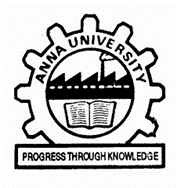
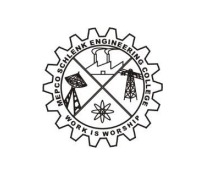
** FILE COMPRESSION AND DECOMPRESSION USING SHARED MEMORY**

**A MINI PROJECT REPORT**

***Submitted by***

**JENNIE R (9517202109023)**

**RAMYA G K (9517202109043)**

**SREWATHI T K (9517202109050)**

**in**

**19AD481 – OPERATING SYSTEM PRINCIPLES**

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

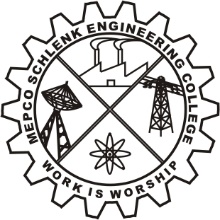
**MEPCO SCHLENK ENGINEERING COLLEGE**

**SIVAKASI**

##### **MAY 2023**

**MEPCO SCHLENK ENGINEERING COLLEGE, SIVAKASI**

**AUTONOMOUS**

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**BONAFIDE CERTIFICATE**

This is to certify that it is the bonafide work of “**JENNIE.R (9517202109023) , RAMYA.G.K (9517202109043) , SREWATHI.T.K (9517202109050)”** for the mini project titled **“FILE COMPRESSTION USING SHARED MEMORY”** in 19AD481 – Operating System Principles during the fourth semester January 2023 – May 2023 under my supervision.

**SIGNATURE** **SIGNATURE**

**Mrs. P. Swathika, Dr. J. Angela Jennifa Sujana,**

**Asst. Professor (Senior Grade), Asso. Professor (Senior Grade)& Head,**

AI&DS Department, AI&DS Department

Mepco Schlenk Engg. College, Sivakasi Mepco Schlenk Engg. College, Sivakas

**ABSTRACT**

In our project, on “FILE COMPRESSION AND DECOMPRESSION WITH SHARED MEMORY”, we have tried to show how this technique used to reduce the size of files by removing redundant information and storing them in a compressed format. This technique utilizes shared memory, which is a memory space that can be accessed by multiple processes simultaneously. By using shared memory, the compression process can be accelerated as the data can be directly transferred between the processes without the need for expensive data copying. This approach also reduces the overall memory usage and improves the performance of the compression algorithm.

In this project, we discuss the implementation of file compression using shared memory and present experimental results to evaluate its effectiveness. The results demonstrate that this technique provides a significant improvement in compression speed and memory usage compared to traditional file compression algorithms.

The process of file compression has become increasingly important with the growing need for efficient storage and transfer of data. In this context, the use of shared memory can significantly enhance the performance of file compression algorithms. This approach allows multiple processes to access a common area of memory, eliminating the need for data copying between processes. This results in faster processing times and reduced memory usage. In this abstract, we explore the benefits of using shared memory for file compression and discuss the implementation of such algorithms. Our findings suggest that the use of shared memory can result in significant improvements in file compression performance, making it an attractive option for applications that require efficient data processing.

|  |  |  |
| --- | --- | --- |
|  |  |  |

**TABLE OF CONTENTS**

**Chapter 1: Introduction**

1.1 Introduction………………………………………………Page No. 5

1.2 Objectives………………………………………………...Page No. 7

1.3 Scope of the project………………………………………Page No. 9

**Chapter 2: Implementation**

2.1 Program Coding…………………………………………….Page No. 10

2.2 Output………………………………………………………Page No. 21

**Chapter 3: Conclusion**

**References**

**CHAPTER 1**

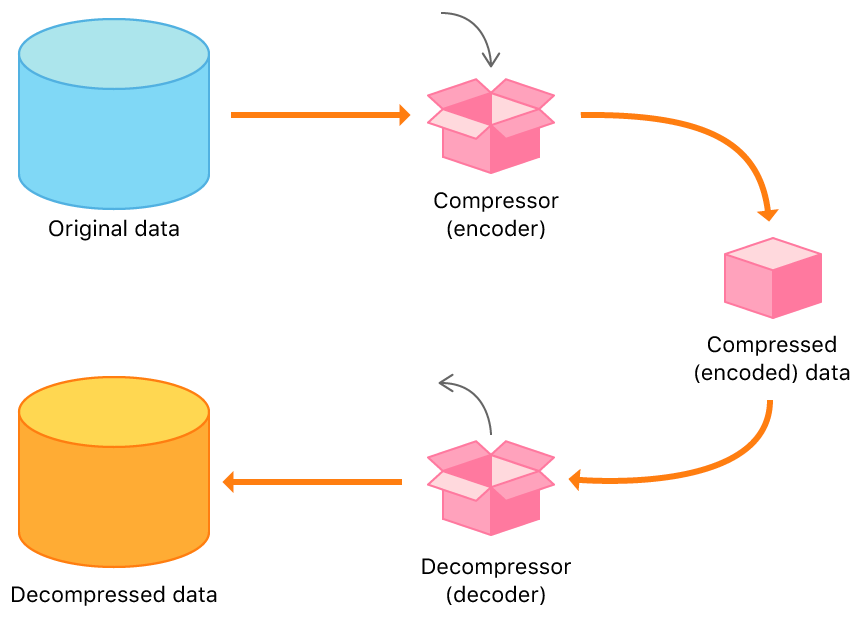
**INTRODUCTION**

**1.1 INTRODUCTION**

Our project on FILE COMPRESSION AND DECOMPRESSION USING SHARED MEMORY, is the process of reducing the size of a file by removing redundant or unnecessary data, while still preserving the essential information needed to reconstruct the original file. This is done through the use of compression algorithms, which analyze the data in the file and identify patterns that can be represented more efficiently.

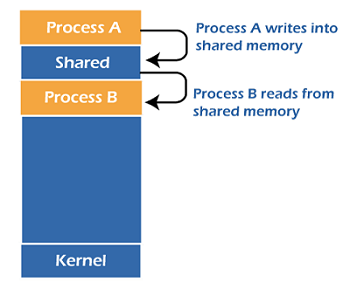
Huffman coding (also known as Huffman Encoding) is an algorithm for doing data compression, and it forms the basic idea behind file compression. This post talks about the fixed-length and variable-length encoding, uniquely decodable codes, prefix rules, and Huffman Tree construction. Every information in computer science is **encoded** as strings of **1s and 0s**. The objective of information theory is to usually transmit information using fewest number of bits in such a way that every encoding is unambiguous. This tutorial discusses about fixed-length and variable-length encoding along with Huffman Encoding which is the basis for all data encoding schemes. Encoding, in computers, can be defined as the process of transmitting or storing sequence of characters efficiently.

File compression is useful for a variety of reasons. It can help to save disk space, reduce bandwidth usage when transferring files over the internet, and speed up backups. However, it's important to note that compression isn't always appropriate for every type of file, and it may not always provide significant savings in terms of file size. Additionally, compressed files may take longer to access or decompress, depending on the algorithm used and the hardware being used to read the files.



**Figure 1.1 Process Diagram**

Shared memory allows multiple processes to share virtual memory space. This method is the fastest to coordinate but not necessarily the easiest for processes to communicate with one another. In general, one process creates/allocates the shared memory segment. The size and access permissions for the segment are set when it is created. The process then attaches, or opens, the shared segment, causing it to be mapped into its current data space. If needed, the creating process then initializes the shared memory. Once created, and if permissions permit, other processes can gain access to the shared memory and map it into their data space.



**Figure 1.2 Shared Memory**

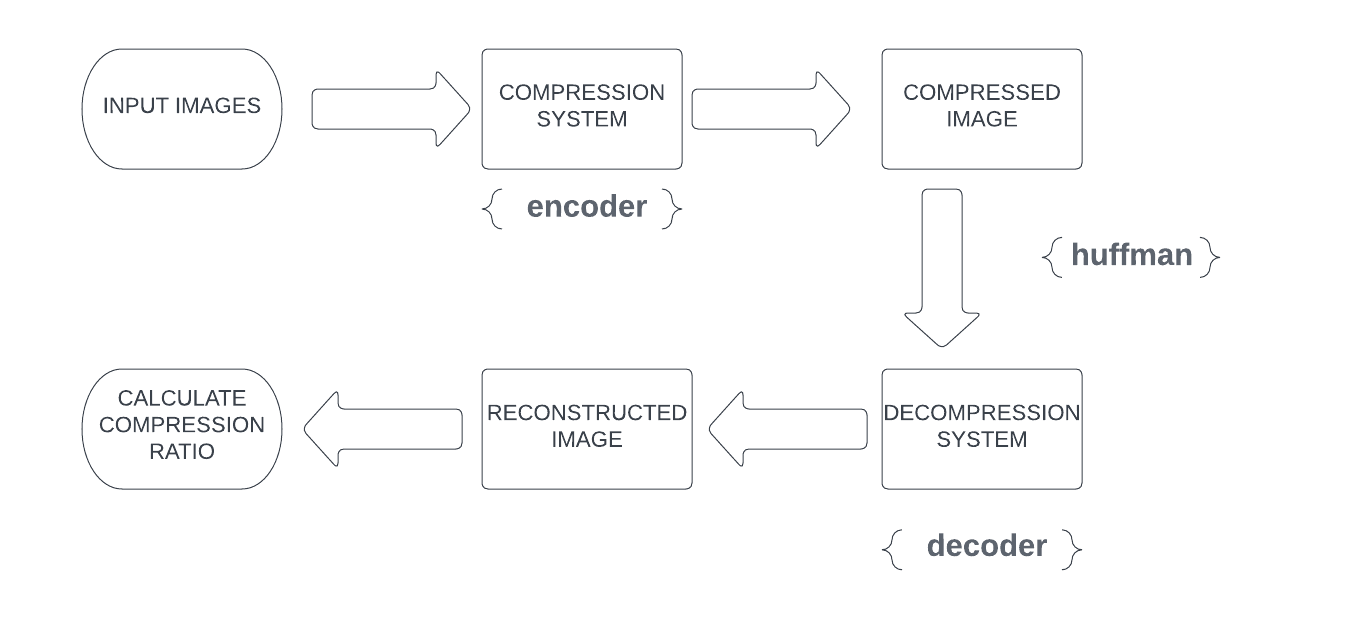
**1.2 OBJECTIVES**

The main objective of this project is to design a compression software which significantly reduces the size of a file so that it can be easily shared over mail even in slow internet speed. This software works in the same way as the winrar and winzip, which are popular compression tools. Each byte of the file will be compressed and takes quite less memory on the disk.

Presently, the files cannot be sent or shared across mail or even across network if their size is above a certain limit. Even if the size lies in the allowable limit, then also sharing is difficult during low internet connection. These constraints create a lot of problem when something important needs to be shared urgently and the person is bounded by the size limit. Such situations require a compression tool which one should have handy.

We sometimes face problems related to memory issues while saving or transferring any document or a file. Hence, there is a need to compress it so that it is reduced in size and can be transferred and processed easily. Data Compression and Decompression provide users to store data in less space and also easier for the user to transfer over the network. The Compression reduces the file of the size so that we can share it easily over any network even at a slow speed. This project aims at compressing and decompressing different types of files stored in our computer system at a particular destination location.

Huffman Algorithm is an efficient way for file Compression and Decompression. This program exactly follows Huffman algorithm. It reads frequent characters from input file and replaces them with shorter binary codeword. The original file can be produced again without losing any bit.

****

**Figure 1.3 Block Diagram**

Shared memory is an operating-system feature that allows the database server threads and processes to share data by sharing access to pools of memory. The database server uses shared memory for the following purposes: To reduce memory usage and disk I/O. To perform high-speed communication between processes. The fundamental feature of a shared-memory computer is that all the CPU-cores are connected to the same piece of memory. This is achieved by having a memory bus that takes requests for data from multiple sources (here, each of the four separate CPU-cores) and fetches the data from a single piece of memory.

Shared memory allows cooperating processes to access the same pieces of data concurrently. Using shared memory, also speed ups the computation power of the system as the long task can be divided into smaller sub-tasks and can be executed in parallel. A shared memory segment is a portion of physical memory that is shared by multiple processes. In this region, processes can set up structures, and others may read/write on them. When a shared memory region is established in two or more processes, there is no guarantee that the regions will be placed at the same base address.

**1.3 SCOPE OF THE PROJECT**

The software to be developed deals with creating a Hotel Management

system which will automate the major hotel operations such as generating

COD, billing and keeping track of records of daily transaction.Admin have

the authority to control and modify the database

The software to be developed deals with creating a Hotel Management

system which will automate the major hotel operations such as generating

COD, billing and keeping track of records of daily transaction.Admin have

the authority to control and modify the database

In the present system, there are so many problems while transferring or sharing a file of large size. These files cannot be sent over a network if they are above certain size limits. Even if the size of the file is small but if the internet connection speed is slow, then too it is difficult to send them. These limitations create a lot of problems when we need to share an urgent file and we just can’t due to the size limit of the file. Therefore, there is a need to build a system that can compress and decompress the files easily so they can be shared efficiently without any constraint.

  Data compression is most consideration thing of the recent world. We have to compress a huge amount of data so as to carry from one place to other or in a storage format. That is why data has to compress. This Huffman coding technique has improved the efficiency of file (like .txt, .docx, .pptx) compression with the concepts of Typecasting and Data Normalization.

**CHAPTER 2**

**IMPLEMENTATION**

**2.1 PROGRAM CODING**

**// ENCODING COMPRESSION (Server)**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/shm.h>  
#include <unistd.h>  
  
typedef struct node\_t {  
        struct node\_t \*left, \*right;  
        int freq;  
        char c;  
} \*node;  
int n\_nodes = 0, qend = 1;  
struct node\_t pool[256] = {{0}};  
node qqq[255], \*q = qqq-1;  
char \*code[128] = {0}, buf[1024];  
int input\_data=0,output\_data=0;  
node new\_node(int freq, char c, node a, node b)  
{  
        node n = pool + n\_nodes++;  
        if (freq != 0){  
                n->c = c;  
                n->freq = freq;  
        }  
        else {  
 n->left = a, n->right = b;  
                n->freq = a->freq + b->freq;  
        }  
        return n;  
}

void qinsert(node n)  
{  
        int j, i = qend++;  
        while ((j = i / 2)) {  
                if (q[j]->freq<= n->freq) break;  
                q[i] = q[j], i = j;  
        }  
        q[i] = n;  
}  
node qremove()  
{  
        int i, l;  
        node n = q[i = 1];  
        if (qend< 2) return 0;  
        qend--;  
        while ((l = i \* 2) <qend) {  
                if (l + 1 <qend&& q[l + 1]->freq< q[l]->freq) l++;  
                q[i] = q[l], i = l;  
        }  
        q[i] = q[qend];  
        return n;  
}

void build\_code(node n, char \*s, int len)  
{  
        static char \*out = buf;  
        if (n->c) {  
                s[len] = 0;  
                strcpy(out, s);  
                code[(int)n->c] = out;  
                out += len + 1;  
                return;  
        }  
        s[len] = '0'; build\_code(n->left,  s, len + 1);  
        s[len] = '1'; build\_code(n->right, s, len + 1);  
}  
  
void import\_file(FILE \*fp\_in, unsigned int \*freq){  
        char c,s[16]={0};  
        int i = 0;  
        printf("File Read:\n");  
        while((c=fgetc(fp\_in))!=EOF){  
                freq[(int)c]++;  
                putchar(c);  
        }  
        for (i = 0; i< 128; i++)  
                if (freq[i]) qinsert(new\_node(freq[i], i, 0, 0));  
        while (qend> 2)  
                qinsert(new\_node(0, 0, qremove(), qremove()));  
        build\_code(q[1], s, 0);  
}  
void encode(FILE\* fp\_in, FILE\* fp\_out, unsigned int \*freq )

{  
  
        char in,c,temp[20] = {0};  
        int i,j=0,k=0,lim=0;  
        rewind(fp\_in);  
        for(i=0; i<128; i++){  
                if(freq[i])     lim += (freq[i]\*strlen(code[i]));  
        }  
        output\_data = lim;  
        fprintf(fp\_out,"%04d\n",lim);  
        printf("\nEncoded:\n");  
        for(i=0; i<lim; i++){  
                if(temp[j] == '\0'){  
                        in = fgetc(fp\_in);  
                        strcpy(temp,code[in]);  
                        printf("%s",code[in]);  
                        j = 0;  
                }  
                if(temp[j] == '1')  
                        c = c|(1<<(7-k));  
                else if(temp[j] == '0')  
                        c = c|(0<<(7-k));  
                else  
                        printf("ERROR: Wrong input!\n");  
                k++;  
                j++;  
                if(((i+1)%8 == 0) || (i==lim-1)){  
                        k = 0;  
                        fputc(c,fp\_out);  
                        c = 0;  
                }  
        }  
        putchar('\n');  
}  
void print\_code(unsigned int \*freq){  
        int i;  
        printf("\n---------CODE TABLE---------\n----------------------------\nCHAR  FREQ  CODE\n----------------------------\n");  
        for(i=0; i<128; i++){  
                if(isprint((char)i)&&code[i]!=NULL&&i!=' ')  
                        printf("%-4c  %-4d  %16s\n",i,freq[i],code[i]);  
                else if(code[i]!=NULL){  
                        switch(i){  
                                case '\n':  
                                        printf("\\n  ");  
                                        break;  
                                case ' ':  
                                        printf("\' \' ");  
                                        break;  
                                case '\t':  
                                        printf("\\t  ");  
                                        break;  
                                default:  
                                        printf("%0X  ",(char)i);  
                                        break;  
                        }  
                        printf("  %-4d  %16s\n",freq[i],code[i]);  
                }  
        }  
        printf("----------------------------\n");  
}

int main(int argc, char\* argv[]){  
        FILE \*fp\_in, \*fp\_out;  
        char file\_name[50]={0};  
        unsigned int freq[128] = {0},i;  
        int shmid;  
        char \*shm\_ptr;  
        shmid=shmget((key\_t)5900,sizeof(char),0666|IPC\_CREAT);  
        printf("Key of shared memory shmid1:%d\n",shmid);  
        if (shmid< 0) {  
        printf("ERROR: Failed to create shared memory segment.\n");  
        return 0;  
    }  
        shm\_ptr = (char\*)shmat(shmid, NULL, 0);  
        printf("Please enter the file to be compressed\t: ");  
        scanf("%s",file\_name);  
        shm\_ptr=file\_name;  
        printf("Process attached at %p\n",shm\_ptr);  
        printf("Text file sent to shared memory:%d\n\n",(char \*)shm\_ptr);  
       if (shm\_ptr == (char\*)-1) {  
        printf("ERROR: Failed to attach shared memory segment.\n");  
        return 0;  
    }  
        if((fp\_in = fopen(file\_name,"r"))==NULL){  
                printf("\nERROR: No such file\n");  
                return 0;  
        }  
        import\_file(fp\_in,freq);  
        print\_code(freq);  
        strcat(file\_name,".huffman");  
        fp\_out = fopen(file\_name,"w");  
        encode(fp\_in,fp\_out,freq);  
        fclose(fp\_in);  
        fclose(fp\_out);  
        strcat(file\_name,".table");  
        fp\_out = fopen(file\_name,"w");  
        for(i=0; i<128; i++){  
                fprintf(fp\_out,"%c",(char)freq[i]);  
        }  
        for(i=0; i<128; i++)    input\_data += freq[i];  
        fclose(fp\_out);  
        printf("\nInput Text file size:\t\t%dkb\n",input\_data);  
        output\_data = (output\_data%8)? (output\_data/8)+1 : (output\_data/8);  
        printf("Compressed text file size:\t\t%dkb\n",output\_data);  
  
        printf("\nCompression ratio:\t%.2f%%\n\n\n",((double)(input\_data-output\_data)/input\_data)\*100);  
        return 0;  
}

**// DECODING COMPRESSION (Client)**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <sys/shm.h>  
#include <unistd.h>  
  
typedef struct node\_t {  
        struct node\_t \*left, \*right;  
        int freq;  
        char c;  
} \*node;  
int n\_nodes = 0, qend = 1;  
struct node\_t pool[256] = {{0}};  
node qqq[255], \*q = qqq-1;  
char buf[1024];

void import\_table(FILE \*fp\_table, unsigned int \*freq){  
        char c;  
        int i = 0;  
        while((c=fgetc(fp\_table))!=EOF){  
                freq[i++] = (unsigned char)c;  
        }  
        for (i = 0; i< 128; i++)  
                if (freq[i]) qinsert(new\_node(freq[i], i, 0, 0));  
        while (qend> 2)  
                qinsert(new\_node(0, 0, qremove(), qremove()));  
}

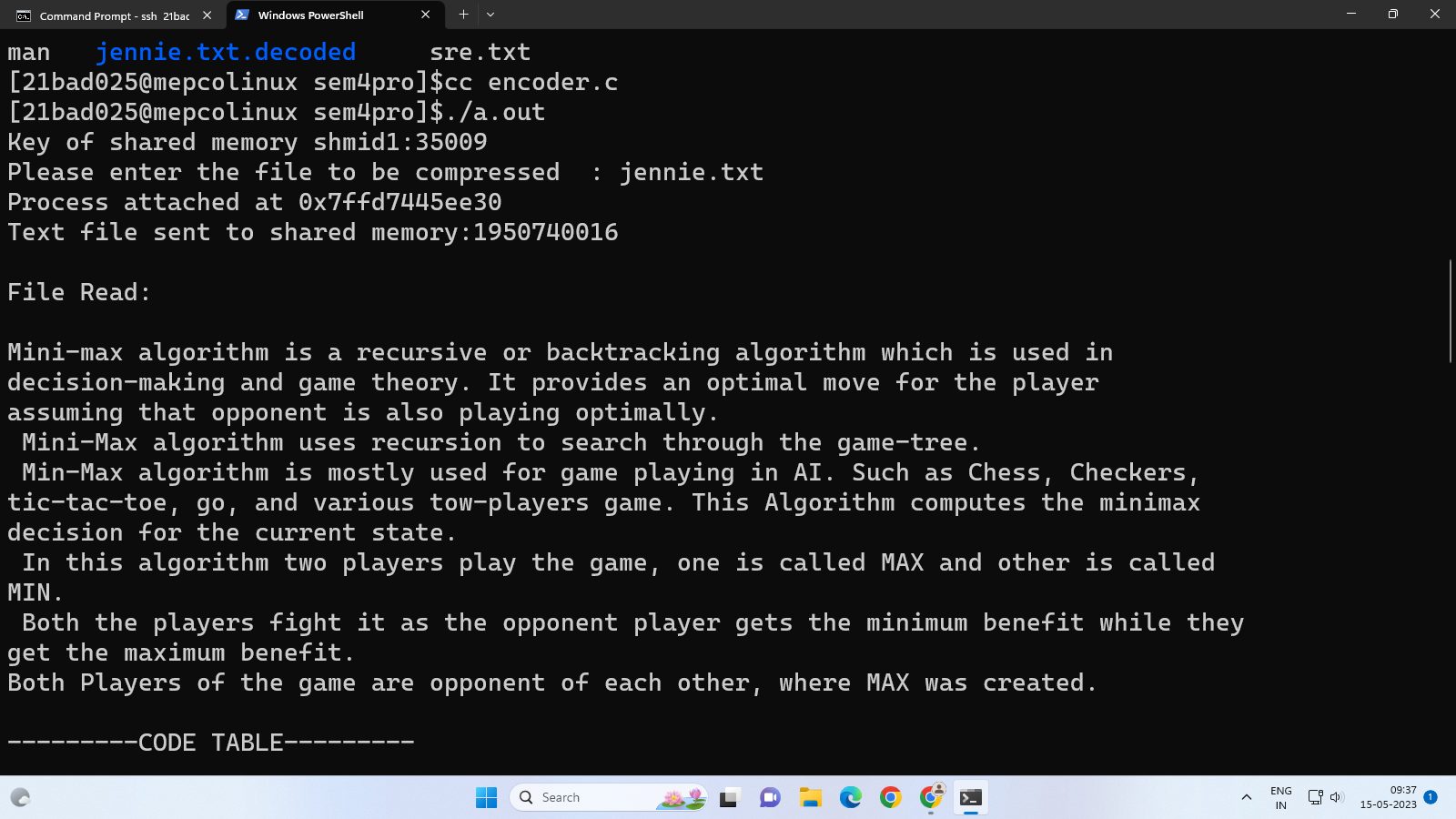
void decode(FILE \*fp\_huffman,FILE \*fp\_out)

{  
        int i=0,lim=0,j=0;  
        char c;  
        node n = q[1];  
        fscanf(fp\_huffman,"%d",&lim);  
        fseek(fp\_huffman,1,SEEK\_CUR);  
        printf("Decoded : \n");  
        for(i=0; i<lim; i++){  
                if(j==0)  
                        c = fgetc(fp\_huffman);  
                if(c&128)       n = n->right;  
                else            n = n->left;  
                if(n->c){  
                        putchar(n->c);  
                        fputc(n->c,fp\_out);  
                        n = q[1];  
                }  
                c = c<<1;  
                if(++j>7)  
                        j = 0;  
        }  
        putchar('\n');  
        if (q[1] != n) printf("garbage input\n");  
}  
int main(int argc, char\* argv[]){  
        FILE \*fp\_table,\*fp\_huffman,\*fp\_out;  
        char file\_name[50]={0},temp[50]={0};  
        unsigned int freq[128] = {0};  
        int shmid;  
        char \*shm\_ptr;  
        shmid=shmget((key\_t)5900,sizeof(char),0666|IPC\_CREAT);  
        printf("Key of shared memory shmid1:%d\n",shmid);  
        shm\_ptr = (char\*)shmat(shmid, NULL, 0);  
        printf("Please enter the file to be compressed\t: ");  
        scanf("%s",file\_name);  
        printf("Process attached at %p\n",shm\_ptr);  
        printf("Text file recieved from  shared memory:%d\n\n",(char \*)shm\_ptr);  
       if (shm\_ptr == (char\*)-1) {  
        printf("ERROR: Failed to attach shared memory segment.\n");  
        return 0;  
    }  
        if (shmid< 0) {  
        printf("ERROR: Failed to create shared memory segment.\n");  
        return 0;  
    }  
        system("clear");  
        if( argc == 2 ) {  
                strcpy(file\_name,argv[1]);  
                if(strstr(file\_name,"huffman") == NULL){  
                        printf("\nERROR:wrong file format!\n");  
                        return 0;  
                }  
        }  
        else if( argc> 2 ) {  
                printf("Too many arguments supplied.\n");  
        }  
        else {  
                if(strstr(file\_name,"huffman") == NULL){  
                        printf("\nERROR:wrong file format!\n");  
                        return 0;  
                }  
        }  
        if((fp\_huffman = fopen(file\_name,"r"))==NULL){  
                printf("\nERROR: No such file\n");  
                return 0;  
        }  
        strcat(file\_name,".table");  
        if((fp\_table = fopen(file\_name,"r"))==NULL){  
                printf("\nERROR: Frequency table cannot be found\n");  
                return 0;  
        }  
        import\_table(fp\_table,freq);  
        \*strstr(file\_name,".huffman") = '\0';  
        strcpy(temp,"mkdir ");  
        strcat(temp,file\_name);  
        system(strcat(temp,".decoded"));  
        strcpy(temp,"./");  
        strcat(temp,file\_name);  
        strcat(temp,".decoded/");  
        if((fp\_out = fopen(strcat(temp,file\_name),"w"))==NULL){  
                printf("ERROR:Creating decoded file failed\n");  
                return 0;  
        }  
        decode(fp\_huffman,fp\_out);

        fclose(fp\_huffman);  
        fclose(fp\_table);  
        fclose(fp\_out);  
        return 0;  
}

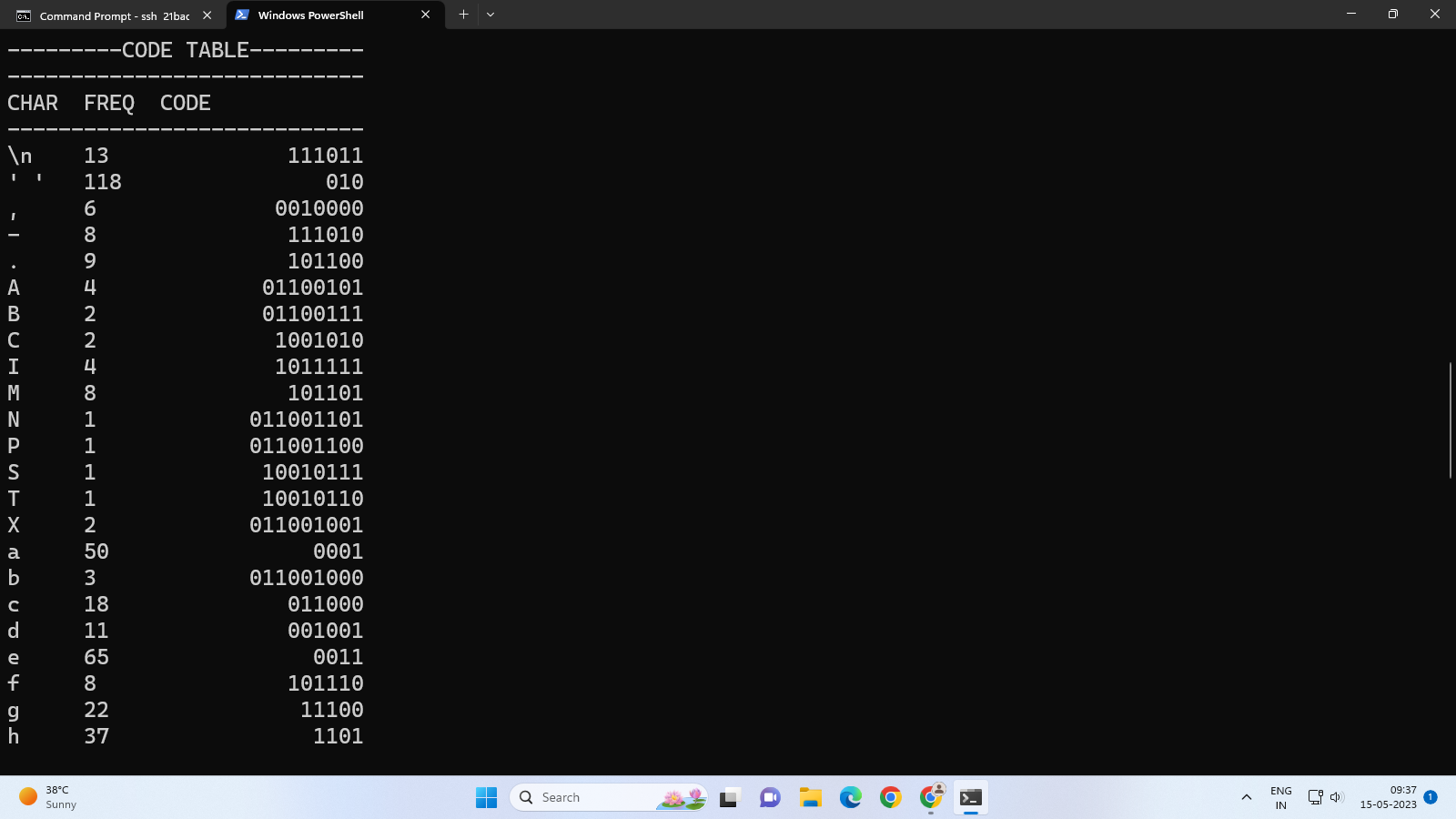
**2.2 OUTPUT**

**// text file used to compress**

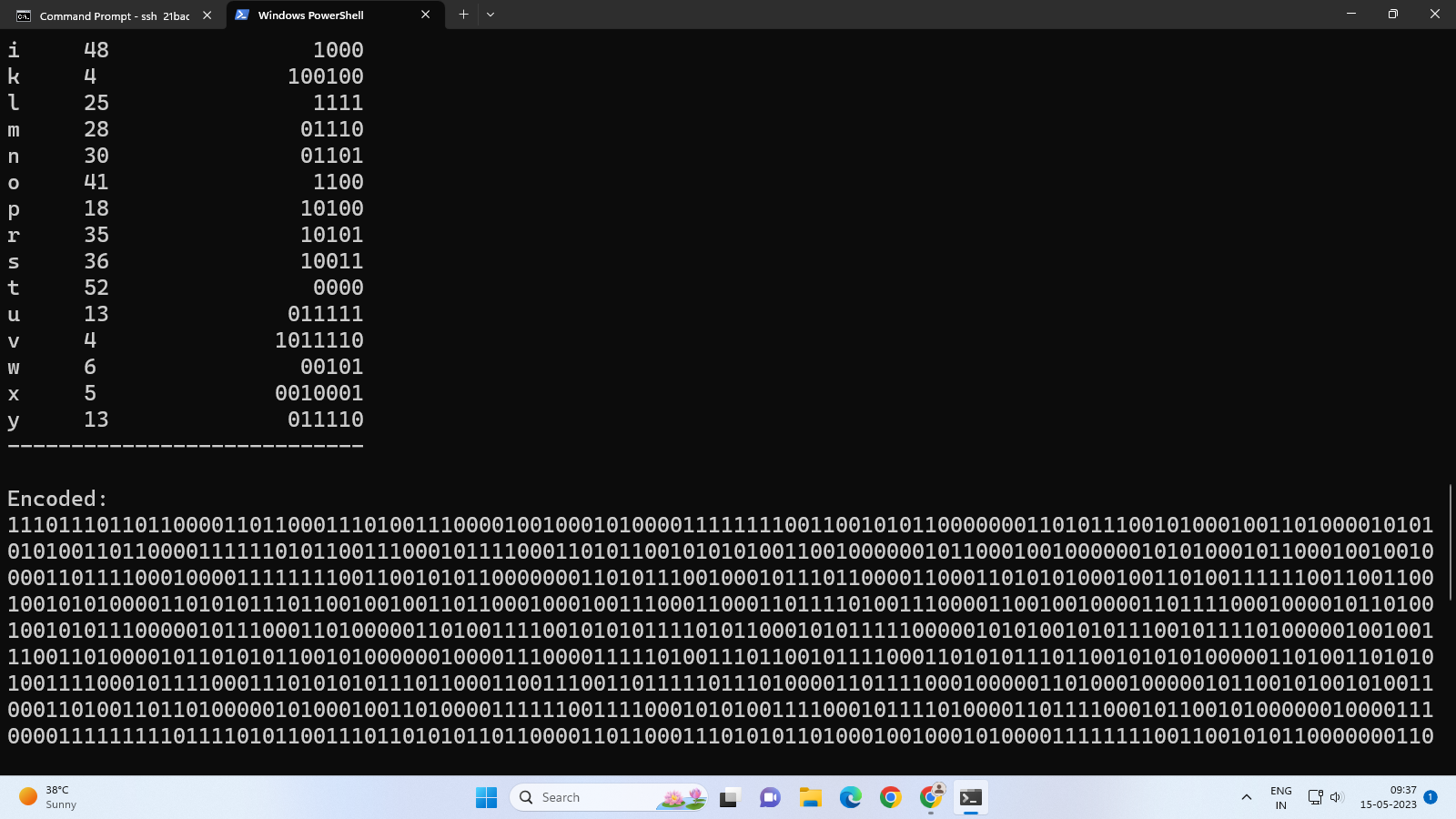


**Figure 2.1 Actual Text File**

**// Huffman Frequency Table**

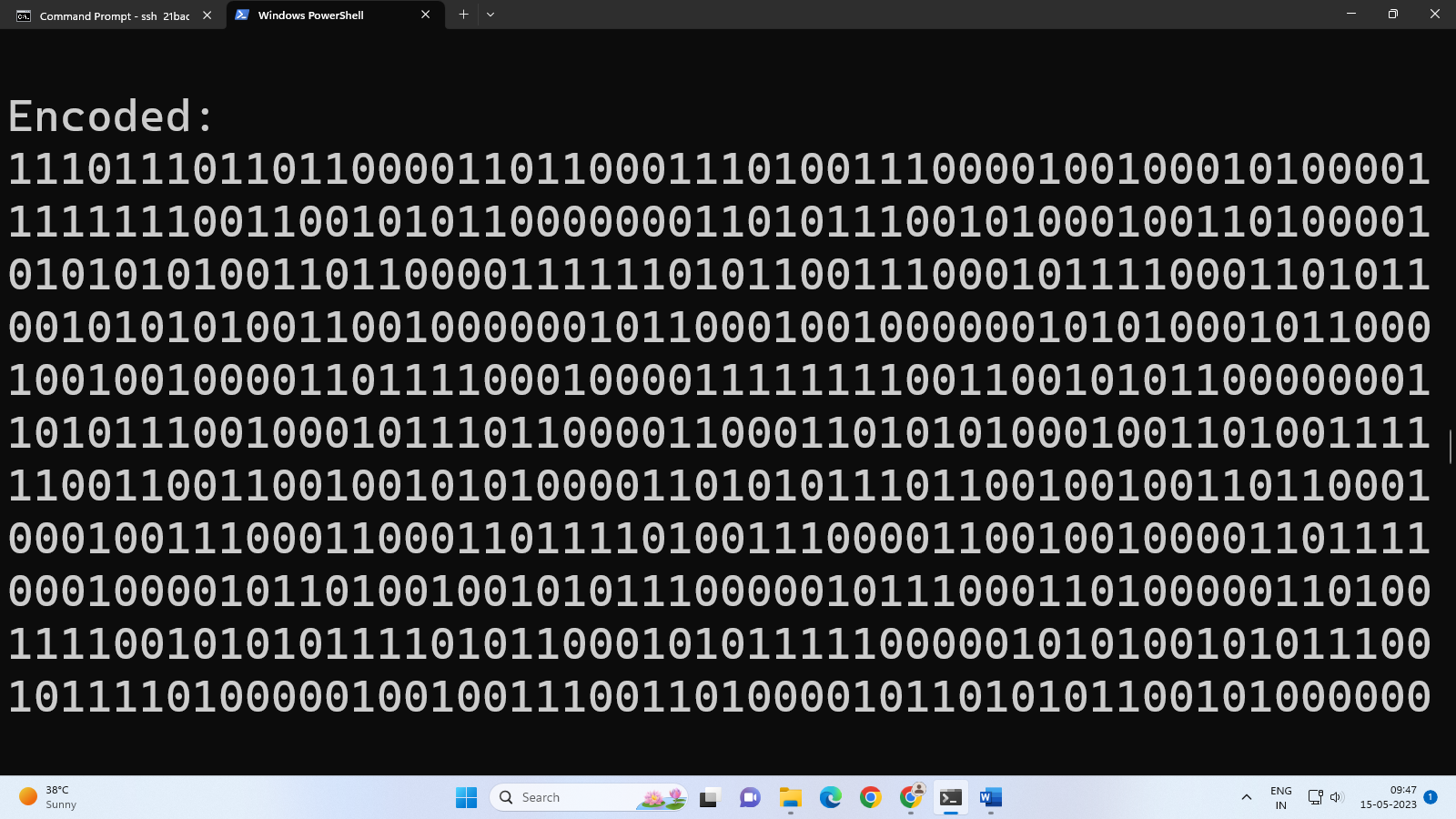


**Figure 2.2 Frequency Table 1**

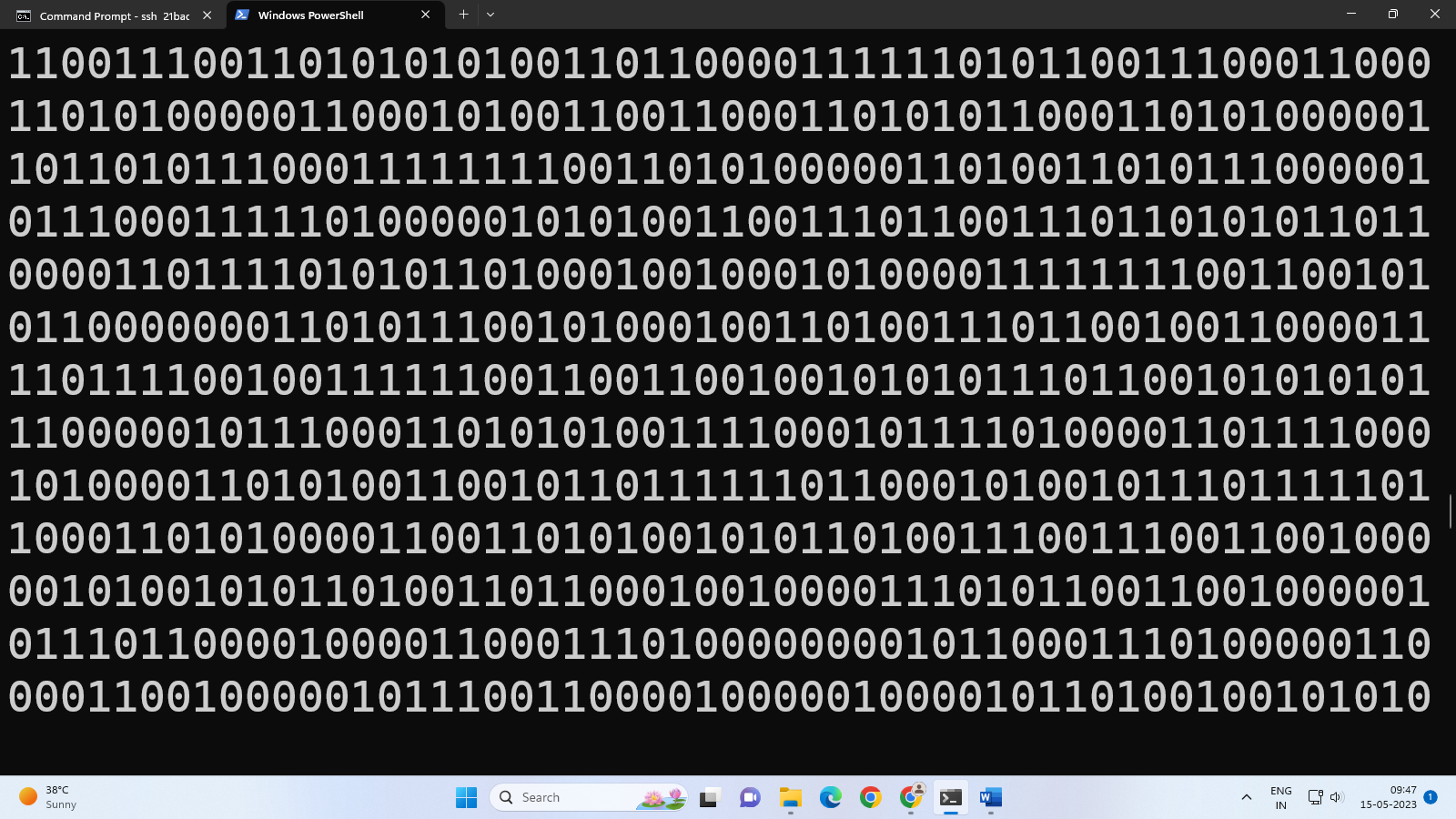


**Figure 2.3 Frequency Table 2**

**// Encoded file**

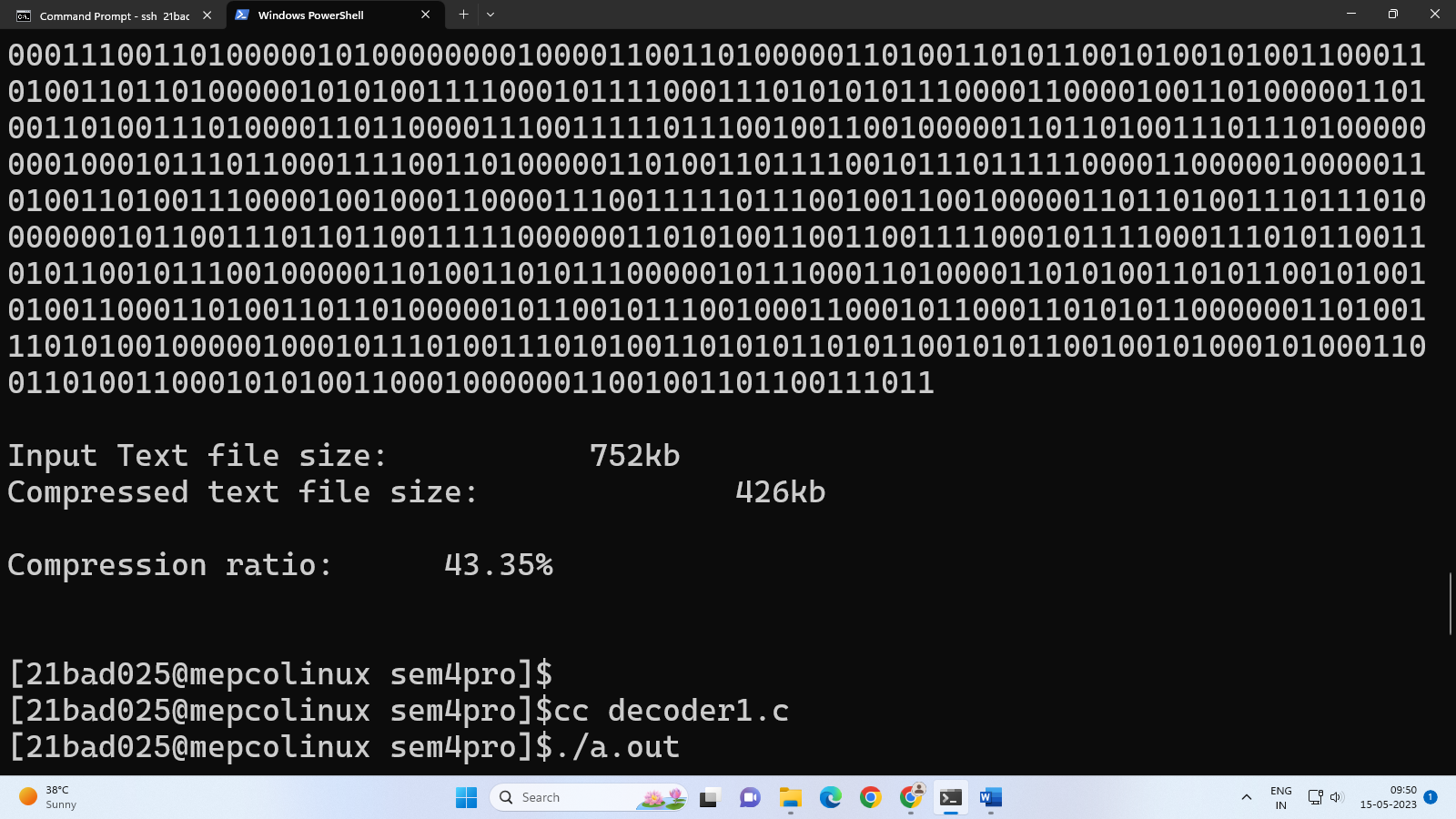


**Figure 2.4 Encoded Output 1**



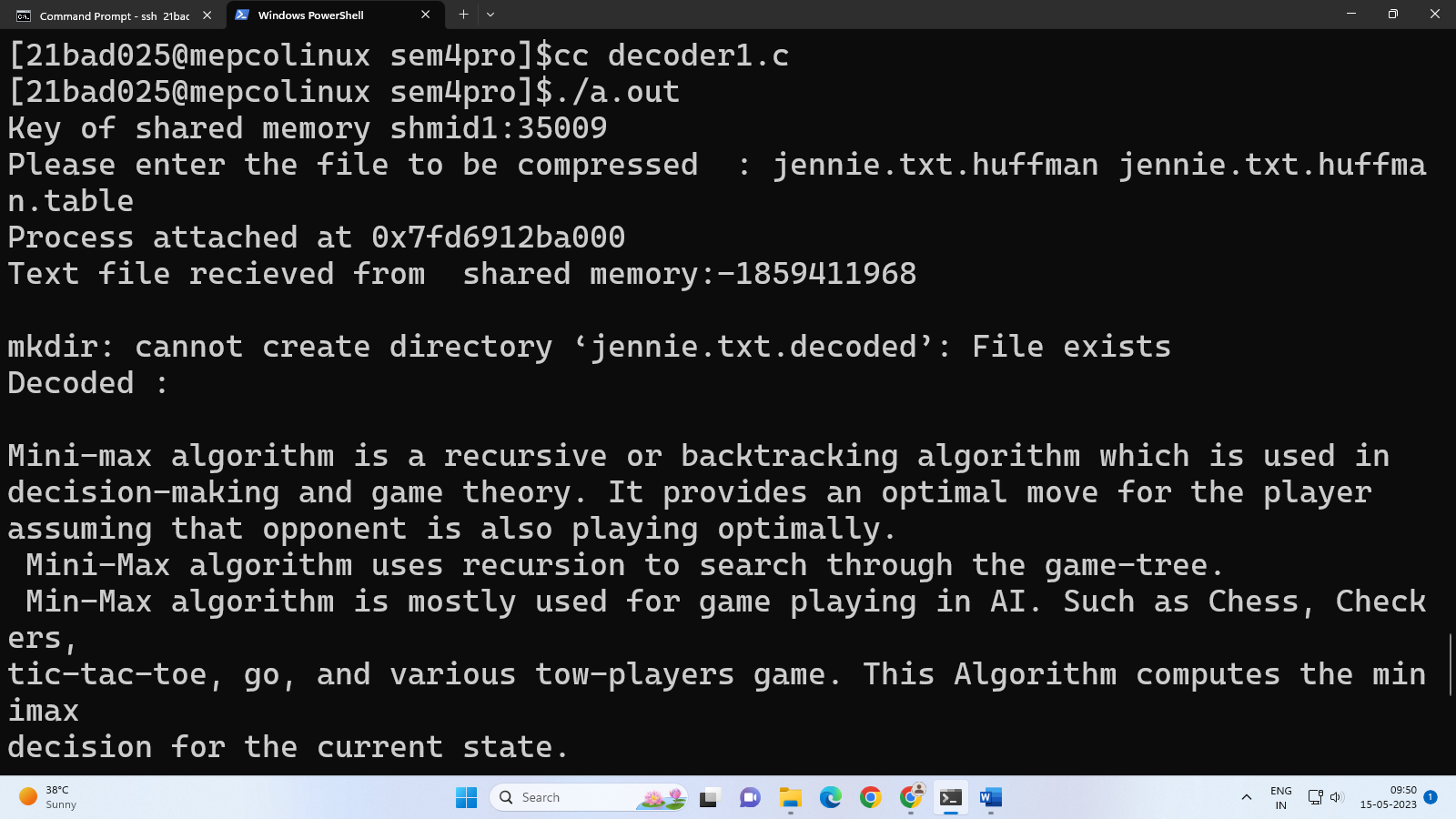
**Figure 2.5 Encoded Output 2**

**//Compression ratio**



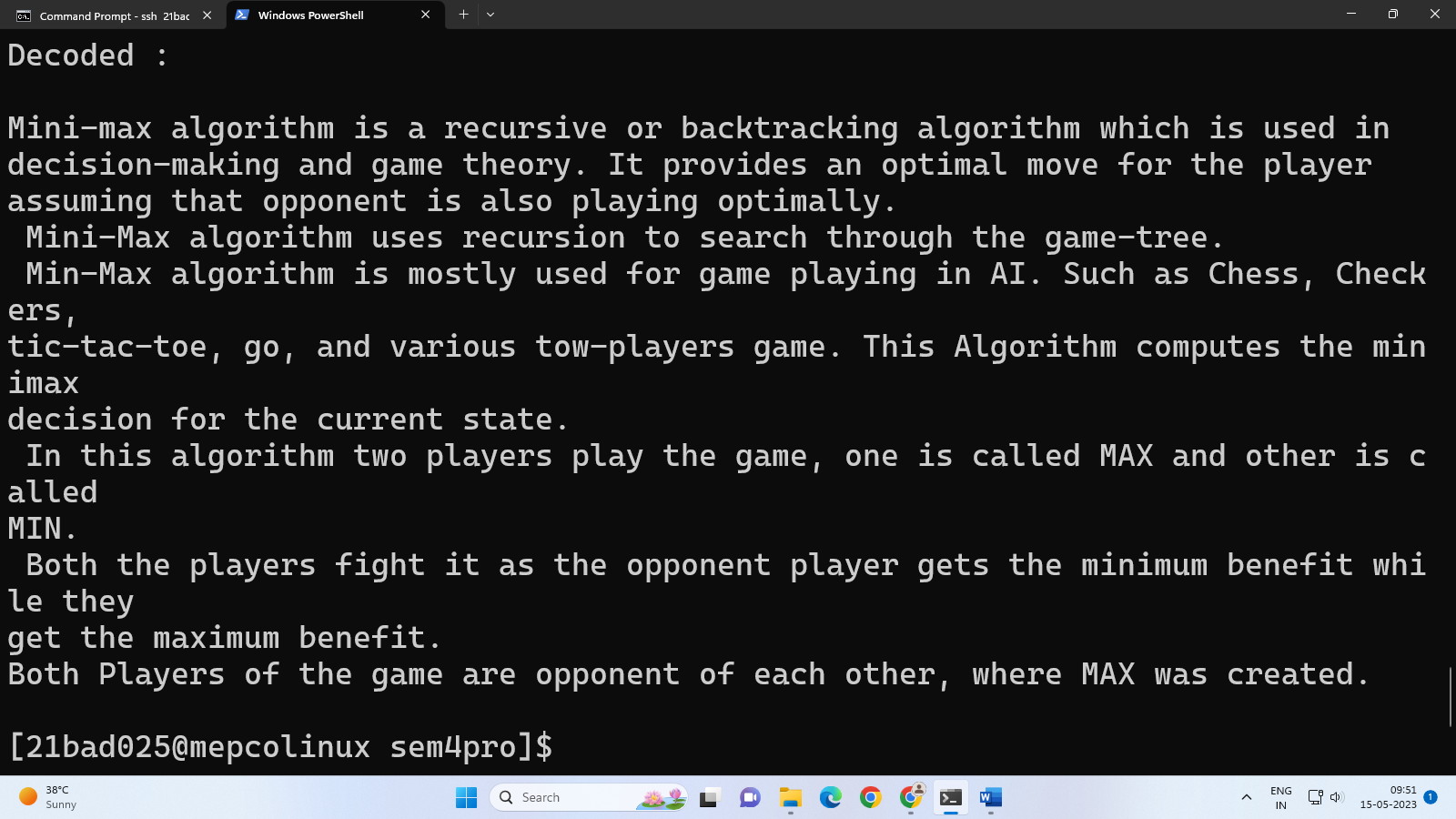
**Figure 2.6 Compression ratio**

**//Decoded output**



**Figure 2.7 Decoded output**

**//Decoded Text file**



**Figure 2.8 Decoded Text File**

**CHAPTER 3**

**CONCLUSION**

From this project, we can easily reduce the large size file into a file of smaller size using compression and send them easily over the network. We can also decompress the compressed file to get the original file and the receiver’s end. File compression lets you pack more data into a given amount of storage space. In addition to saving space on hard drives and other media, compression can dramatically improve the speed of file downloads.

The act of compressing a file makes it unreadable to most programs until the file is uncompressed. A file reduced in size through the application of a compression algorithm, commonly performed to save disk space. The act of compressing a file will make it unreadable to most programs until the file is uncompressed. By compressing a file, data takes up less space, and files can be sent and received a lot more quickly.

Many files, such as log files, can not only begin to clog up directories, but they can also take up too much space.

To save storage space, it is very useful to compress the files and folders on your desktop and transfer them quickly. You can create a zip file easily via File Explorer and unzip it at any time. Besides that, making a self-extracting archive like an EXE file is possible.

**REFERENCES**

1. https://byjus.com/gate/huffman-coding-notes/

2. https://stackoverflow.com/questions/23255210/simple-file-compression-in-c

3. https://www.tutorialspoint.com/what-is-shared-memory-in-the-os

4. https://www.studytonight.com/data-structures/huffman-coding

5. https://www.technipages.com/what-happens-when-you-compress-a-file/

6. https://www.easeus.com/knowledge-center/what-is-file-compression.html

7. https://www.geeksforgeeks.org/ipc-shared-memory/

8. https://www.codeguru.com/dotnet/compressing-and-decompressing-files-with-c/