044B95044951327B0D9ABBD7E4E3719CC1AE59B57DF059945616219C1

There are four files provided for your use. Use you program to determine which of these four files has the Merkle root of:

A5A74A770E0C3922362202DAD62A97655F8652064CCCBE7D3EA2B588C7E07B58.

These files are at https://www.andrew.cmu.edu/~mm6/
95-771/Homeworks/homework1_S21/smallFile.txt
95-771/Homeworks/homework1_S21/CrimeLatLonXY.csv
95-771/Homeworks/homework1_S21/CrimeLatLonXY1990_Size2.csv
95-771/Homeworks/homework1_S21/CrimeLatLonXY1990_Size3.csv

In the comments of your main routine, show all four Merkle roots and be sure to say which one of these four files has the Merkle root that we seek.

Part 3. Grading:

You need to submit:
Working code: 70%
Comments with your name, course, and assignment number
Comments in your code describing the tree building process: 20%
Screenshots demonstrating program execution: 5%
Good style and a clean submission: 5%
Be sure to site the source of any code that you copy.

Note: A complete submission of Project 1 will contain three directories submitted to Canvas. The three directories will all be contained within a single zipped file named <yourAndrewID>Project1.zip.