

# Assignment 8

Ethan Fidler 3/5/2023

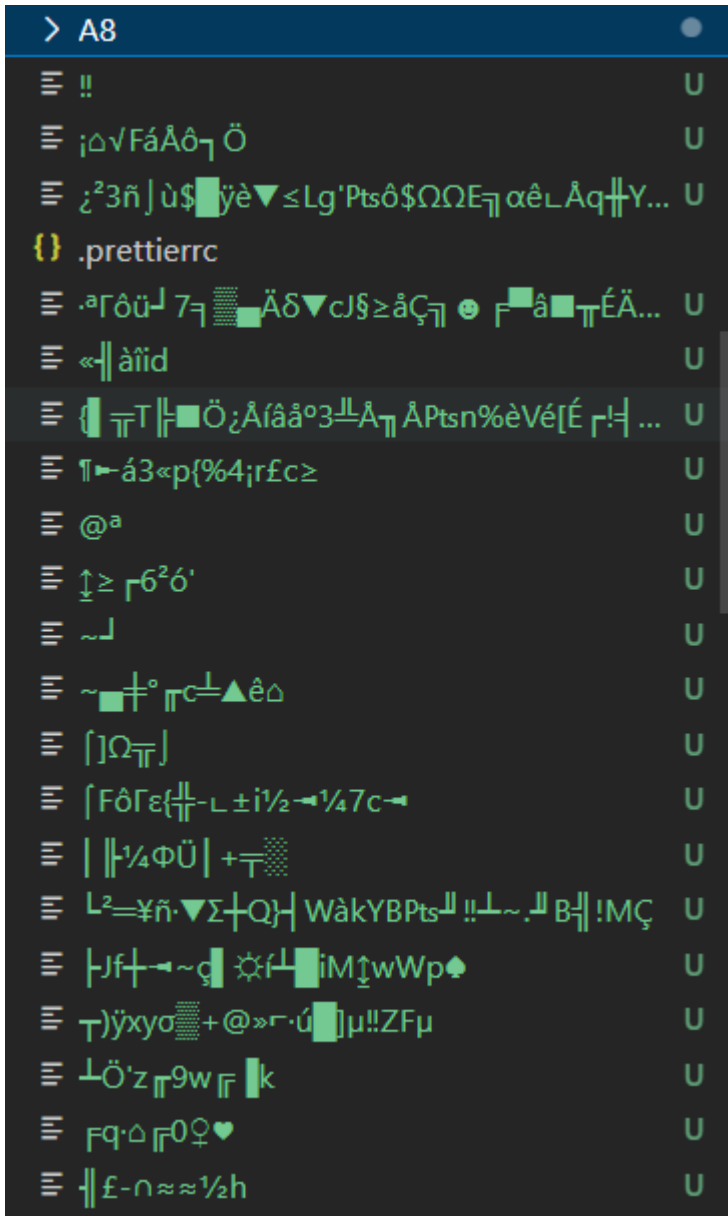
## Output

The test file was loaded properly, but I don't think that the terminal was able to output the entire encoded string; included is an image of the tail of the encoded string.

```
@er@)n=0?nz=u*à|†UST|QE±~+†I I Y mß0E↓≥nl e0e0 0akAic&éπ††|β0d†±ΣQ000NG~|ñ2C«0y9 GT(≈|V00JHo·^hmmA=ñ·0i²%o 0n4|J0c"(-J 1n²|δ_Ti%-
r-i¿3q-|:φfx±Q0= rñé±±††f±^9i¿A± Aw00=0|n=±0°co|↓±N>†jé|Úf|D|±0±±δ v nã8L~Hh0t6^6!!ç|v2-µ-0q FÜ#iπBZ@a ≈v°t▲†Σ3ç†cç|D†%90~*†ák{ rüq]pñ'è±|†ÜY4
[Ta2yÿg=I1φ0trjΣ≤_r,0εεt9^9?N_rZb| 7ca±Nπ0%r±jY2|†0p|1>fèç»ñG|†πÜVS| i±0± hml 6i¿!ÜH-Qnq i^"y±N 0G·| D6â±-| "Aw°±-èA†de iY0i ñ≈π-µ1J2±Y«=NNT L-0I
]†ov iTx+±|δ000==Y_±, ±i|†0R|el_05q c†±N=Nx±x0)| y^4±E± N_ wtkZL±[0±²|âc|†0±f|EâST^n²|δ=J#M±_|Ü|±010ac.■|, %A†0±!âAß±9@f4Z0»Hrb|†c j i j 0'†0±_o 0Bè
70R±†E_fJ1††fñ0±; (=09w|†c0±†v±z0±|†EZ=†Tr††_A tCF0±0~â0±è±Y0²;†|)0ècma°H±†f0v i° h_rv~è9!†rúφq|†|¼çeu±L4i"†c_ Aw^n±0$ \±E±†A† f=n0w3d0sf±_r0%±
N_±n0w=nU|_††0æS~±≥S_r†|_±v_ E~X0±AqiiHdçX!üv_†_†_0%5S~>C±bN1|†±†_†_>~c i f E(hxcE| 0E_†_†|†±±0±²_†_>^n^N_r_▲±0±0±U0_r0_Ü±)â_r_(g#G±'fδ/±=Σ|±0içn9†
EY0±0~1±†φ/0y=1_r0±: 6V0±g±µy0±; rS
Cw#| y| ±yê±;|ê0±:YiA0°0±-ç_f+N_r†0±†±†±±v&â?±-E iâ±†Lz^n²|ERKA·|/|)R_±J0±Ew^n
†|_0±0µ2 ±_▲±ay%±ü†R0T{±| SY7±0±J±Y:ε0y9 F±±v±R_±±±i~±0H0E ±v&CbA±†Z†_±dosy|é_r-0±_▲±0±0|·9|A}7fg±(±5f>ÿ~|?D±fLm_9bf|_9±bg*; 0"0g°<_»}±±≤±/S
±±±U=ê±G|wâ ±y i i±±0±K0â±sgs iâImAr}±iüA0±SgmNhy†C†T0yall=±NÑ!A0pS±T±V7(±iü| p†0±o/Kg«Z|
u±sâ±N±QjÜ±±0N·n±ü±±Mb±A±S±-|0±†_±3ü|0±µ|B±0±; S±0±Zy0â±_ 8%tY0,0±v±Y_±†c{0±d0±_±|>φ·N0±|0±"m
x~ |â|v| ±%±pBV|B±N±Qj0±09|8wB·m·9±0mü>±±_±w†±±_08S0±#±0±p±ñs0±±±D±fK±nq|y_r_±i^kE±0±|;β±±_±0±|J±t±Yc ±†±eh°j±~v|U±_±|±?±_0 i±f.58r0±V(†y±±>±)ö=
H-†i30±_1±)i9_±e~5~A^n†0±A1q†††††L†k#±!±±†,q±v_±_±filesize = 2250910; bits generated = 17602372; bitspersymbol = 7.820113642926638
```

Inputting both the entire encoded message & the tail of the encoded message did not result in any output in the terminal, but it did create a large amount of junk files. Testing on a test message I defined myself caused the program to crash.

```
C:\Users\Ethan Fidler\Desktop\Data Encoding\CS5125> cmd /C ""C:\Users\Ethan Fidler\AppData\Local\Programs\Eclipse Adoptium\jdk-17.0.5.8-hotspot\bin\java.exe" -XX:+ShowCodeDetailsInExceptionMessages -cp "C:\Users\Ethan Fidler\AppData\Roaming\Code\User\workspaceStorage\fe8f402d851bca048f3125949912f681\redhat.java\jdt_ws\CS5125_68b77586\bin" DE12B "
Cw#| y| ±yê±;|ê0±:YiA0°0±-ç_f+N_r†0±†±†±±v&â?±-E iâ±†Lz^n²|ERKA·|/|)R_±J0±Ew^n
†|_0±0µ2 ±_▲±ay%±ü†R0T{±| SY7±0±J±Y:ε0y9 F±±v±R_±±±i~±0H0E ±v&CbA±†Z†_±dosy|é_r-0±_▲±0±0|·9|A}7fg±(±5f>ÿ~|?D±fLm_9bf|_9±bg*; 0"0g°<_»}±±≤±/S
±±±U=ê±G|wâ ±y i i±±0±K0â±sgs iâImAr}±iüA0±SgmNhy†C†T0yall=±NÑ!A0pS±T±V7(±iü| p†0±o/Kg«Z|
u±sâ±N±QjÜ±±0N·n±ü±±Mb±A±S±-|0±†_±3ü|0±µ|B±0±; S±0±Zy0â±_ 8%tY0,0±v±Y_±†c{0±d0±_±|>φ·N0±|0±"m
x~ |â|v| ±%±pBV|B±N±Qj0±09|8wB·m·9±0mü>±±_±w†±±_08S0±#±0±p±ñs0±±±D±fK±nq|y_r_±i^kE±0±|;β±±_±0±|J±t±Yc ±†±eh°j±~v|U±_±|±?±_0 i±f.58r0±V(†y±±>±)ö=
H-†i30±_1±)i9_±e~5~A^n†0±A1q†††††L†k#±!±±†,q±v_±_±
```



## Code

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

class Node implements Comparable<Object>{
    Node left, right;
    int symbol;
    int frequency;
    public Node(Node l, Node r, int s, int f){
        left = l; right = r; symbol = s; frequency = f;
    }
    public int compareTo(Object obj){
        Node n = (Node)obj;
        return frequency - n.frequency;
    }
}
```

```

public class DE12A{

    static final int numberOfSymbols = 256;
    static final int blockSize = 4096;
    int[] freq = new int[numberOfSymbols];
    Node tree = null;
    String[] codewords = new String[numberOfSymbols];
    int[][] codetree = null; // Huffman tree with actualNumberOfSymbols leaves
    int buf = 0; int position = 0; // used by outputbits()
    int actualNumberOfSymbols = 0; // number of symbols with freq > 0
    int filesize = 0;
    int sizeOfCompressed = 0;

    void count(String filename){ // count symbol frequencies
        byte[] buffer = new byte[blockSize];
        FileInputStream fis = null;
        try {
            fis = new FileInputStream(filename);
        } catch (FileNotFoundException e){
            System.err.println(filename + " not found");
            System.exit(1);
        }
        int len = 0;
        for (int i = 0; i < numberOfSymbols; i++) freq[i] = 0;
        try {
            while ((len = fis.read(buffer)) >= 0){
                for (int i = 0; i < len; i++)
                    freq[Byte.toUnsignedInt(buffer[i])]++;
                filesize += len;
            }
            fis.close();
        } catch (IOException e){
            System.err.println("IOException");
            System.exit(1);
        }
    }

    void entropy(){
        double sum = 0;
        for (int i = 0; i < numberOfSymbols; i++) if (freq[i] > 0){
            actualNumberOfSymbols++;
            sum += freq[i] * Math.log(((double)freq[i]) / filesize);
        }
        sum /= filesize * Math.log(2.0);
        System.err.println("actual number of symbols = " + actualNumberOfSymbols + ";
entropy = " + -sum);
    }

    void makeTree(){ // make Huffman prefix codeword tree
        PriorityQueue<Node> pq = new PriorityQueue<Node>();
        for (int i = 0; i < numberOfSymbols; i++) if (freq[i] > 0){
            actualNumberOfSymbols++;
            pq.add(new Node(null, null, i, freq[i]));
        }
    }
}

```

```

int nodeLabel = numberOfSymbols;
while (pq.size() > 1){
    Node a = pq.poll(); Node b = pq.poll(); // remove two subtrees
    pq.add(new Node(a, b, nodeLabel++, a.frequency + b.frequency));
    // add the merged subtree
}
tree = pq.poll(); // root of tree as the last single subtree
}

void dfs(Node n, String code){ // generate all codewords
    if (n.symbol >= numberOfSymbols){ // internal node
        dfs(n.left, code + "0"); dfs(n.right, code + "1");
    }else codewords[n.symbol] = code; // leaf
}

void makeCodewords(){
    dfs(tree, "");
}

void buildTreeArray(){ // make an array for the tree
    codetree = new int[actualNumberOfSymbols * 2 - 1][2];
    int treeSize = 1;
    for (int i = 0; i < actualNumberOfSymbols * 2 - 1; i++){
        codetree[i][0] = codetree[i][1] = 0;
    }
    for (int i = 0; i < numberOfSymbols; i++){
        if (codewords[i] != null){
            int len = codewords[i].length();
            int k = 0;
            for (int j = 0; j < len; j++){
                int side = codewords[i].charAt(j) - '0';
                if (codetree[k][side] <= 0) codetree[k][side] = treeSize++;
                k = codetree[k][side];
            }
            codetree[k][1] = i;
        }
    }
}

void outputTree(){
    System.out.write(actualNumberOfSymbols); // number of used symbols
    for (int i = 0; i < actualNumberOfSymbols * 2 - 1; i++){ // Huffman tree
        System.out.write(codetree[i][0]);
        System.out.write(codetree[i][1]);
    }
    int fs = filesize;
    for (int i = 0; i < 3; i++){ // three bytes for filesize
        System.out.write(fs & 0xff);
        fs >>= 8;
    }
}

void encode(String filename){ // compress filename to System.out
    byte[] buffer = new byte[blockSize];
    FileInputStream fis = null;
    try {

```

```

        fis = new FileInputStream(filename);
    } catch (FileNotFoundException e){
        System.err.println(filename + " not found");
        System.exit(1);
    }
    int len = 0;
    try {
        while ((len = fis.read(buffer)) >= 0)
            for (int i = 0; i < len; i++)
                outputbits(codewords[Byte.toUnsignedInt(buffer[i])]);
        fis.close();
    } catch (IOException e){
        System.err.println("IOException");
        System.exit(1);
    }
    if (position > 0){ System.out.write(buf << (8 - position)); }
    System.out.flush();
    System.err.println("filesize = " + filesize + "; bits generated = " +
sizeofCompressed + "; bitspersymbol = " + ((sizeofCompressed * 1.0)/filesize));
}

void outputbits(String bitstring){ // output codeword
    sizeofCompressed += bitstring.length();
    for (int i = 0; i < bitstring.length(); i++){
        buf <<= 1;
        if (bitstring.charAt(i) == '1') buf |= 1;
        position++;
        if (position == 8){
            position = 0;
            System.out.write(buf);
            buf = 0;
        }
    }
}

public static void main(String[] args){
    if (args.length < 1){
        System.err.println("Usage: java DE12A original > encoded");
        return;
    }
    DE12A de12 = new DE12A();
    de12.count(args[0]);
    de12.entropy();
    de12.makeTree();
    de12.makeCodewords();
    de12.buildTreeArray();
    de12.outputTree();
    de12.encode(args[0]);
}
}

```

```

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

public class DE12B{

    int[][] codetree = null;
    int buf = 0; int position = 0;
    int actualNumberOfSymbols = 0;
    int filesize = 0;

    void readTree(){ // read Huffman tree
        try{
            actualNumberOfSymbols = System.in.read();
            codetree = new int[actualNumberOfSymbols * 2 - 1][2];
            for (int i = 0; i < actualNumberOfSymbols * 2 - 1; i++){
                codetree[i][0] = System.in.read();
                codetree[i][1] = System.in.read();
            }
            for (int i = 0; i < 3; i++){ // read filesize
                int a = System.in.read();
                filesize |= a << (i * 8);
            }
        } catch (IOException e){
            System.err.println(e);
            System.exit(1);
        }
    }

    int inputBit(){ // get one bit from System.in
        if (position == 0)
            try{
                buf = System.in.read();
                if (buf < 0) return -1;
                position = 0x80;
            } catch (IOException e){
                System.err.println(e);
                return -1;
            }
        int t = ((buf & position) == 0) ? 0 : 1;
        position >>= 1;
        return t;
    }

    void decode(){ // Your two lines of code for updating k are needed for this to
        work.
        int bit = -1; // next bit from compressed file: 0 or 1. no more bit: -1
        int k = 0; // index to the Huffman tree array; k = 0 is the root of tree
        int n = 0; // number of symbols decoded, stop the while loop when n == filesize

        while ((bit = inputBit()) >= 0){
            // Your code: replace k by the index of a child according to bit (Walk down
            tree)

```

```
k ^= bit;
if (codetree[k][0] == 0){ // leaf
    System.out.write(codetree[k][1]);
    if (n++ == filesize) break; // ignore any additional bits
    // Your code: restart for the next symbol by move to the root (Go up to
root)
    k = 0;
}
}
System.out.flush();
}

public static void main(String[] args){
    DE12B de12 = new DE12B();
    de12.readTree();
    de12.decode();
}
}
```