Data Structures and Algorithms

CS-250

Project

Due Date: Wednesday 20-12-2023 @ 05:00 pm

Student Details:			
Name:	Muhammad Haroon & Muhammad Ibtesam		
NUST ID:	420639 & 416646		
Semester:	Fall 2023	Batch:	2022

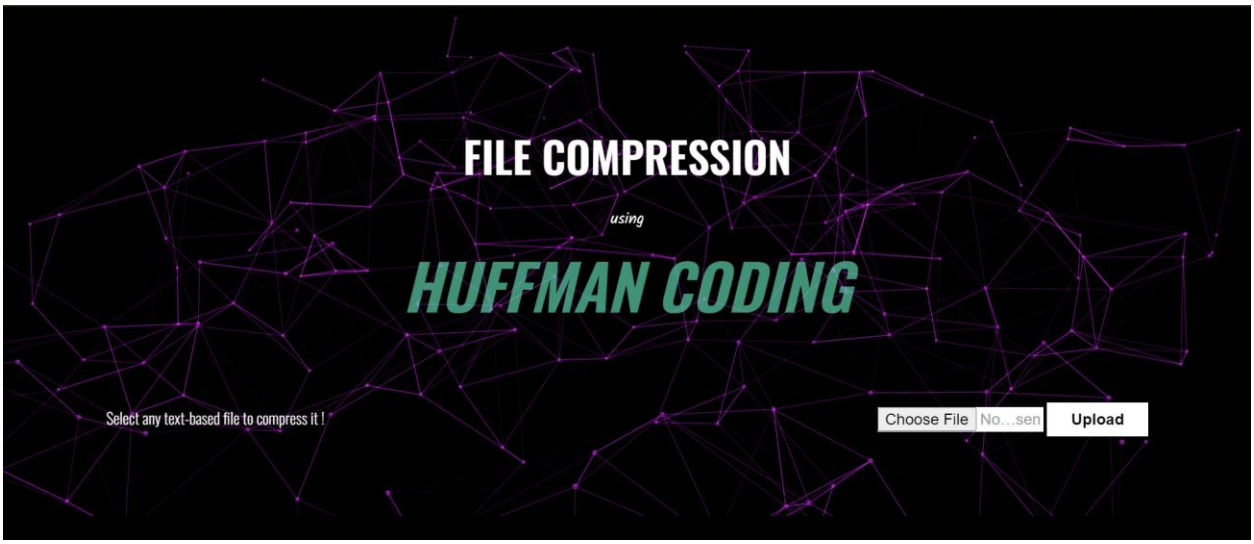
Obtained Marks:	
Total Marks:	30

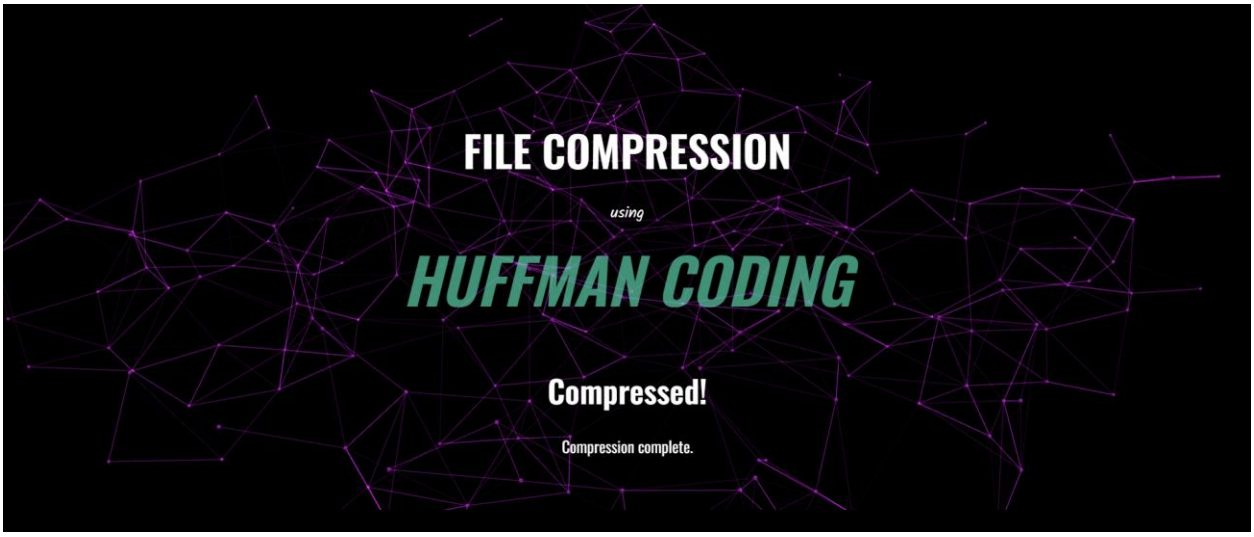
Important Instruction

Fill your information on the cover page.
Make your work clear and well-presented.
LATE submission will result in marks-deduction.
Copying from other students or resources will result in ZERO mark.
Number all the subsequent pages properly.

Introduction:

This is a File Compression Tool developed in C++ using the Huffman Algorithm, which is used to reduce the size of text-based files.





Implementation:

This project is implemented using concepts of Data Structure and Algorithms such as linked lists, trees, Huffman Algorithm, and Priority Queue (Max Heap) concept of data structure. The user interacts with main functions like **compressfile()** and **cgiMain().**

Code Content:

- class Node

```
Private:
```

```
void traversehelper (const Node *node, string &&code) const{}
Public:
      Node (char d, int f){}
Node (const Node *l, const Node *r){}
bool operator< (const Node &a) const{}
void traverse (string &&code) const{}
```

- class PriorityQueue

```
public:
PriorityQueue (int size){}
~PriorityQueue (){}
void push (const Node &node){}
void pop (){}
Node top (){}
int size (){}
```

- class HuffmanCoding

```
private:
      void generatecodes (const Node *root, string &&code){}
public:
void compressfile(const string &inputfile, const string &outputfile){}
```

- int cgiMain(){}

- int Main(){}

Error Handling:

- The system checks for a valid text-based input file.
- The file should already exist.
- The file should not have the same name.

Time Complexity Analysis:

The time complexity of the application's key functionalities is as follows:

Building Priority Queue (PriorityQueue class):

- The push and pop operations in the priority queue take O(log n) time, where n is the number of elements in the priority queue.
- The loop in the push operation may run up to log(n) times.
- The push and pop operations are performed 2n-1 times (while building the Huffman tree).

- Therefore, the time complexity for building the priority queue is **O(n log n).**

Generating Huffman Codes :

- The generatecodes function traverses the Huffman tree, and each node is visited exactly once.
- The time complexity of this operation is **O(n)**, where n is the number of nodes in the Huffman tree.

Compressing File :

- The time complexity of reading the input file and calculating character frequencies is O(i), where i is the size of the input file.
- Building the Huffman tree takes O(n log n), where n is the number of unique characters.
- Generating Huffman codes using the tree takes O(n).
- The loop that processes the input file and writes the compressed output file takes O(m) since each character is processed once.
- Overall, the time complexity for compressing the file is dominated by the Huffman tree construction, i.e., **O(n log n)**.

Overall Complexity:

The overall complexity of this code depends on the number of unique characters in the input file O(n) and the total number of characters in the file O(m). The complexity is influenced by the characteristics of the input data.

Conclusion:

The original file size can be decreased by using File Compression. Depending on the number of unique characters in the input file, the time complexity of producing Huffman codes is O(n). This effective feature of Huffman coding adds to its usefulness in compression scenarios. By using more effective data structures and algorithms, it can be made better.