

## Homework 4: Huffman Decoder

Due: Feb. 8 at 11:55pm

This program should be completed using pair programming. (See Pair Programming Guidelines in Moodle.)

David Huffman was a founding faculty member of the Computer Science Department at UC Santa Cruz. In 1951, when he was a student at MIT, David invented a method of encoding characters using a binary code (a code composed entirely of ones and zeros.) Huffman Code depends on the frequencies of the characters in the alphabet. For efficiency, the most frequently-used characters are given shorter codes. Huffman Code is very widely used and has been shown to be the most efficient binary encoding for a string of characters in which each character has a unique code.

A message encoded in Huffman Code can be decoded by a finite state machine (FSM). An FSM has a fixed number of states, and transitions from one state to another based on reading symbols from an input stream. On each state transition, the FSM may generate an output symbol. When decoding a message in Huffman Code, the input stream consists of the ones and zeros of the encoded message, and the output symbols are the characters of the decoded message.

The FSM to decode the Huffman Code for a given alphabet can be described by a state transition table that lists all the possible states for the FSM. For each state, the state transition table specifies the next state and output symbol if the input symbol is 0, and the next state and output symbol if the input symbol is 1. Some state transitions generate no output and have no output symbol.

Your assignment is to write a program named `Huffman.java` that implements a finite state machine to decode Huffman-coded messages. Your program must do the following:

1. Read a state transition table from an input file named `states.txt`. Save this information in a set of arrays that will control the actions of your machine as it reads the input message.
2. Read an encoded message from an input file named `message.txt`. As you read the ones and zeros from the message, implement the state transitions of the FSM. Use `print()` and `println()` to print the output symbols of the decoded message.

The `states.txt` file has the following format:

1. The first line contains an integer that specifies the number of states. If there are  $N$  states, the states are numbered from 0 to  $N-1$ . The FSM always starts in state 0.
2. The next  $N$  lines each describe one state, in order from state 0 to state  $N-1$ . Each line consists of the following parts, separated by whitespace:
  - a. The number of the state.
  - b. The number of the next state if the input symbol is 0.

- c. A string specifying the output symbol if the input symbol is 0, represented as follows (with no quotes):
  - If the first five characters of the string are "char:", the character following the colon is the output symbol. For example, "char:w" indicates that the output symbol is w.
  - "space" indicates that the output symbol is a blank space.
  - "newline" indicates that the output symbol is a newline character (causes a new line to begin in the output message.)
  - "none" indicates that there is no output symbol (no output for this state transition.)
- d. The number of the next state if the input symbol is 1.
- e. A string specifying the output symbol if the input symbol is 1, using the same representation described in (c) above.

The example below shows two lines from a possible `states.txt` file.

- The first line describes state 6. When the FSM is in state 6, if the input symbol is 0, the machine transitions to state 0 and outputs the character k. If the input symbol is 1, the machine transitions to state 0 and outputs the character v.
- The second line describes state 7. When the FSM is in state 7, if the input symbol is 0, the machine transitions to state 5 and generates no output. If the input symbol is 1, the machine transitions to state 0 and outputs the character b.

```
6 0 char:k 0 char:v
7 5 none 0 char:b
```

You and your partner should complete the following steps:

1. Download the files `states.txt` and `message.txt` from Moodle. Look at the content of these files and make sure you understand it.
2. Write a program that reads the two files, decodes the message, and prints the decoded message. If you get English words in your decoded message, your program is probably working. The graders will test your program by decoding a different message.
3. Submit one file named `Huffman.java` to Moodle before the assignment deadline. You do not need to submit the output of your program. Each partner should submit a file, following Pair Programming Guidelines.

**Tips for this assignment:**

1. To read the content of a file, you may create a Scanner as in the following examples. The Scanner then behaves just as if it were scanning input from the keyboard.

```
Scanner scan1 = new Scanner(new File("states.txt"));
Scanner scan2 = new Scanner(new File("message.txt"));
```

2. A method that scans a file must declare that it might throw an "exception" if the file does not exist. This is done as in the following examples:

```
static void readStateTable()
    throws java.io.FileNotFoundException
{
    // body of method goes here
}
```

A method must contain this declaration even if it calls another method that scans a file.

3. A program that scans a file must import the Java input/output package as well as the utility package:

```
import java.io.*;
import java.util.*;
```