# Assignment 11: Huffman Codes

**CS230**
**Fall 2000**

**Allen B. Downey**
**Computer Science Department**

Due: Wednesday 29 November

For this assigment you are going to write a program that reads a sample string and constructs a Huffman code for it. The Huffman code can then be used to encode and decode Strings. Before starting, read the handout from Standish's book.

In order to get the whole thing working, you need three pieces:

- A frequency table (like Table 8.48).
- A Huffman tree (as seen on pages 299-301).
- A code table, which contains the Huffman code for each character (like Table 8.51).

## The Frequency table

Create a class called `FreqTab` that builds a table that contains the frequency of letters in a text sample.

1. You can use an array, Vector or Hashtable to implement the frequency table.
2. You might want to pass the text sample as a parameter to the `FreqTab` constructor, and make the frequency table an instance variable.
3. You should write a method that prints the contents of the table.
4. Rather than allow client code direct access to the contents of the table, you should provide accessor methods. For example, you might write `frequency`, which takes a letter (character) and returns the number of times it appears in the sample text.

## The HuffTree

Create a class called `HuffTree` to represent a Huffman Tree as seen on pages 299-301. Here's a suggestion about what the instance variables might be:

```
public class HuffTree implements Comparable {
  char c;
  int frequency;
  HuffTree left, right;
}
```

Since HuffTrees implement the Comparable interface, they can be inserted into a Heap, which is useful for assembling the tree.

## Huffman

Make a class called `Huffman` that will serve as the central organizing class. Here is what its public interface might look like:

```
public class Huffman {
    private FreqTab ft;
    private HuffTree ht;
    private CodeTab ct;

    public Huffman (String sample);
    public String encode (String s);
    public String decode (String s);
}
```

Here are some suggestions for how to proceed:

1. Write `buildHuffTree` first and use the print method to check it. Keep in mind that you might not get exactly the same tree as in the book, but it should be similar in the sense that each letter should have the same code length as in the book, even if the code is not identical.

2. You can use your Heap implementation or mine. Either way, you should not have to make any changes to Heap.java—that's the nice thing about ADTs!

3. Write the method `decode` next, since you don't need the `CodeTab` to test it.

4. Write `buildCodeTable` next. You can use an array, Vector, or Hashtable to implement the code table.

   HINT: to build the code table, you have to traverse the HuffTree and keep track of the path as you go, since the path to each leaf is its Huffman code.

5. Write `encode` last. If `buildCodeTable` works, `encode` is trivial.