Overview

A program that implements a compression and decompression Hoffman algorithm on ASCII files.

Introduction

This document holds the technical design of the CSCI312\_Project\_Schmidt program and serves as a pseudo lessons learned for the changes made to the initial design. The program is intended to compress and decompress an ASCII file.

## Scope

Compress and decompress an ASCII file.

### PROCESSING

The user enters encode or decode, and an ASCII file address

Compressing:

The program creates a List of nodes and begins adding or incrementing nodes to the list based on every character in the ASCII file to be compressed. Then it sorts the list based off frequency, adds the two smallest to a new parent node that is the sum of the smallest two frequencies. It repeats this until there is only one node left in the list.

The program maintains the list of nodes though, so that it can immediately create the data needed for the decompression header and to recreate the node tree.

Then the program goes through all the characters in the ASCII file again, finds the path to their node from the root, and adds it to a list of bools. All of those bools are then written into a string in their decimal representation (1s and 0s).

Tree Header:

The tree header is the decimal representation of the Boolean path to each of the characters. This is used to recreate the tree structure when decoding.

Decompressing: The program recreates the tree structure based off the tree header which is the path to the node and the node char. If a node does not exist on the path, then the program creates an empty node. Then for every 1 and 0 in the compressed file, it follows the tree structure until it finds a node that has no children and prints that node’s char. It then resets back to the root node and continues its journey.

### DATA

The logical and physical data structure of files should be defined in detail.

Data structure definitions must include the:

description of each element, e.g. name, type, dimension;

relationships between the elements, i.e. the structure;

range of possible values of each element;

initial values of each element.

MyHuffmanTree

Holds the processes for compressing and decompressing the files.

Node

Holds the node information for frequency and character and its two children nodes.

### COMPONENTS

Assignment6Execution

* Holds the static main and initiates the simulator.

ProcessHeap

* Holds the simulator process

|  |
| --- |
| **MyHuffmanTree** |
| -List<Node> Nodes  -Node m\_Root  -List<bool> treeCopy  -List<char> treeChars  -bool copyTreeRan  -string treeInfos |
| +BuildTree(string): void  +Encode(string): string[2]  +Decode(string, string): string  +IsLeaf(Node): bool  +getRoot(): Node  +setRoot(Node): void  +cloneTree(string): void |

### TESTING

Present one or more named scenarios that will be utilized to test the application.

The testing plan should be repeatable.

Describe the scenario in detail, the steps required to execute the test, the input data, the output data, and the success criteria.

Present a summary of the testing scenarios before the details of each scenario.

|  |  |  |
| --- | --- | --- |
| Scenario | Description | Pass/Fail |
| 1st foo bar run | Run compression/decompression of string | Fail |
| 2nd foo bar run | Run compression/decompression of string input | Pass |
| 3rd Other string run | Run compression/decompression of string input | fail |
| 4th File test | Run compression/decompression of file | fail |

Jobs.txt input:

1 4 8

2 9 5

3 10 3

4 1 9

Scenario #1- jobs.txt test

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1. | Enter string | Input: string “foo bar” |
| 2. |  | Output: encoded file |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| EXPECTED OUTPUT | | Some cool encoded message |
| ACTUAL OUTPUT | | Utter gibberish. It did not work, everything was broken. |
| RESULTS – The desired output of a .txt file with the findings was produced. | | Fail |

Scenario #2- wap.txt test

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1. | Enter string | Input: string “foo bar” |
| 2. |  | Output: encoded file |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| EXPECTED OUTPUT | | To put out node paths to be decompressed |
| ACTUAL OUTPUT | | It worked perfecty! |
| RESULTS – The desired output of a .txt file with the findings was produced. | | Pass |

Scenario #3 – job.txt test

##### 

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1. | Enter string | Input: string “I hope this works” |
| 2. |  | Output: encoded file |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| EXPECTED OUTPUT | | To put out node paths to be decompressed |
| ACTUAL OUTPUT | | It did not work, I tried turning the paths into their own bytecodes and then decompressing them and it turned out to only work on “Foo bar” for whatever reason. I had to take out the scrambling. |
| RESULTS – The desired output of a .txt file with the findings was produced. | | Fail |

##### *Scenario #4 – job.txt test*

|  |  |  |
| --- | --- | --- |
| Step | Description | Input/Output |
| 1. | Enter file | Input: file: “Please work this time” |
| 2. |  | Output: encoded file |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| EXPECTED OUTPUT | | To put out node paths to be decompressed |
| ACTUAL OUTPUT | | It worked perfectly, but it doesn’t actually compress the file. Its actually larger than the original file. |
| RESULTS – The desired output of a .txt file with the findings was produced. | | Fail |