Detailed Report

Ultimate File Compression Tool

University of Engineering and Technology, Lahore

Analysis of Algorithms

Group: CS311S20PID01

2018-CS-02

Muhammad Qasim (A)

2018-CS-90

Bilal Ashraf (B)

To:

Prof. Samyan Qayyum Wahla

**Letter of Transmittal**

August 16, 2020

Prof. Samyan Qayyum Wahla

University of Engineering and Technology, Lahore

Respected Prof.

Our semester project for the course of Analysis of Algorithms, has come to the stage of completion. We are submitting this report on our project **“Ultimate Compression Tool”** as per your guidance. This report contains:

* The general overview of project
* The compression algorithm/technique used
* Complexity and correctness of the algorithm
* The information of the backend codebase
* The detailed overview of the GUI
* The issues reported and their solution.
* The information about the contribution of each member

We believe that this report will meet the requirements defined.

Regards

2018-CS-02

Muhammad Qasim

2018-CS-90

Bilal Ashraf

**Acknowledgement**

I am very glad to complete this project and report according to the requirements defined. I would like to thank my fellow group member, who helped me alot to complete this project and report. Mostly the guidance about the project is taken from the Internet.

At the end, I would like to pay my gratitude to our teacher, Prof. Samyan Qayyum Wahla, because without his proper guidance, this project was impossible.

2018-CS-02

Muhammad Qasim

2018-CS-90

Bilal Ashraf

**Table of Contents**

Table of Contents Page No.

1. Letter of Transmittal 1
2. Acknowledgement 2
3. Overview 4
   1. Business Case 4
   2. End User of Product 4
   3. Motivation for Projects 5
4. Description of Project Description 6
   1. Impacts 6
   2. Functional Requirements 6
5. Benefits 7
6. Implementation Details 8
   1. Link to GitHub repository 8
   2. Total Commits 8
   3. Contribution of each member 9
   4. Details of commits 9
7. General Overview of Software 10
   1. Format of Input 10
   2. Validations 10
   3. Format of Output 10
   4. Deployment 10
8. Details of Algorithm 11
   1. Pseudo Code 11
   2. Description 12
   3. Complexity 12
   4. Correctness 12
9. Interface of Project 15
   1. Home Window 15
   2. Compression 16
   3. Decompression 19
   4. Help 21
10. Integration 22
11. Change Requests 22
12. Testing 23
13. Technology 24
    1. Programming Language 24
    2. Platform 24

***Ultimate File Compression Tool***

***Project* Executive Summary**

**Overview**

This project is about file compression and decompression. Some files in our computer have huge size which can cause storage problems. So, what can we do about this problem? We need some kind of tool that can minimize the sizes of these files. Here comes the concept of file compression. File compression tool can minimize the size of a file up to 70% without any loss of data. Therefore, it is also known as***lossless data compression.*** *T*he purpose is not just to minimize the size but also to transmit the files easier and faster to other locations over a network or the internet.

In this project, we will implement file compression using***Huffman Encoding***algorithm. Basically, this algorithm works in a way that it reads the whole file, recognizing similar data and replacing these by unique binary codes. There are several different approaches used for file compression, but Huffman encoding is the best way to compress files because it ensures lossless data compression. The length of binary code depends upon the frequency of relative character. Therefore, the least frequency characters are assigned a long bit of code while the characters with highest frequency are assigned a short bit of code. The binary code for each character is unique because they are prefix-free (no code is a prefix of another). Decompressing files just requires the reverse procedure of compression i.e. reading compressed file, decoding binary code streams and getting the original file without loss of any data.

**Business Case**

Data is always growing exponentially, as compared to the physical devices used to store the actual data. For efficient storage of large amounts of data into some less capacity physical devices, the data needs to be compressed or converted to some smaller form with no loss of actual data. The Data Compression tool uses some advanced compression techniques to convert the actual data into encrypted form, which can be restored to the original form using the decompression technique.

**End user of the product**

Any **common computer user** is the end user of this software, it's too simple to use with guided Graphical User Interface. Educational as well as the industry can use this File Compression Tool for efficient data compression.

The application GUI is self-explanatory, still there is a user manual for users who will face a problem while using software.

**Motivation for Project**

The choice of project is not an easy task, we had 3 choices for the projects, all are best and demanded in the real world. What's the motivation behind choosing the File Compression Tool?

* Efficient data storage is in more demand than any other data processing software. So, more data is available.
* Other software facilities are data dependent, so having accurate and large amounts of data is more necessary. So other facilities have more data available.
* Compression exactly needs computing power; no mind power can efficiently do compression tasks.
  + Activity Scheduling can easily be done by keeping various activities in view using Mind power.
  + Plagiarism Checking can be done by thoroughly reading the copied documents and the available helper document.
  + Generating alternate keys for each character depending upon their frequency and using them to generate a compressed version of a file is pretty difficult for a human mind.
* Reduced size of data can speed up our processing and save processing time used to do actions on original uncompressed files. So, time is saved.

**Description of the project objectives:**

The ease of **data storage** is the driving objective behind the development of this Compression Tool Development. Some of other major objectives are listed below:

* Minimize file size to release up more disk space for other data storage.
* Easier, faster data processing
* Saved processing time.
* Data Security by compression.

**State the level of impact expected should the project proceed and implications of not proceeding.**

A lot of benefits are abstracted in just a layer of data compression. From physical storage to CPU efficiency, from more security to better security there are many impacts on operational, strategic and performance level. That impacts vary form Implementation of file compression tool provides us with more

* More physical storage facility
  + By reducing file size at binary level
* Less processing time required
  + Smaller data size needs less time to get actions done on it.
* Improved security of data
  + Data is hidden under a layer of abstraction, so data is secure.
* Encryption and Compression at same level
  + Using variable binary codes for each character gives a sort of encryption and compression of original data at same time.

**Functional Requirements**

File compression tool has some dependencies to run, the system will have **C#** execution ability and **Visual Studio Community** Version to perform its execution.

This **v1.0beta** is just made for **‘.txt’** file compression. An input path of the text file is given to the software, it generates the output compressed **‘.bin’** file at the same location along with the original file.

For decompression, the path of the compressed file is given to the software, it will decompress the compressed **.bin** file to again **.txt** file and save at its corresponding location along the compressed file.

Text files having extension **‘.txt’** compressed into **‘.bin’** file and the same procedure will be applied for decompression while **‘.bin’** file is given as input which provides **‘.txt’** file as output.

**Benefits**

The expected benefits of the project are

* Users with an academic background always require some space, so it can provide more disk space for their work.
* Industries dealing in a bigger amount of data can achieve more space for larger data files and can easily transmit data.

**Implementation Details**

**Link to GitHub Repository**

Here’s the link to the project private repository.

[**https://github.com/mrqasimasif/CS311S20PID01**](https://github.com/mrqasimasif/CS311S20PID01)

**Total commits**

Total commits of the project repository are **41**.

Muhammad Qasim: 2018-CS-02 ( **30** commits )

Bilal Ashraf: 2018-CS-90 ( **8** commits)

Abdul Samad-Testing: 2018-CS-110 ( **3** commits )

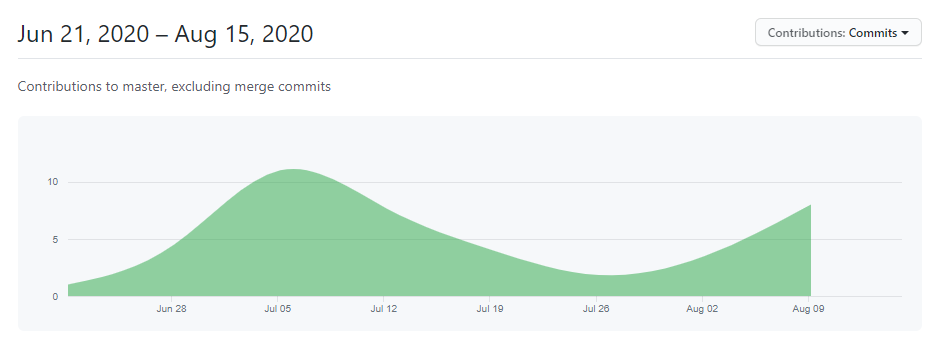
**Exact contribution of each member**

Contribution of both members is nearly equal. From backend code design to Front end UI development, both members have played their equal roles in it. The number of commits may vary but the contribution of each member is nearly same. Just Qasim pushed after each small change and Bilal pushed after complete development.

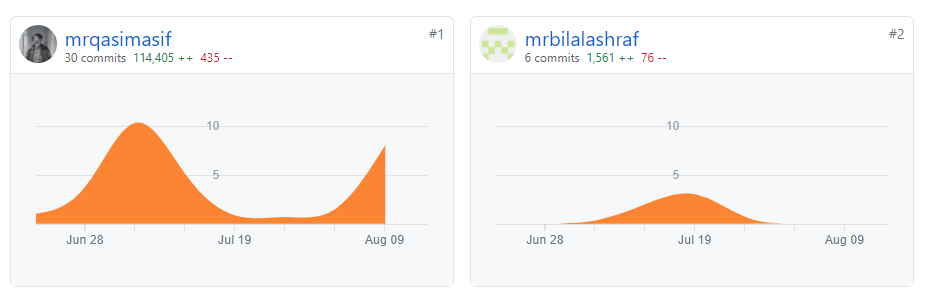
Mostly development was on our local desktop computer and pushed to remote GitHub after completion of a phase. So, the exact number of commits is generally less. The project has C# backend code base with configured modern C# UI design.

**Detailed chart of commits on GitHub**

The Contribution Chart of the repository shows the continuous work of both members on the project.



On individual basis, the contribution chart is nearly equal for both group members, the contribution does not mean in terms of commits, but in terms of the actual work done. 2018-CS-02 used to commit frequently but 2018-CS-90 push the commit after completion of some task. Overall, the collaboration of both members is nearly equal.



**2018-CS-02 50% 2018-CS-90 50%**

**Have you used build in algorithms, or you have implemented yourself?**

Project is developed and evaluated by own efforts, from Huffman Coding program and unique UI design in C#, all is possible due to the combined effort of both group members.

**Format of Input**

For **compression**, the input file should have **.txt** extension and you can easily access .txt files on your system by clicking on the **browse** option that we have provided in application.

For **decompression**, the input file should have **.bin** extension and you can easily access .bin files on your system by following the same procedure described above.

**Validations**

On compress window, the browse option shows only **.txt** files in the file explorer to select files. And the decompression browse option enables only compressed **.bin** files to be selected.

Moreover, it shows an error message if compress or decompress button is pressed without selecting a file.

**Format of output**

The whole software deals with files. Compression process produces a compressed file with name **“original name-compressed.bin”** while the output of decompression is a decompressed file named **“original name-decompressed.txt”.**

**Deployment**

We deployed this project on overall three releases while we cover all the milestones of this project.

1. **v1.0-beta:** In this release, we covered both the implementation of backend and frontend. We developed backend with **Python** programminglanguage and for frontend (UI), we used **C# (.NET Framework) windows forms application**. We also covered pseudo code, time complexity and correctness of **Huffman Coding** algorithm.
2. **v2.0:** In this release, we replaced backend Python code with **C#** (**.NET Framework**). The reason for this replacement is described in the **Change Requests** section. We also integrated both the frontend and backend. Project is now in proper working state.
3. **v2.1:** We completed both the project and project report in this release. Project report is finalized now with all the details of the project and overall commits and changes that we had done in the remote repository on GitHub.

**Details of algorithms**

This project is based on the Huffman Coding algorithm. Pseudo code of the algorithm is given below.

**Pseudo Code**

**compress( X ):**

file\_size = X.size

keys\_queue = priority\_queue()

for i = 1 to file\_size:

value = node( X [ i ] )

keys\_queue.enqueue(value)

end for

while keys\_queue.size is not equal to 1

new\_node = node()

new\_node.left = a = keys\_queue.dequeue()

new\_node.right = b = keys\_queue.dequeue()

new\_node.freq = a.freq + b.freq

keys\_queue.enqueue(new\_node)

end while

return keys\_queue

**decompress ( root, bit\_stream ):**

stream\_size = bit\_stream.size

for i = 1 to stream\_size

current\_node = root

while current\_node.left != NULL and current\_node.right != NULL

If bit\_stream[i] = ‘0’

current\_node = current\_node.left

else

current\_node = current\_node.right

endif

i=i+1

endwhile

print(current\_node.dchar)

endfor

**Description**

**Compressing** takes an input character array. Adds each element to priority queue depending upon their decreasing frequency and a binary tree is created for characters that assign them a new encoding Symbol. The tree is traversed bottom-up in compression.

**Decompressing** takes the root node and the compressed bitstream and traverses the tree from a top-down manner to find the corresponding symbol for each compressed character to find the original file.

**Time Complexity**

This algorithm runs in **O(nlogn)** time: Putting the ﬁrst **n** pairs into a variable i.e. valueusing an array **X**,and creating the **n** leaves takes **O(n)** time, and turning value into a heap using **priority\_queue()** also takes **O(n)** time. The for loop is repeated **n−1** times. In each iteration we perform two **dequeue()** operations and one **enqueue()** operation, each of which takes **O(logn)** time. So, the loop takes **O(nlogn)** time, and the entire algorithm takes **O(n) + O(n) + O(nlogn) = O(nlogn)** time.

**Correctness**

An algorithm is said to be correct if:

● For every input, it produces a meaningful output.

● It halts after each execution cycle.

We prove the correctness of the algorithm by the number of characters **n** in the alphabet. The base case is **n = 2**, because there is only one possibility of the solution that both codes are 1 digit long, which is exactly optimal. We need to show for exactly **n** number of characters. Suppose that the algorithm produces an optimal tree for alphabets with **n − 1 ≥ 2** symbols and their associated frequencies. We will prove that it produces an optimal tree for alphabets with n symbols and their associated frequencies.

Let **T** be any prefix code tree with two siblings b and c at the maximum depth of tree with smallest frequencies. Two more alphabets with small frequencies are x and y too.

● Assume **p(b) <= p(c)** and **p(x) <= p(y).**

Since x and y are the smallest frequencies, it follows as:

● **p(x) <= p(b)** and **p(y) <= p(c)** may be the same.

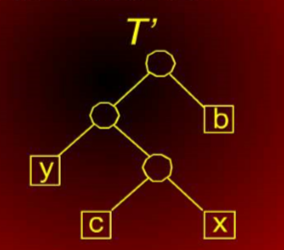
The nodes b and c are at the deepest level of the tree, x and y be one level up in tree.

● **d(b) >= d(x)** and **d(c)>=d(y),** **d** be the depth of the tree.

● **p(b) - p(x) >=0** and **d(b) - d(x) >=0**

● **B(T) = (** **p(b) - p(x) . d(b) - d(x) ) >=0**

Hence their cost in the form of product is non-negative. Tree construction **T** is complete now.



Now swap the position of **x** and **b** in the tree. The result is a new **T’** tree. The cost of the tree is **B(T’).**

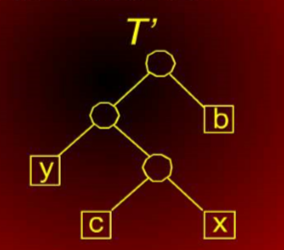
● **p(x) <= p(c)** and **p(b) <= p(y).**

● **p(b) <= p(x)** and **p(y) <= p(c).**

● **d(x) >= d(b)** and **d(c)>=d(y),** **d** be the depth of the tree.

● **B(T’) = (** **p(x) - p(b) . d(x) - d(b) ) >= 0**

Tree construction of the **T'** tree is complete. By interchanging the nodes, the cost of tree **T’** doesn't change, showing that **T’** tree is optimal.



The cost of both trees **T and T’** is nearly the same. This proves the algorithm is **correct** and

produces the optimal result in every case.

**Interfaces for your project**

An interactive GUI is created in C# with 3 main options:

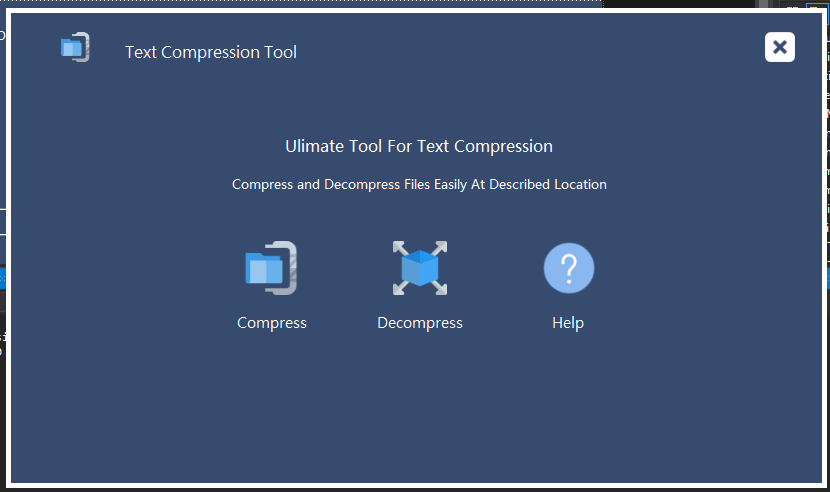
* Compress
* Decompress
* Help

Close button is at the top right. The UI windows have the facility to be dragged anywhere on the screen.

**1. Home Window**

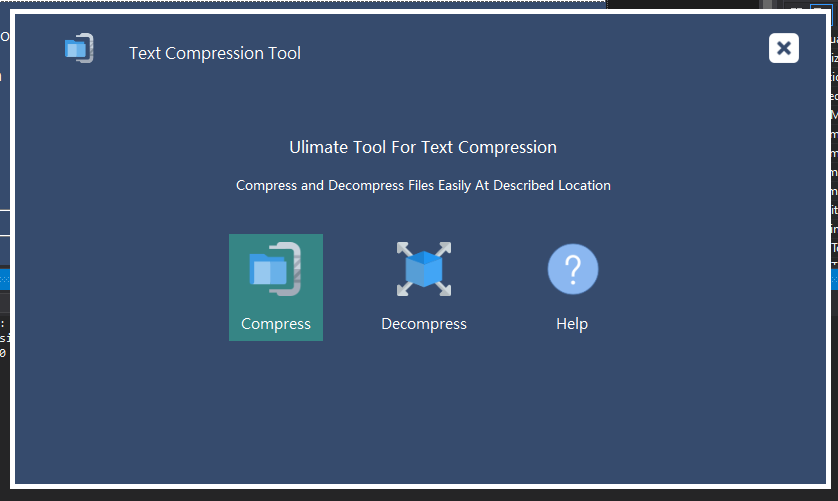
Home window contains selectable buttons that are COMPRESS, DECOMPRESS and HELP that are interactive hover buttons, that change its color on mouse hover.

The Compress and Decompress Windows have a return button at the top left corner to move back to the original Home window.

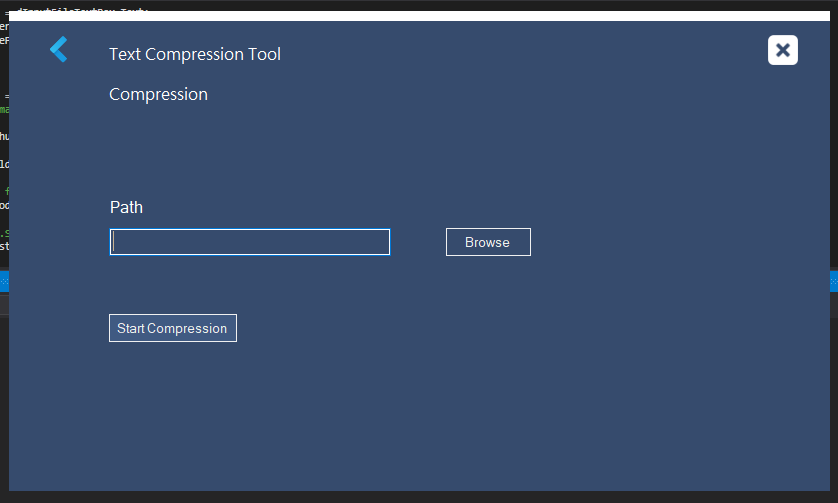


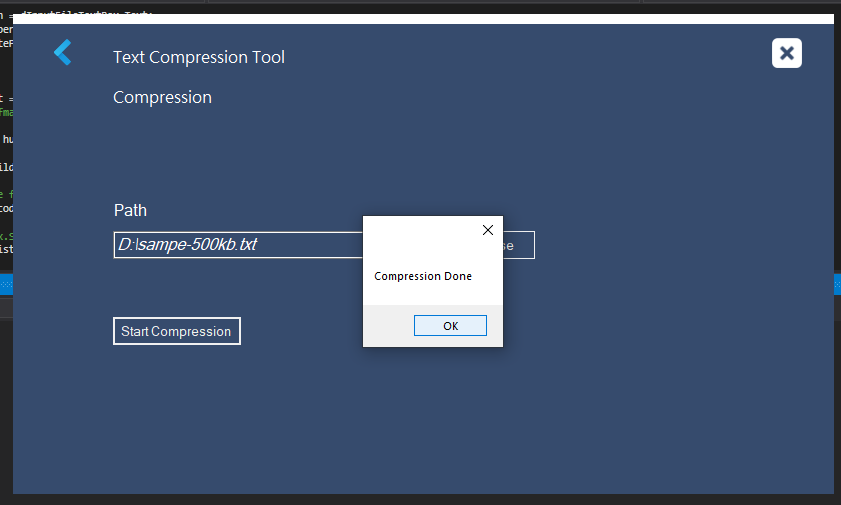
**2. Compression**

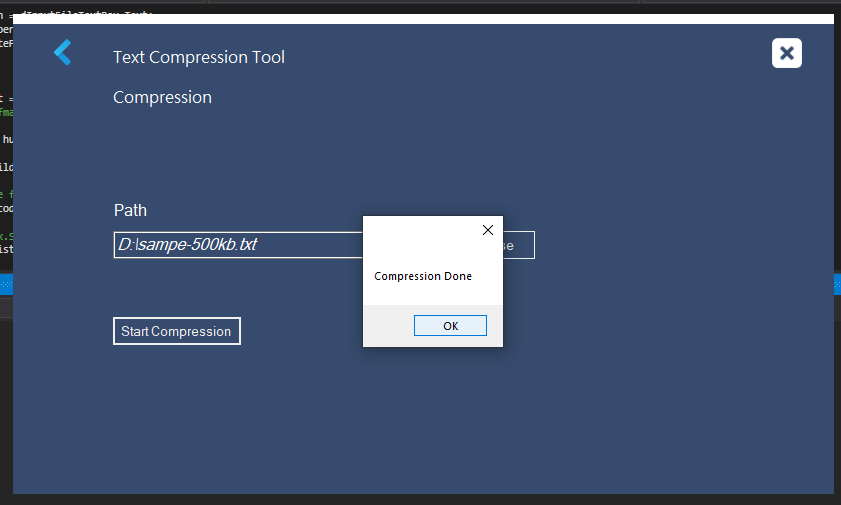
First of all, the file should be compressed, on clicking the Compress button, a new compression window will appear.

**2.1)** A new window with Browse option Appears. The valid path can be typed as input or file can be selected through browse.

**2.2)** Exception handling is done throughout the project application. If the file path is wrong or file is not selected and compression is started, Message box alert pops up.



**2.3)** After the right path is selected, compression can be started.

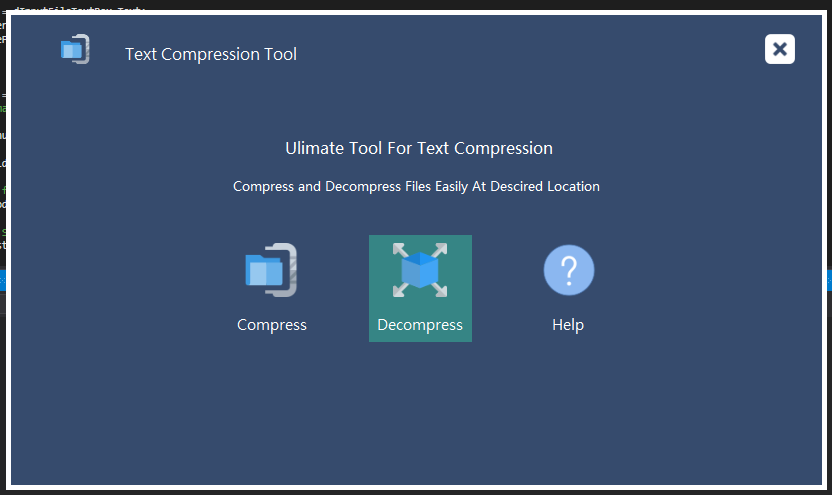
**2.4)** Compression starts and a completion message pops up when done.

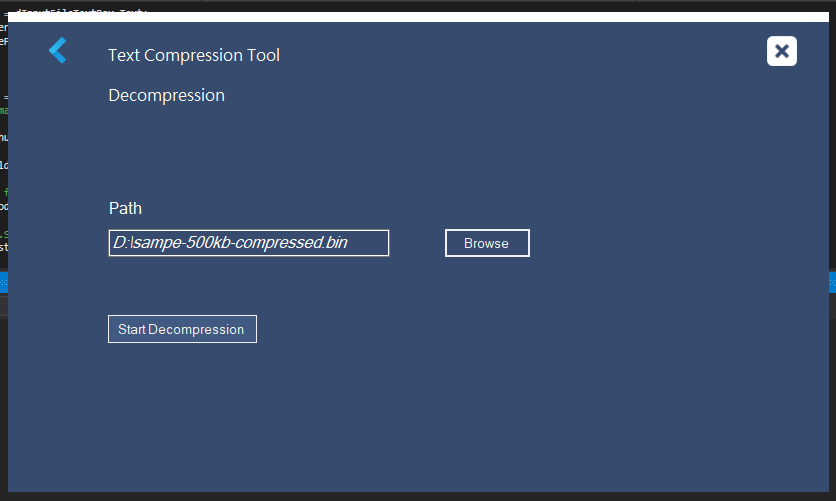
The compressed file is stored at the described location along with the original file, with the name as **“original file name-compressed.bin”.**

**3. Decompression**

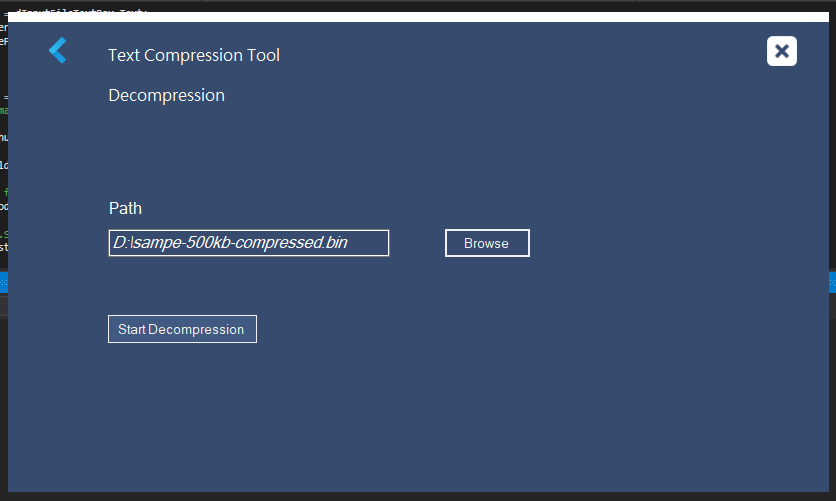
The 2nd home page button is Decompression. Same procedure follows to decompress a file.

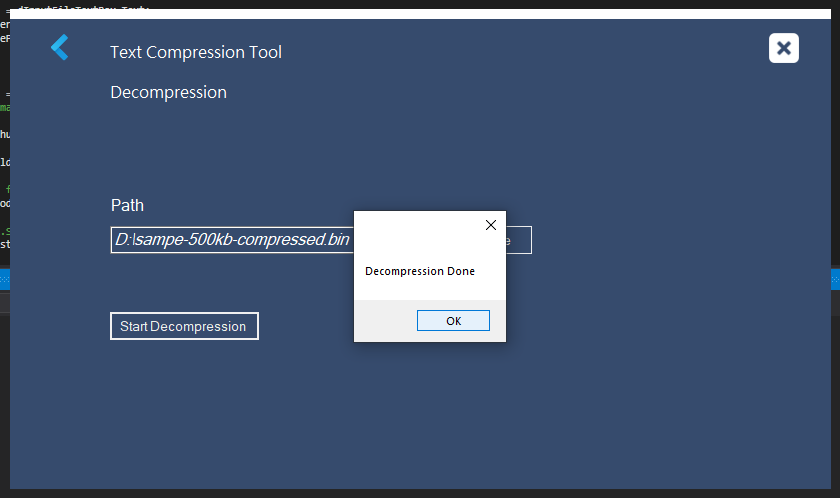
* Click Decompress Icon
* Enter or browse a compressed file, Exception handled.
* Click the Decompress and the completion message box will appear.
* The decompressed file will be stored at the describes location where the compressed file is stored, with the name **“original file name-decompressed.txt”**





**3.1)** Path of the compressed file can be entered or Browsed.

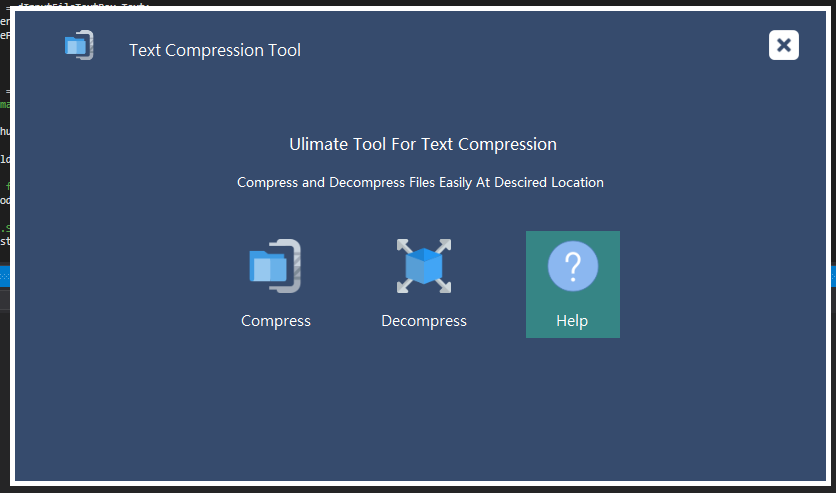


**3.2)** On completion, the Message Box appears to show the decompression done.

**4. Help**

Help button is the link to the public report git repository, that contains this detailed report about the software, its backend used, and all other technologies used.

The link to the repository is: https://github.com/mrqasimasif/CS311S20PID01-PublicReport/



The description of project UI is complete here.

**Integration of Backend with Frontend**

**Phase 1**

At the time of project proposal, we were confident about the python backed working with C# UI. So, an interactive UI was created in C#, with the backend code base of the Python Huffman Coding program. Both front end UI and backend Python program worked well separately, the UI was responsive with all types of exception handling and the backend program base was successfully doing compression and decompression on text files and producing binary coded files.

At the phase of integration, there was a great challenge for us to integrate both C# UI with Python code base. We came up with two solutions

* Running Python Script in Shell
* Using IronPython

In progress with running Python Script in Shell, the C# UI events were unable to communicate with the Python backend script, because we need to pass a path as an argument and the flag bit to denote its Compression or Decompression Process. The main error we faced in integration of Python code base with C# UI was **“Can’t open Python script compress.py”** with arguments **“path”.**

Then we preferred the IronPython, which was installed in the project solution using NuGet Extension Manager. It allowed us to directly call some python Script using some System commands in UI events like button click. But the main issue was the complete construction of a new GUI with IronPython Desktop Project. That didn't seem feasible near the deadline.

**Phase 2**

So the Final solution we proposed was the complete change of Backend code base from Python to C# that allowed the classes structure inside the project and the class objects were easily called in UI events and allowed the easy integration of the backend with the frontend.

**Change Requests**

Yes, we have changed the coding language from Python to C#. Because we were facing difficulties in integrating Python with C# .net framework. After so many efforts, we had not been able to resolve this issue. Hence, we felt it easier to use another coding language than the whole change in GUI, that could integrate properly with C# .net framework windows forms application.

**Testing**

Testing of our project was performed by Group 5; testing was performed before release. 4 issues were reported by the testing team, that contains major 2 issues.

The issues reported were:

1. [Just working on small inputs](https://github.com/mrqasimasif/CS311S20PID01/issues/1)
2. [gui not working](https://github.com/mrqasimasif/CS311S20PID01/issues/2)
3. [didn’t run on slightly large file (maybe infinite loop)](https://github.com/mrqasimasif/CS311S20PID01/issues/3)
4. [don't gave same file back if input file is larger than 1.5 mb (empty file)](https://github.com/mrqasimasif/CS311S20PID01/issues/4)

Looking at the issues there are major 2 issues

1. GUI was not working.
2. Did not run on relatively large inputs.

Both issues were resolved and closed on Github. On 8th of August, issues were reported, the issue of GUI got fixed by just after some control events management, GUI worked well, and the issue was closed on 9th of August.

The second major issue was file size not supported for compression, now after the change of code base from Python to C#, the file size limit is waived off for the compression process. The C# code base provided efficient compression for files larger than 1MB. The issue was opened on 8th of August and was closed on 14th of August.

So, the software Application is now in working state and performs space efficient compression and decompression.

**Technology**

We preferred to use the technologies that are highly rated, user friendly and nowadays used by big software and technological companies. The technologies used in this project are described below.

* **Programming Language**

We used C# .Net Framework for implementing Huffman Coding algorithm and C# Windows forms application for creating UI because of the two main reasons.

* First and foremost, we are studying this language in our currently ongoing academic semester.
* Secondly, it is one of the highly rated, and mostly used modern programming languages that runs on the .NET Framework to build desktop, mobile, and web applications.
* **Platform**

The application development main platform is Visual Studio Community using C# Windows form application, we preferred to build a **desktop application** because the main use is compression at academics level, and developing the desktop application waives off the mandatory internet connection to use the application and can perform the offline compression and decompression tasks.

End of Report