qwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmrtyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnmqwertyuiopasdfghjklzxcvbnm

|  |
| --- |
| genie_logo_lam.png File Genie  Data Structures and algorithms project  Syndicate  Raja Hamza Iqbal  Raja Aneeq Ashraf  Muhammad Umar Farooq Qureshi  Submitted To  Ms Shahela Saif  Lec Bilal Rauf |

Abstract

The File Genie is a file compression and decompression program based on Huffman’s Algorithm. The application implements some of the standard data structures taught in the course. It has a Java based Graphical User Interface and uses C++ at the back end to implement the core functionality.

Table of Contents

[1 Introduction 4](#_Toc313463130)

[2 Requirements 4](#_Toc313463131)

[3 Design 4](#_Toc313463132)

[3.1 Browse 5](#_Toc313463133)

[3.2 Password 5](#_Toc313463134)

[3.3 OK 5](#_Toc313463135)

[3.4 Help 5](#_Toc313463136)

[3.5 Confirmation Dialog 6](#_Toc313463137)

[3.6 Error Messages 6](#_Toc313463138)

[4 Limitations 7](#_Toc313463139)

[5 Internal Concepts and Implementation 7](#_Toc313463140)

[5.1 Architecture Diagram 8](#_Toc313463141)

[5.2 Graphical User Interface 8](#_Toc313463142)

[5.3 Compressor 8](#_Toc313463143)

[5.4 Decompressor 9](#_Toc313463144)

[5.5 Background of Huffman’s Algorithm 9](#_Toc313463145)

[5.5.1 Huffman Coding as a Greedy Algorithm 9](#_Toc313463146)

[5.5.2 Huffman’s Algorithm Steps 10](#_Toc313463148)

[5.5.3 Pseudo code 10](#_Toc313463149)

[5.6 Important Member functions 10](#_Toc313463150)

[5.6.1 Huffman (char name []) 10](#_Toc313463151)

[5.6.2 void writeOutHuff(char []) 11](#_Toc313463152)

[5.6.3 void HuffHuff () 11](#_Toc313463153)

[5.6.4 void Output1Bit (int bit,ofstream& out) 11](#_Toc313463154)

[5.6.5 void Compress1Element (string code,ofstream& out) 11](#_Toc313463155)

[5.6.6 void unHuff (char password [25]="") 11](#_Toc313463156)

# 1 Introduction

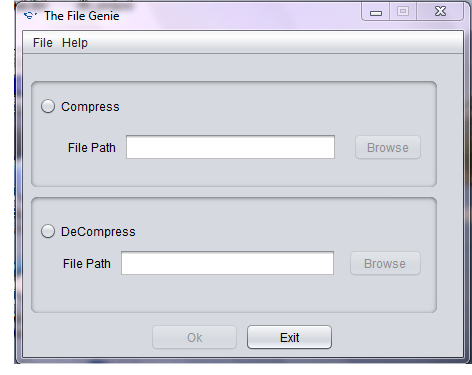
The File Genie is a file compression and decompression utility which implements the following standard data structures to achieve its functionality:

1. Hash Table
2. Trees
3. Stack
4. Priority Queues

# 2 Requirements

1. Add support for file compression.
2. Add support for file decompression.
3. Provide a GUI to select source file.
4. Implement using standard data structures taught during the course.

# 3 Design

The user runs the application by clicking on the .jar file. After the application launches, user is allowed to compress and decompress a specified text file. The following figure shows a screenshot of the application GUI. 

The user is allowed to choose between compress or decompress facility through Compress and Decompress radio buttons. Based on the user’s selection, the file path text box and ‘Browse’ button for that feature are enabled.

## 3.1 Browse

The user is then required to click on the browse button and select the file that is required to be compressed or decompressed. In case of Compression, the input file has a .txt extension. For decompression, the input file has a .huff extension. In order to decompress successfully, the .header and .bitPass files for the specified source file must be present in the same location as the .huff file.

## 3.2 Password

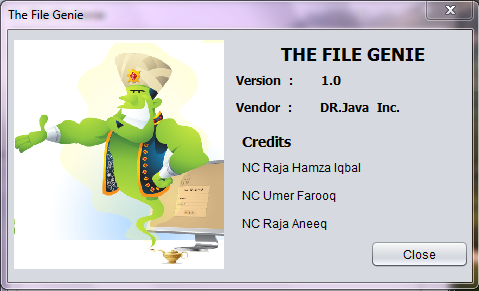
This program also provides the facility of Password but is only available when the executable is directly run with command line and the password is passed as second command line argument.

## 3.3 OK

If the user selects compress option, a compressed file is generated in the same location as the source file. On the other hand, if the user has opted for decompression, the decompressed file is generated in same directory as the .huff file.

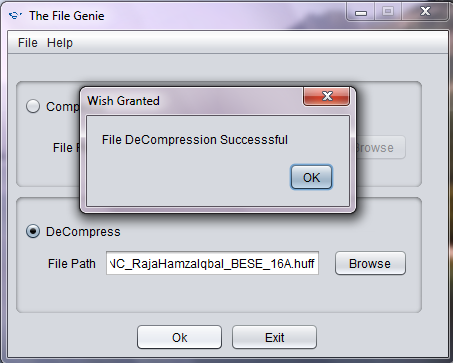
## 3.4 Help

In the help menu the information about the authors of file genie is provided.



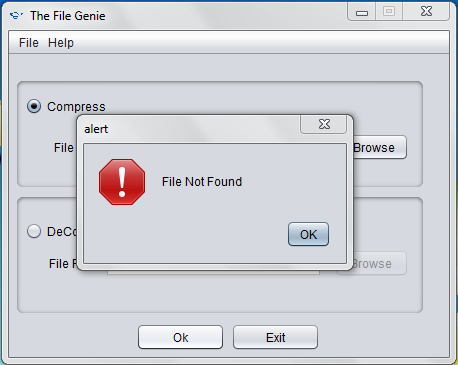
3.5 Confirmation Dialog

In case of successful compression or decompression of file, an operation successful message box is displayed to inform the user of the result of the operation.



## 3.6 Error Messages

In case of an exception, an appropriate error notification is displayed.



# 4 Limitations

1. The program works only for text files with ASCII encoding. If the encoding for the text file is different, it will still compress the file but when decompressing, the format will be changed to ASCII which can cause data loss.
2. Since the GUI for the application is implemented using Java 7, it requires Java Runtime Environment 7 to execute.

# 5 Internal Concepts and Implementation

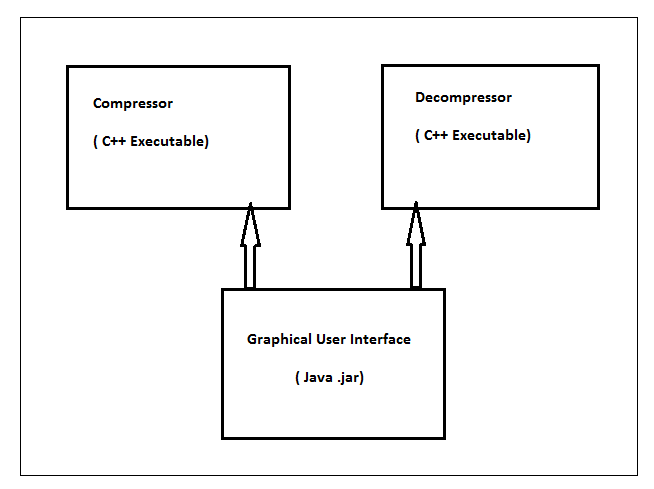
The application integrates a JAVA front end with a back-end implemented using C++. This concept of mixing multiple programming languages in one application is called **mixed-mode programming.**

The text file is compressed by generating a new unique bit code for each character used in the file. This code assigns the shortest code to the most frequently occurring symbol. This is approach is called the **Huffman’s Algorithm**.

The resultant compressed file is generated with .huff extension along with associated .header and .bitPass files. All these resultant files are then stored in a new folder created in the same file path as the original source file. The program has the following three modules:

1. GUI
2. Compressor
3. Decompressor

## 5.1 Architecture Diagram

The following architectural diagram explains the relationship between the modules:

## 5.2 Graphical User Interface

The GUI for the application is built using Java Runtime Environment. Based on the user selection of compress or decompress option, the .jar executable then invokes either the compressor or decompressor exe and passes the file path as argument.

## 5.3 Compressor

The compressor module functions as follows:

1. Takes absolute path of the file to be compressed as a command line argument (this is provided by the GUI module).
2. Reads the text based file and generates the frequency table for each character.
3. A forest with each tree representing a character is built and enqueued on a priority queue.
4. Builds the Huffman tree with help of priority queue.
5. The tree is traversed and encoded string for each valid character is obtained and stored in hash table named ‘lookup’.
6. The compressed encoding for each character is written on file in units of bytes by using an unsigned char variable as a buffer.

## 5.4 Decompressor

The functionality of the decompressor is as below:

1. Takes the absolute path of the file containing the compressed bits with the .huff extension as command line argument ( this is provided by the GUI module).
2. Reads the .huff file written by ‘Compressor’ executable along with .bitPass and .header.
3. Reads the frequency hash table from .header file and generates the huffman tree.
4. Also reads the number of Bits written on .Huff file from .bitPass.
5. A byte is read from file which consists of bits i.e 1’s and 0’s into an unsigned char variable that acts as a buffer.
6. On the basis of those bits the tree is traversed with the convention that ‘0’ means left and ‘1’ means right.
7. When we get a leaf the character from leaf is written on file and the pointer used for traversing the tree is again set to the root.
8. The process continues until all bits specified by .bitPass are processed.

In addition to using the data structures as required by the course, the application also utilizes the following additional programming techniques to achieve its functional requirements:

* Java Beans
* Exception Handling
* Filing
* Mixed-mode Programming

## 5.5 Background of Huffman’s Algorithm

The Huffman’s algorithm works on the principle that since all possible characters are not used in every file and that some characters like vowels are used far more frequently. These facts can be utilized to reduce the file size if we generate unique bit sequences representing each used character for a particular file and assign shorter bit sequences to more frequently used characters instead of just storing each character with an eight bit sequence.This can drastically reduce the amount of space used to store a particular file.

### 5.5.1 Huffman Coding as a Greedy Algorithm

### Huffman's algorithm is an example of a greedy algorithm. It's called greedy because the two smallest nodes are chosen at each step, and this local decision results in a globally optimal encoding tree. In general, greedy algorithms use small-grained, or local minimal/maximal choices to result in a global minimum/maximum.

### 5.5.2 Huffman’s Algorithm Steps

The Algorithm is as follows:

1. Begin with a forest of trees. All trees are one node, with the weight of the tree equal to the weight of the character in the node which is equal to the frequency of that character. Thus characters that occur most frequently have the highest weights and Characters that occur least frequently have the smallest weights.
2. Repeat this step until there is only one tree:

Choose two trees with the smallest weights. Create a new tree whose root has a weight equal to the sum of the weights T1 + T2 and whose left sub tree is T1 and whose right sub tree is T2.

1. The single tree left after the previous step is an optimal encoding tree.

### 5.5.3 Pseudo code

* Create Priority Queue F from singleton trees with each tree containing the character it represents and its frequency as its weight.
* Those with the smallest weights have the highest priority.
* WHILE (F has more than one element)
* DO
* Dequeue T1 and T2 with Min weight
* Construct new tree T by creating a new node and setting T1 and T2 as its children
* Let the sum of the values associated with the roots of T1 and T2 be associated with the root of T
* Add T to F

## 5.6 Important Member functions

### 5.6.1 Huffman (char name [])

/\*\*

\*Requires a character array acting as a C-string representing the absolute path of the file to be

compressed.

\*Initializes all class variables

\*Constructs a hash table representing the frequencies of each character

\*creates a new directory of the same name as the file in the same dir as the to be compressed

file.

\*creates a file with the same name as the file to be compressed in the above mentioned

directory with .header extension

\*writes the above mentioned hash table to file with extension .header

\*creates a new Tree Node for each Active Symbol and enqueues it into the priority queue

\*/

### 5.6.2 void writeOutHuff(char [])

 /\*\*

\*creates the .huff binary file

\*opens the file that is to be compressed

\*reads the input file into a buffer and then closes it

\*runs a loop for the number of bytes in buffered

\*calls the 'Compress1Element' function to compress each byte

\*calls 'Output1Bit' with -1 as argument to write the last byte with padding

\*/

### 5.6.3 void HuffHuff ()

/\*\*

\*Requires that all Tree Nodes representing Active Characters have been created

\*And Enqueued into the Priority Queue

\*Generates a huffman tree by combining the nodes in the PQ

\*the root of the tree is at the head of the priority queue

\*/

### 5.6.4 void Output1Bit (int bit,ofstream& out)

/\*\*

\*a function that writes the bits to the output file

\*takes an integer represent the bit value to be written as argument

\*An of stream objects that has an opened file associated with it is also required by reference

\*a bit value of -1 indicated that the buffer be filled with required padding bits and then written

\*/

### 5.6.5 void Compress1Element (string code,ofstream& out)

/\*\*

\*Requires a string containing a valid character code representing a

\*character

\*Requires a ofstream object with an opened file associated with it

\*Writes the code consisting of bits to the file through the ofstream object

\*/

### 5.6.6 void unHuff (char password [25]="")

/\*\*

\*requires a generated huffman tree with its root in the Priority queue

\*Opens the .huff file in binary format

\*Now it reads a byte into a 1 byte buffer and the traverses the tree

\*every time it reaches a leaf it writes that character to the output file

\*if the counter representing the no of bits processed reaches zero

\*the next byte is read into the buffer and the counter is reset

\*if EoF is reached or required no of bits has been processed the loop terminates

\*/