

MINOR-1 PROJECT

Report

Lossless Compression of High-Density data and Simulating the transfer over Optimal Path.

Submitted By:

Name	Roll No	Branch
Aanchal Sharma	R110218002	CSE-CCVT
Abhirup Kumar	R110218005	CSE-CCVT
Arnav Sharma	R110218032	CSE-CCVT
Dhairya Chugh	R110218047	CSE-CCVT

Under the guidance of:

Ms. Avita Katal
Assistant Professor- Senior Scale
Department of Virtualization
School of Computer Science

UNIVERSITY OF PETROLEUM AND ENERGY STUDIES



**Bidholi Campus, Energy Acres
Dehradun-248007**

CANDIDATE’S DECLARATION

I/We hereby certify that the project work entitled Lossless Compression of High Density and Simulating the transfer over Optimal Path is partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering with Specialization in Cloud Computing and Virtual Technology and submitted to the Department of Virtualization at School of Computer Science, University of Petroleum and Energy Studies, Dehradun, is an authentic record of my/our work carried out during a period from August, 2020 to December, 2020 under the supervision of Avita Katal,

The matter presented in this project has not been submitted by me/us for the award of any other degree of this or any other University.

Aanchal Sharma
R110218002

Abhirup Kumar
R110218005

Arnav Sharma
R110218032

Dhairya Chugh
R110218047

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Date:06/12/2020

Dr. Deepshikha Bhargava
Head of Department
Department of Virtualization
School of Computer Science

Ms. Avita Katal
Assistant Professor-Senior Scale
Department of Virtualization
School of Computer Science

Acknowledgement

We wish to express our deep gratitude of our guide Avita Katal Ma'am, for all the advice, encouragement and constant support she has given us throughout our project work. This work would not have been possible without her support and valuable suggestions.

We sincerely thank our Head of Department Dr. Deepshikha Bhargava , for great support in doing our project at SoCS.

We are also grateful to Avita Katal Assistant Professor- Senior Scale, Department of Virtualization and Dr. Manish Prateek, Dean of SoCS ,UPES for giving us the necessary facilities to carry out our project work successfully.

We would like to thank all our friends for their help and constructive criticism during our project work. Finally we have no words to express our sincere gratitude to our parents who have shown us this world and for every support they have given us.

Aanchal Sharma
R110218002

Abhirup Kumar
R110218005

Arnav Sharma
R110218032

Dhairya Chugh
R110218047

Project Proposal Approval Form (2019-2020)

Minor 1

Project Title:

Lossless Compression of High-Density data and Simulating the transfer over Optimal Path.

ABSTRACT

Broadcasting, streaming, cloud storage, fetching backups, social networking everything works on transferring data items from one place to another.

So, transfer optimization has become the major area of modification to reduce costs and increase speed. Compression and optimal route selection are some of the processes used to do this. In this project the compression technique called Huffman Coding is used. It is used for lossless data compression. It compresses the data items at the sender's end and decompresses it again at the receiver's end. Lossless data compression ensures that the user does not face any quality degradation while interpreting the message. To send this compressed data, an optimal path is selected from the randomly generated topology with a fixed number of vertices to transfer the data item. If the path for transfer is the optimal path minimum traffic, the transfer will be fast and the transmission time will be less. If the transmission time is less, then the probability of losing data during transmission will be less and the transfer process will also be optimized.

Table Of Content

Topic	Page No.
1. Introduction	7
2. Literature Review	8
3. Problem Statement	9
4. Objectives	10
5. System Requirements	11
6. Methodology	12-15
6.1 Huffman Coding	
6.2 Network topology creation with optimal path selection	
7. Implementation	16-29
7.1 Data Flow Diagram	
7.2 Text File compression using Huffman Algo.	
7.3 Image Compression using Huffman Algo.	
7.4 Network topology creation with optimal path selection	
7.5 Implementing Network Programming	
7.6 Integrated Results	
8. Pert Chart	30
9. Code	30
10. References	31

Table Of Figures

Topic	Page No.
6.1 Huffman Algorithm FlowChart	14
7.1 Level 0 DFD	16
7.2 Level 1 DFD	16
7.3 Input Text File	17
7.4 Input Text File Size	18
7.5 Compressed File	18
7.6 Output Text File Size	18
7.7 Decompressed File	19
7.8 Decompressed File Size	19
7.9 Huffman Key	20
7.10 Implementation of Image Compression	21
7.11 Generation of Uncompressed File	21
7.12 Decompressed File	21
7.13 Network Simulation Variables	22
7.14 Random Graph Generation	23
7.15 Assigning IP addresses	24
7.16 Implementation of Dijkstra Algorithm	24
7.17 Network Parameters	25
7.18 Single server and multiple clients	26
7.19 Size of received files	27
7.20 Client Side for Text	28
7.21 Server Side for Text	29
7.22 Client Side for Text	30
7.23 Server Side for Image	30
7.24 Client Side for Image	31
7.25 Client Side for Image	32
8.1 Pert Chart	33

1.INTRODUCTION

Data Compression is a reduction in the number of bits needed to represent data. Compressing data can save storage capacity, speed up file transfer, and decrease costs for storage hardware and network bandwidth. In this project, the advantages of compression and optimal path selection is given importance.

Compression is broadly of two types: lossy data compression and lossless data compression. In this project lossless data compression is used since the integrity and completeness of data during transfer is an important aspect. To implement lossless data compression Huffman coding is used. Huffman coding is a method for the construction of minimum-redundancy codes. The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source file(image,text etc.). The algorithm derives this table from the estimated probability or frequency of occurrence for each possible value in the source file. More common symbols are generally represented using fewer bits than less common symbols.

A random topology is generated with a given number of nodes. These nodes represent sender and receivers with multiple paths between them. It gives a view of how multiple systems are connected in various different topologies and it can be difficult for the sender to send the data to a particular receiver. A random graph generator is used for generating a topology.

Random IP addresses are assigned to all the nodes in the graph to create a stimulating environment for transfer of files. The vertices are considered as computational nodes.

The Dijkstra's algorithm is used to find out the path with minimum traffic between the sender and receiver. It is used to optimize the transfer process, i.e. to reduce the transmission time and reduce the loss of data during transmission.

The integration of Huffman coding with optimal route selection will bring in a significant change in the transfer of data.

2.LITERATURE REVIEW:

- I. The analysis of Huffman coding is done to do compression using MATLAB programming software. In [1], Huffman coding was applied in various scenarios. Huffman was used in compression of five random numbers. In the second scenario, 26 english alphabets were used. In the third scenario, image compression was done using the Huffman algorithm.
- II. It is usually difficult to achieve a balance between speed and memory usage using variable-length binary Huffman code. In [2], the Quaternary tree is used to produce optimal codewords that speed up the way of searching. To analyze the performance of algorithms, Huffman-based techniques were used in terms of decoding speed and compression ratio.
- III. In [3], Dijkstra Algorithm based numerical methods were used to deal with ray tracing in closed space such as tunnels or undergrounds. This method has successively been applied to ray tracing in an open space such as a random rough surface.
- IV. [4] Highlights the idea of using a storage medium to store the solution path from the Dijkstra algorithm, then uses it to find the implicit path at an ideal time cost. Performance of Dijkstra Algorithm was improved using an appropriate Data Structure.
- V. Drivers select a path, normally by just considering a single criterion like distance, without considering other factors which affect traffic congestion. [5] provides a suggestion with a new approach which is to include multi-criteria in selecting the best path. The multi-criteria in selecting the best path in this study are the degree of saturation (level of service), route distance and time travel of the path.

3.PROBLEM STATEMENT

Transferring data in an optimised way is an integral part of most computer technologies. Another important aspect of data transfer is maintaining accuracy of data transferred. For fast transmission of data, reducing the number of bits can be the solution. But reducing the number of bits should not result in loss of actual data.

4.OBJECTIVES

- Compressing text/image file using huffman coding.
- Generating a graph with random values representing nodes in a network.
- Finding an optimal path with minimum traffic between two given nodes for transfer simulation.
- Network programming to send compressed files.
- Decompressing the data after transfer simulation is complete.

5. SYSTEM REQUIREMENT

- **Hardware Interface:**
 - 32-bit / 64-bit processor architecture supported by windows.
 - RAM requirement for proper functioning is around 2 GB.
- **Software Interface:**
 - Windows/Linux based Operating System
 - GCC Compiler.

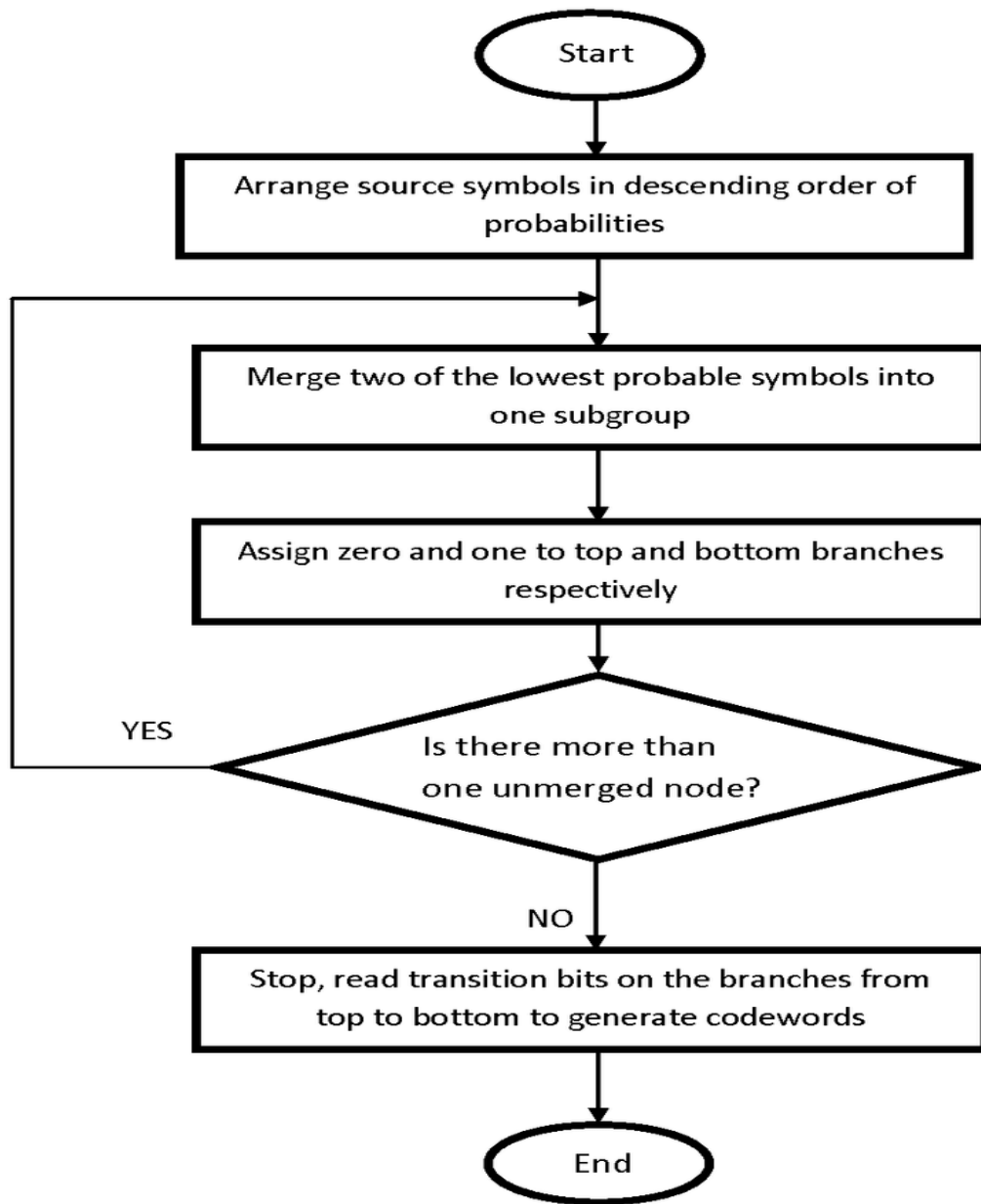
6. METHODOLOGY

- I. In this project the software development use is the Agile model.
- II. For high density lossless data(image/text) compression, Huffman coding is used to reduce the number of bits required to represent the data item.
- III. Random weighted graphs are generated with a fixed number of vertices and multiple edges to represent a network topology.
- IV. Shortest path algorithm, Dijkstra's algorithm is used to find the shortest and optimal path for the transfer of data items.
- V. Network programming to send this compressed data from client to the server using TCP protocol.
- VI. Decompression of the data item is done at the receiver end after the transfer of data item so that the transferred data is readable and the receiver does not encounter data loss or any quality degradation.

6.1 Huffman Coding

Huffman algorithm is an example of a greedy algorithm.

- I. Start from calculating the frequency of each character/pixel intensities in the input image/text file and keeping a record of it.
- II. Once frequencies have been determined, sort the characters in increasing order of those frequencies/weights. Store the result in a priority queue/structure node.
- III. Find each unique character in the queue and make leaf nodes for each of them or in case of image find two leaf nodes with minimum weight and combine them to form a new node keeping in mind the order of the node.
- IV. Now we create an empty node, say X. Provide minimum frequency to the left child of X while the second minimum frequency to the right child of X. After this, give a value to X which will be the sum of these 2 minimum frequencies.
- V. Hence it is time to remove these 2 minimum frequencies from the queue we created in the above steps and add the sum to the list of frequencies.
- VI. Insert this X node in the tree.
- VII. Finally, repeat the steps 3 to 5 for all the characters remaining in the input file.
- VIII. Once we have the tree formed, we will assign 0 to each leaf edge and 1 to each edge of each node and write the encoded values in the image/text file.
- IX. After this we will take out the compressed code from this tree and along with the key which is a table which contains corresponding values of each character to its coded value will be ready to be used by other modules.



6.1 Huffman Algorithm Flowchart

6.2 Network topology creation and selection of optimal path

An undirected weighted graph is generated to represent a network topology. This topology is generated to simulate the transfer of the data(text,images) between computational nodes. After the graph is generated, the nodes of the graph are attached with IP addresses. A source node and destination node is selected at random for file transfer. The edge weights in the graph are considered as traffic and the path with minimum traffic is selected using Dijkstra's Algorithm.

- I. A random value is selected to use as the number of vertices in the graph.
- II. After selecting the number of vertices, edge weights are assigned randomly between a specified range.
- III. IP address is assigned to all the vertices in the graph and displayed.
- IV. A source node and destination node is selected randomly.
- V. The IP address of the source node and destination node is displayed.
- VI. Dijkstra's algorithm is applied to display the minimum traffic values and path from the selected source node to all nodes.
- VII. Network parameters are displayed for the compressed and uncompressed file.

7. Implementation

7.1 Data Flow Diagram

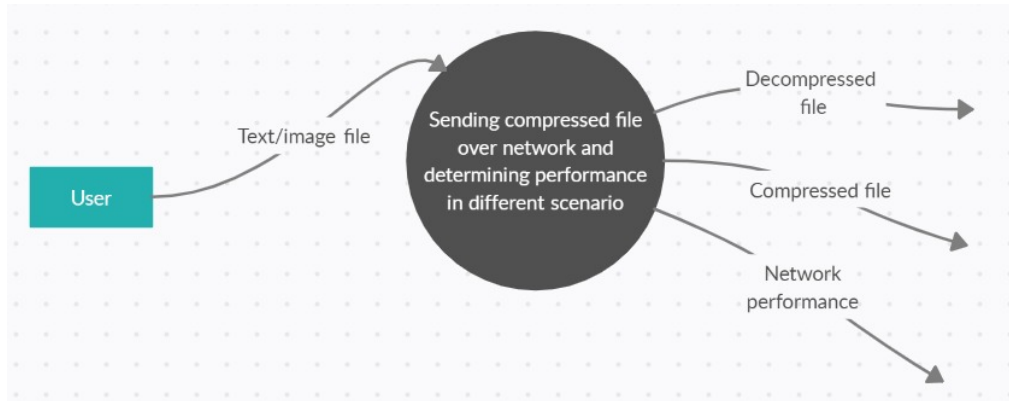


Figure 7.1 Level 0 DFD

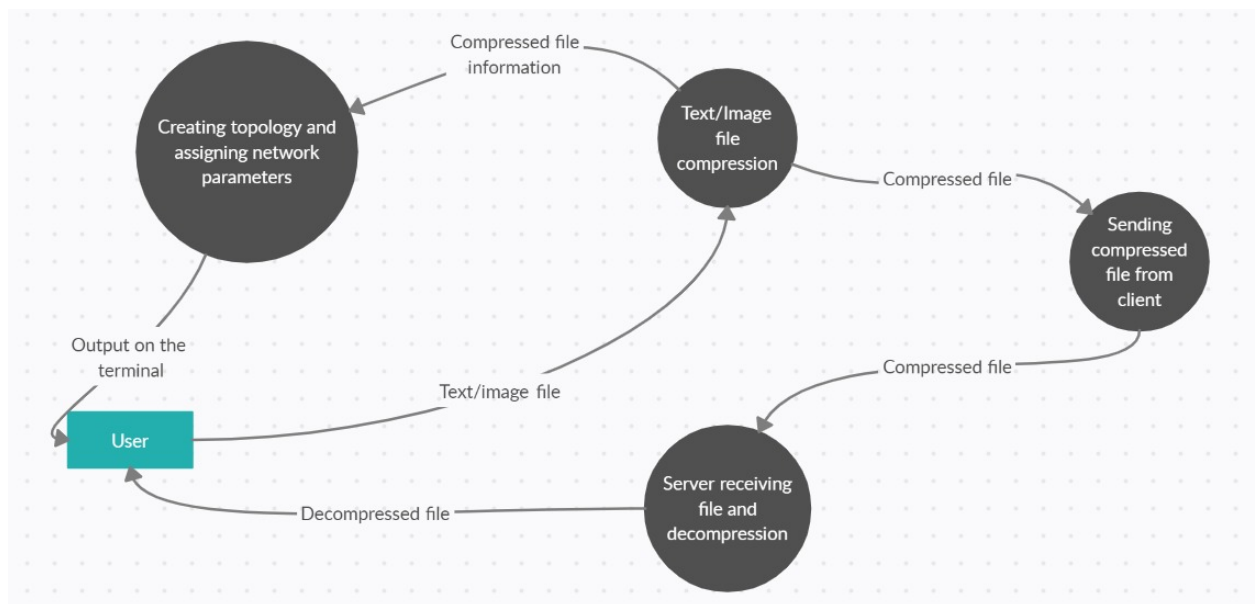


Figure 7.2 Level 1 DFD

7.2 Text File compression using Huffman Algorithm

The program starts by if condition directs program flow towards text file compression. This means that the user has wished to compress a text file in the process. For testing use a sample file which can be seen in screen shot. This file is read by the program using file handling. It reads each character in the file and prepares a frequency table. That is half of the need for the input file. Next the tree was prepared containing all the file's characters at the base of the tree. This base while containing each input character also contains frequency of each character which determines its weight in the file. Higher the weight, more the frequency thus smaller the code we need to represent it in the file to achieve compression. Once the tree has been built a single node at topmost location will be formed, through which the rest of the tree can be travelled straight to the bottom-most point to get to the character that will be represented and written to the decompressed file. Actually here, Huffman codes are not being formed like the one happened in the works of midsem. Due to the problem of 'prefix-match' of codes, this idea has to be dropped. The tree travelled to reach the desired character through bitwise operations and was employed to not only write the code into the file but also to read them back and find corresponding characters to it.

Opening the original input file which needs to be compressed. This is a plain text file in Figure 7.3 whose content will be compressed by the program.

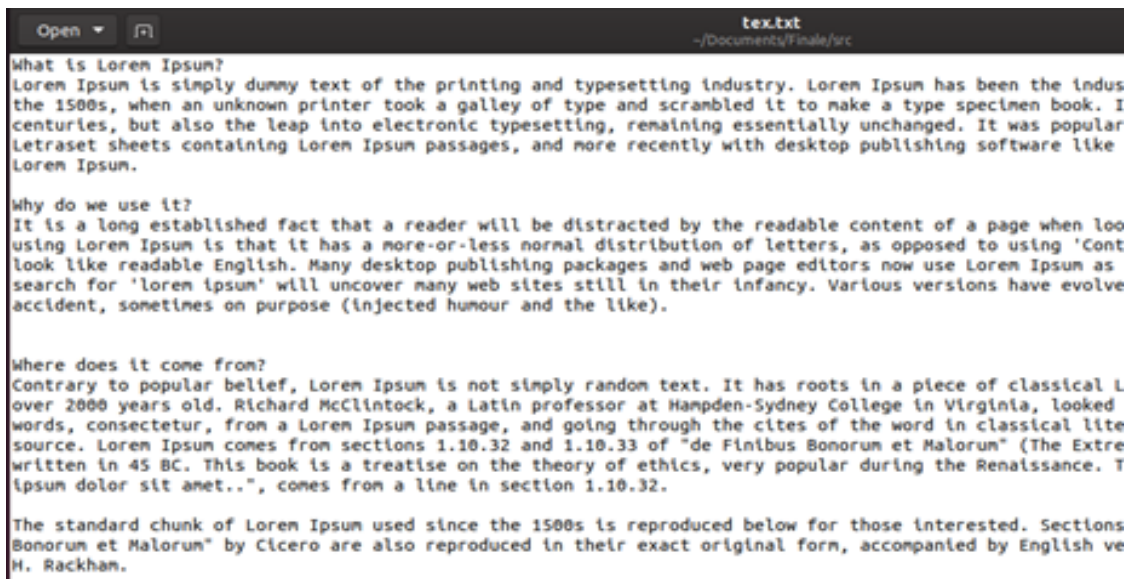


Figure 7.3 Input Text File

Figure 7.4 represents statistics of the input file, the original file size.

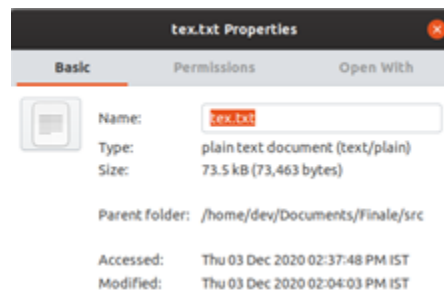


Figure 7.4 Input File Size

Figure 7.5 represents opening the compressed file after implementing the huffman algorithm.

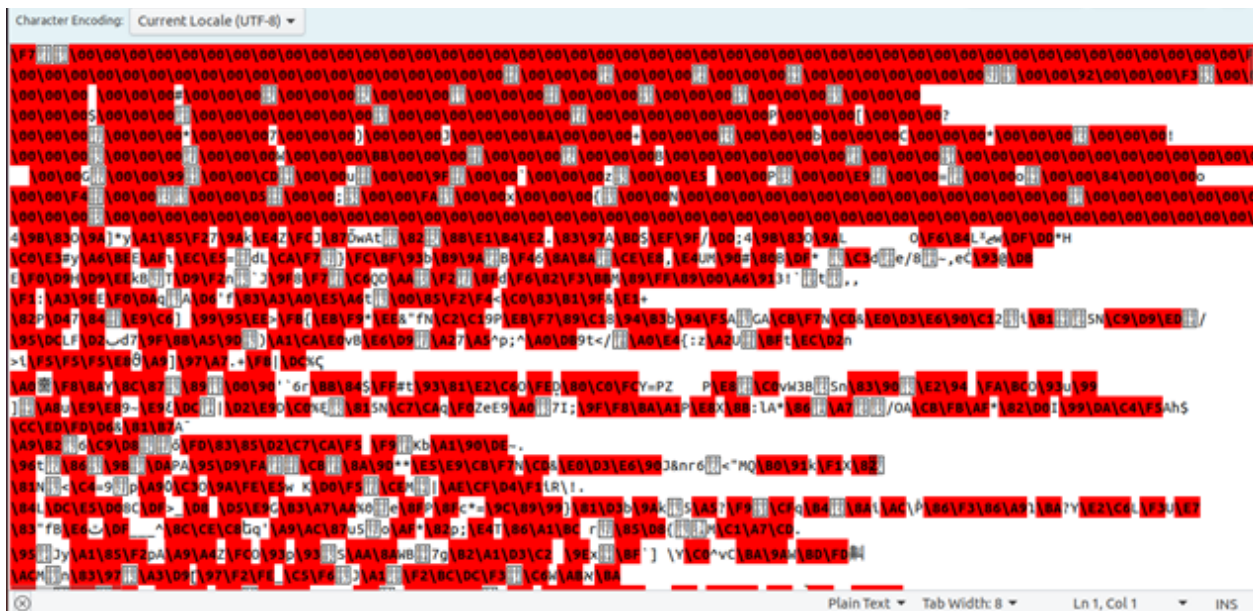


Figure 7.5 Compressed File.

Figure 7.6 represents statistics of the output file, the new compressed size.

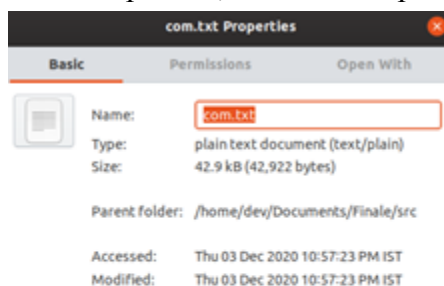


Figure 7.6 Compressed File Size.

After the complete execution of the code, the decompression part has been executed, this results in retrieval of the original text file represented in Figure 7.7 which was once coded to an unreadable format.

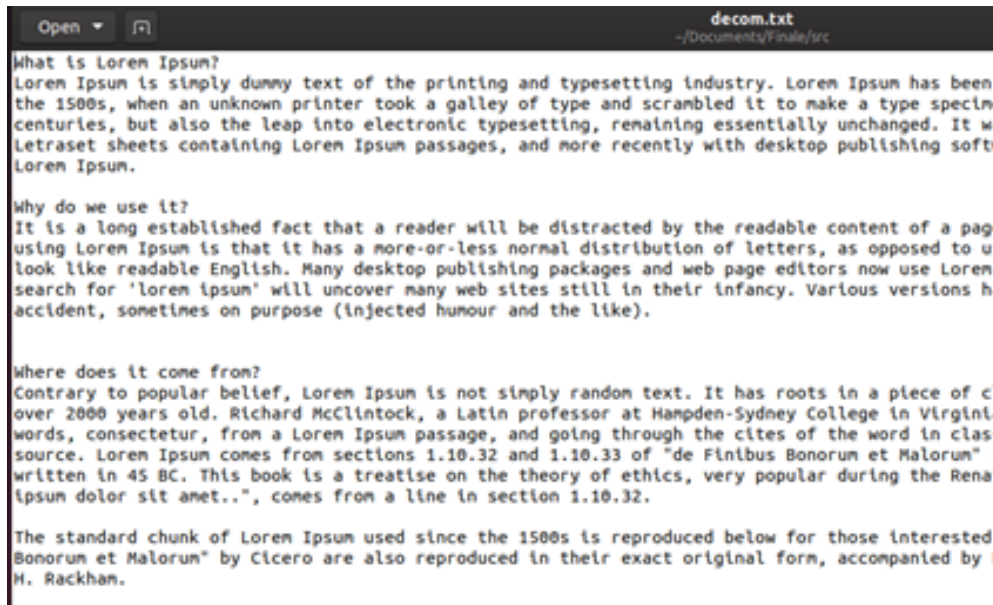


Figure 7.7 Decompressed File.

Figure 7.8 represents statistics of a new decompressed file, which shows that it has been decompressed to original size, back to normal.

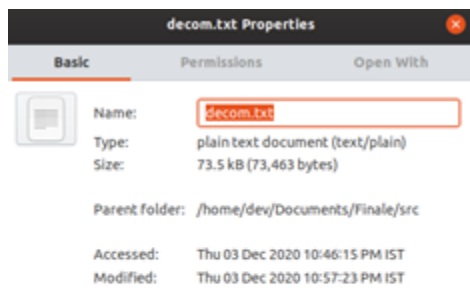


Figure 7.8 Decompressed File Size.

7.3 Image Compression using Huffman Coding

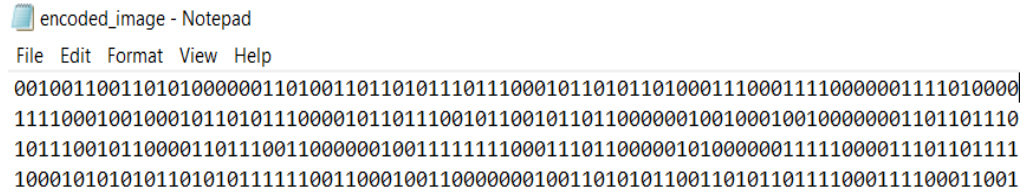
We have used array and struct data structure to represent the image in the encoded format. We are using File Handling to read image into binary format and to convert it into 2D array of pixel intensities for Huffman tree generation. Building a Huffman tree requires combining leaf nodes aka pixel intensities values into a single node then sorting them and repeating combining and sorting until we get a single node. We then start backtracking and assigning '0' or '1' to each intermediate node, till we reach the leaf nodes. After assigning it comprises of three steps:

- I. Creating a Huffman key/table consisting of pixel intensities with their corresponding encoded code

```
C:\project>gcc huff.c
C:\project>a
Huffmann Codes:
Pixel Intensity Values - ASCII Code
79      -> 011100
80      -> 010100
84      -> 110010
85      -> 110011
86      -> 110100
88      -> 011001
90      -> 011110
98      -> 011111
99      -> 11100
100     -> 010110
103     -> 010111
104     -> 00101
105     -> 10010
111     -> 00001
112     -> 110000
113     -> 110001
114     -> 110110
115     -> 110111
117     -> 0001
118     -> 110101
119     -> 011010
121     -> 011011
124     -> 011000
125     -> 11101
126     -> 00110
127     -> 00111
129     -> 00100
130     -> 011101
131     -> 010010
135     -> 010011
137     -> 010000
138     -> 010001
140     -> 1111
141     -> 00000
142     -> 10011
143     -> 010101
144     -> 101010
145     -> 101011
150     -> 101000
153     -> 101001
156     -> 101110
160     -> 101111
```

Figure 7.9 Implementation of Huffman Key

- II. Writing the encoded code of each pixel in the image to a file which is known as encoded_image file.



```

00100110011010100000011010011011011101110001011010110100011100011110000001111010000
1111000100100010110101110000101101110010110010110110000001001000100100000001101101110
101110010110000110111001100000010011111110001110110000010100000011111000011101101111
10001010101011010101111110011000100110000000100110101100110101101111000111100011001

```

Figure 7.10 Implementation of Image Compression

- III. Using the generated huffman key to get back the uncompressed image file







 a	04-12-2020 14:54	Application	93 KB
 encoded_image	04-12-2020 14:54	Text Document	1 KB
 file	30-11-2020 20:36	BMP File	1 KB
 huff.c	04-12-2020 13:28	C File	9 KB
 huffman	01-12-2020 12:08	H File	1 KB
 uncompressed	04-12-2020 14:54	BMP File	1 KB

Figure 7.11 Generation of uncompressed file



Figure 7.12 decompressed file

- IV. Calculating the size of compressed file and difference between actual and compressed file using file handling and passing these variables as a result of function call for ip addressing.
- V. Calculating the size of decompressed file and passing the variable by function calling for sending the file to multiple client

```
There are total 48 unique pixel intensities in the image  
  
Original filesize: 0.064000  
Compressed filesize: 0.031375  
Decompressed file size:-0.062500
```

Figure 7.13 Getting the variables required for network simulation

7.4 Network topology creation and selection of optimal path

A random weighted graph is generated to represent a network topology. The nodes of the graph represent the computing nodes which will send or receive data and the edges represent the traffic between these nodes.

In this project representation of the graph is implemented using adjacency matrix.

A function is used to randomly generate the number of vertices within a specified range.

The number of vertices is then passed to a function to generate edges of random weight between the vertices within a specified range. It returns a 2D array.

The output printed shows the adjacency matrix representation of the graph.

```
D:\minor\project\final.exe
Selecting random number of vertices to create a graph topology...
Number of vertices: 15
Assigning weights to the edges between vertices...
Matrix representation of the graph generated
```

8	5	1	10	5	9	9	3	5	6	6	2	8	2	2
5	3	8	7	2	5	3	4	3	3	2	7	9	6	8
1	8	9	10	3	8	10	6	5	4	2	3	4	4	5
10	7	10	9	8	5	3	8	8	10	4	2	10	9	7
5	2	3	8	7	1	3	5	9	7	6	1	10	1	1
9	5	8	5	1	4	5	5	7	1	7	7	2	9	5
9	3	10	3	3	5	3	10	2	4	6	10	9	5	1
3	4	6	8	5	5	10	3	1	10	8	4	8	3	7
5	3	5	8	9	7	2	1	3	1	1	2	5	7	1
6	3	4	10	7	1	4	10	1	10	10	9	2	9	3
6	2	2	4	6	7	6	8	1	10	10	5	8	9	4
2	7	3	2	1	7	10	4	2	9	5	10	9	5	2
8	9	4	10	10	2	9	8	5	2	8	9	4	8	2
2	6	4	9	1	9	5	3	7	9	9	5	8	4	5
2	8	5	7	1	5	1	7	1	3	4	2	2	5	1

Figure 7.14 Implementation of random graph generation

```

Attaching IP addresses to all nodes for simulating transfer...

IP address of node 0 is 198.243.180.161
IP address of node 1 is 8.236.209.143
IP address of node 2 is 214.127.72.162
IP address of node 3 is 178.52.173.209
IP address of node 4 is 85.77.53.71
IP address of node 5 is 180.193.31.36
IP address of node 6 is 137.84.47.165
IP address of node 7 is 183.206.97.41
IP address of node 8 is 5.128.34.252
IP address of node 9 is 129.141.206.123
IP address of node 10 is 122.36.76.89
IP address of node 11 is 242.171.97.164
IP address of node 12 is 230.2.50.215
IP address of node 13 is 93.94.11.153
IP address of node 14 is 141.99.210.103

Selecting a source node to simulate the transfer...

Source node is node-7

Selecting a destination node for simulating the transfer...

Destination node is node-10

Source ip address 183.206.97.41
Destination ip address 122.36.76.89

```

Figure 7.15: Assigning IP addresses to all nodes and selecting source node and destination node.

```

Applying Dijkstra algorithm to find the minimum traffic from source to all nodes...

Vertex    Traffic    Path
7 -> 0    3          7 0
7 -> 1    4          7 1
7 -> 2    4          7 8 10 2
7 -> 3    5          7 8 11 3
7 -> 4    3          7 8 14 4
7 -> 5    3          7 8 9 5
7 -> 6    3          7 8 6
7 -> 7    0          7
7 -> 8    1          7 8
7 -> 9    2          7 8 9
7 -> 10   2          7 8 10
7 -> 11   3          7 8 11
7 -> 12   4          7 8 14 12
7 -> 13   3          7 13
7 -> 14   2          7 8 14

```

Figure 7.16: Implementation of Dijkstra algorithm

```

Bandwidth=20 kbps
Collision percentage = 10
Packet size=2 kb
The number of packets created is 50.
Packets that need to be resend 5.
Total transfer time = 5.000000 seconds.

```

Figure 7.17: network parameters with file size 100 kb

7.5 Network Programming

- I. This part of the project focuses on sending compressed files using network programming.
- II. TCP protocol is used for sending files.
- III. This is achieved by creating a server and a client. Compression done at client side. Then this compressed file is sent to the server. At the server side, the file is decompressed.
- IV. Multiple clients can connect to the server at the same time. Each of the clients gets a separate port upon connection. In this way we separate clients from each other.
- V. Server runs an infinite loop, to accept the connections from the client all the time. Concept of process identification(pid_t) is used to differentiate the clients.

The image displays two terminal windows side-by-side, illustrating a network communication process. The left terminal window, titled 'Terminal', shows a server running on localhost. It starts with the command `./server`, which outputs `[+]Server Socket is created.` and `[+]Bind to port 4444`. It then enters a listening state, receiving multiple connections from clients at various ports (36668, 36670, 36672, 36674, 36676). Each connection results in `[+]Data written in the file successfully.` and `Connection accepted from client through port no: [port number]`. The right terminal window, also titled 'Terminal', shows a client running on localhost. It starts with the command `./client`, which outputs `[+]Client Socket is created.` and `[+]Connected to Server.`. The client then sends data to the server, with each successful transmission outputting `Client: [+]File data sent successfully.` and `[+]Closing the connection.`. This sequence of operations is repeated multiple times, demonstrating a continuous client-server interaction.

Figure 7.18 Network Programming demonstrating a single server and multiple clients

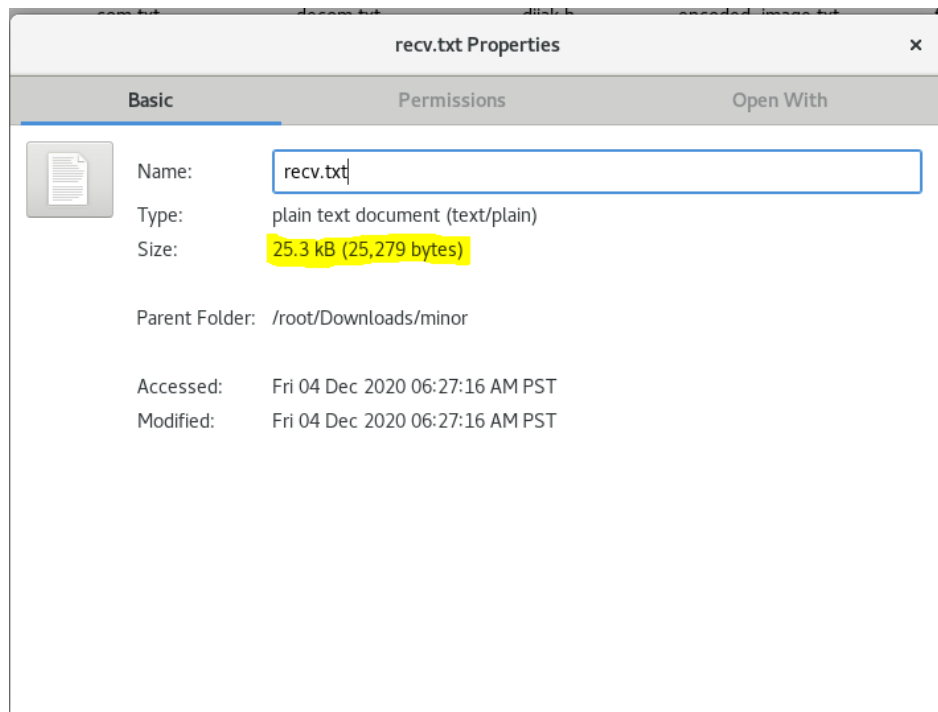


Figure 7.19 Size of both the files is same

7.6 Integrated Results

For Text

```
[+]Data written in the file successfully.  
Enter 1 if it is image  
Enter 0 if it is text  
Enter your choice:  
0  
  
Bandwidth= 10 kbps  
Collision percentage = 10%  
Packet size=2 kb  
The number of packets created is 72.  
Packets that need to be resend 8.  
Total transfer time = 15.000000 seconds.  
  
[+]Data written in the file successfully.  
Connection accepted from client on port no:41186
```

Figure 7.20 Client Side for text

```
[root@localhost (final_day)_og_minor_(ambitious)]# ./multiple_client
press 0 for text file
press 1 for image file
enter your choice:
0
Selecting random number of vertices to create a graph topology...
Number of vertices: 11
Assigning wiegths to the edges between vertices...
Attaching IP addresses to all nodes for simulating transfer...
IP address of node 0 is 111.142.147.71
IP address of node 1 is 201.65.219.172
IP address of node 2 is 207.151.137.243
IP address of node 3 is 81.5.121.113
IP address of node 4 is 44.14.13.96
IP address of node 5 is 107.89.213.89
IP address of node 6 is 225.108.193.185
IP address of node 7 is 85.99.66.68
IP address of node 8 is 113.85.11.186
IP address of node 9 is 151.230.104.103
IP address of node 10 is 254.113.91.80
Selecting a source node to simulate the transfer...
Source node is node-3
Selecting a destination node for simulating the transfer...
Destination node is node-9
Source ip address 81.5.121.113
Destination ip address 151.230.104.103
```

Figure 7.21 Server side for text

```
Selecting a source node to simulate the transfer...
Source node is node-3
Selecting a destination node for simulating the transfer...
Destination node is node-9
Source ip address 81.5.121.113
Destination ip address 151.230.104.103
Applying Dijkstra algorithm to find the minimum traffic from source to all nodes...
Vertex    Traffic    Path
3 -> 0     4          3 0
3 -> 1     3          3 4 1
3 -> 2     3          3 2
3 -> 3     0          3
3 -> 4     2          3 4
3 -> 5     3          3 5
3 -> 6     6          3 4 1 6
3 -> 7     4          3 7
3 -> 8     2          3 8
3 -> 9     6          3 2 9
3 -> 10    5          3 10
FOR COMPRESSED FILE
Bandwidth= 10 kbps
Collision percentage = 10%
Packet size=2 kb
The number of packets created is 42.
Packets that need to be resend 5.
Total transfer time = 9.000000 seconds.
[+]Client Socket is created.
[+]Connected to Server.
[+]Connection established between 81.5.121.113 and 151.230.104.103
[+]File data sent successfully.
[+]Closing the connection.
[root@localhost (final day)_og_minor_(ambitious)]#
```

Figure 7.22 Client Side for text

```
[root@localhost (final_day)_og_minor_(ambitious)]# ./multiple_server
Size of the file is 94208 bytes
[+]Server Socket is created.
[+]Bind to port 4444
[+]Listening...
Enter 1 if it is image
Enter 0 if it is text
Enter your choice:
1

Bandwidth= 5 bps
Collision percentage = 10%
Packet size=10 kb
120.586243The number of packets created is 13.
Packets that need to be resend 2.
Total transfer time = 26.000000 seconds.

[+]Data written in the file successfully.

Connection accepted from client on port no:41184
```

Figure 7.23 Server Side for image

```
[root@localhost (final_day)_og_minor_(ambitious)]# ./multiple_client

press 0 for text file
press 1 for image file

enter your choice:
1
The size of original file is: 402653.187500 bytes
There are total 255 unique pixel intensities in the image
The difference between the actual file and compresses file is: 402425.718750 bytes
Selecting random number of vertices to create a graph topology...

Number of vertices: 11

Assigning wieghts to the edges between vertices...

Attaching IP addresses to all nodes for simulating transfer...

IP address of node 0 is 111.142.147.71
IP address of node 1 is 201.65.219.172
IP address of node 2 is 207.151.137.243
IP address of node 3 is 81.5.121.113
IP address of node 4 is 44.14.13.96
IP address of node 5 is 107.89.213.89
IP address of node 6 is 225.108.193.185
IP address of node 7 is 85.99.66.68
IP address of node 8 is 113.85.11.186
IP address of node 9 is 151.230.104.103
IP address of node 10 is 254.113.91.80

Selecting a source node to simulate the transfer...

Source node is node-3

Selecting a destination node for simulating the transfer...

Destination node is node-9
```

Figure 7.24 Client Side for image

```

Source ip address 81.5.121.113
Destination ip address 151.230.104.103

Applying Dijkstra algorithm to find the minimum traffic from source to all nodes...

Vertex    Traffic    Path
3 -> 0      4          3 0
3 -> 1      3          3 4 1
3 -> 2      3          3 2
3 -> 3      0          3
3 -> 4      2          3 4
3 -> 5      3          3 5
3 -> 6      6          3 4 1 6
3 -> 7      4          3 7
3 -> 8      2          3 8
3 -> 9      6          3 2 9
3 -> 10     5          3 10

FOR COMPRESSED FILE
Bandwidth= 5 Bps
Collision percentage = 10%
Packet size= 10 bytes
The number of packets created is 23.
Packets that need to be resend 3.
Total transfer time = 48.000000 seconds.
[+]Client Socket is created.

[+]Connected to Server.
[+]Connection established between 81.5.121.113 and 151.230.104.103

[+]File data sent successfully.
[+]Closing the connection.

```

Figure 7.25 Client Side for image

8. PERTCHART

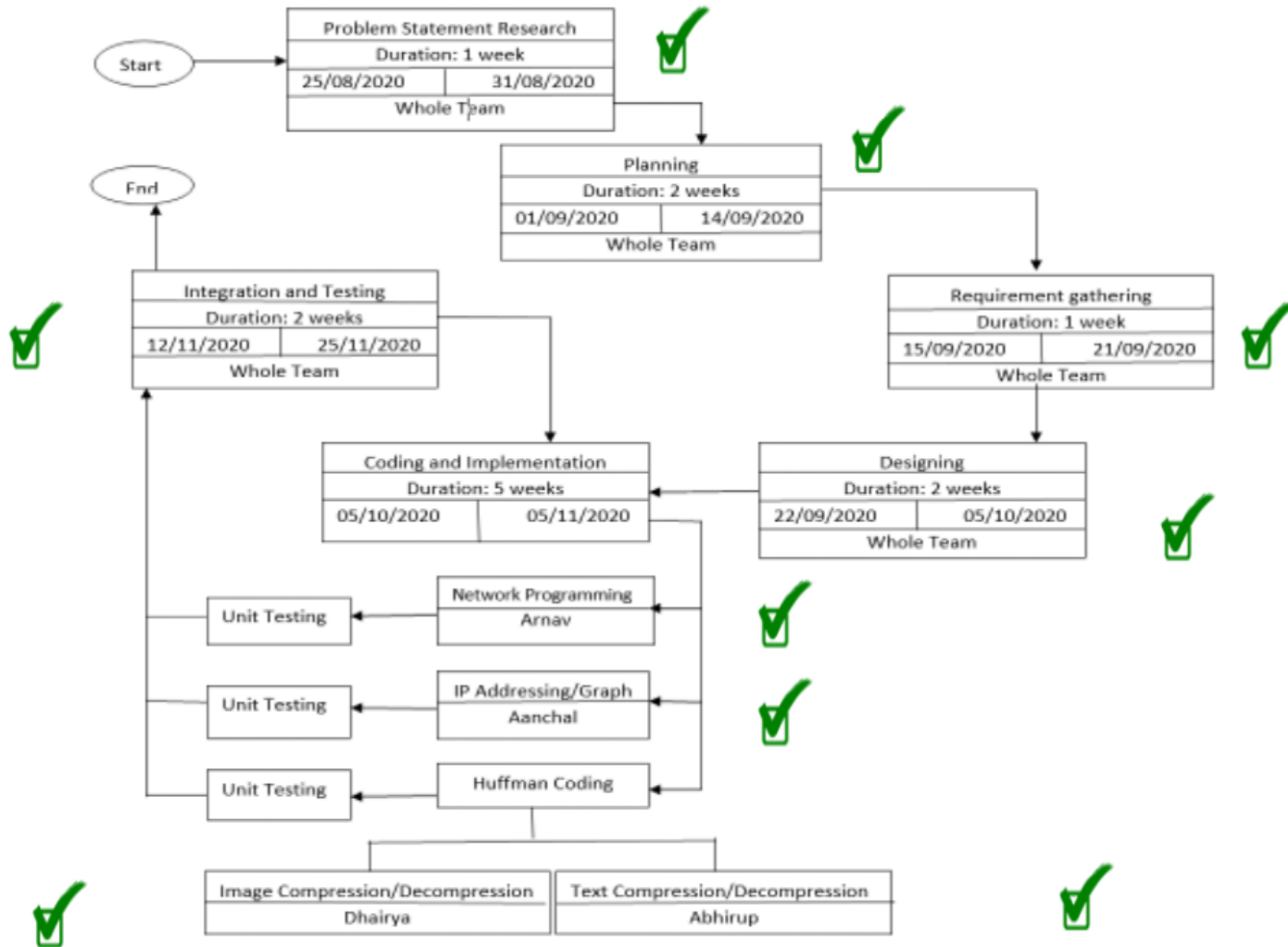


Figure 8.1 Pert Chart

9. Code

Link for Code: https://github.com/Abhirup-Kumar/Minor_1.git

10. REFERENCES

- [1] Ali Tariq Bhatti and Dr. Jung Kim, "Implementation of Lossless Huffman Coding: Image compression using K-Means algorithm and comparison vs. Random numbers and Message check methodology," IRJET, Volume: 02 Issue: 05, Aug-2015.
- [2] Ahsan Habib and Mohammad Shahidur Rahman, "Balancing decoding speed and memory usage for Huffman codes using quaternary trees," Applied Informatics, Issue 1/2017.
- [3] K. Uchida and L. Barolli, "Dijkstra Algorithm Based Ray Tracing: A Case Study for Tunnel Structures," 2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA), Krakow, 2018, pp. 82-87, doi: 10.1109/WAINA.2018.00067..
- [4] Alashoury, Ibtusam & Amarif, Mabroukah. (2019). An Effect of Using A Storage Medium in Dijkstra Algorithm Performance for Ideal Implicit Path Cost. International Journal of Computer Science and Information Technology. 11. 27-41. 10.5121/ijcsit.2019.11403.
- [5] Abdullah, Noraini & Hua, Ting. (2018). Weighted Methods of Multi-Criteria Via Dijkstra's Algorithm in Network Graph For Less Congestion, Shorter Distance and Time Travel in Road Traffic Network. Global Journal of Engineering Science and Researches. 5. 46-57. 10.5281/zenodo.1305010.
- [6] Sharma, Mamta. (2010). compression using huffman coding. 10. 133-141.
- [7] <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/> [accessed on 10/09/2020]
- [8] <https://searchstorage.techtarget.com/definition/compression> [accessed on 9/09/2020]
- [9] https://en.wikipedia.org/wiki/Huffman_coding [accessed on 9/09/2020]
- [10] <https://www.geeksforgeeks.org/compression-using-huffman-coding/> [accessed on 8/09/2020]
- [11] <https://www.programiz.com/dsa/huffman-coding> [accessed on 9/09/2020]
- [12] https://www.khronos.org/opengl/wiki/Main_Page [accessed on 8/09/2020]
- [13] <https://www.gatevidyalay.com/huffman-coding-huffman-encoding> [accessed on 8/09/2020]

Synopsis Draft Verified By

Dr. Deepshikha Bhargava

Head of Department
Department of Virtualization
School of Computer Science



University of Petroleum and Energy Studies

HOD
(Dr. Deepshikha Bhargava)

Ms. Avita Katal

Assistant Professor-Senior Scale
Department of Virtualization
School of Computer Science



University of Petroleum and Energy Studies

Project Guide
(Ms. Avita Katal)