

# CIS 121 - Fall 2009

Homework 4 – Friday, October 16

Due Friday, October 23, 10AM

## 4 Required Problems (100 points)

In this assignment you will implement Huffman coding as discussed in lecture. For the sake of simplicity we will use `String` objects to represent bit-sequences as opposed to real sequences of bits. JavaDoc will be provided on the course website.

You may use any built-in Java data structures for problems 2 - 4, but you may not use a `PriorityQueue` or any other built-in implementation of a heap for problem 1.

### Problem 1: 25 Points; file to submit `BinaryMinHeap.java`

Write your own Heap data structure by implementing the `BinaryMinHeapI` interface below. You can use any of the data structures you are familiar with except, of course, the `PriorityQueue` or any other Java implementation of a Heap.

**Important:** For grading purposes, please include a default constructor in your implementation.

```
public interface BinaryMinHeapI<K extends Comparable<? super K>>
{
    public boolean isEmpty();
    public K removeMin() throws NoSuchElementException;
    public void insert(K k);
}
```

### Problem 2: 20 Points; file to submit `CodeBook.java`

Implement the interface `CodeBookI` below. A `CodeBookI` contains the symbols, frequencies and encodings for character calculated by the `HuffmanBuilder` you will implement in Problem 3. Once these are known, your `CodeBook` should also be able to return the encodings of single characters or Strings. Remember, this class does not calculate the frequencies and encoding, it just stores them.

**Important:** For grading purposes, please include a default constructor in your implementation.

```
public interface CodeBookI
{
    public Set<Character> getAlphabet();
    public double getProbability(char symbol) throws InvalidSymbolException;
    public double expectedEncodingLength();
    public void putSymbol(char symbol, double frequency, String encoding)
        throws IllegalArgumentException;
    public String encode(char symbol) throws InvalidSymbolException;
    public String encode(String text) throws InvalidSymbolException;
}
```

**Problem 3: 30 Points;** file to submit `HuffmanBuilder.java`

This part is broken into two parts. You will be computing the frequencies of symbols in part (a) and their encodings in part (b). Be sure to implement both methods.

- (a) Implement the method `blankCodeBook`. Given a `String` input, calculate the frequencies (and probabilities) of all the symbols. Encodings for each symbol will be computed in part (b), so set the encoding equal to `""`. The resulting tuples (`symbol`, `frequency`, `blank_encoding`) are stored in a new `CodeBookI`.
- (b) Implement the method `buildHuffmanCode`. This method receives as input the `CodeBookI` computed in part (a) containing an alphabet and the respective probabilities of each symbol. It returns a new `CodeBookI` containing the appropriate Huffman encodings. This `CodeBookI` can then be used to encode arbitrary characters and strings.

**Important:** Your implementation of `HuffmanBuilder` might depend on your implementation for `CodeBook`. It may also depend on your implementation of `BinaryMinHeap`, if you feel uncomfortable using it (you think it may not be fully correct), then you are allowed to use the Java `PriorityQueue`. We will try to award partial credit in the case `CodeBook` isn't correct.

```
public class HuffmanBuilder
{
    public static CodeBookI blankCodeBook(String text) {}
    public static CodeBookI buildHuffmanCode(CodeBookI codebook) {}
}
```

**Problem 4: 25 Points;** file to submit `VLDecoder.java`

Implement the class `VLDecoder` below. A `VLDecoder` can decode a `String` given a `CodeBookI`. Note that `VLDecoder` assumes that the data is encoded using a variable length encoding.

```
public class VLDecoder
{
    public VLDecoder(CodeBookI codebook) {}
    public String decode(String codedString)
        throws InvalidSymbolException, IllegalArgumentException {}
}
```