Homework 4
Huffman Coding

Due 11:59 PM Monday March 27
Huffman Coding

• Implement Huffman Encoding and Decoding and the classes shown on the following slides. You will also need to use the Stack class and the LinkedList or ArrayList class.

• HuffmanEncode expects 2 command line arguments: the name of the source file (the file to be compressed) and the name of the destination file (the file to which the compressed file must be written)
Huffman Coding

• HuffmanDecode expects 2 command line arguments: the name of the source file (a file that contains a compressed file created by your encoder) and the name of the destination file (the file to which the uncompressed file must be written)

• If your programs work the contents of decoder’s destination file should be exactly the same as the original source file used by encoder
Huffman Coding

- I will post some large files for you to test with, but you should first test with small files you create.
- You will demonstrate your implementation to me.
import java.io.*;
public abstract class BitInputStream {

    protected DataInputStream d;

    public BitInputStream(String filename) {
        try {
            d = new DataInputStream(new FileInputStream(filename));
        }
        catch (IOException e) {
        }
    }

    public abstract int readBit();

    public abstract void close();
}
import java.io.*;
public abstract class BitOutputStream {

    protected DataOutputStream d;

    public BitOutputStream(String filename) {
        try {
            d = new DataOutputStream(new FileOutputStream(filename));
        }
        catch (IOException e) {
        }
    }

    public abstract void writeBit(int bit);

    public abstract void close();
}
import java.io.*;
public class HuffmanInputStream extends BitInputStream {

    private String tree;
    private int totalChars;

    public HuffmanInputStream(String filename) {
        super(filename);
        try {
            tree = d.readUTF();
            totalChars = d.readInt();
        } catch (IOException e) { }
    }
}

Huffman Coding
public int readBit() {
}

public String getTree() {
}

public int totalChars() {
}

public void close() {
}
import java.io.*;
public class HuffmanOutputStream extends BitOutputStream {
    public HuffmanOutputStream(String filename, String tree, int totalChars) {
        super(filename);
        try {
            d.writeUTF(tree);
            d.writeInt(totalChars);
        } catch (IOException e) {
        }
    }
    public void writeBit(int bit) {
        //PRE bit == 0 || bit == 1
    }
    public void close() {
    }
}
import java.util.*;
public class HuffmanTree {

    private class Node {
        private Node left;
        private char data;
        private Node right;

        private Node(Node L, char d, Node R) {
            left = L;
            data = d;
            right = R;
        }
    }

    private Node root;
    private Node current;
Huffman Coding
Huffman Tree (continued)

public HuffmanTree() {
    root = null;
    current = null;
}

public HuffmanTree(char d) {
}

public HuffmanTree(String t, char nonLeaf) {
    // Assumes t represents a post order representation of the tree
    // where a node is either a leaf or has two children. nonLeaf
    // is the char value of the data in the non-leaf nodes
}

public HuffmanTree(HuffmanTree b1, HuffmanTree b2, char d) {
}
Huffman Coding
Huffman Tree (continued)

//The following methods allow a user object to follow a path in the tree.
//Each method except atLeaf and current changes the value of current
//atLeaf returns true of the current position is a leaf, otherwise it returns false
//current returns the data value in the current Node

public void moveRoot() {
}
public void moveLeft() {
}
public void moveRight() {
}
public boolean atLeaf() {
}
public char current() {
}
Huffman Coding

Huffman Tree (continued)

//Inner class to create an iterator. The iterator allows the user class to find all paths from
//the root to a leaf. The paths are sequences of 0s and 1s. 0 means left and 1 means right
//You will find it easier to find all the paths when the iterator is created.

public class PathIterator implements Iterator<String> {
    public PathIterator() {

    }
    public boolean hasNext() {
    }
    public String next() {
    }
    public void remove() {
        //optional method not implemented
    }
}
public Iterator<String> iterator() {
// return a PathIterator object
}

public String toString() {
}
}
import java.io.*;
import java.util.*;

public class HuffmanEncode {

    public HuffmanEncode(String in, String out) {
        // Implements the huffman encoding algorithm
        // Add private methods as needed
    }

    public static void main(String args[]) {
        new HuffmanEncode(args[0], args[1]);
    }
}
public class HuffmanDecode {

    public HuffmanDecode(String in, String out) {
        // Implements the huffman decoding algorithm
        // Add private methods as needed
    }

    public static void main(String args[]) {
        new HuffmanDecode(args[0], args[1]);
    }
}
Huffman Coding

• In addition to the classes on the previous slides you should modify the BinaryHeap class I discussed in class so it can be used as a priority queue to build the Huffman tree.
Homework 4 Submission

• You will demonstrate your program to me either on your own machine or on a machine in the CS lab.

• If you demo between Monday Mar. 27 and Thursday Mar. 30 the program is on time. I will not be able to do demos for everyone on the last day so you should not wait until the last day to demo.

• You can demo from Friday Mar. 31 until Thursday Apr. 6 for up to 50% of the points.