Palestine Polytechnic University

College of Engineering & Technology
Computer & Electrical Engineering Department

Software Project Report

Data Compression and Decompression using Visual C#

Project Team
Anwar A. Fakhouri **Wael F. Takrouri**
Yousef H. Shroukh

Project Supervisor
Eng. Amal Mohmmad Al-dweik

Hebron - Palestine

June 2004

Palestine Polytechnic University

College of Engineering & Technology
Computer & Electrical Engineering Department

Software Project Report

Data Compression and Decompression using Visual C#

Project Team
Anwar A. Fakhouri **Wael F. Takrouri**
Yousef H. Shroukh

Project Supervisor
Eng. Amal Mohmmad Al-dweik

Hebron - Palestine

June 2004

جامعة بوليتكنيك فلسطين

الخليل – فلسطين

كلية الهندسة والتكنولوجيا

دائرة الهندسة الكهربائية و الحاسوب

اسم المشروع:
ضغط و إزالة ضغط البيانات

اسماء الطلبة
انور الفاخوري وائل التكروري
يوسف الشروخ

بناء على نظام كلية الهندسة و التكنولوجيا و إشراف ومتابعة المشرف المباشر على المشروع و موافقة أعضاء اللجنة الممتحنة تم تقديم هذا المشروع إلى دائرة الهندسة الكهربائية و الحاسوب و ذلك للوفاء بمتطلبات درجة البكالوريوس في الهندسة تخصص هندسة أنظمة الحاسوب

توقيع المشرف

م. أمل الدويك .....

توقيع اللجنة الممتحنة

د. نبيل عرمان ..... م. مراد أبوصبيح .....

توقيع رئيس ا لدائرة

د. غاندي المناصره .....

Dedication

To our Parents ……

To our Families ……

To our Friends ……

To our Supervisor ……

To all Lecturers in our University……

To our University ……

Anwar
Wael
Yousef

Acknowledgement

To our supervisor Eng. Amal Mohammad Al-Dweik for her guidance, support and encouragement, to every one who provides us with helpful suggestions.

Abstract

Data Compression and Decompression (DCD) using C# system aims to use Microsoft Visual C# as a new programming language to compress data (Text files and image), the system has the ability to decompress data after compressing it so that the original data will be available after this stage, the system uses 3 different compression algorithms (Run Length Encoding or RLE, Huffman Encoding, Adaptive Huffman).

DCD system has the ability to provide statistical information summarizing the results obtained after compression and decompression applied to particular file (i.e. original file size, compressed file size compression ratio), a demonstration show how a particular algorithm works is also provided by the system.

الملخص

ضغط وإزالة ضغط البيانات و الصور باستخدام #C هو نظام يهدف الى استخدام لغة البرمجة (Microsoft® Visual C#) لضغط نوعين من الملفات(الملفا ت النصية والملفات الصورية)، وبإمكان النظام أن يزيل الضغط عن الملفات بعد ضغطها وبالتالي نحصل على البيانات الأصلية، يتعامل النظام مع ثلاثة أنواع من الخوارزميات (Run Length Encoding or RLE, Huffman Encoding and Adaptive Huffman) ، في عمليات الضغط و إزالة الضغط لملفات البيانات و الصور.

يمكن لهذا النظام أن يزود المستخدم بمعلومات إحصائية تتعلق بالملف الذي تم عمل له ضغط وإزالة ضغط(من هذه **المعلومات مثلا حجم الملف الأصلي وحجم الملف المضغوط ومعامل الضغط**) ، ويوفر النظام مشهد توضيحي يوضح ألية عمل خوارزية معينة.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

BMP	Bitmap
DCD	Data Compression Decompression
DFD	Data Flow Diagram
GB	GigaByte
GIF	Graphics Interchange Format
GUI	Graphical User Interface
JPEG	Joint Photographic Experts Group
MB	MegaByte
OO	Object Oriented
PC	Personal Computer
RLE	Run Length Encoding
SVGA	Super Video Graphics Array

Palestine Polytechnic University

College of Engineering & Technology
Computer & Electrical Engineering Department

Software Project Report

Data Compression and Decompression using Visual C#

Project Team
Anwar A. Fakhouri **Wael F. Takrouri**
Yousef H. Shroukh

Project Supervisor
Eng. Amal Mohmmad Al-dweik

Hebron - Palestine

June 2004

جامعة بوليتكنيك فلسطين

الخليل – فلسطين

كلية الهندسة والتكنولوجيا

دائرة الهندسة الكهربائية و الحاسوب

اسم المشروع:
ضغط و إزالة ضغط البيانات

اسماء الطلبة
انور الفاخوري وائل التكروري
يوسف الشروخ

بناء على نظام كلية الهندسة و التكنولوجيا و إشراف ومتابعة المشرف المباشر على المشروع و موافقة أعضاء اللجنة الممتحنة تم تقديم هذا المشروع إلى دائرة الهندسة الكهربائية و الحاسوب و ذلك للوفاء بمتطلبات درجة البكالوريوس في الهندسة تخصص هندسة أنظمة الحاسوب

توقيع المشرف

م. أمل الدويك .....

توقيع اللجنة الممتحنة

د. نبيل عرمان ..... م. مراد أبوصبيح .....

توقيع رئيس ا لدائرة

د. غاندي المناصره .....

Dedication

To our Parents ……

To our Families ……

To our Friends ……

To our Supervisor ……

To all Lecturers in our University……

To our University ……

Anwar
Wael
Yousef

Acknowledgement

To our supervisor Eng. Amal Mohammad Al-Dweik for her guidance, support and encouragement, to every one who provides us with helpful suggestions.

Abstract

Data Compression and Decompression (DCD) using C# system aims to use Microsoft Visual C# as a new programming language to compress data (Text files and image), the system has the ability to decompress data after compressing it so that the original data will be available after this stage, the system uses 3 different compression algorithms (Run Length Encoding or RLE, Huffman Encoding, Adaptive Huffman).

DCD system has the ability to provide statistical information summarizing the results obtained after compression and decompression applied to particular file (i.e. original file size, compressed file size compression ratio), a demonstration show how a particular algorithm works is also provided by the system.

الملخص

ضغط وإزالة ضغط البيانات و الصور باستخدام C# هو نظام يهدف الى استخدام لغة البرمجة (Microsoft® Visual C#) لضغط نوعين من الملفات(الملفا ت النصية والملفات الصورية)، وبإمكان النظام أن يزيل الضغط عن الملفات بعد ضغطها وبالتالي نحصل على البيانات الأصلية، يتعامل النظام مع **ثلاثة** أنواع من الخوارزميات (Run Length Encoding or RLE, Huffman Encoding and Adaptive Huffman) ، في عمليات الضغط و إزالة الضغط لملفات البيانات و الصور.

يمكن لهذا النظام أن يزود المستخدم بمعلومات إحصائية تتعلق بالملف الذي تم عمل له ضغط وإزالة ضغط(من هذه **المعلومات مثلا حجم الملف الأصلي وحجم الملف المضغوط ومعامل الضغط**) ، ويوفر النظام مشهد توضيحي يوضح ألية عمل خوارزية معينة.

Table of Contents

List of Tables

List of Figures

List of Tables

List of Abbreviations

BMP	Bitmap
DCD	Data Compression Decompression
DFD	Data Flow Diagram
GB	GigaByte
GIF	Graphics Interchange Format
GUI	Graphical User Interface
JPEG	Joint Photographic Experts Group
MB	MegaByte
OO	Object Oriented
PC	Personal Computer
RLE	Run Length Encoding
SVGA	Super Video Graphics Array

Chapter One

Introduction

1.1 Preface

1.2 System Objectives

1.3 Previous Work

1.4 Main Points of Contribution

1.5 Report Outline

Chapter Two

Project Planning

2.1 Preface

2.2 Project Organization

2.3 Risk Analysis

2.4 Hardware and Software Resource Requirement

2.5 Work Activities

2.6 Project Schedule

Chapter Three

System Specification and Analysis

3.1 Software Requirement Specification

3.2 System Requirement Specification

3.3 Software Requirement Analysis

3.4 System Models

Chapter Four

Software Design

4.1 Preface

4.2 Objects and Object Classes

4.3 O-O design process

4.4 System Interface Design

Chapter Five

Implementation

5.1 Preface

5.2 Algorithms

5.3 System Pseudo code

5.4 System Flow Chart

5.5 System Results

5.6 Conclusions

Chapter Six

Testing

6.1 Introduction

6.2 Unit Testing

6.3 Integration Testing

Chapter Seven

Conclusion and Future Work

7.1 Conclusion

7.2 Future Work

Resources

Appendices

Appendix A Glossary

Appendix B DCD User Interface

Appendix C DCD Source Code

Appendix A

Glossary

Appendix B

DCD User Interface

Appendix C

DCD Source Code

Chapter One

Introduction

1.1 Preface

1.2 System Objectives

1.3 Previous Work/Previous System

1.4 Main Points of Contribution

1.5 Report Outline

1.1 Preface

Data and image compression was and still one of the subjects that many of software researches and projects work on, this comes form the fact that data and image compression provides great advantages, these advantages come into two main things:

- Save size on disks.
- Reduce cost of transmitting data over the Internet or networks in general.

The main idea behind data compression is to eliminate redundant data, redundancy of data exists specially in image files, and so if there is a technique to eliminate this redundancy, this will reduce the size of data file.

1.2 System Objectives

The system is expected to accomplish the following

- Compression of text files (files with extension .txt) and image files (Bitmap (.bmp), JPG (.jpg) and GIF (.gif)) using a predefined compression algorithms.
- Decompression of the compressed file so it will provide the original data file.
- Provides statistical information after applying compression to a data file. This information includes the size of original file, the size of compressed file and compression ratio.
- Provides quick demonstration that will explain and illustrate how Huffman algorithm works in an animated mode
- Learn Microsoft Visual C# as a new programming technology.

1.3 Previous Work

There is a project that works on the image compression using matlab, this project “Image Compression using Discrete Transform” by Jawad Sabha, Shadi Arafeh and

Wanes Abu Safa which presented at 2002 works on images using Discrete Cosine Transform algorithm, Hartly Transform Hadamard Transform and Haar Transform.

1.4 Main Points of Contribution

The system will be able to do the following:

- Reduces the size of text and image files using three well known algorithms (Run Length Encoding, Huffman Encoding and Adaptive Huffman), so it will reduce the storage requirements, it also reduces the cost required to transmit data over the networks.
- Provides user with statistical information, so it will make it easy for the user to make comparisons between the algorithms and make judge which algorithm is more efficient for a particular file.
- Provides the user with demonstration that will illustrate how a particular algorithm works using animated show.

1.5 Report Outline

Report consists of seven chapters, the following is a brief description of the topics that will be covered in each chapter.

Chapter One Introduction presents an idea about system objectives, the previous work that is related to the current system, also it introduces the main points of contribution of the system.

Chapter Two Project Planning presents an idea about how project team is organized, risks that faces the project team, suggestions to reduce these risks and gives description about project schedule that is how project activities are related and depended between each other.

Chapter Three System Specification and Analysis introduces both the Software and system requirement specification, it also covers software requirement analysis and the system models (i.e. Data Flow Diagrams (DFDs), Object Models).

Chapter Four Software Design Shows objects and classes used by the system, it describes the design process of Object Oriented (OO), and it also describes the software interface design.

Chapter Five Implementation provides description of the features and characteristics of each algorithm used in the system, system pseudo code and flowcharts are included.

Chapter Six Testing explains levels of test unit component and defect testing, and it provides OO testing.

Chapter Seven Conclusion and Future Work presents conclusions that concluded form design and development of the project and it also provide suggested ideas to be implemented for future work.

Chapter Two

Project Planning

2.1 Preface

2.2 Project Organization

2.3 Risk Analysis

2.4 Hardware and Software Resource Requirement

2.5 Work Activities

2.6 Project Schedule

2.1 Preface

The system will be able to implement the following

- Compression of text and image files and decompress the compressed files
- Provides statistical information about the compressed files
- Provides illustration of how Huffman algorithm works

The system will be dominated with the following constraints:

- The system development time will be 16 weeks

2.2 Project Organization

The development team consists of three developers

- Anwar A. Fakhouri
- Wael F. Takrouri
- Yousef H. Shroukh

The work division and role of each developer will be as follow

- Anwar A. Fakhouri responsible for project planning, scheduling and requirement specification and analysis.
- Wael F. Takrouri responsible for project design and implementation.
- Yousef H. Shroukh responsible for requirement specification and analysis and for project testing.

Developer	Role
Anwar	Planning, scheduling and requirement specification and analysis
Wael	Design and implementation
Yousef	Requirements specification and analysis and project testing

Table 2.1 Team Work Division

2.3 Risk Analysis

2.3.1 Project Risks

Risk can be expected as follows

- The way of how to deal with image, reading the content of image using Visual C#
- The documentation and material related to data compression using C# is not available.
- Illness of one of the team members.

2.3.2 Reduction Strategies

Risks can be reduced using the suggested ideas

- By knowing how other object oriented languages like java programming language deal with different compression algorithm and how they deal with images.
- Searching for other documentations that illustrate how other programming languages build compression algorithms.
- Make a new plan that increases the efforts on the other team members to perform the tasks of the sick member.

2.4 Hardware and Software Resource Requirement

2.4.1 Hardware Development Resources

One PC with the following specifications:

- Pentium III 1000MHz
- 256MB RAM
- 30 GB Hard Disk Drive (HDD)
- 15” SVGA Monitor
- Keyboard
- Mouse

2.4.4.1 Hardware Resources Cost Estimation

	Component	Estimated cost ($)
1-	Pentium III PC	340.0
2-	Printer	150.0
Total cost		490.0

Table 2.2 Hardware Resource Cost Estimation

2.4.2 Software Development Resources

The development of project requires the following packages

- Microsoft Windows XP professional
- Microsoft Visual Studio.Net 2003
- Microsoft Office 2003

2.4.2.1 Software Resources Cost Estimation

	Component	Estimated cost ($)
1-	Microsoft Windows XP Professional	290.0
2-	Microsoft Visual Studio .Net 2003	990.0
3-	Microsoft Office 2003	420.0
4-	Macromedia Flash	250.0

Total cost	1950.0

Table 2.3 Software Resource Cost Estimation

2.4.3 Human Development Resources

The development of project will be implemented by a team of 3 developers

2.4.3.1 Human Resources Cost Estimation

The project team work through 16 weeks, with 6 days a week, and 5 hours a day.

Total number of hour of work: 16week X 6days X 5hours= 480 hour

480 hour X $20= $9600 per individual

Total human cost=$9600 X 3 individuals= $28800

2.4.4 Internet

Total Internet cost = 6 week X 5 hours X $1 = $30

2.4.5 Transportation

Total transportation cost = 16 week X 5days X $6 = $480

2.4.6 Project Total Cost

	Resource	Estimated cost ($)
1-	Hardware	490.0
2-	Software	1950.0
3-	Human	28800.0
4-	Internet	30.0
5-	Transportation	480.0

Total cost	31750.0

Table 2.4 Project Total Cost

2.5 Work Activities

The system development involves a group of tasks, the following points describe these tasks and the working period of each task

T1: Collection of Data	6 Weeks
T2: Requirements Specification	4 Weeks
T3: Requirements Analysis	5 Weeks
T4: System Design	7 Weeks
T5: System Implementation	6 Weeks
T6: System Testing	5 Weeks
T7: documentation	16 week

Requirements Specification involves the following activities

- Requirements definition: the functional and non functional requirements are described

System Design involves the following activities

- Design of classes and objects.
- Design of software interface which involves the design of Graphical User Interface (GUI) components.

System Implementation involves the following activities

- Implementing algorithms
- Implementing flow charts

System Testing involves the following activities

- Defect testing which involves Black Box testing, structural testing and interface testing
- Object Oriented testing which involves testing objects

2.6 Project Schedule

A) Task Schedule

The following bar chart describes the project tasks and the duration of each task

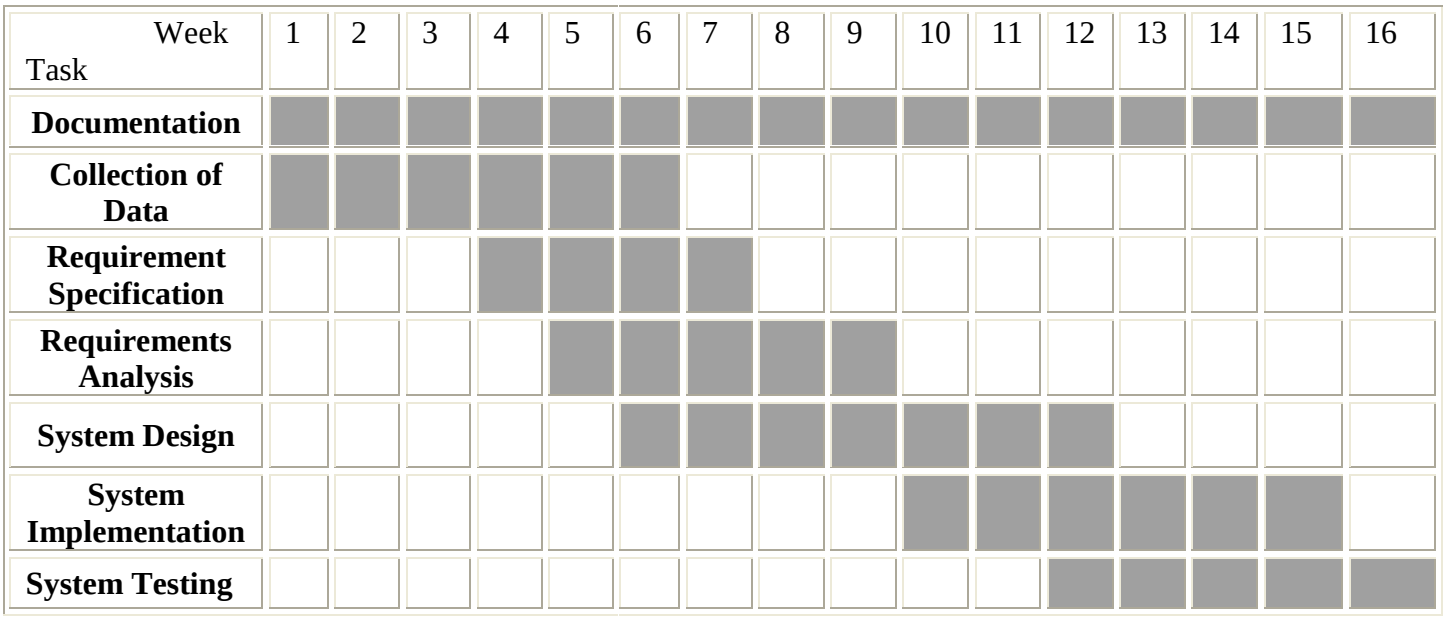


Figure 2.1 Bar Chart Task Schedule

B) Team Schedule

Task	Developer
T1	Anwar, Wael and Yousef
T2	Anwar and Yousef
T3	Anwar and Yousef

T4	Wael
T5	Wael
T6	Anwar and Yousef
T7	Anwar, Wael and Yousef

Table 2.5 Team Schedule

Chapter Three

System Specification and Analysis

3.1 Software Requirement Specification

3.2 System Requirement Specification

3.3 Software Requirement Analysis

3.4 System Models

3.1 Software Requirement Specification

3.1.1 System Definition

DCD is Data Compression and Decompression system which has the ability to compress text and image data and decompress it after compression, using three different encoding decoding techniques (Run Length Encoding (RLE), Huffman Encoding, and Adaptive Huffman), also the system provides statistical information after a particular compression algorithm applied to a particular file (i.e. it provides quality of data, and compression ratio), a demonstration shows how a Huffman algorithm works is also provided by the system.

The system requirements in the DCD are categorized into two main parts, these requirements are:

- Functional Requirements
- Non-functional Requirements

We will consider both of them and describe them in details through this chapter.

3.1.2 Functional Requirements

An illustration of the functional requirements will be provided through this section, functional requirements definition and functional requirements specification will be considered.

Functional Requirements Definition

The functional requirements involve accomplishing the following requirements:

- Allow the user to compress text files (files with extension .txt) and image files (bitmap .bmp, JPG .jpg and GIF .gif) using the predefined compression algorithms.
- Allow the user to decompress compressed file using the same algorithm that was compressed by, so it will provide the original data file.
- Provides statistical information after applying compression to a data file. This information includes quality of data and compression ratio.
- Provides quick demonstration that explains and illustrates how Huffman algorithm works in an animated mode.

3.1.2.1 Allow the User to Compress Data Definition

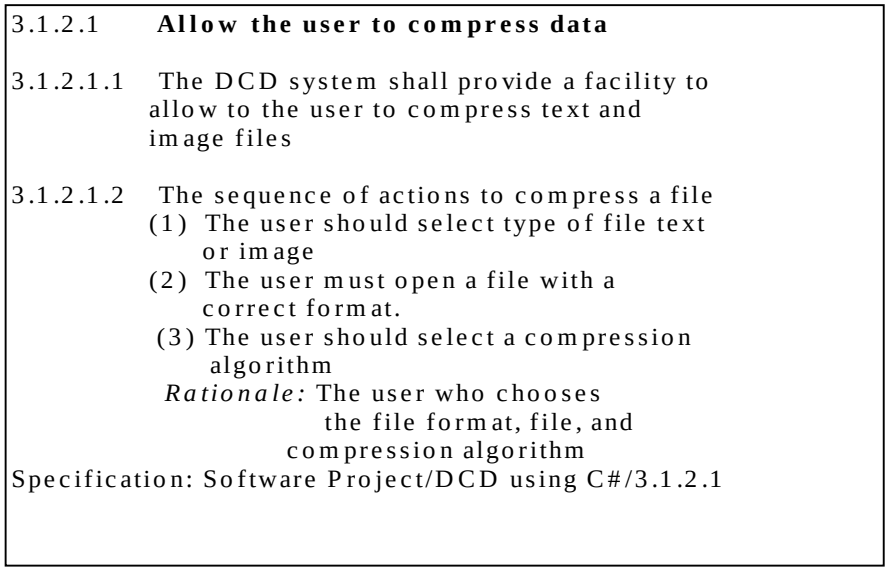


Figure 3.1 Requirements Definition for Data Compression

3.1.2.2 Allow the User to Decompress Data Definition

3.1.2.2 **Allow the user to decompress data**

3.1.2.2.1 The DCD system shall provide a facility to allow to the user to decompress compressed files.

3.1.2.2.2 The sequence of actions to decompress a file
(1) The user should select type of file format
(2) The user must open a file with a correct format (compressed file).
(3) The user should select a decompression algorithm
Rationale: The user who chooses the file format, file, decompression algorithm

Specification: Software Project/DCD using C#/3.1.2.2

Figure3.2 Requirements Definition for Data Decompression

3.1.2.3 Provides Statistical Information Definition

3.1.2.3 **Provides Statistical Information**

3.1.2.3.1 The DCD system shall provide a facility to allow to the user to request statistical information on a compressed file

3.1.2.3.2 The sequence of actions to request statisticla information
(1) The user should select type of file format
(2) The user must open a file with a correct format (compressed file).
(3) The user should select a compression algorithm
(4) The user Should request statistical information.
Rationale: The user who chooses the file format, file, compression algorithm,and request Statistical information

Specification: Software Project/DCD using C#/3.1.2.3

Figure 3.3 Requirements Definition for Requesting Statistical Information

3.1.2.4 Provides a Quick Demonstration Definition

3.1.2.4	**Provides a quick demonstration**
3.1.2.4.1	The DCD system shall provide a facility to allow to the user see a quick demonstration explains how Huffman algorithm works
3.1.2.4.2	The sequence of actions to request demonstration The user should request demonstration *Rationale:* The user who requests the demonstration
Specification: Software Project/DCD using C#/3.1.2.4	

Figure 3.4 Requirements Definition for Requesting Demonstration

3.1.3 Non – Functional Requirements

- Friendly user interface
- Using Visual C# as development environment
- Portability
- Reliability
- Performance

3.1.3.1 Usability (Friendly User Interface)

DCD provides the user with easily and friendly graphical component by using help tips and help pages for each used form, also the use of friendly messages which

directs the user how to recover from errors and how to interact with the system operations.

3.1.3.2 Using Visual C# as Development Environment

Visual C# which is built on .NET technology provides a group of classes and methods to deal with image and text files, which enhances the development of DCD system.

3.1.3.3 Portability

The system works under Microsoft Windows operating systems Windows 9X, Windows 2000 and Windows XP.

3.1.3.4 Reliability

The system is reliable that it manages all the problems that may occur, it also able to deal with exceptions that may results.

3.1.3.5 Performance

The system provides the services to the user in an reasonable time, which makes system efficient.

3.2 System Requirements Specification

3.2.1 Allow the User to Compress Data

Software Project / **Data Compression and Decompression Using C# /** ***3.2.1***

Function: Allow the user to compress data (text or image)

Description: A user can select a text or image file and compress it by a predefined algorithm.

Inputs: Text or image file, compression algorithm selection.

Source: File and compression algorithm are selected by the user.

Outputs: Compressed file (file with extension .data).

Destination: Folder named with the same of the compression algorithm.

Requires: Data (text or image).

Pre-Condition: The user should select a file to be compressed, and then select a compression algorithm.

Post-Condition: None.

Side-effects: None

Definition: Software Project / **Data compression decompression Using C# /** ***3.2.2***

3.2.2 Allow the User to Decompress Data

Software Project / **Data Compression Decompression Using C# *3.2.3***

Function: Allow the user to decompress compressed file

Description: A user can select a compressed file and decompress it by the compression algorithm.

Inputs: Compressed file (file with extension .data), decompression algorithm.

Source: Compressed file and decompression algorithm are selected by the user.

Outputs: The original data (text or image).

Destination: The decompression folder.

Requires: Compressed file.

Pre-Condition: The user should select a compressed file to be compressed, and then select a decompression algorithm.

Post-Condition: None.

Side-effects: None.

Definition: Software Project / **Data compression decompression Using C# *3.2.1***

3.2.3 Provides Statistical Information

Software Project* / Data Compression Decompression Using C# / *3.2.3

Function: Provides statistical information to the user.

Description: Provides statistical information after applying compression to data this information include data quality and compression ratio

Inputs: File, compression algorithm.

Source: File and compression algorithm are selected by the user.

Outputs: Statistical information.

Destination: Data on windows form.

Requires: Data (text or image).

Pre-Condition: The user should select a file to provide related compression information.

Post-Condition: None.
Side-effects: None.

Definition: Software Project* / Data compression decompression Using C# / *3.2.3

3.2.4 Provides a Quick Demonstration

Software Project* / Data compression decompression Using C# / *3.2.4

Function: Provides a quick demonstration.

Description: Provides a quick demonstration that explains and illustrates how Huffman algorithm works in an animated mode

Inputs: None.

Source: None.

Outputs: Macromedia Flash view.

Destination: Sketch on screen.

Requires: None.

Pre-Condition: None.

Post-Condition: None

Side-effects: None.

Definition: Software Project* / Data compression decompression Using C# / *3.2.4

3.3 Software Requirement Analysis

In this section view points of the DCD system will be described, by providing view point hierarchy or structure and view point templates.

3.3.1 View-Point Oriented Elicitation

The DCD system interacts with a user, the user receives some services from the system, in this section the interaction between the user and the system will be explained. Figure 3.5 shows data and control information entered by the user.

Control input	Data input
File format Compression algorithm Decompression algorithm	File Compressed file

Figure 3.5 View Point Data and Control Informatopn

3.3.1.1 Viewpoint Hierarchy

Viewpoint hierarchy shows the services delivered to the user, as shown in Figure 3.6.

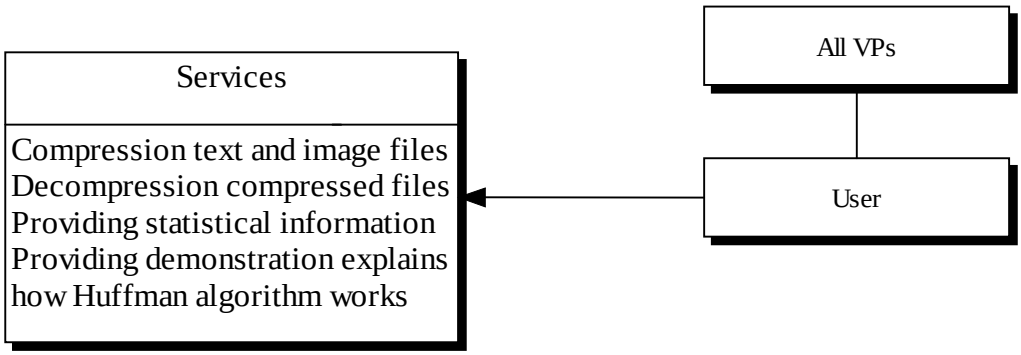


Figure 3.6 Viewpoint Hierarchy

3.3.1.2 Viewpoint Templates

In this section the relation between the viewpoint templates and the service templates provided to the user will be described, as shown in figure 3.7.

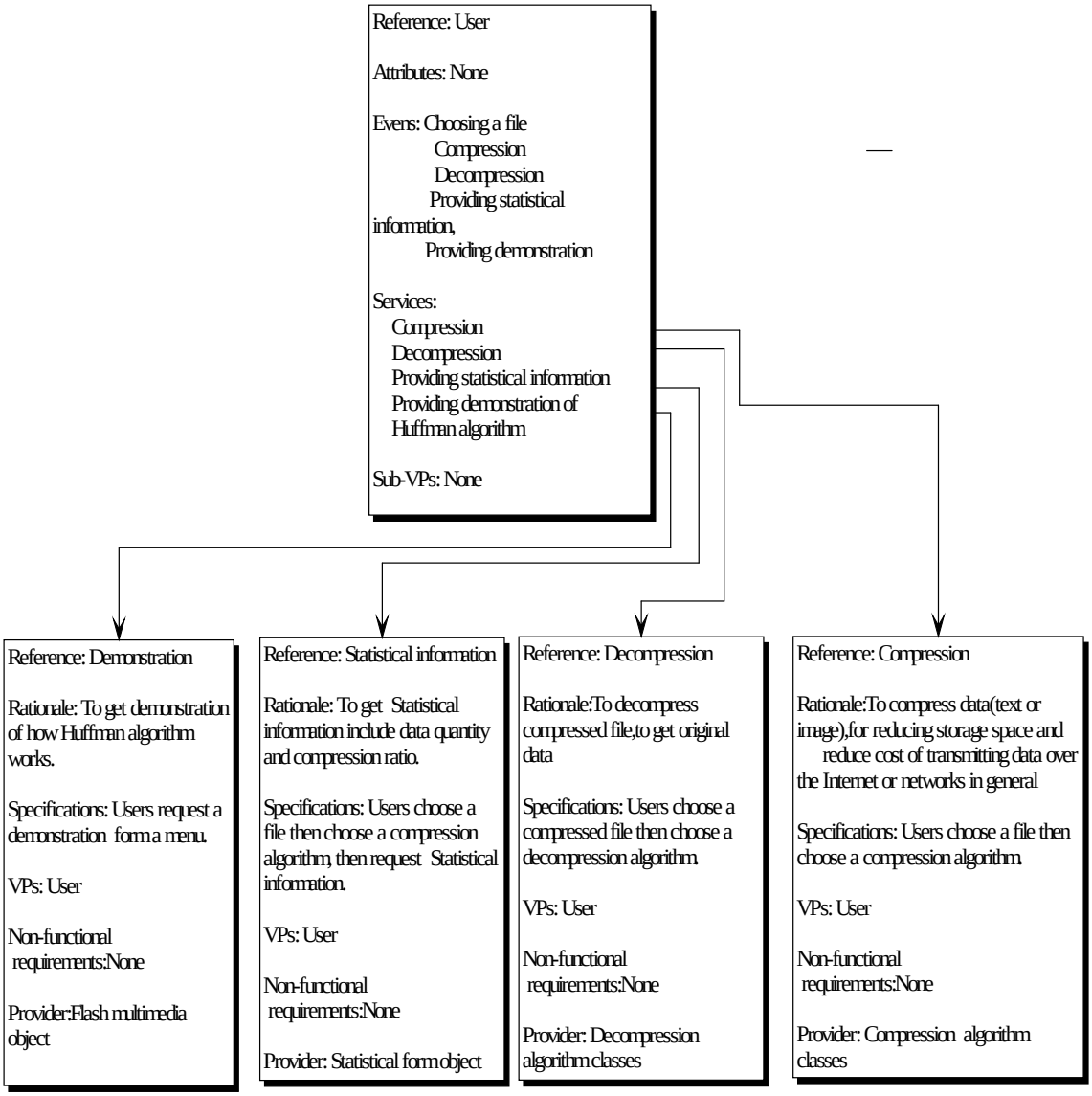


Figure 3.7 Viewpoint Templates

3.3.2 Event Scenario

Event scenario describes how the user interacts with the system, description of data flows and actions of the system and documents the exceptions which can arise.

3.3.2.1 Dealing with Text Files Scenario

The user deals with text files as follow:

(1) The user open file with a correct format, if the user doesn't specify the correct there is an exception:

- Invalid file format

(2) The user performs a set of services on the opened file.

(3) Compress file, if the user select a compression service while not opened file there is an exception.

- Open File

May be the user not select a compression algorithm there is an exception

- Select a compression algorithm

(4) Decompress file, if there is not a compressed file

- Compress file

(5) Request statistics, if there is not a compressed file

- Compress file

(6) Word count, if there is no opened file

- Open file

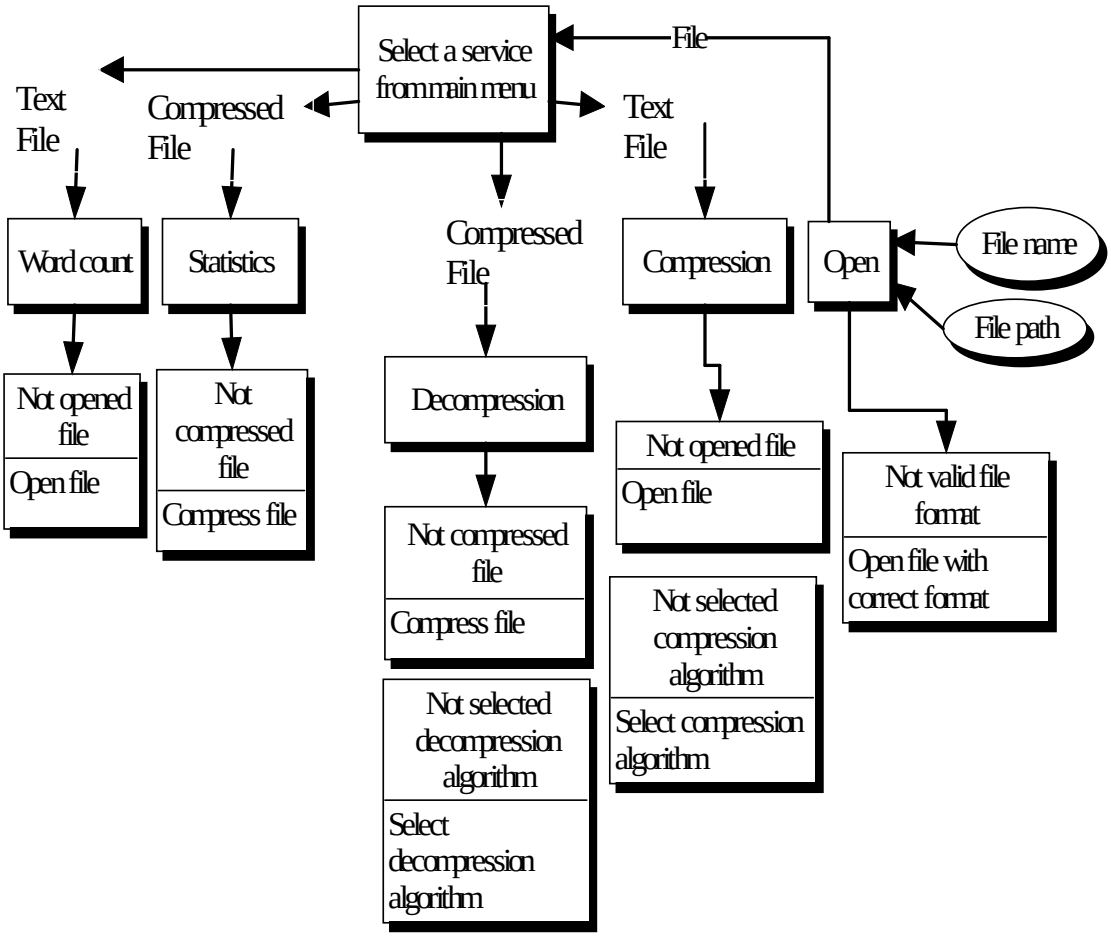


Figure 3.8 Event Scenario – Dealing with Text Files Scenario

3.3.2.2 Dealing with Image Files Scenario

The user deals with image files as follow:

(1) The user open file with a correct format, if the user doesn't specify the correct there is an exception:

- Invalid file format

(2) The user performs a set of services on the opened file.

(3) Compress file, if the user select a compression service while not opened file there is an exception.

- Open File

May be the user not select a compression algorithm there is an exception

- Select a compression algorithm

(4) Decompress file, if there is not a compressed file

- Compress file

(5) Request statistics, if there is not a compressed file

- Compress file

(6) Image properities, if there is no opened file

- Open file

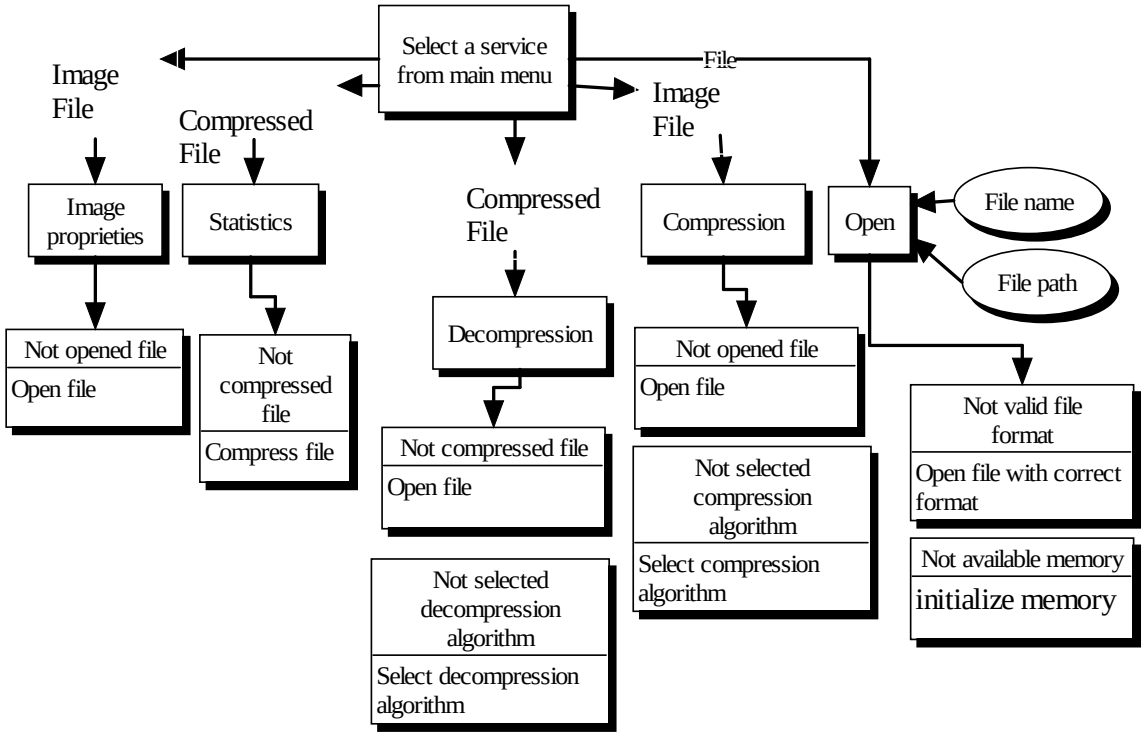


Figure 3.9 Event Scenario – Dealing with Image Files Scenario

3.4 System Models

In this section system models will be explained, the system context model, data flow diagrams, and data/object models.

3.4.1 System Context Model

The DCD system is independent so it hasn't any relation with other systems.

3.4.2 Data Flow Diagrams (DFD)

Data-flow models shows how data flows through a sequence of processing steps, the next diagrams shows how the data flow through the different processes in the DCD system.

3.4.2.1 Compression Process

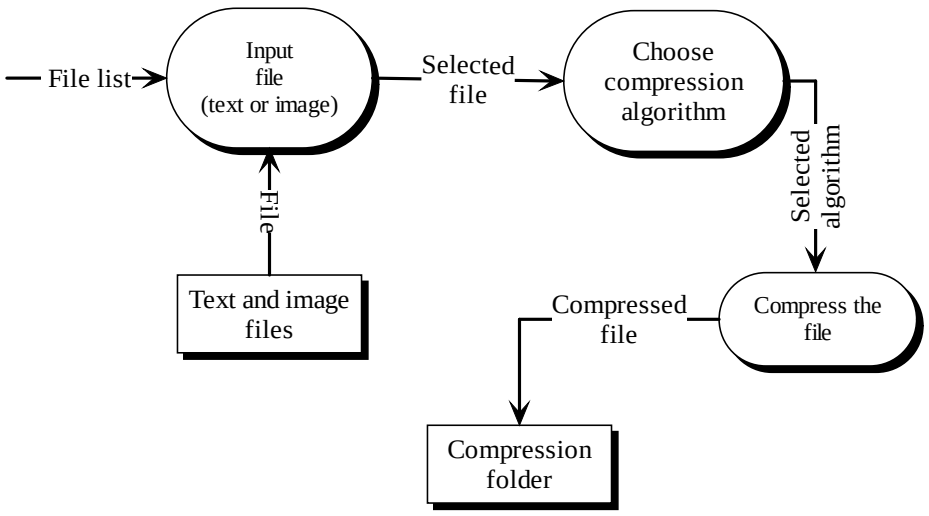


Figure 3.10 Data – Flow Diagram of Compression Process

3.4.2.2 Decompression Process

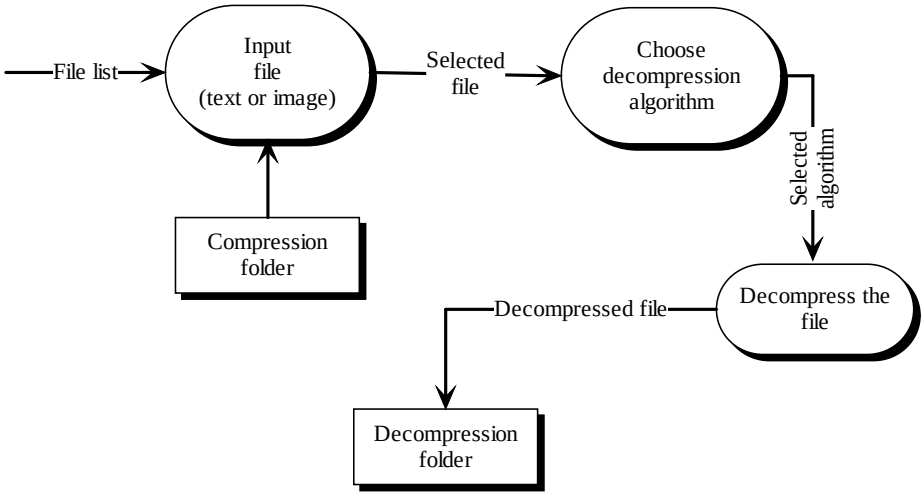


Figure 3.11 Data – Flow Diagram of Decompression Process

3.4.2.3 Request Statistical Information Process

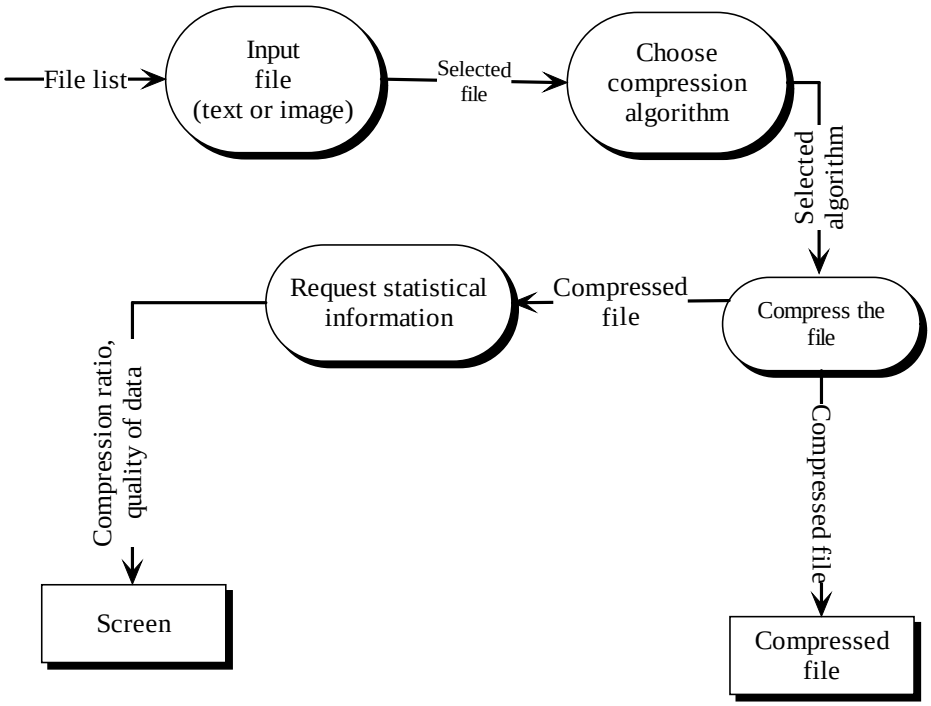


Figure 3.12 Data – Flow Diagram of Statistical Information Request Process

3.4.3 Object Models

Object models shows how entities in the DCD system are composed, and the inheritance relation among the different objects.

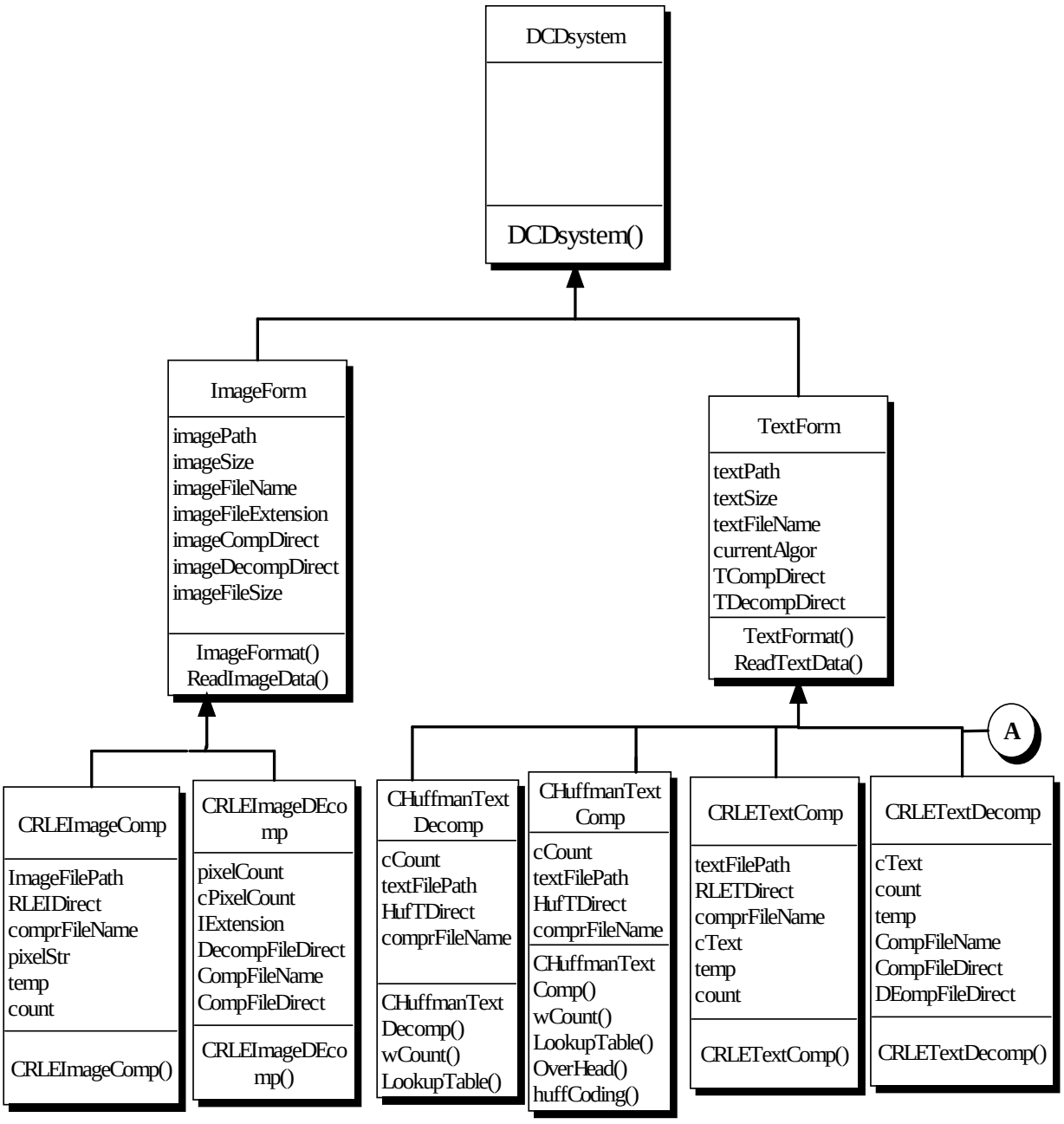


Figure 3.13 DCD System Class Hierarchy

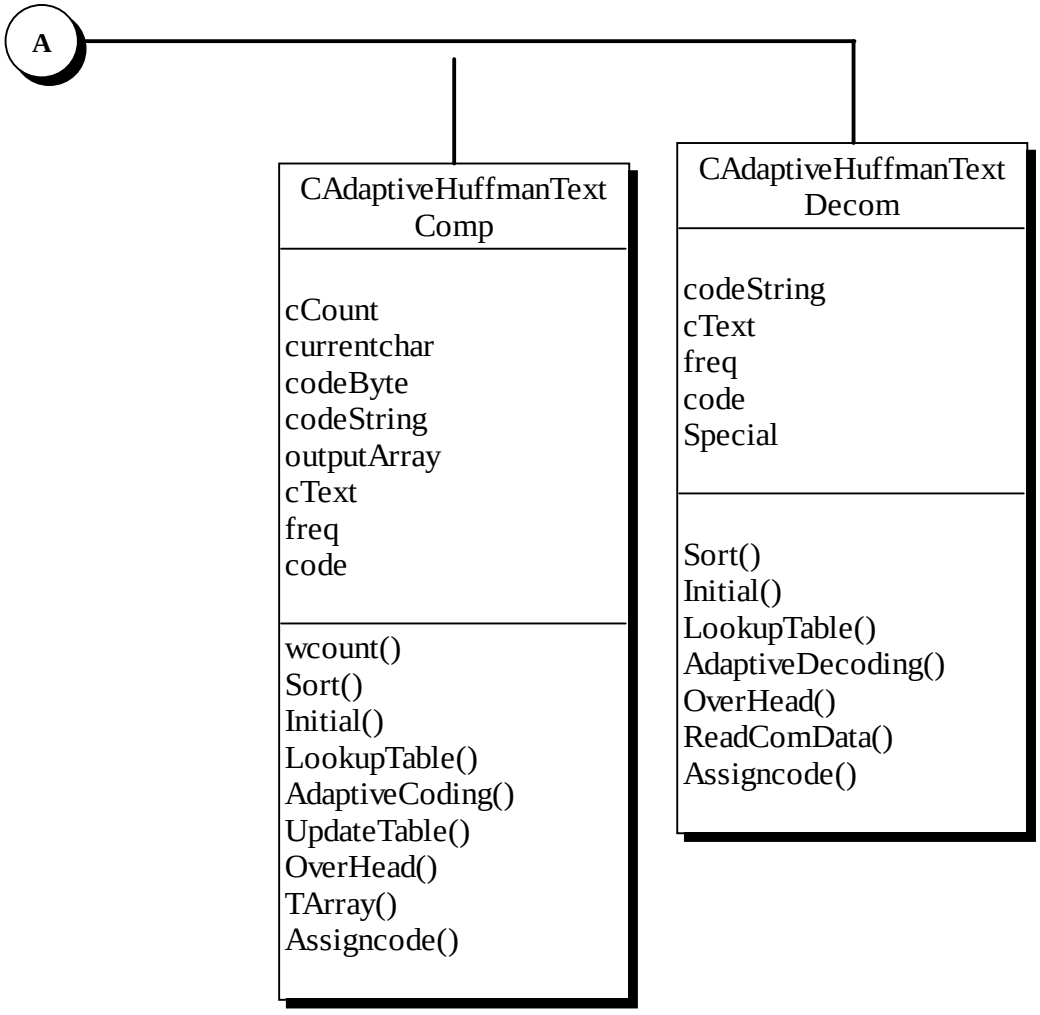


Figure 3.14 Cont. DCD System Class Hierarchy

Chapter Four

Software Design

4.1 Preface

4.2 Objects and Object Classes

4.3 O-O design process

4.4 System Interface Design

4.1 Preface

This chapter shows the system models. In addition, it demonstrates the relations between the system parts. It describes and identifies the different objects used in the system, illustrates the use-case and sequence diagrams for each part of the system, and describes the software interface design used in the system too.

4.2 Objects and Object Classes

This section describes all of the objects used in the DCD system that builds the different parts of the system.

4.2.1 DCD Object

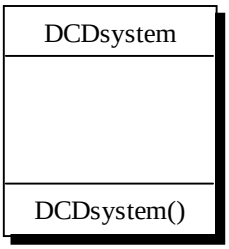


Figure 4.1 DCD Object

4.2.2 ImageFormat Object

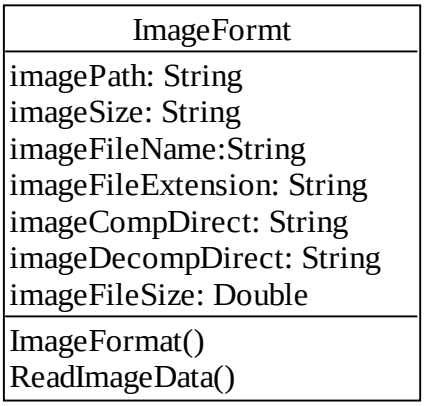


Figure 4.2 ImageFormat Object

4.2.3 TextForm Object

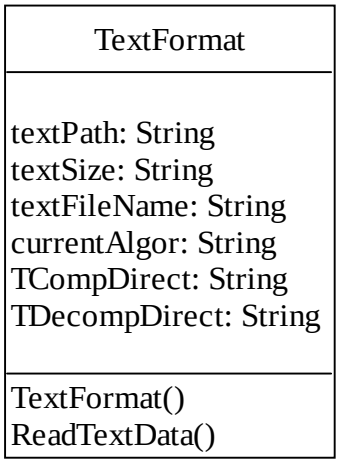


Figure 4.3 TextFormat Object

4.2.4 CRLERextComp Object

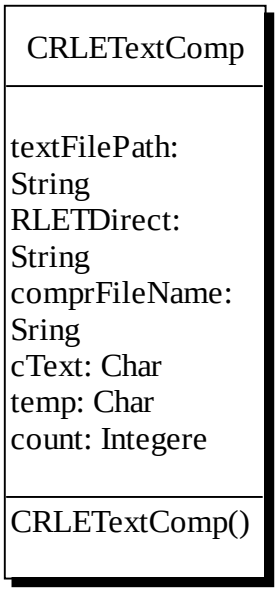


Figure 4.4 CRLETextComp Object

4.2.5 CRLERextDecomp Object

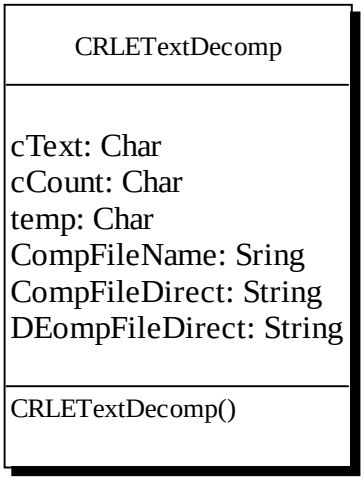


Figure 4.5 CRLETextDecomp Object

4.2.6 CHuffmanTextComp Object

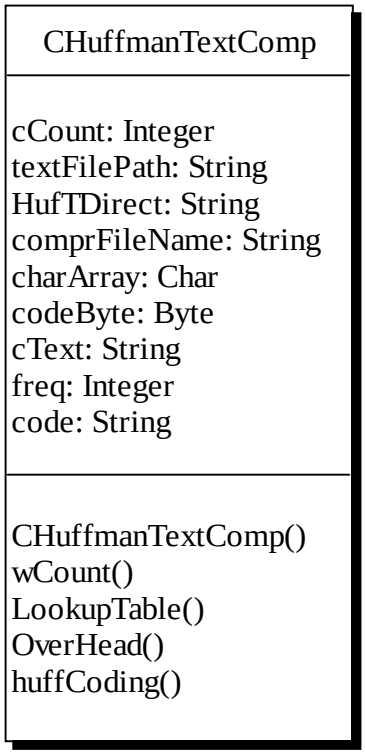


Figure 4.6 CHuffmanTextComp Object

4.2.7 CHuffmanTextDecomp Object

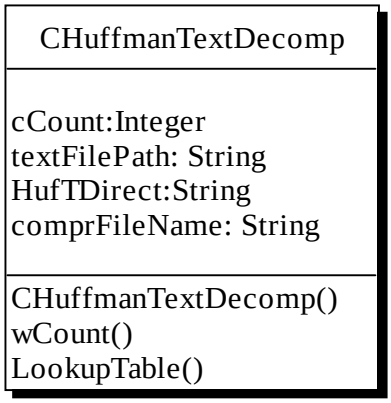


Figure 4.7 CHuffmanTextDecomp Object

4.2.8 CRLEImageComp Object

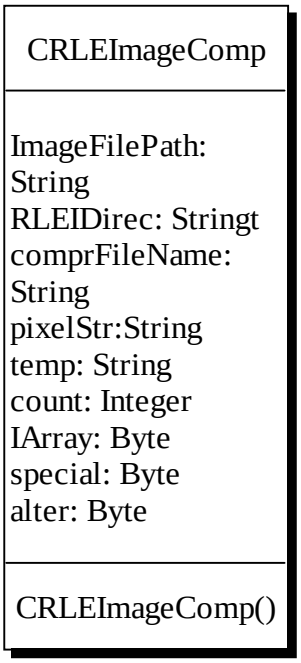


Figure 4.8 CRLEImageComp Object

4.2.9 CRLEImageDecomp Object

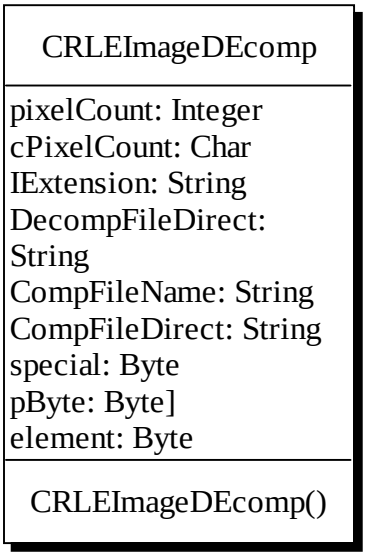


Figure 4.9 CRLEImageDecomp Object

4.2.10 CAdaptiveHuffmanComp Object

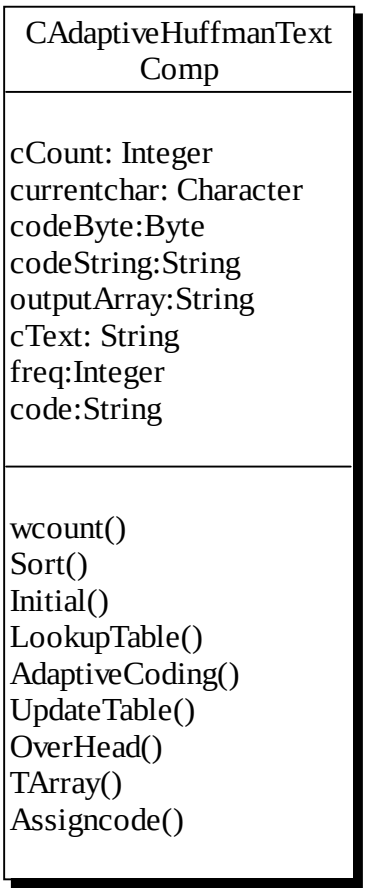


Figure 4.10 CAdaptiveHuffmanComp Object

4.2.11 CAdaptiveHuffmanDecomp Object

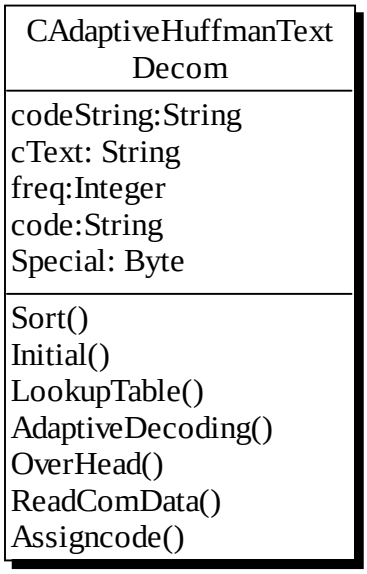


Figure 4.11 CAdaptiveHuffmanDecomp Object

4.3 O-O Design Process

In this section the process pf object-oriented design will be illustrated, it passes through the following stages:

1. Design the system architecture.
2. Identify the objects in the DCD system.
3. Develop design models.
4. Specify object interfaces.

4.3.1 Architectural Design

Architectural design decomposes the system to subsystems, so it describes all the subsystems that compose the DCD system as shown in figure 4.10.

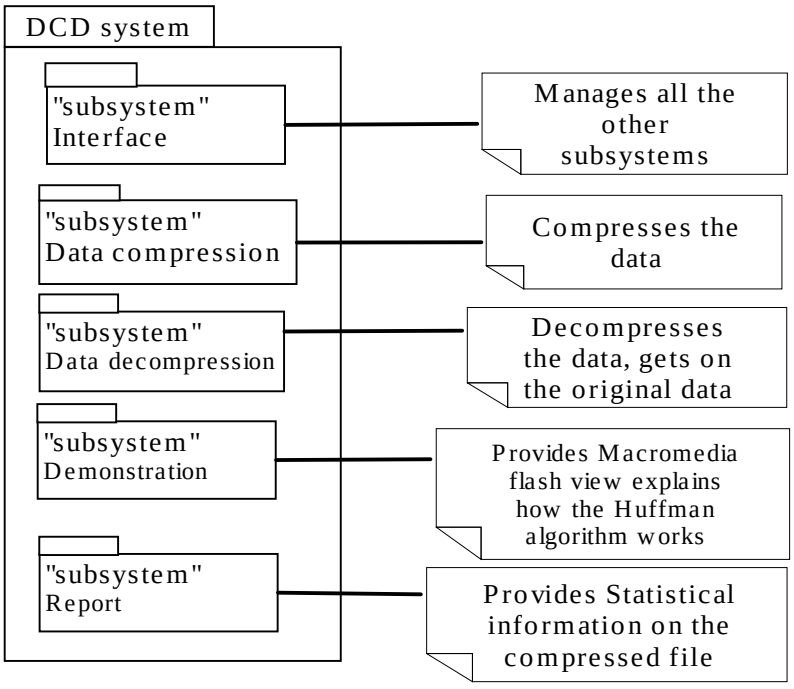


Figure 4.12 DCD System Architecture

4.3.2 Objects Identification

1. The DCD object class provides the basic interface of the DCD system, from it the user can perform the different services that is provided by the system as shown in figure 4.1.
2. The TextFormat object class provides the basic interface that allows the user to deal with the text file format and performs the different services on it as shown in figure 4.2.
3. The ImageFormat object class provides the basic interface that allows the user to deal with the image file formats and performs the different services on it as shown in figure 4.3.
4. The CRLETextComp object class is used to compress text data with the RLE compression algorithm as shown in figure 4.4.
5. The CRLEtextDecomp object class is used to decompress text data with the RLE compression algorithm as shown in figure 4.5.
6. The CRLEImageComp object class is used to compress image data with the RLE compression algorithm as shown in figure 4.6.
7. The CRLEImageDecomp object class is used to decompress image data with the RLE compression algorithm as shown in figure 4.7.
8. The CHuffmanTextComp object class is used to compress text data with the Huffman compression algorithm as shown in figure 4.8.
9. The CHuffmanTextDecomp object class is used to decompress text data with the Huffman compression algorithm as shown in figure 4.9.
10. The CAdaptiveHuffmanTextComp object class is used to compress text data with the adaptive Huffman compression algorithm shown in figure 4.10.

11. The CAdaptiveHuffmanTextDecomp object class is used to decompress text data with the adaptive Huffman decompression algorithm shown in figure 4.11.

4.3.3 Use-Case Models

4.3.3.1 DCD System Use-Case Model

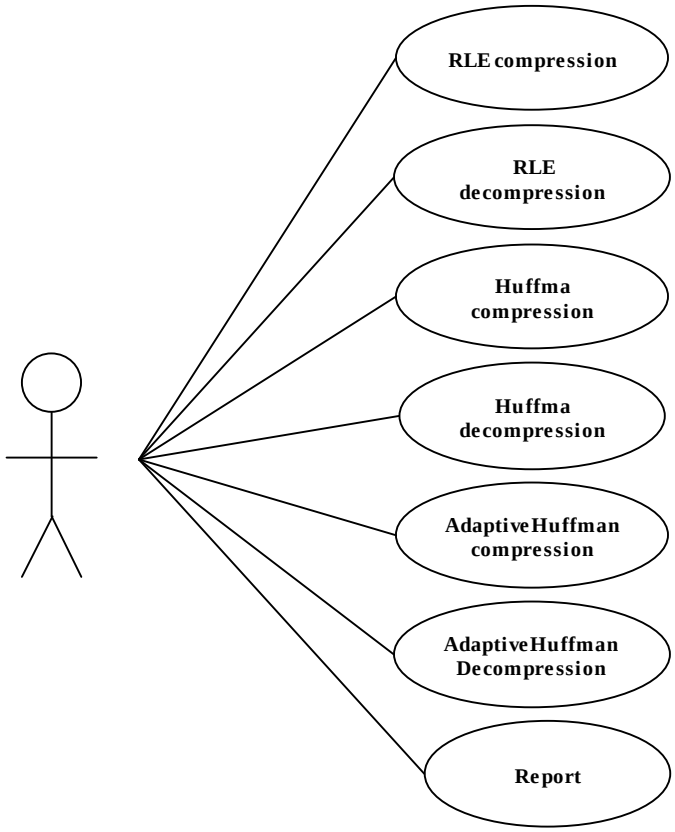


Figure 4.13 DCD Use-Case

4.3.3.2 RLE Compression Use-Case Description

System	Data Compression Decompression (DCD)
Use-Case	RLE compression
Actors	the selected file (text or image)
Data	The user will select the file (text or image) and the RLE algorithm will compress that file
Stimulus	The data repetition in the selected file will change the result
Response	Compressed file
Comments	The result change according to the data repetition in the selected file

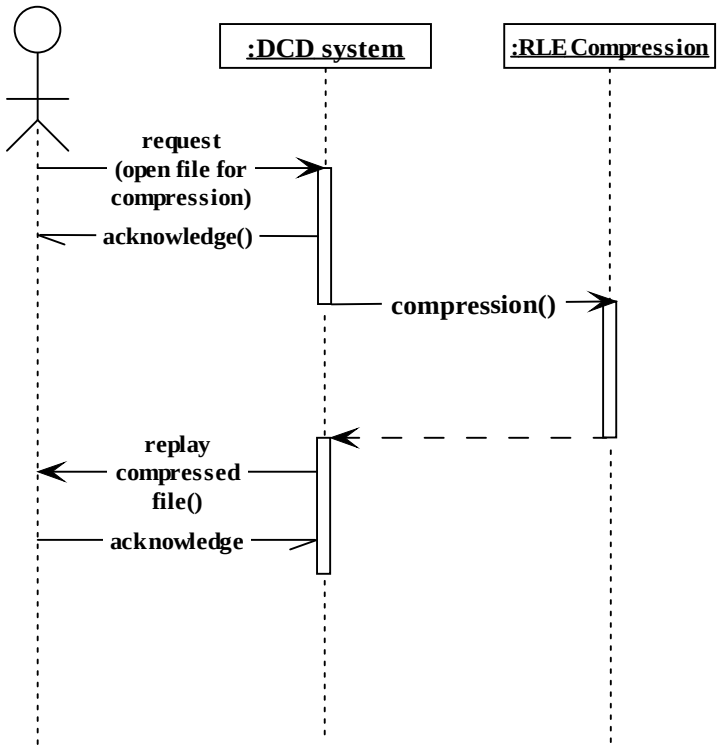


Figure 4.14 Collaboration Diagram for RLE Compression

4.3.3.3 RLE Decompression Use-Case Description

System	Data Compression decompression (DCD)
Use-Case	RLE decompression
Actors	The selected compressed file
Data	The user will select the compressed file and the RLE algorithm will decompress that file
Stimulus	The data repetition in the selected file will change the result
Response	The original file
Comments	The result changes according to the data repetition in the selected file

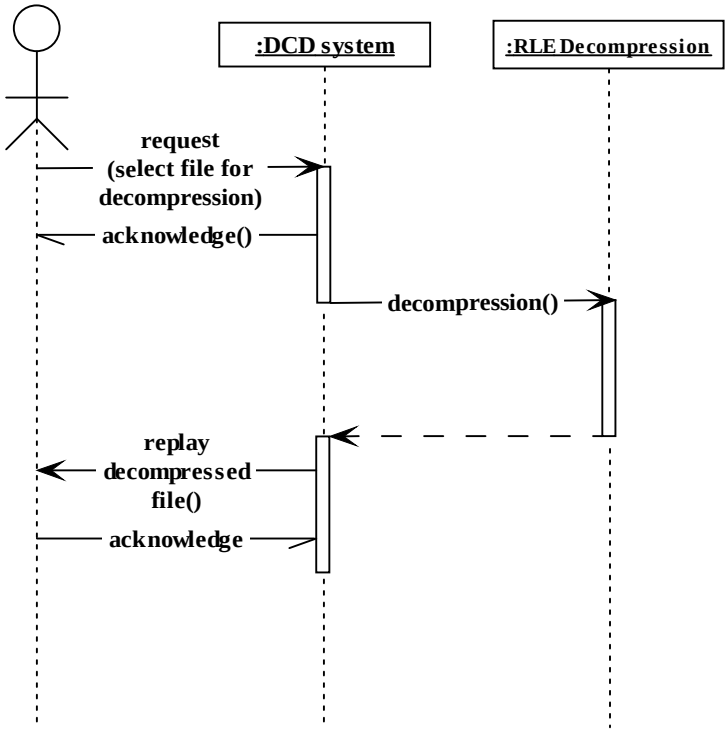


Figure 4.15 Collaboration Diagram for RLE Decompression

4.3.3.4 Huffman Compression Use-Case Description

System	Data Compression decompression (DCD)
Use-Case	Huffman compression
Actors	The selected file (text or image)
Data	The user will select the file (text or image) and the Huffman algorithm will compress that file
Stimulus	The data frequency in the selected file will change the result
Response	Compressed file
Comments	The result changes according to the data frequency in the selected file.

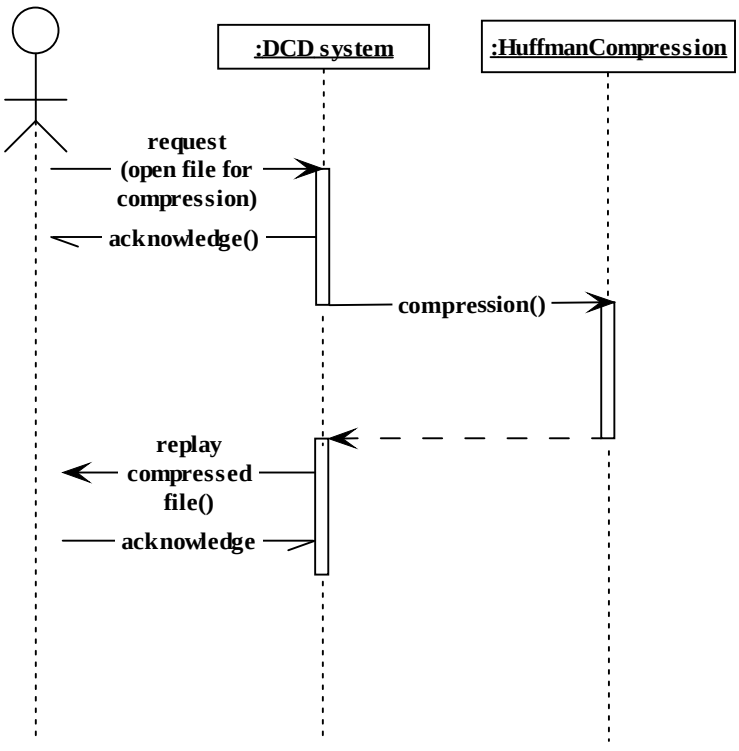


Figure 4.16 Collaboration Diagram for Huffman Compression

4.3.3.5 Huffman Decompression Use-Case Description

System	Data Compression decompression (DCD)
Use-Case	Huffman decompression
Actors	The selected compressed file (text or image)
Data	The user will select the compressed file and the Huffman algorithm will decompress that file
Stimulus	The data frequency in the selected file will change the result
Response	the original file
Comments	The result change according to the data frequency in the selected file

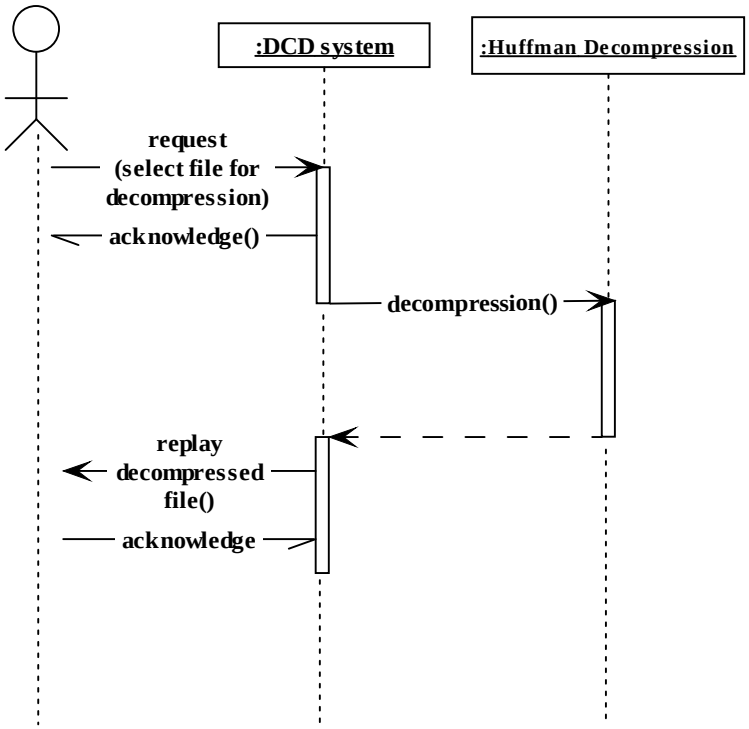


Figure 4.17 Collaboration Diagram for Huffman Decompression

4.3.3.6 Adaptive Huffman Compression Use-Case Description

System	Data Compression decompression (DCD)
Use-Case	Adaptive Huffman compression
Actors	the selected file (text or image)
Data	The user will select the file(text or image) and the Adaptive Huffman Will compressed that file
Stimulus	the data frequency in the selected file will change the result
Response	Compressed file
Comments	The result change according to the data frequency in the selected file

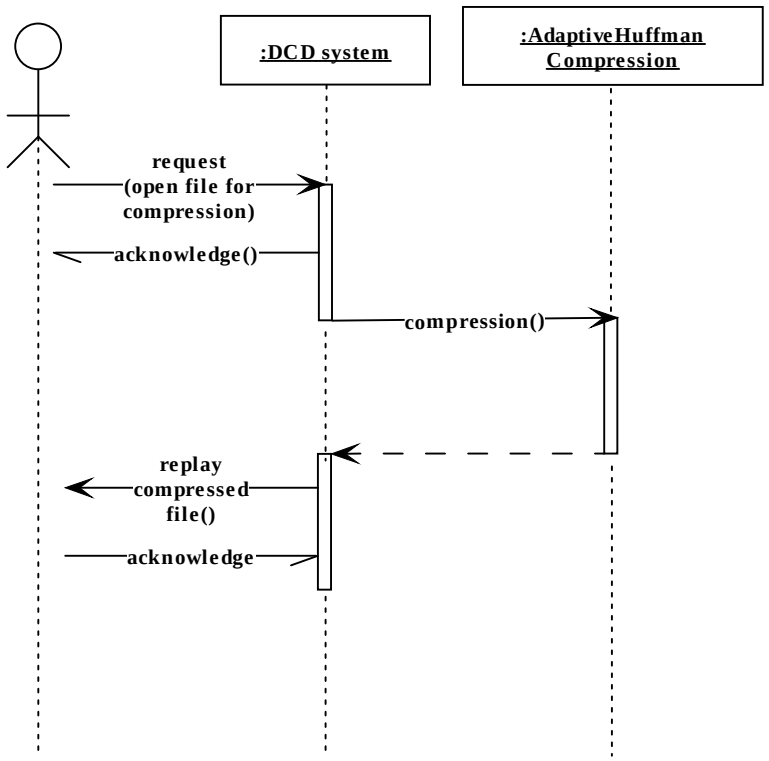


Figure 4.18 Collaboration Diagram for Adaptive Huffman Compression

4.3.3.7 Adaptive Huffman Decompression Use-Case Description

System	Data Compression decompression (DCD)
Use-Case	Adaptive Huffman decompression
Actors	the selected compression file (text or image)
Data	The user will select the compression file(text or image) and the Adaptive Huffman will decompressed that file
Stimulus	the data frequency in the selected file will change the result
Response	the original file
Comments	The result change according to the data frequency in the selected file

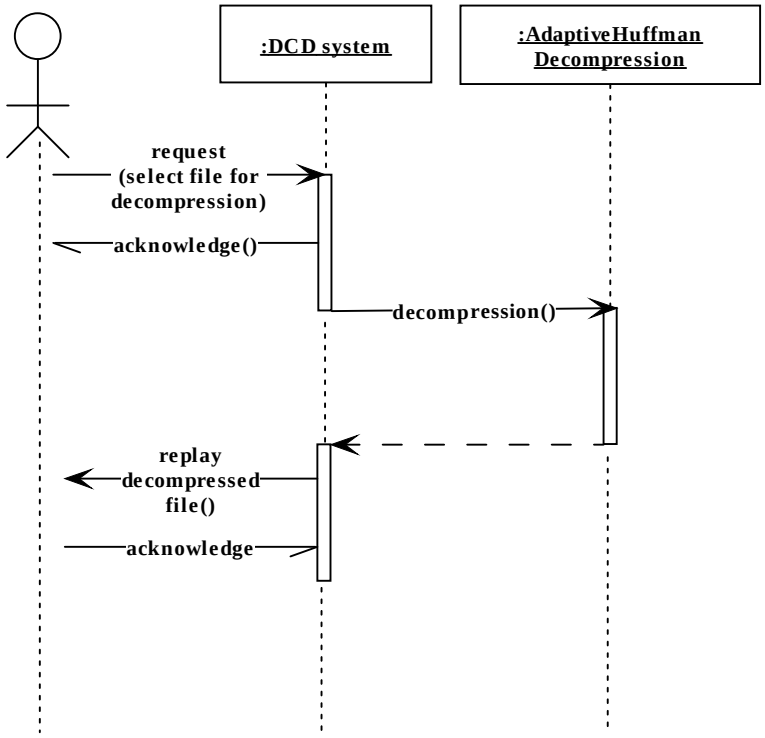


Figure 4.19 Collaboration Diagram for Adaptive Huffman Decompression

4.3.3.8 Report Use-Case Description

System	Data Compression decompression (DCD)
Use-Case	Report
Actors	Data compression algorithm, statistical information
Data	Statistical class sends report contains the compression ratio, file proprieties and data quality
Stimulus	The selected algorithm will change the result
Response	Statistical information will be given contains compression ratio, file proprieties and data quality
Comments	The result change according to the selected algorithm and file

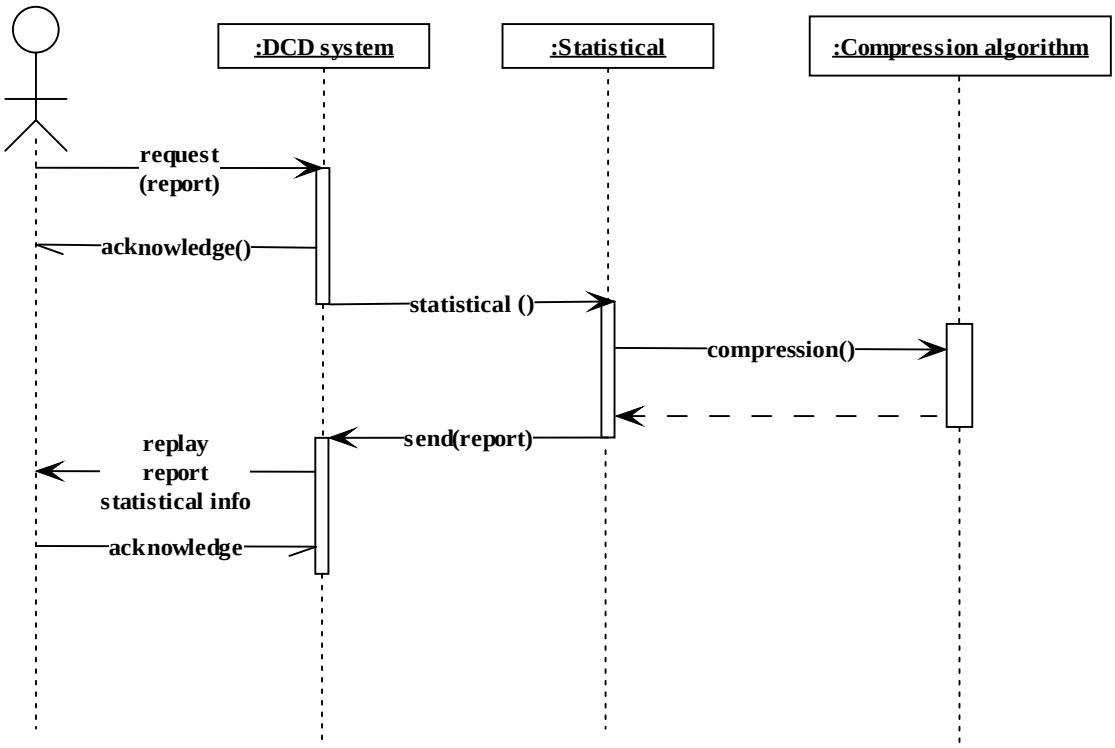


Figure 4.20 Collaboration Diagram for Report Providing

4.4 Object Interface Specification

This section specifies the interface between the different components in the design, so that objects and other components can be designed in parallel.

```
Interface DCD {
    Public void DCD();
    Public void RLECompression(File  F);
    Public void HuffmanCompression(File  F);
    Public void Report();
    Public void RLEDecompression();
    Public void RLEDecompression(File  F);
    Public void HuffmanDeompression();
    Public void HuffmanDeompression((File  F);
}// DCD
```

4.5 System Interface Design

This section shows how DCD's user interface will look like. It shows all of the screens that the user will interact with them.

1) DCD Main Form

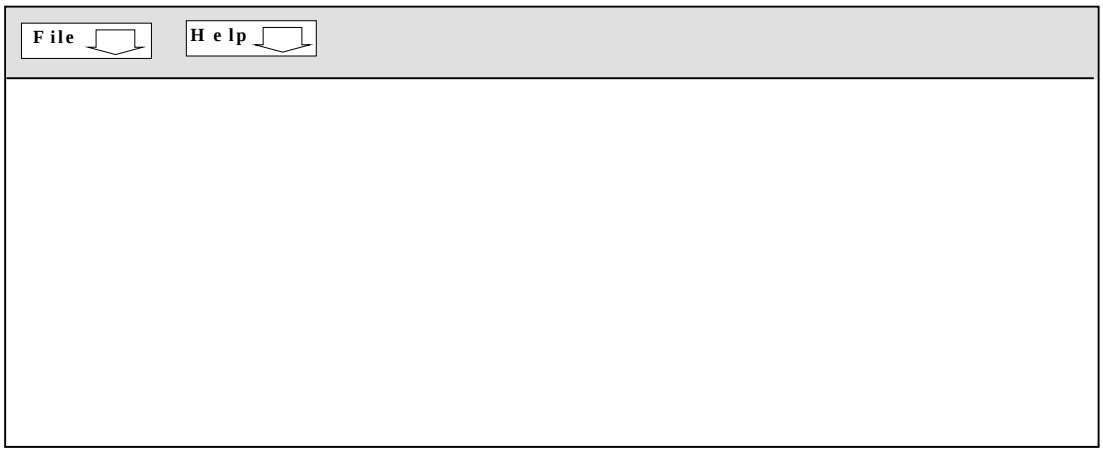


Figure 4.21 DCD Main Form

This form allows the user to select Text File Form or Image File Form from File menu, and allows the user to view About DCD Form from Help menu.

2) Text File Form

This form allows the user to select desired text file to compress it by one of the algorithms, decompress compressed file, view file word count, view statistical information, and get more information about compression algorithms.

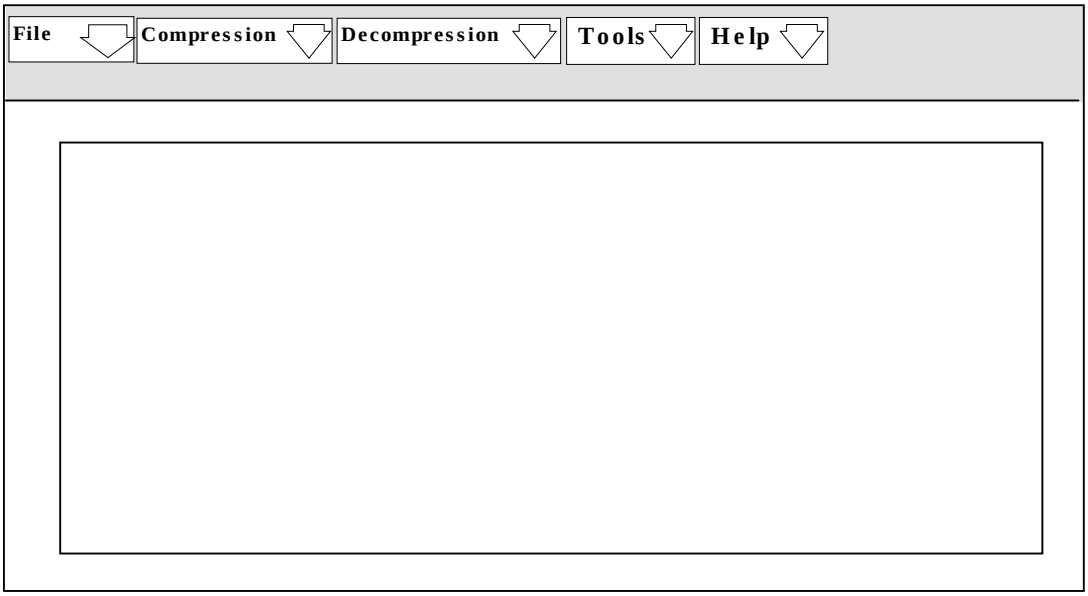


Figure 4.22 Text File Form

3) Image File Form

This form allows the user to select any image file and make compression, decompression, view image properties and view statistical information.

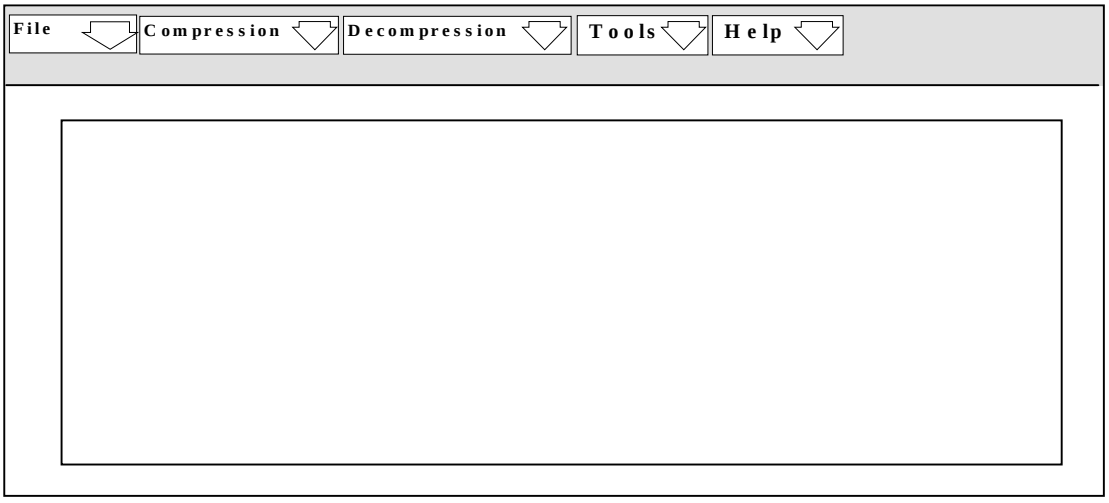


Figure 4.23 Image File Form

4) Word Count Form

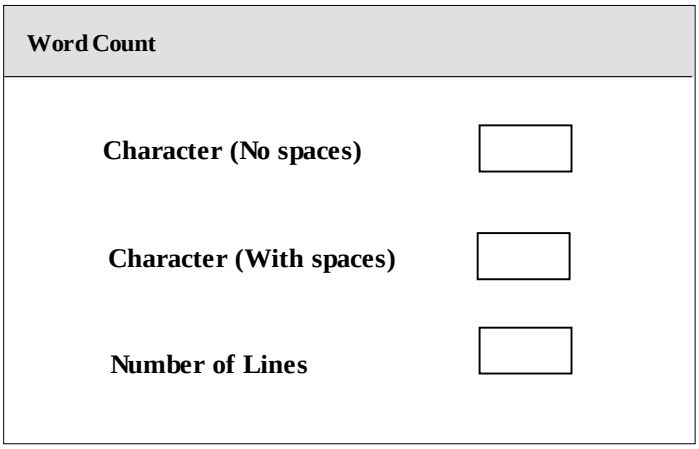


Figure 4.24 Word Count From

This form shows text file properties include the number of characters with spaces, without spaces and the number of lines in the file.

5) Text Statistical Information Form

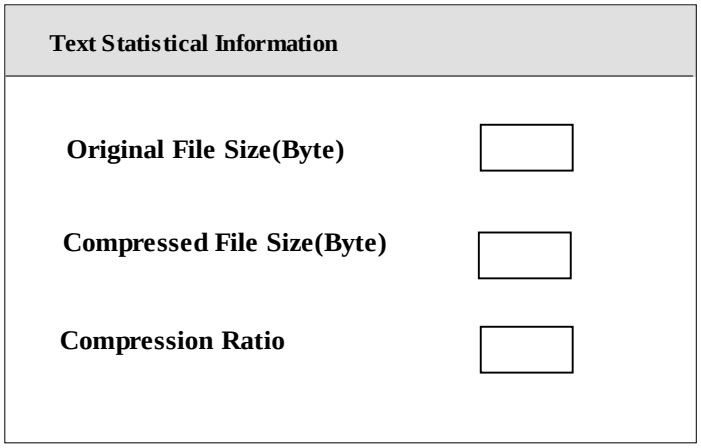


Figure 4.25 Text Statistical Information Form

This form shows the text statistical information includes the original file size, file size after the compression, and compression ratio

6) Image Properties Form

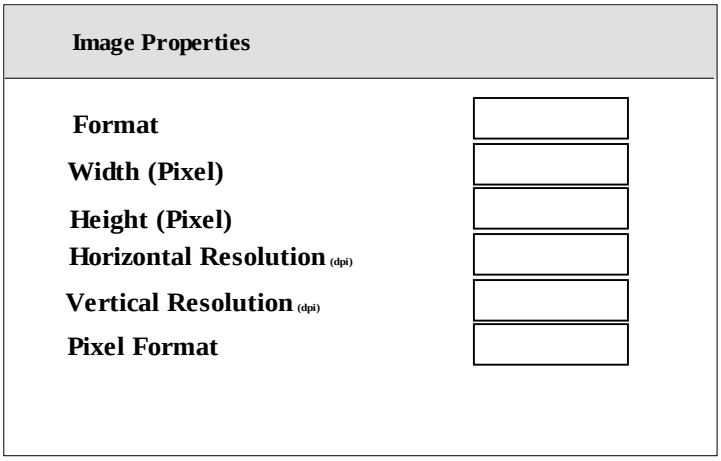


Figure 4.26 Image Properties Form

This form shows properties include the image format, width, height, Resolution and pixel format for the selected image.

7) Image Statistics Information Form

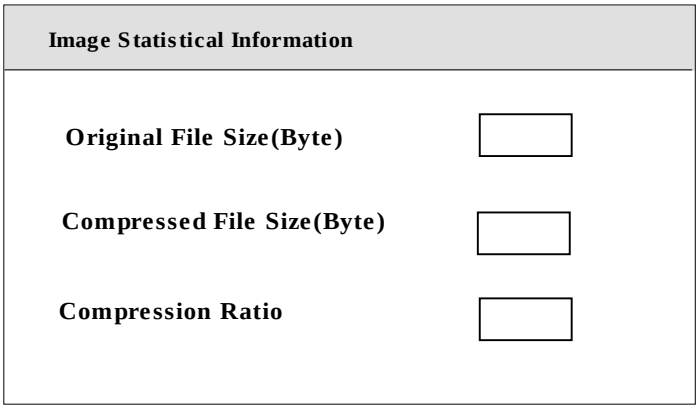


Figure 4.27 Image Statistics Information Form

This form shows image statistical information includes the original file size, file size after the compression, and compression ratio.

8) About DCD Form

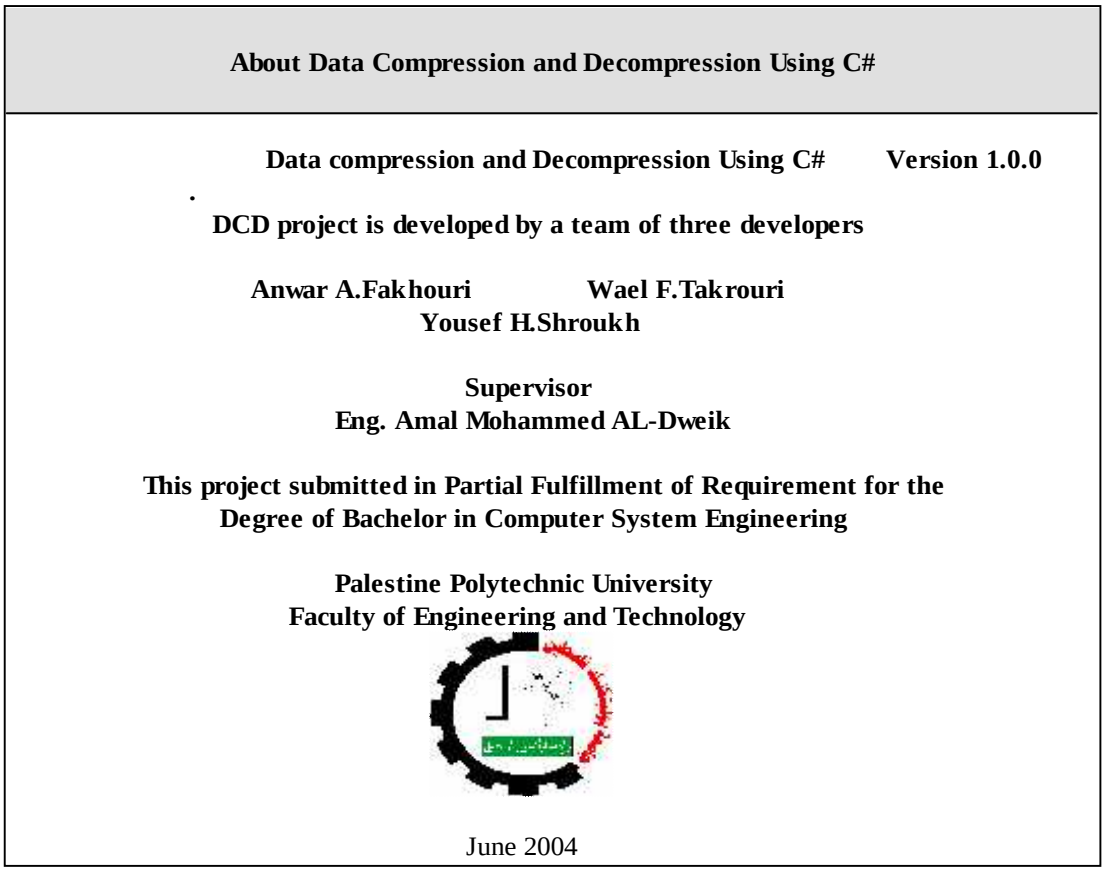


Figure 4.28 About DCD Form

This form shows information about the DCD Project includes the programmers name, supervisor name, and the university and college name.

Chapter Five

Implementation

5.1 Preface

5.2 Algorithms

5.3 System Pseudo code

5.4 System Flow Chart

5.5 System Results

5.6 Conclusions

5.1 Preface

This chapter describes algorithms used in the DCD system and features of each algorithm, it presents Pseudo code and flow chart of each algorithm, the results after implementing algorithms on selected data will be provided, finally resulted data will be analyzed to get conclusion.

5.2 Algorithms

5.2.1 Run Length Encoding

Run Length Encoding (RLE) which is one of the lossless compression algorithms that reduces the repeated data sequence after a level of occurrence.

RLE encounters a repeated data sequence that is above a level of occurrence it will reduce this occurrence by replacing them with a special character (compression indicator character) followed by a one of the repeating data and finally the count which represent the number of times the repeated data occurred in the data sequence.
The selection of the special character should be based on the fact that it will not occurred in the data stream, which makes it easy to refer to the compressed data and also make it easy when we try to decompress the data.

Sc	X	Count

Where Sc: the special character.

X: the repeated data.

Count: the number of times the repeated data occurred in data sequence.

Description of RLE Compression

1- The algorithm will start to read the content of text file until it reaches the end of file
2- In the first reading the read data will be considered as temporary data, and the comparisons will be applied between the temporary data and the next read data.
3- If the next data differs from the temporary data, the temporary data will be written to the compressed file, and the new data will be considered as temporary data.
4- If they are the same then the count which represent the number of repetition times will be incremented by one, the compressed data will be written to compress file when a different data is read or when the count reaches 255.
5- The compressed format is same as the previous figure, with the special character (ESC with ASCII code 27_{10} and $1B_{16}$).
6- The compression output will be written to *filename*.data which represents a binary file.

Notes:

1. Text files
    A) We deal with characters size of each one byte, because the compression format takes three bytes so the repeated data must be four or more to get efficient reduction of data.
    B) If the repeated data is more than 255(1 byte), we compress the first 255 repeated data separately then deal the next also separately, because the compression format takes only 3 bytes, one for count value.

2. Image files

   A) We deal with pixels size of each three bytes (1 byte for RED, 1 byte for GREEN, 1 byte for BLUE).

   B) The compression format here is 5 bytes because each symbol in the repeated data takes 3 bytes.

   C) Because the compression format takes 5 bytes so the repeated data (Pixels) must be two or more to get efficient reduction of data.

   D) If the repeated data is more than 255(1 byte), we compress the first 255 repeated data separately then deal the next also separately, because the count value is one byte.

   E) Because image data contains all values from 0-255, so there is no a special character(byte) that will not occur in the data stream, so we choose a value(27) to indicate a special character and if found in the data replace it by 26 value.

Description of RLE Decompression

1- The algorithm start to read the content of compressed file until it reaches the end of the file (a file with the extension .data).
2- If the data is the special character then the algorithm read the next byte which represents the data, and read the next byte which represent the count, then it will decompress the compressed data to text file.
3- If the read data is not special character it will be written to the text file normally.

5.2.2 Huffman Encoding

Huffman encoding was introduced by prof. David A. Huffman (1925-1999), Huffman encoding is a lossless compression algorithm that is there is no loss of information after the data is compressed i.e. after we decompress the data, that original information will be available, lossless compression is desired while compressing text documents, bank records …etc.

The main idea behind Huffman encoding is to assign smaller codes (a sequence of zeros and ones) to frequently used data and longer codes for data that are less frequently used which provides considerable compression results, so the Huffman encoding is called variable length coding.

Implementation of Huffman encoding should use binary tree precisely complete binary tree, where the form of each node in the tree will as in the following figure.

Typical node of a Huffman Tree

Description of Huffman Compression

1- The algorithm will start to read the content of text file until it reaches the end of file.

2- Frequency table is build which contains all symbols and their frequency, and then the binary tree will be built starting from the leaves until reaching the root.
3- The code will be provided by starting from root of tree going on assigning 1 to every left branch and 0 to every right branch.
4- The code of each symbol can be determined by moving form the root to the desired symbol if traversing toward the left branch 0 is applied else 1 is applied.
5- Each symbol in the input file then will be replaced by the code generated by the tree.
6- The compressed file will be a binary file(file with extension .data)

Description of Huffman Decompression

1- Starting form the root of the tree that is generated by the compression process
2- The next input is read from the compressed file.
3- If the input equals to 1, then move to the left child of the tree.
4- If the input equals to 0, then move to the right child of the tree.
5- If the node is leaf, then output the symbol and return to step number 1
6- If the node is not leaf, then go to step number 2.

5.2.3 Adaptive Huffman Encoding

Adaptive Huffman or Dynamic Huffman encoding introduced to solve the problems or disadvantages in Huffman encoding, where in adaptive Huffman there is no need to know the symbol frequency table in advance.

Description of Adaptive Huffman Compression

1- The algorithm will start to read the content of text file until it reaches the end of file.
2- Every symbol are initially has frequency of 1, and the binary tree is formed.
3- After symbol is encoded its frequency incremented by one and the tree is rebuilt, and from the newly updated tree the new symbol decoded, the process continues until all symbols are encoded.
4- As the frequency of symbol increase the bits needed to decode the symbol decreases.

Description of Adaptive Huffman Decompression

1- The algorithm will build binary tree same as in the compression step where the frequency of all symbols equals to one.
2- Starting form the root of the tree.
3- The next input from the compressed file will be examined.
4- If the input equals to zero, then move to the left child node, if not where it will be one then move to the right child node.
5- If the node is child, then the symbol will be output to the decompressed file and the frequency symbol will be incremented by one, and the tree will be updated, then continue to step number 2 until the end of file is reached.
6- If it is not leaf node, then continue to step number 3.

Note: Why using file type binary or the file with ***extension .data*** in writing compressed data.

Binary file or the file with the extension .data allows data to be read and written as binary, which allows the value of count to be written as one byte and therefore when the file will be decompressed it will be easier to read the count as a byte rather than a sequence of character in the case of Text file.

5.3 System Pseudo code

5.3.1 RLE Compression Pseudo Code

```
Step1: Read symbol from the file
Step2:  Set count to zero
Step3:   Set symbol as temp variable
Step4:  Repeat
            Read next symbol
            If symbol same as temp and count less than 255
                    Increment count
             Else
                Begin
                    If count   less than 4 (2 for image data)
                                Write temp to compressed file
                    Else
                        Begin
                            Set compression format
                                Write compressed data to the file
                          End
                End
   Step5: Until reach end of the file
```

5.3.2 RLE Decompression Pseudo Code

Step1: Repeat
Step2: Read a symbol from compressed file
Step3: If read symbol equals to special character
    Read symbol
    Read count
    Repeat
        Output symbol to decompressed file
        Decrement count
Step4: Until count equals one
Step5: Else
    Output symbol to decompressed file
Step5: Until reach end of file

5.3.3 Huffman Compression Pseudo Code

Step1: Read the different symbols from the file
Step2: Determine symbols frequency
Step3: Arrange frequencies in ascending order
Step4: Repeat
    Assign 0 to 2nd smallest frequency
    Assign 1 to smallest frequency
    Combine the two smallest frequencies
    Arrange frequency in ascending order
Step5: Until all frequencies are combined
Step6: Generate code for each symbol
Step7: Write the code for each symbol to compressed file

5.3.4 Huffman Decompression Pseudo Code

Step1: Read the compressed file

Step2: Start from the root of the tree

Step3: Examine the next element in the input

Step4: If it is 1, move to the left child.

Step5: If it is 0, move to the right child.

Step6: If it is leaf node

Output its symbol

Go to step2

Step7: If it is not a leaf node then go to step3

5.3.5 Adaptive Huffman Compression Pseudo Code

Step1: Read the different symbols from the file

Step2: Initialize frequency to one for each symbol

Step3: Build tree

Step4: Read next symbol

Step5: Code symbol

Step6: Increment count of read symbol

Step7: Go to step 3 until reach end of file

5.3.6 Adaptive Huffman Decompression Pseudo Code

Step1: Make initial Huffman tree having frequency count of all symbols 1.

Step2: Start from the root of the tree.

Step3: Examine the next element in the input

Step4: If it is 1, move to the left child.

Step5: If it is 0, move to the right child.

Step6: If it is leaf node then

Output its symbol.

Increment its count by 1.

Update the Huffman tree.

Go to step 2

Step7: If it is not a leaf node then go to step 3

5.4 System Flow Charts

5.4.1 RLE Compression Flow Chart

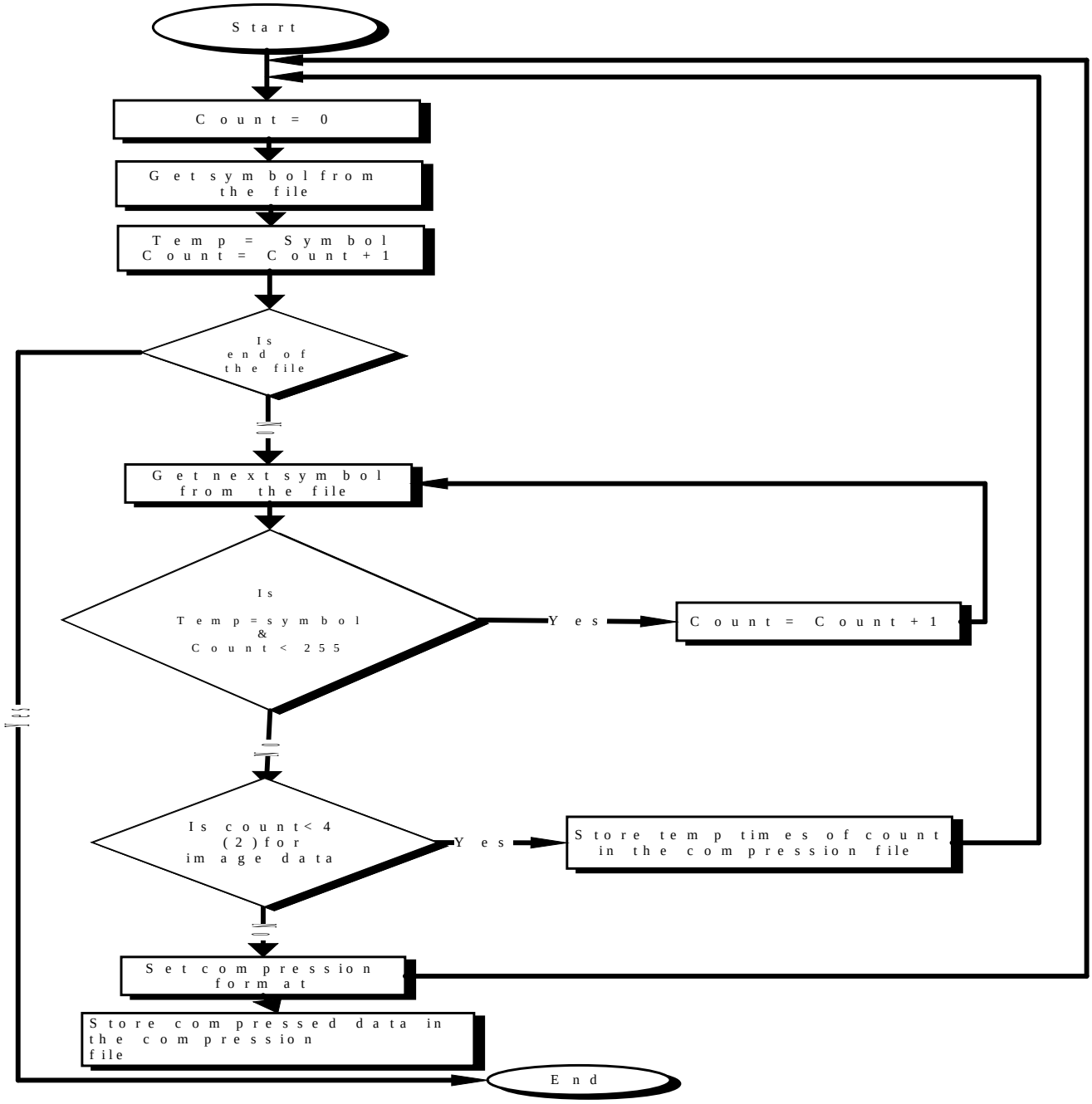


Figure 5.1 RLE Compression Flow Chart

5.4.2 RLE Decompression Flow Chart

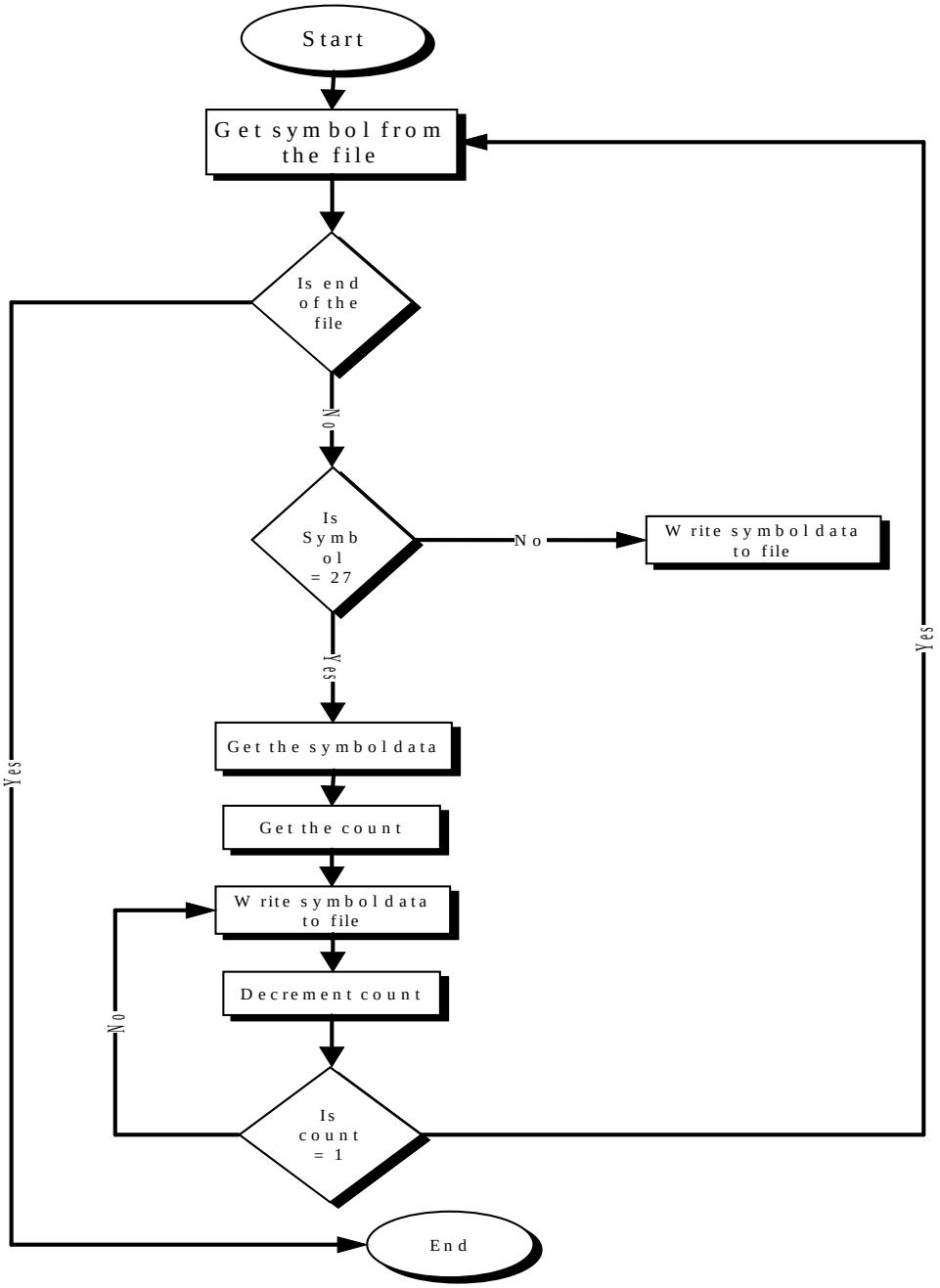


Figure 5.2 RLE Decompression Flow Chart

5.4.3 Huffman Compression Flow Chart

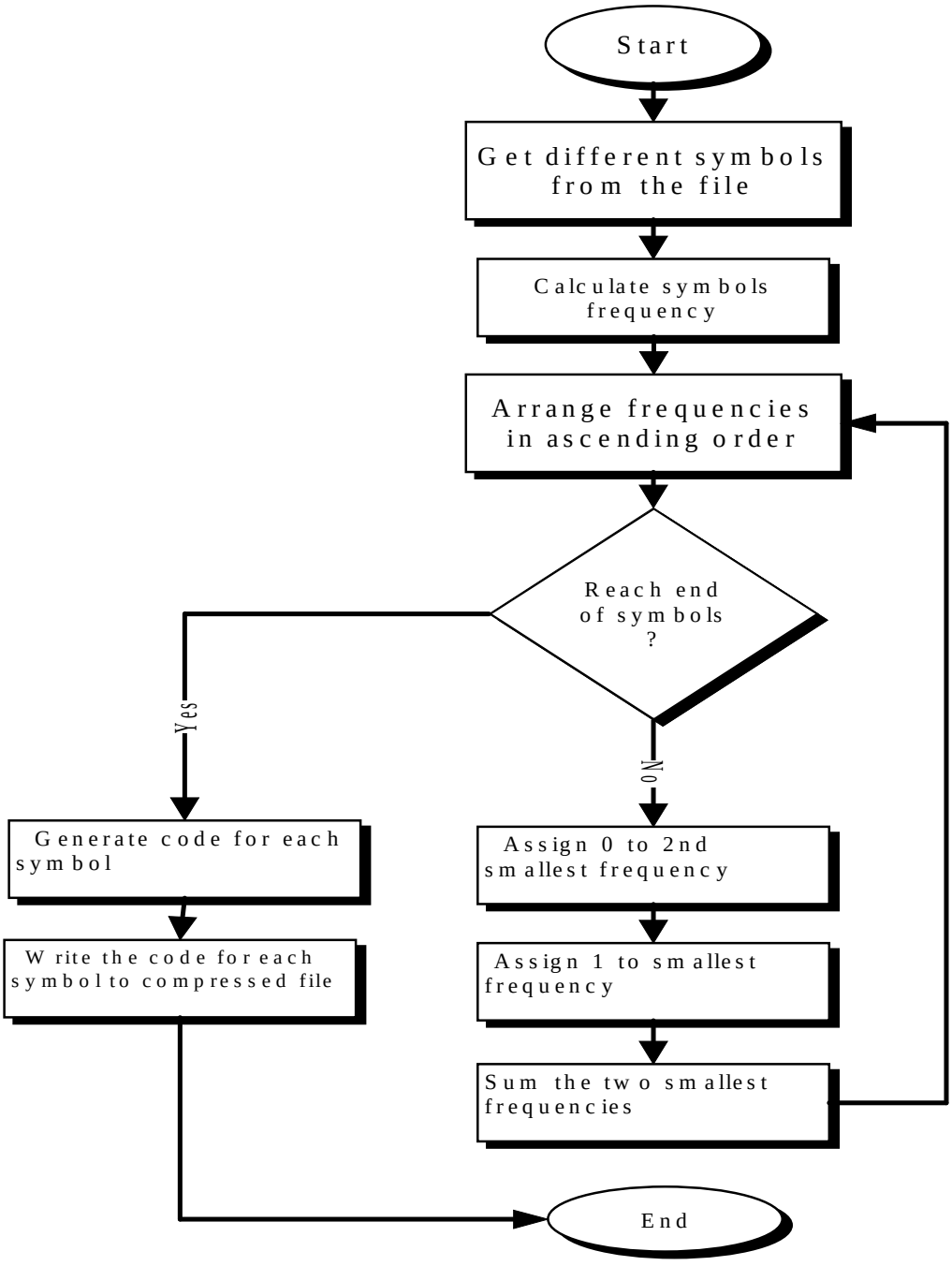


Figure 5.3 Huffman Compression Flow Chart

5.4.4 Huffman Decompression Flow Chart

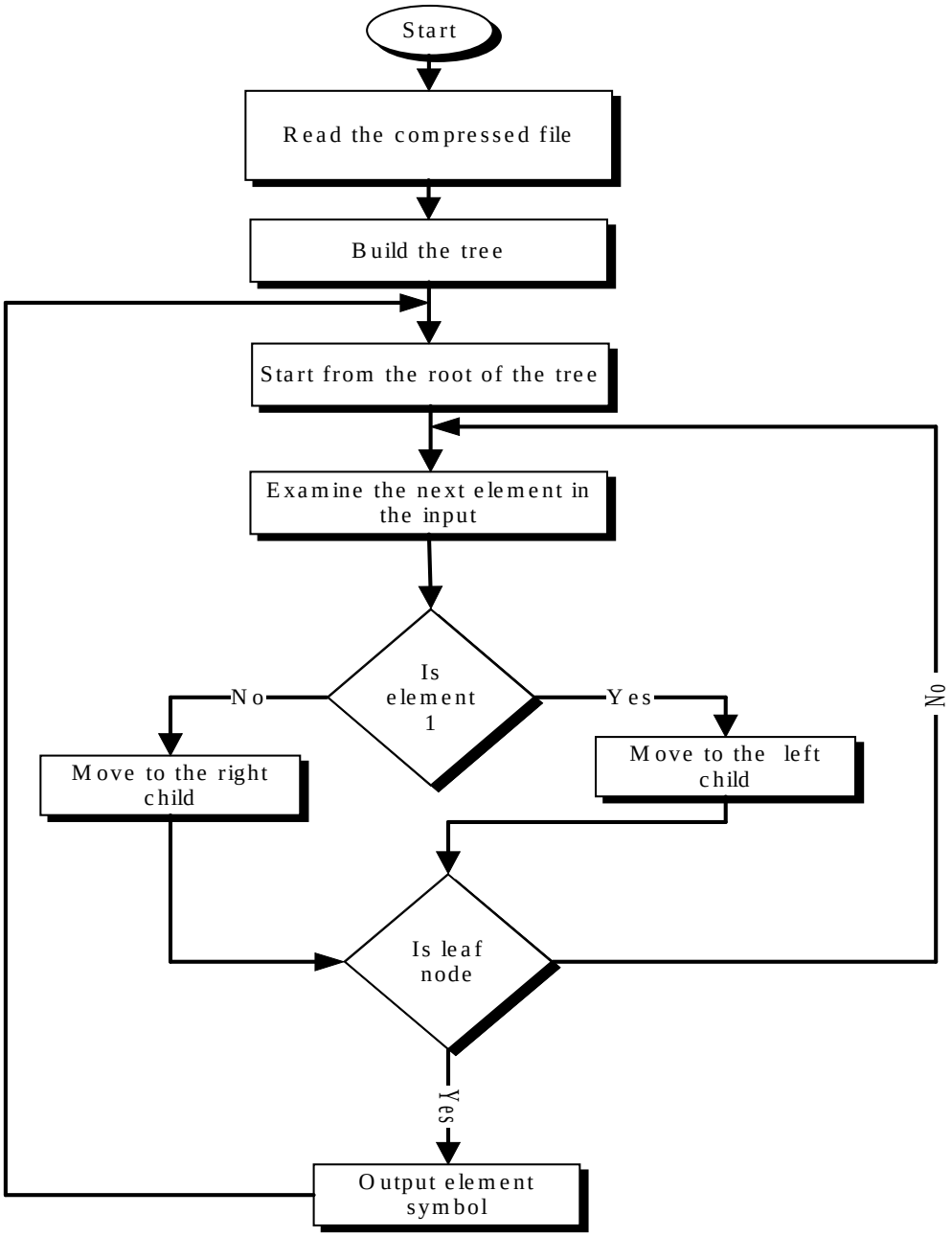


Figure 5.4 Huffman Decompression Flow Chart

5.4.5 Adaptive Huffman Compression Flow Chart

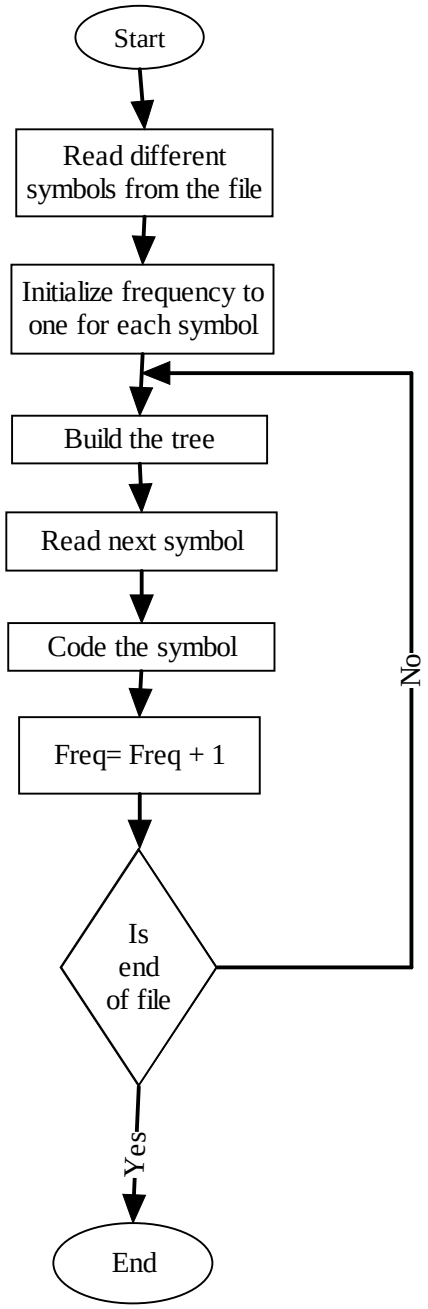


Figure 5.5 Adaptive Huffman Compression Flow Chart

5.4.6 Adaptive Huffman Decompression Flow Chart

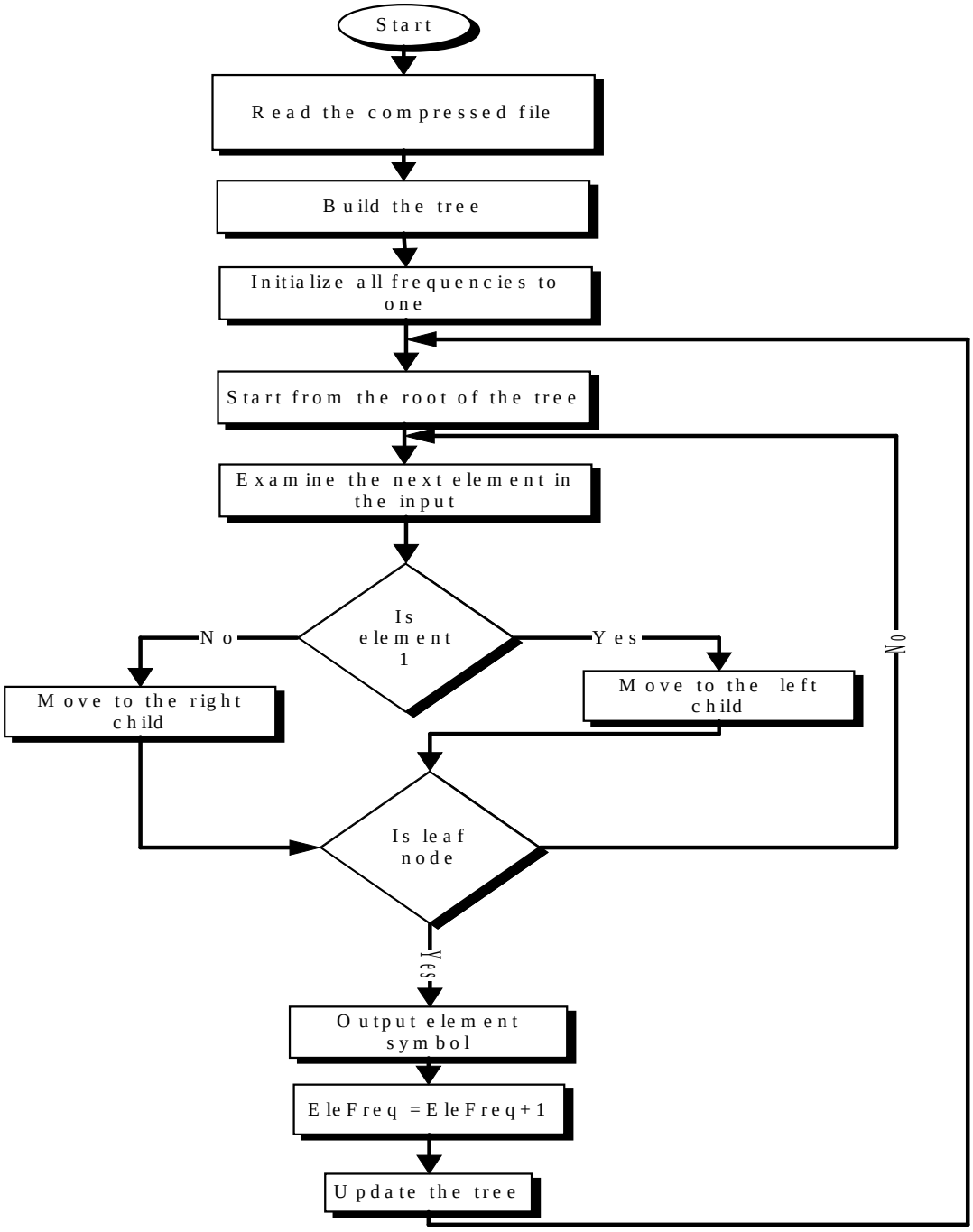


Figure 5.6 Adaptive Huffman Decompression Flow Chart

5.5 System Results

This section provides statistical information on different algorithms that applied to different data files to examine algorithms efficiencies.

5.5.1 RLE Text

A) Best Case Sample

```
dddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
dddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
dddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
ddddddddddddddddddddddddddddddddddddddd
```

Analysis:

The data includes one symbol repeated 255 times, so the compression format is (27 d 255) where:

27: indicates the special character.

d: The repeated symbol

255: Count of repeated data.

So:

- The compressed data is three bytes.
- The best case happens when there is one symbol in the data and repeated 255 times (Count value stores in bytes 255 = 1 byte) and that rarely occurs in text files.

B) Average Case Sample

```
This is software project DCD using C#,which implements RLE,Huffman,Adaptive


palestine polytechnic university.






hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhiiiiiiii
iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
```

Analysis:

The sample shows that:

- The first 128 symbol has no repetition so it has no compression

- Next the data contains the symbol (h) repeated 69 times, so it compressed to three bytes (27 h 69).

- Finally the data contains the symbol (i) repeated 58 times, so it compressed to three bytes (27 i 58).

- So the compressed total data is 128 + 3 + 3 = 134 bytes.

The result indicates that the average case happens when the data contains a number of repeated data and some without repetition, this case occurs normally when the data contains a lot of spaces between the words.

C) Worst Case Sample

This is software project that uses C# as development environment

it implements RLE,Huffman and Adaptive compression algorithms

developed by a team of three computer systems engineering:

Anwar A. Fakhouri
Wael F. Takrouri
Yousef H. Shroukh

Analysis:

Sample shows that the data has no repetition so the output data is the same as the original data. So worst cast happens when the data has no any repetition, this case occurs in normal data, data normally has no repetition, so the compressed data is the same as the original data.

Case/Size	**Original text file (Byte)**	**Compressed file (Byte)**	**Compression ratio**
Best	255	3	= 255 : 3 85 : 1
Average	255	134	= 255 : 134 19 : 10
Worst	255	255	= 255 : 255 1 : 1

Table 5.1 RLE Text Statistics

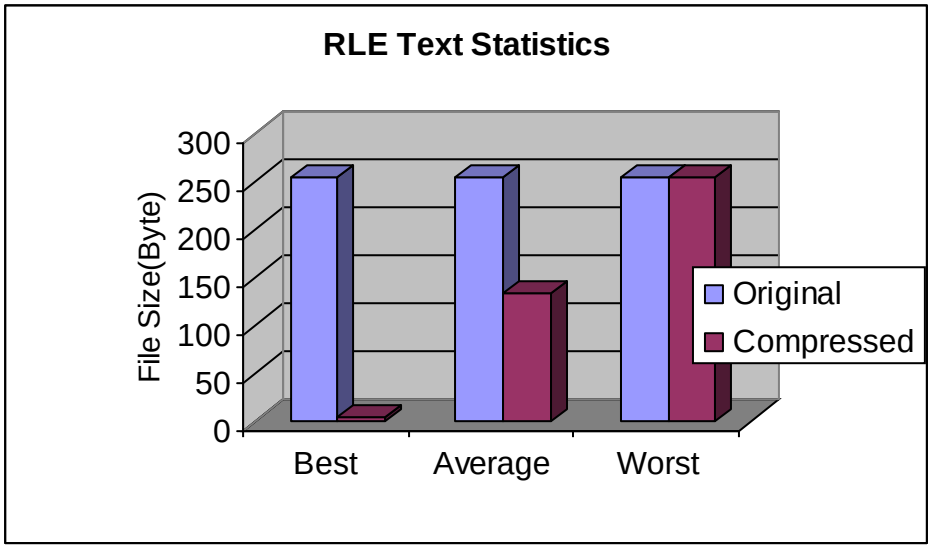


Figure 5.7 RLE Text Statistics

5.5.2 RLE Image

1) BMP

A) Best Case Sample

Dimensions: 312 x 132

Figure 5.8 RLE (BMP) Best Case

Analysis:

-The original data size is 245340 bytes, the compressed data is 1664 bytes.

- The bitmap header = 54 bytes.

- So the image data is 245340 – 54 = 245286

- Each pixel has 24 bits (3 bytes).

- So the image data has 245286 / 3 = 81762 pixels

- Since the image has on symbol (pixel) is red so all image data is repeated but the system deals every 255 repeated data separately so it has a number of repeated sequences equals to:

81762 / 255 = 321

- So the compressed data size equals

54 bytes (bitmap header) + 321(sequence) * 5(compression format) = 1659 bytes

B) Average Case Sample

Dimensions: 800 X 600

Figure 5.9 RLE (BMP) Average Case

Analysis:

Sample shows that the image data has a small number of colors, so there is also a lot number of repeated data, so it has a medium compression ratio.

C) Worst Case Sample

Dimensions: 256 X 256

.

Figure 5.10 RLE (BMP) Worst Case

Analysis:

Sample shows that the image data has a lot number of colors, so it has rarely repetitive data, so it has a small compression ratio.

Case/Size	**Original text file (Byte)**	**Compressed file (Byte)**	**Compression ratio**
Best	245340	1664	245340 : 1664 = 147 : 1
Average	479436	13167	479436 : 13167 = 36 : 1
Worst	391788	196454	391788 : 19788 = 2 : 1

Table 5.2 RLE Image (BMP) Statistics

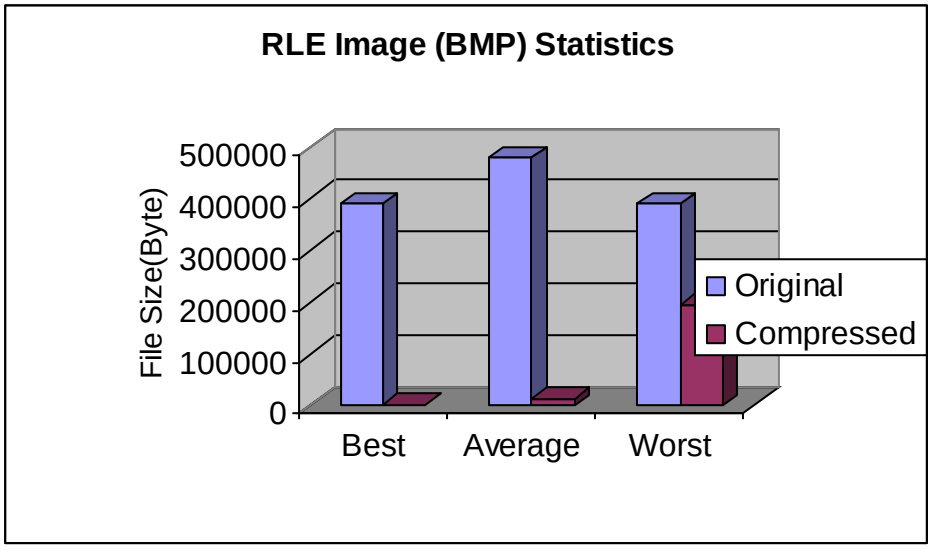


Figure 5.11 RLE Image (BMP) Statistics

2) JPG

A) Best Case Sample

Dimensions: 800 X 600

Figure 5.12 RLE (JPG) Best Case

Analysis:

-The original data size is 2875308 bytes, the compressed data is 18854 bytes.

- The bitmap header = 54 bytes.
- So the image data is 2875308 – 54 = 2875254
- Each pixel has 24 bits (3 bytes).
- So the image data has 2875254 / 3 = 958418 pixels
- Since the image has on symbol (pixel) is green so all image data is repeated but the system deals every 255 repeated data separately so it has a number of repeated sequences equals to:

958418 / 255 = 37581

- So the compressed data size equals

54 bytes (bitmap header) + 37581(sequence) * 5(compression format) = 18849 bytes

B) Average Case Sample

Dimensions: 750 X 750

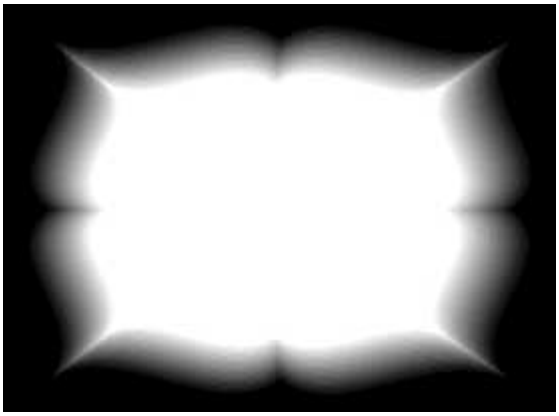


Figure 5.13 RLE (JPG) Average Case

Analysis:

Sample shows that the image data has a small number of colors, so there is also a lot number of repeated data, so it has a medium compression ratio.

C) Worst Case Sample

Dimensions: 800 X 600

Figure 5.14 RLE (JPG) Worst Case

Analysis:

Sample shows that the image data has a lot number of colors, so it has rarely repetitive data, so it has a small compression ratio.

Case/Size	**Original text file (Byte)**	**Compressed file (Byte)**	**Compression ratio**
Best	2875308	18854	2875308 : 18854 = 153 : 1
Average	1128652	203493	1128652 : 203493 = 11 : 2
Worst	2875308	1280723	2875308 : 1280723 = 22 : 10

Table 5.3 RLE Image (JPG) Statistics

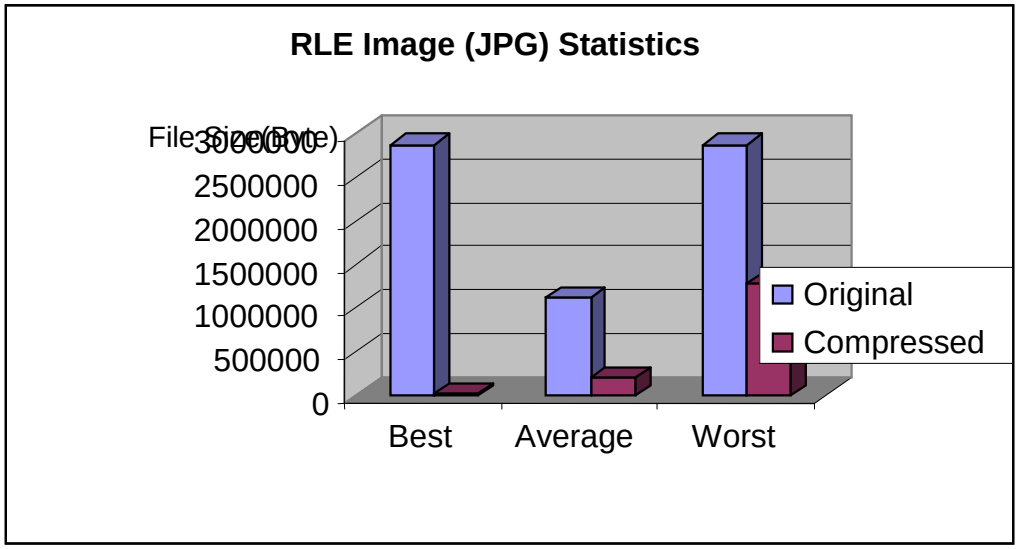


Figure 5.15 RLE (JPG) Statistics

3) GIF

A) Best Case Sample

Dimensions: 360 X 216

Figure 5.16 RLE (GIF) Best Case

B) Average Case Sample

Dimensions: 200 X 48

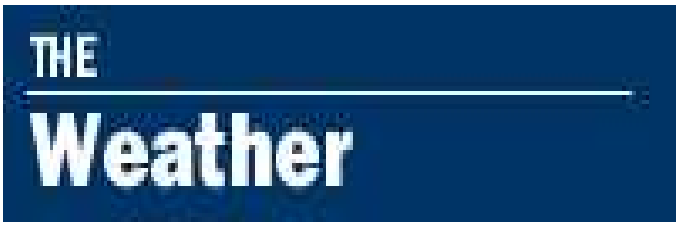


Figure 5.17 RLE (GIF) Average Case

B) Worst Case Sample

Dimensions: 125 X 90

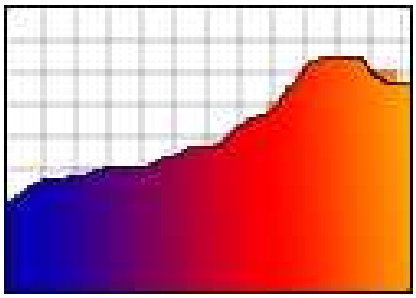


Figure 5.18 RLE (GIF) Worst Case

Case/Size	**Original text file (Byte)**	**Compressed file (Byte)**	**Compression ratio**
Best	131072	944	131072 : 944 = 139 : 1
Average	19164	1954	19164 : 1954 = 10 : 1
Worst	17248	11885	17248 : 11885 = 14 : 10

Table 5.4 RLE Image (GIF) Statistics

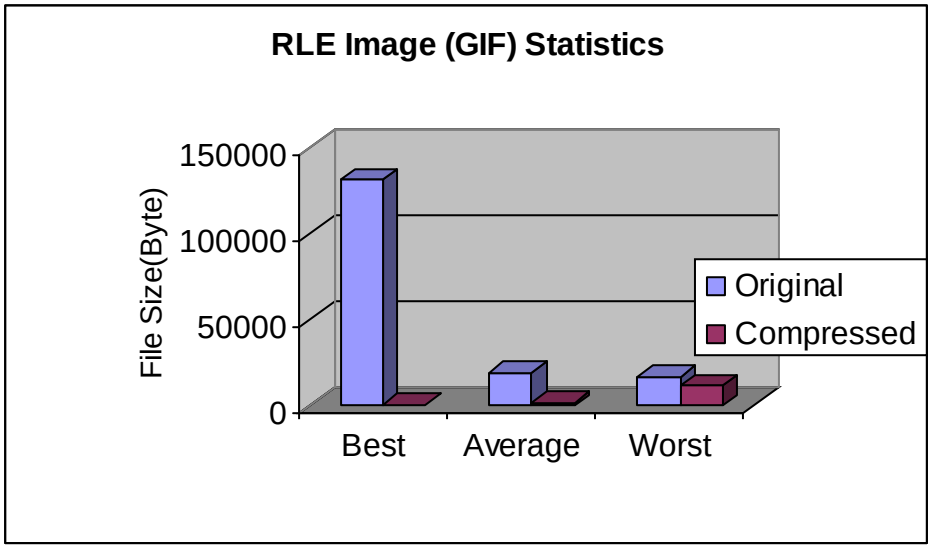


Figure 5.19 RLE Image (GIF) Statistics

Conclusions:

Table 5.2, 5.3, 5.4 shows the statistics for image data from it we see:

1) Best case for images occurs when the image data consist of one color so it is completely corresponds a repetitive data.

2) Average case occurs if the image data has a small number of colors, so there is also a lot number of repeated data.

3) Worst case occurs if the image data has a lot number of colors, so it has rarely repetitive data.

5.5.3 Huffman Text

A) Best Case Sample

dddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
dddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
dddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddddd
dddddddddddddddddddddddddddddddddddddddddddddddddddddd

B) Average Case Sample

This is software project

data compression and decompression using viusal c# which implements RLE,huffman

and adaptive huffman, the project developed a team of three
computer system engineers

anwar a fakhouri

wael f takrouri

yousef h shroukh

supervisor
eng. amal mohammad al dweik
palestine polytechnic university

college of engineering and technology

computer and electical department

one two three four five six seven eight nine ten eleven twelve
abcdefghijklmnopqrstuvwxyz

huffman codes are indeed very efficient
we use huffman codes in virtually every application which involves compression
of digital data huffman compression can compress data up to around ninty percent
huffman compression is implemented in computer networks, modems and fax machines
huffman compression is also used in multimedia applications, Various multimedia
standards like JPEG and MPEG use Huffman compression

data compression is one of the most renowned branches of computer science
over the years a lot of research has been done in this field to compress data
in numerous ways and many standards have been developed data compression can
be defined as reducing of the amount of storage space required to store
a given amount of data, data compression comes with a lot of advantages,
it saves storage space, bandwidth, cost and TIME required to transmit data
from one place to another, in this article we will throw light on Huffman
coding for data compression

C) Worst Case Sample

abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ123456789

Case/Size	**Original text file (Byte)**	**Compressed file (Byte)**	**Compression ratio**
Best	255	37	= 255 : 37 7 : 1
Average	1517	962	= 1517 : 962 3 : 2
Worst	61	171	= 61 : 171 1 : 3

Table 5.5 Huffman Text Statistics

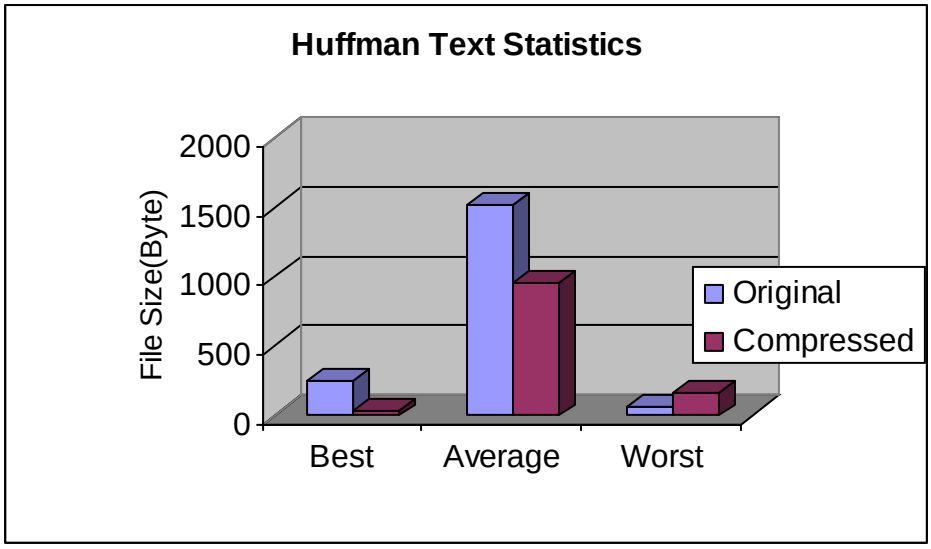


Figure 5.20 Huffman Text Statistics

Huffman Analysis:

For analysis Huffman code consider the following example which shows Symbol Frequency table of a text file containing a total of 100 characters.

Symbol	Frequency of occurrence
A	40
B	30
C	10
D	10
E	6
F	4

Table 5.6 Frequency Symbol Table

1) First we create the leaf nodes from the symbol frequency table and sort them. Symbols ‘f’ and ‘e’ have the least frequencies, 4 and 6 respectively; these 2 nodes are combined to make a node ‘fe’ having frequency 4+6=10.

2) This new node ‘fe’ is now the parent node of the nodes ‘f’ and ‘e’, and ‘fe’ replaces ‘f’ and ‘e’. Again we sort the nodes.

3) Now ‘fe’ and ‘c’ have least frequencies i.e. 10 each. This time we combine ‘fe’ and ‘c’ to create a new node ‘fec’ having frequency 20. Nodes ‘fe’ and ‘c’ are replaced to be their parent ‘fec’.

4) After sorting again, we now combine‘d’ and ‘fec’ to create ‘dfec’ having frequency 30, 10 of ‘d’ and 20 of ‘fec’. This ‘dfec’ becomes parent of ‘d’ and ‘fec’ and replaces both of them.

5) Now combines the nodes ‘dfec’ and ‘b’ as they have least frequency. New node will be ‘dfecb’ having frequency of 60, again ‘dfec’ and ‘b’ will be replaced by ‘dfecb’.

6) Now only two nodes are left namely ‘dfecb’ and ‘a’. We again sort them and combine both of them to form ‘adfecb’ which has frequency count of 100.

7) After making ‘adfecb’ parent of ‘a’ and ‘dfecb’ and replacing them with ‘adfecb’, we have created the Huffman tree for the symbols in Table 5.6. Node ‘adfecb’ is the root of the tree.

8) To assign codes to the tree created, start from the root of tree and go on assigning 1 to every left branch and 0 to every right branch.

9) To find the code for a particular symbol, start from that symbol and traverse in up direction towards the root. As soon as you encounter a branch output the code assigned

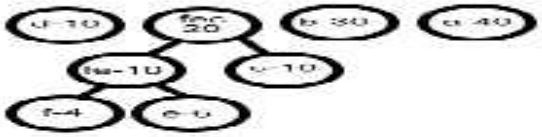
to that branch (1/0), when you reach the root you will have the code for that symbol from LSB to MSB. Suppose we need to find Huffman code for ‘f’, we start from ‘f’ and move

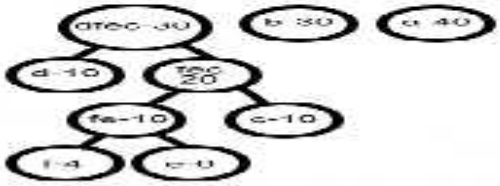
toward root i.e. f -> fe -> fec -> dfec -> dfecb -> adfecb; we get code 1, 1, 0, 1, 0 respectively (i.e. 1 for f->fe , 1 for fe -> fec, 0 for fec -> dfec and so on). Note that this

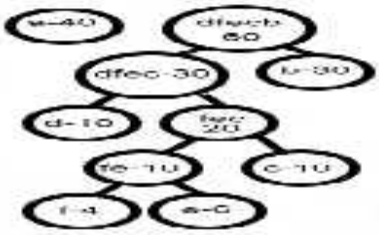
Huffman Tree Creation

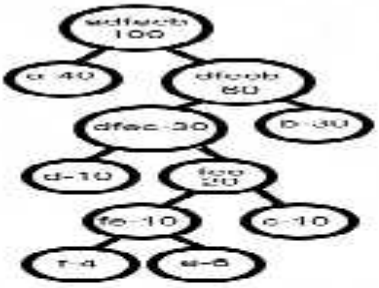
A)Create leaf nodes and sort them.

B) Combine nodes with least frequencies and again sort; repeat until only one node is left.

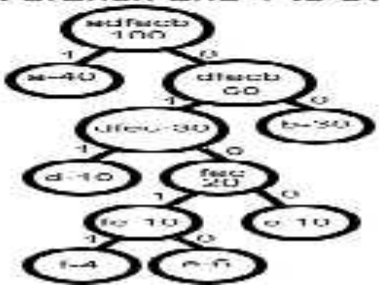








C) Assign code the Tree created, 0 to every right branch and 1 to every left branch



code that we have is from LSB to MSB so the final code for ‘f’ will be “01011”. table 5.7 shows Huffman code for all the symbols.

Symbol	Frequency of occurence	Huffman code
a	40	1
b	30	00
c	10	0100
d	10	011
e	6	01010
f	4	01011

Table 5.7 Huffman Code Table

Conclusion: if you notice carefully you will see that no codeword is also a prefix of another codeword. E.g. codeword for b is 00, now there is no other codeword which begins with 00. Codes having this property are called as Prefix codes. In prefix codes no codeword in the set is a prefix to another codeword. Huffman codes are Prefix codes. This property makes Huffman codes easy to decode as we will find later.

Now suppose we want to encode the following string using Huffman “abdceabedf”, 1 for ‘a’, 011 for ‘d’ and so on. Following table shows what will be encoded for each of the symbols:

a	b	D	c	E	a	b	e	D	F
1	00	011	0100	01010	1	00	01010	011	01011

So the compressed code will look like “1000110100010101000101001101011”. There are 10 characters and it takes only 31 bits to code them. If we use normal ASCII code i.e. 8 bit fixed length code then it will take 8*10=80 bits to code the same string. So in our case, use of Huffman codes saved 49 bits.

Decoding Huffman Codes

How to recover the original data from the given Huffman code? To decode the Huffman codes we need the Symbol Frequency Table which was used while compressing the data. Without the symbol frequency table we can not recover the data back.

Suppose we want to decode the stream “1000110100010101000101001101011” for the above created Huffman tree.

1) Start from the root (node “adfecb”), as first bit is 1 we go to the left child which is node ‘a’, now this ‘a’ is a leaf node so we output the symbol ‘a’ and again start our way from the root.
2) This time input is 0, so we move to right child of the root i.e. node ‘dfecb’; this node is not a leaf node so we continue our search.
3) Again next input bit is 0 so we again visit the left child of node ‘dfecb’ which is node ‘b’; now node ‘b’ is a leaf node so we output the symbol ‘b’ and restart the search. In the similar way we find the symbols for the above string as follows:

1	00	011	0100	01010	1	00	01010	011	01011
a	b	D	c	E	a	b	e	D	F

So the final output symbols string is “abdceabedf”.

5.5.4 Adaptive Huffman Text

Adaptive Huffman applied on the same cases in Huffman.

Case/Size	Original text file (Byte)	Compressed file (Byte)	Compression ratio
Best	255	34	255:34 = 15:2
Average	1517	910	1517:910 = 16:10
Worst	61	111	61 : 111 = 1:2

Table 5.8 Adaptive Huffman Text Statistics

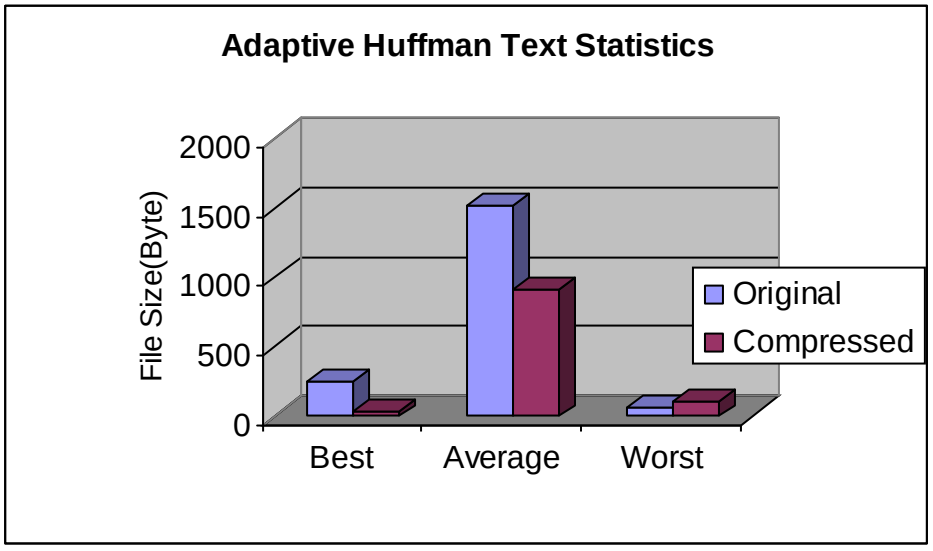


Figure 5.21 Adaptive Huffman Text Statistics

Adaptive Huffman Analysis:

For illustration Adaptive Huffman consider the following example: suppose we have 4 symbols a, b, c and d which occur as “abbbccdccc".

1) Initially all the symbols will be having a frequency count of 1 and an initial tree will be build as show in fig (a); symbol ’a’ will be encoded as 01 and its frequency count will be incremented to 2.
2) The tree is updated as in fig (b) and this time we will encode ’b’ as 101. This process continues until we are finished with encoding all the symbols.

As frequency count of a symbol increases, bits needed to encode the symbol decreases i.e. in adaptive Huffman coding bits needed to encode a symbol increase or decrease dynamically and hence the name Dynamic Huffman Encoding. The final bit stream will be 011010011111000011010 01.

a	B	b	b	c	c	d	c	C	c
01	101	00	1	111	000	011	01	00	1

Decoding adaptive Huffman codes:

First of all, we build an initial Huffman tree, like the one we built while encoding the symbols. From that tree we decode the first symbol. After decoding the symbol we increment its count by 1 and we update the tree. From this newly updated tree, we decode the next symbol. This process continues till we decode all the symbols.

Adaptive Huffman Encoding

For data pattern "abbbccdccc"

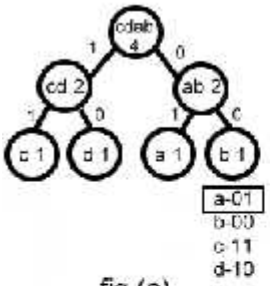


fig (a)

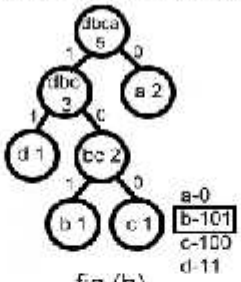


fig (b)

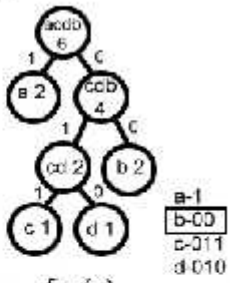


fig (c)

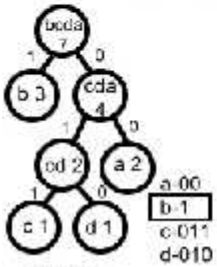


fig (d)

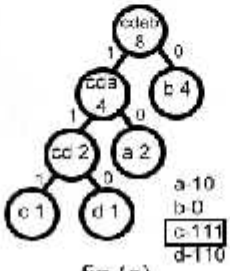


fig (e)

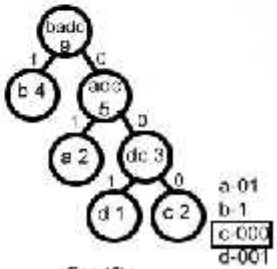


fig (f)

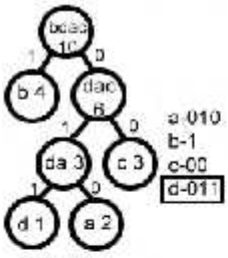


fig (g)

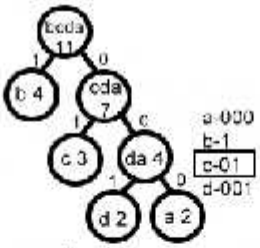


fig (h)

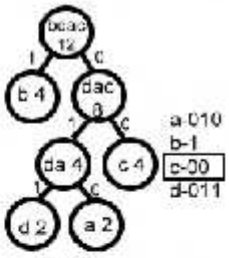


fig (i)

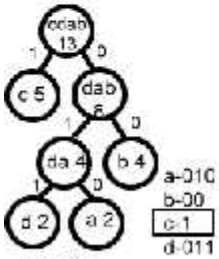


fig (j)

Encoded bit pattern :011010011110000110100 1

Suppose for example; we want to decode “01101001111000011010 01”. We first create the initial tree as in fig (a). From that tree we decode 01 as ’a’; procedure for decoding the code basically remains same as the one used for static Huffman decoding. We then increment the frequency count of ’a’ to 2 and rebuild/ update the tree (fig. b). From this updated tree we then decode 101 as ’b’ and then increment count of ’b’ to 2 and again update the tree as in the fig (c). We continue this process until the whole input stream is decoded.

01	101	00	1	111	000	011	01	00	1
A	B	b	b	c	c	d	c	c	C

So the final output stream becomes “abbbccdccc".

Conclusion:

Table 5.5 and table 5.6 show the statistics for Huffman and Adaptive Huffman text compression:

1) Best case occurs if the data has a high frequency so the compressed symbols has a small number of bits.

2) Average case occurs if the data has a lot number of symbols with different values of frequencies.

3) Worst case occurs if the data has a lot number of symbols with the nearly the same frequencies.

5.6 Conclusions

1) Run Length Encoding is actually not useful for compression text files, since a typical text file doesn't have a lot of long repetitive character strings.
2) RLE it is very useful for compression bytes of monochrome image data (give high performance in compression the black-and-white image).
3) RLE result always the compressed data size is lower or equal the original data size if there is a repeated data.
4) Huffman and adaptive Huffman compression algorithms is mainly efficient in compression text files, because they depends on the frequency of data regardless of it's location in the string.
5) RLE compression efficiency depends on the type of arrangement of data in the file, but Huffman and Adaptive Huffman compression has the same efficiency what ever how the data arranged in the file.
6) Adaptive algorithms do not need a priori estimation of probabilities,

   they are more useful in real applications.

7) Implementations of Static Huffman encoding faces some problems

- As discussed earlier symbol frequencies must be known in advance. Now this can be a real problem, consider a network in which Huffman compression is applied on the data, before sending them on the communication channel. We can not build the Symbol Frequency table in advance as in most cases we do not know the nature of data being transmitted i.e. we do not know whether next character that will be transmitted is an 'a' or 'z' or something else. The

compression must be applied on the fly, which is not possible with the static Huffman encoding.

- Symbol frequency table must be stored /transmitted along with the code words. This table is needed while decoding the Huffman codes.
- If we use the universal symbol frequency table we have an advantage that this table may not be stored/ transmitted along with Huffman codes as this table is known in advance to parties involved in the process. Disadvantages of this approach is that in an specific situation input stream may totally mismatch with what is there in universal frequency table, e.g. a text message in which there are more ‘z’ than ‘e’. In that situation we do not get good compression ratio.
- While compressing a file using static Huffman encoding we need to do two passes on the file; in the first pass we make the Symbol Frequency table and in the second pass we actually compress the data.
- For large files containing a large number of characters, building a symbol frequency table is time consuming as disk access is quite slow.

9) While compressing files with adaptive Huffman encoding we only need to have one pass on the file as compared to two required by static Huffman coding. This can save time required for additional disk I/O operation.

Chapter Six

Testing

6.1 Introduction

6.2 Unit Testing

6.3 Integration Testing

6.1 Introduction

The DCD system must be tested to ensure that every unit or component in the project works as it is expected to do and checks functionality, this grantees that the project works properly.

6.2 Unit Testing

DCD system consists of many classes each represent a form or a class these class are tested separately to insure the functionality of these components, classes contains either compression or decompression algorithm, these algorithms were build using Console Application in Visual C# then after that an input is implemented to the algorithm if the desired output is gotten.

6.2.1 RLE Text

```
This is a Data Compression Decompression using C#


salammmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
maaaaaaaaaaaaaaatttttttt
```

Figure 6.1 Input File RLE Text

```
This is a Data Compression Decompression using C#


salammmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
maaaaaaaaaaaaaaatttttttt
```

Figure 6.2 Output File RLE Text

Property	Value
Input file path	C:\Documents and Settings\Wael\Desktop\RLE Text.txt
Input file size	131 byte
Compressed file path	C:\Documents and Settings\Wael\My Documents\Visual Studio Projects\DataComprDecompr\bin\Debug\Text\Compresssion\ RLE\ RLE Text.data
Compressed file size	72 byte
Decompressed file path	C:\Documents and Settings\Wael\My Documents\Visual Studio Projects\DataComprDecompr\bin\Debug\Text\Decompression \RLE\\RLE Text.txt
Decompressed file size	131 byte

Table 6.1 RLE Text Testing Results

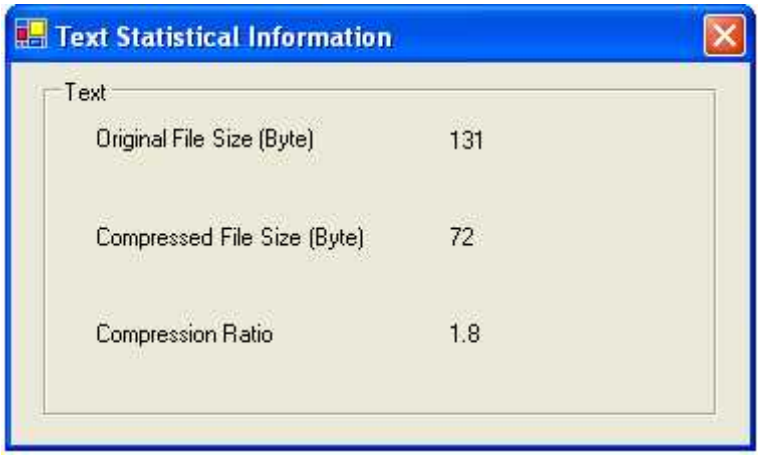


Figure 6.3 RLE Text Statistical Information

6.2.2 RLE Image

Figure 6.4 Input File RLE Image

Figure 6.5 Output File RLE Image

Property	Value
Input file path	C:\Documents and Settings\Wael\Desktop\RLE Image.bmp
Input file size	229932 byte
Compressed file path	C:\Documents and Settings\Wael\My Documents\Visual Studio Projects\DataComprDecompr\bin\Debug\Image\Compresssion\RLE\ RLE Image.data
Compressed file size	1564 byte
Decompressed file path	C:\Documents and Settings\Wael\My Documents\Visual Studio Projects\DataComprDecompr\bin\Debug\Image\Decompression\RLE\RLE Image.bmp
Decompressed file size	229932 byte

Table 6.2 RLE Image Testing Results

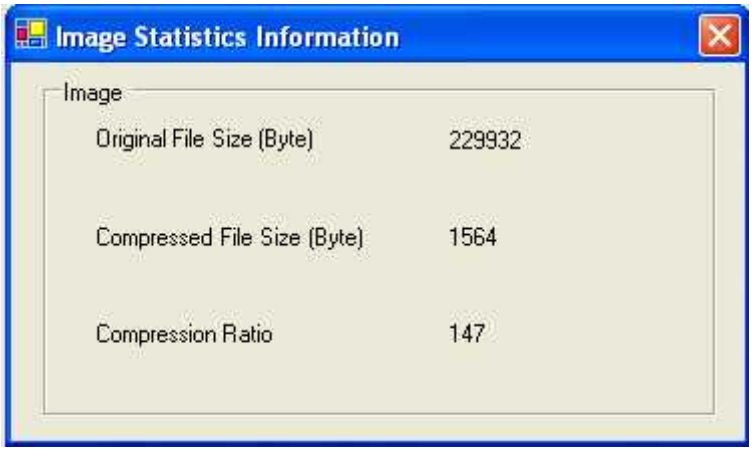


Figure 6.6 RLE Image Statistical Information

6.2.3 Huffman Text

this is a test page this is a dcd system

this is software project

this is a test page

Figure 6.7 Input File Huffman Text

this is a test page this is a dcd system

this is software project

this is a test page

Figure 6.8 Output File Huffman Text

Property	Value
Input file path	C:\Documents and Settings\Wael\Desktop\Huffman Text.txt
Input file size	107 byte
Compressed file path	C:\Documents and Settings\Wael\My Documents\Visual Studio Projects\DataComprDecompr\bin\Debug\Text\Compresssion\ Huffman\ Huffman Text.data
Compressed file size	92 byte
Decompressed file path	C:\Documents and Settings\Wael\My Documents\Visual Studio Projects\DataComprDecompr\bin\Debug\Text\Decompression \Huffman\Huffman Text.txt
Decompressed file size	107 byte

Table 6.3 Huffman Text Testing Results

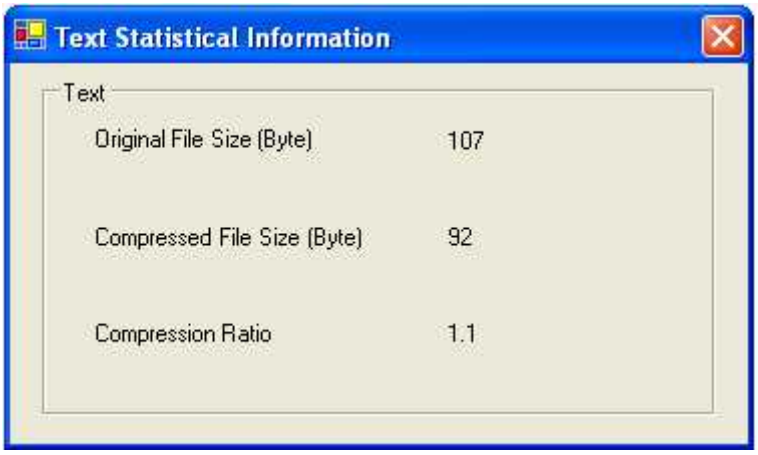


Figure 6.9 Huffman Text Statistical Information

6.2.4 Adaptive Huffman Text

this is a test page this is a dcd system

this is software project

this is a test page

Figure 6.10 Input File Adaptive Huffman Text

this is a test page this is a dcd system

this is software project

this is a test page

Figure 6.11 Output File Adaptive Huffman Text

Property	Value
Input file path	C:\Documents and Settings\Wael\Desktop\Adaptive Text.txt
Input file size	107 byte
Compressed file path	C:\Documents and Settings\Wael\My Documents\Visual Studio Projects\DataComprDecompr\bin\Debug\Text\Compresssion\ Adaptive\ Adapive Text.data
Compressed file size	77 byte
Decompressed file path	C:\Documents and Settings\Wael\My Documents\Visual Studio Projects\DataComprDecompr\bin\Debug\Text\Decompression \Adaptive\Huffman Text.txt
Decompressed file size	107 byte

Table 6.4 Adaptive Huffman Text Testing Results

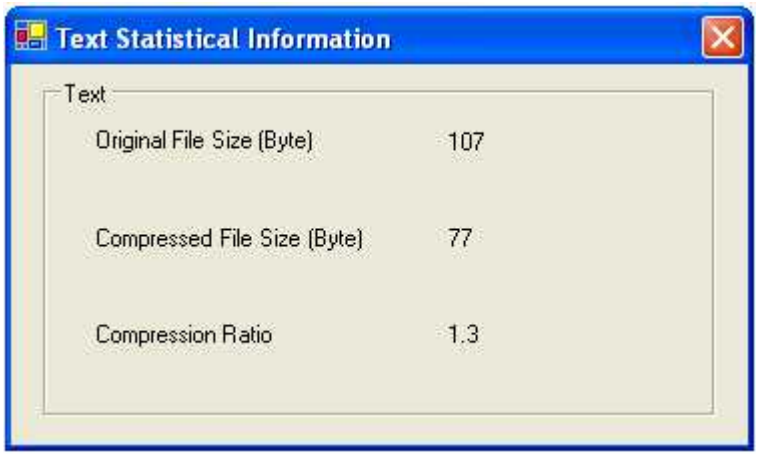


Figure 6.12 Adaptive Huffman Text Statistical Infromation

Chapter Seven

Conclusion and Future Work

7.1 Conclusion

7.2 Future Work

7.1 Conclusion

There are many conclusions that are concluded after working with this project and in this section a description of the resulted conclusions will be pointed:

- In normal cases the occurrence of redundant data in images is greater than text files.
- Each algorithm has best case, average case and worst case, this comes from the fact that each algorithm has its own way to deal with data.
- RLE works perfectly when redundant data occurred frequently data, where as when talking about normal cases where redundant data occurred rarely compression ratio is small.
- Huffman Algorithm and Adaptive Huffman algorithm comes to solve the lower compression ratio in the RLE.
- Working with this project increases the knowledge of dealing with streams to read data and write form files (text or image).
- Working with this project focuses on how to implement different data structures (i.e. building trees using Visual C# without using pointers).
- Working with this project it could implement three well kwon algorithms in classes (RLE, Huffman, and Adaptive Huffman) where until this moment no classes where implemented using Visual C#, so any future work on DCD could use these classes(class reusability), with no need to build it another time(why to reinvent the wheel).
- Working with this project make it possible to make a distinction between the algorithms, the user compress a file using the algorithms, then request statistical information, so it is possible for him to make comparison.

- Working with this project forces the way of how to program using Visual C#, that is C# is has leading rank than its competitive object oriented languages.
- Working with this project focuses the concepts of visual programming, where DCD system uses the visual components which are available in wide manner.

7.2 Future Work

DCD system could be improved and modified, and the following points can be implemented as future work:

- Implementing the DCD using more number of algorithm (i.e. Modified Huffman),
- Applying Quantization to images before compressing these files so that it could get better results than the way without using Quantization.
- Implementing some of tools on images (i.e. Inverting color, gray scaling…), where the implementation of these tools requires accessing image data and changes these data according to the tool.

Resources

Books

1] **C# for Java Programmers** Brian Bagnall Philip Chen Stephen Goldberg, Syngress publishing Inc. 800 Hingham Street Rockland, MA 02370, 2002

2] **C# your visual blueprint for building .NET applications** Eric Butow, Tommy Ryan Hungry Minds Inc. 909 Third Avenue New York NY 10022, 2002

3] **Visual C# .NET Developer's Handbook** John Paul Mueller, SYBEX Inc., 1151 Marina Village Parkway, Alameda, 2002

4] **Data Compression Techniques and Applications Hardware and Software Considerations** 3rd Edition, Gilbert Held, JOHN WILEY & SONS Inc. 605 Third Avenue, New York NY 10158-0012 USA, 1994

5] **Visual Studio.NET 2003 Combined Help Collections**

6] **msdn training, Course 2124C Programming with C#,** Microsoft Corporation USA 2002.

Web Sites

1] http://www.codeproject.com/cs/media/#General

2] http://www.csharp-station.com

3] http://www.datacompression.com

4] http://www.csharpindex.com

5] http://www.c-sharpcorner.com

6] http://www.codehound.com/csharp

7] http://www.csharphelp.com

Appendices

Appendix A Glossary

Appendix B DCD User Interface

Appendix C DCD Source Code

Appendix A
Glossary

Adaptive Huffman Encoding: is a modified version of Huffman encoding that is used to build the tree dynamically and the is called variable length coding that is the generated code is variable according to frequency of each symbol.

Binary tree: it is a special type of inverted tree in which each element has only two branches below it.

Bitmap (BMP): is a standard format used by Windows to store device-independent and application-independent images. The number of bits per pixel (1, 4, 8, 15, 24, 32, or 64) for a given BMP file is specified in a file header. BMP files with 24 bits per pixel are common. BMP files are usually not compressed and therefore are not well suited for transfer across the Internet.

Compression Ratio: defined as the ratio of original data size to the compressed data size, this is an indication of how compression algorithm efficiency.

Data Compression: defined as reducing of the amount of storage space required to store a given amount of data, it comes with a lot of advantages, it saves storage space, bandwidth, cost and time required to transmit data from one place to another.

Data Decompression: defined as the process of getting the original data form compressed formats it is the reverse process of compression.

Graphics Interchange Format (GIF): is a common format for images that appear on Web pages, GIFs work well for line drawings, pictures with blocks of solid color, and pictures with sharp boundaries between colors, GIFs are compressed, but no information is lost in the compression process; a decompressed image is exactly the same as the original. One color in a GIF can be designated as transparent, so that the image will have the background color of any Web page that displays it. A sequence of GIF images can be

stored in a single file to form an animated GIF. GIFs store at most 8 bits per pixel, so they are limited to 256 colors.

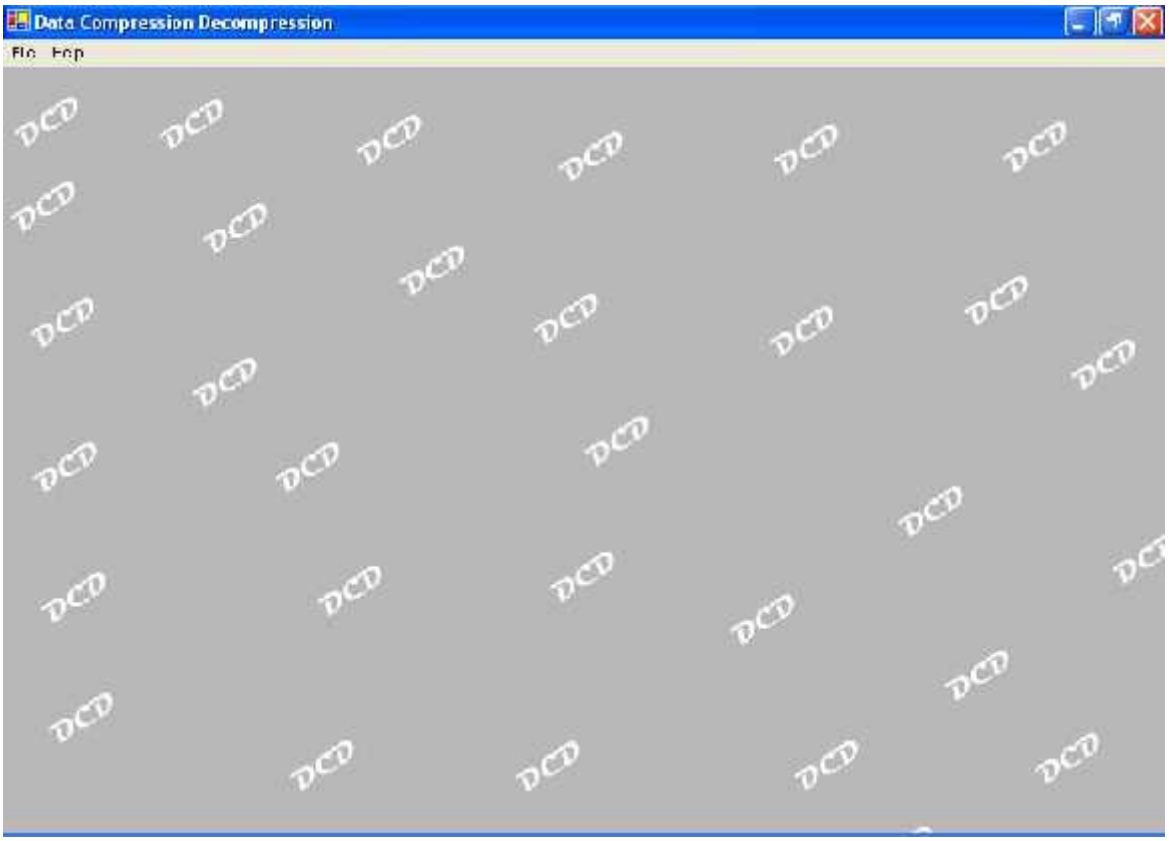
Huffman Encoding: one of the algorithms that is used to calculate the frequency of each count in file then building a binary tree that, each node of this tree has the following items the symbol, frequency of each symbol and code for each node.

Joint Photographic Experts Group (JPEG): JPEG is a compression scheme that works well for natural scenes such as scanned photographs. Some information is lost in the compression process, but often the loss is imperceptible to the human eye. JPEGs store 24 bits per pixel, so they are capable of displaying more than 16 million colors. JPEGs do not support transparency or animation.

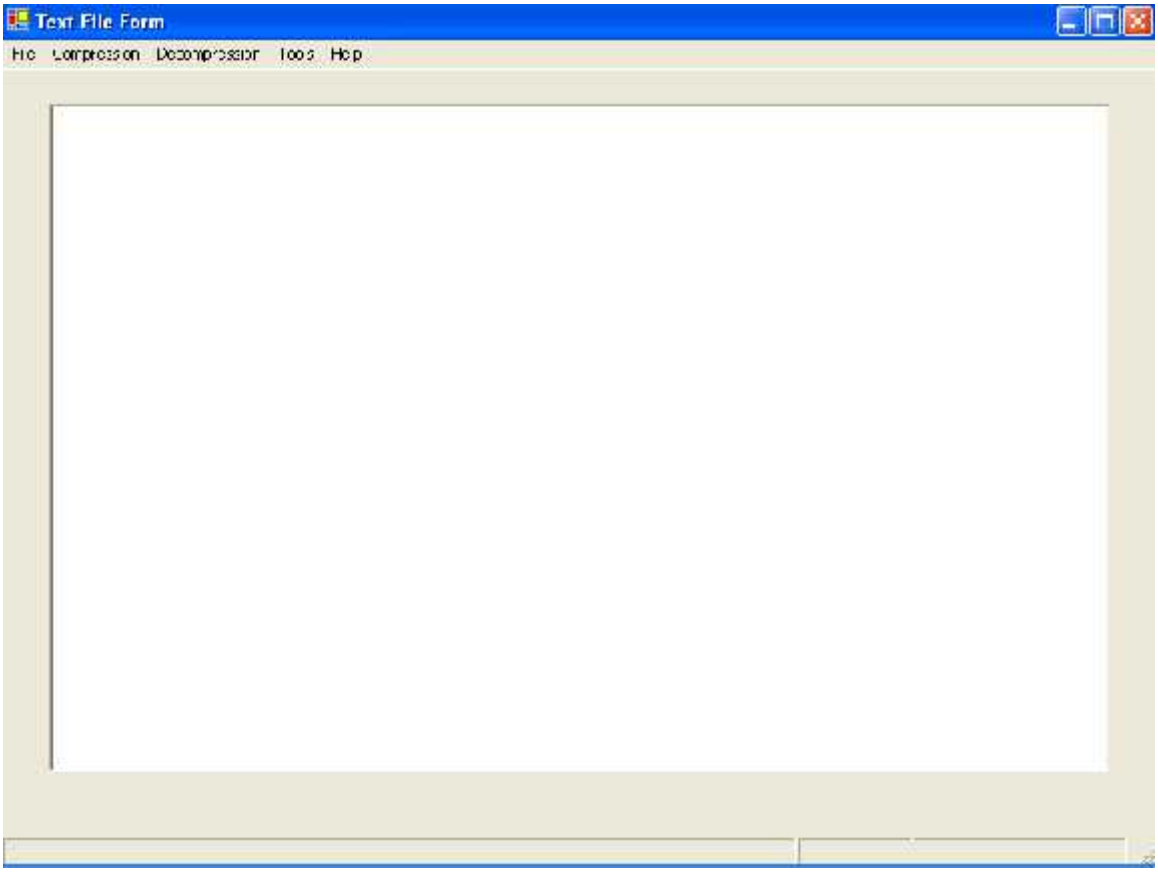
Run Length Encoding (RLE): one of the algorithms that is used to compress the redundant data with a format that take less number of symbols, the general format of compressed data is special character, symbol then count or frequency of symbol.

Tree: A type of data structure in which each element is attached to one or more elements directly beneath it. The connections between elements are called branches, trees are often called inverted trees because they are normally drawn with the root at the top.

Appendix C
DCD User Interface



DCD Main Form



Text File Form

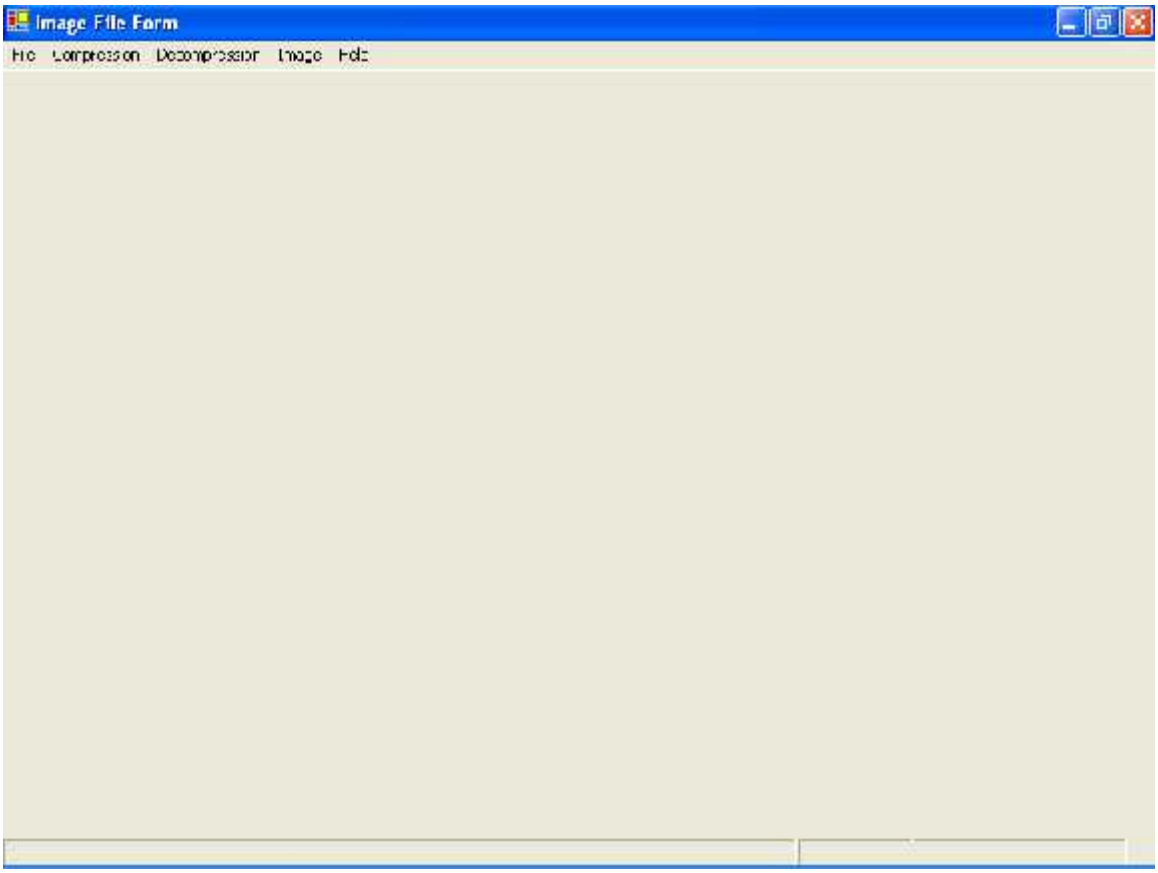
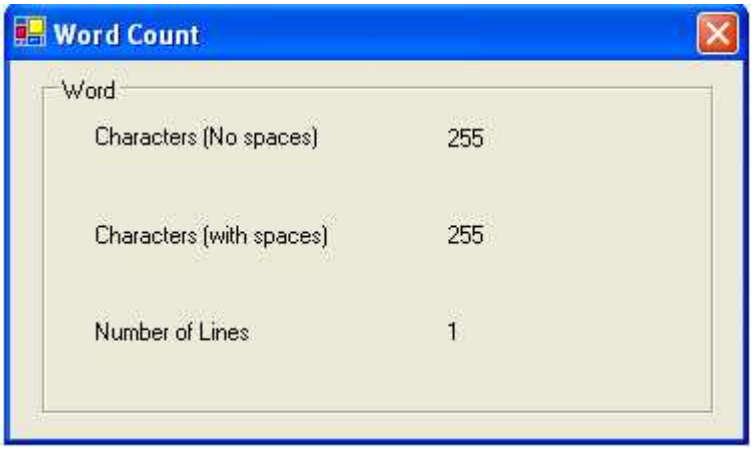
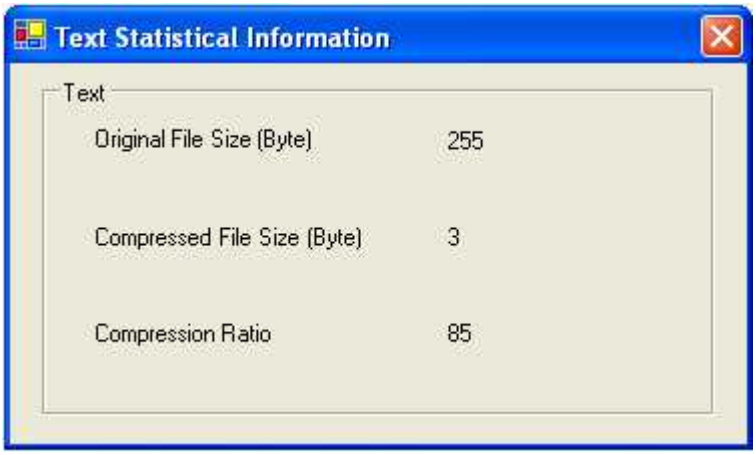


Image File Form



Word Count From



Text Statistical Information Form

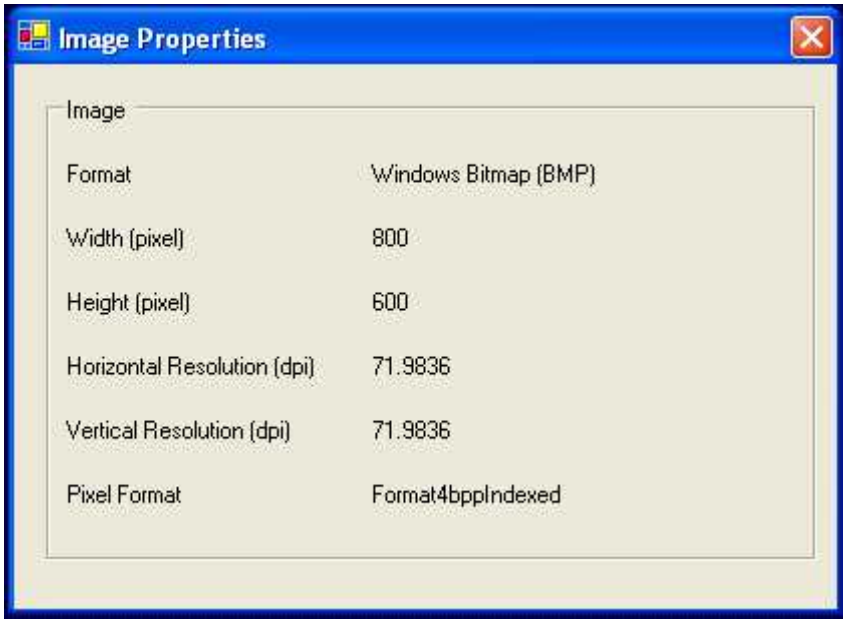


Image Properties Form

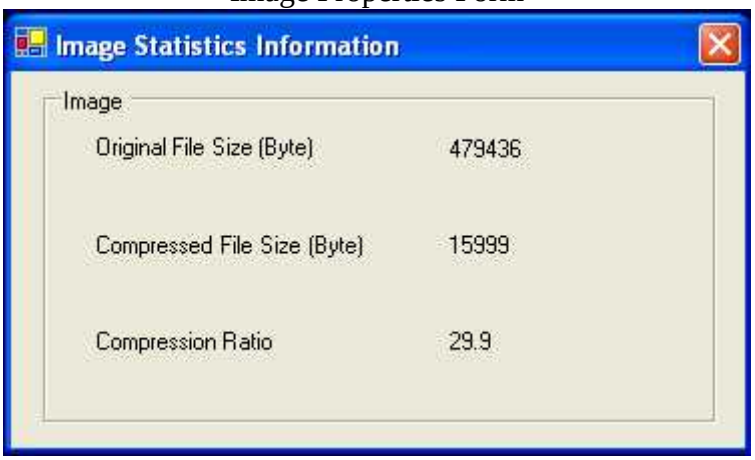


Image Statistics Information Form

About DCD Form

Appendix C
DCD Source Code

RLE Text Compression Class

```
using System;
using System.IO;

namespace DataComprDecompr
{
	/// <summary>
	/// Summary description for RLETextComp.
	/// </summary>
	public class CRLETextComp
	{
		public CRLETextComp()
		{
			char cText;
			char temp=']';
			//count represents the number of times the repeated
data occurred
			int count=0;

			String textFilePath=frmText.textPath;
			String RLETDirect=frmText.TCompDirect;
			String comprFileName=frmText.textFileName;
			StreamReader sr=new StreamReader(textFilePath);

			FileStream fs=new
FileStream(RLETDirect+"\\"+comprFileName+".data",FileMode.OpenOrCreate,
FileAccess.Write);
			BinaryWriter bw=new BinaryWriter(fs);
			while(sr.Peek()!=-1)
			{
				if(count==0)
				{
					cText=Convert.ToChar(sr.Read());
					temp=cText;
					count+=1;
				}
				else
				{
					cText=Convert.ToChar(sr.Read());
					if(temp!=cText | count==255)
					{
						if(count<4)
						{
							for(int i=0;i<count;i++)
								bw.Write(temp);

						}
```

```
                                    else
                                    {
                                          bw.Write(Convert.ToByte(27));
                                          bw.Write(temp);

        bw.Write(Convert.ToByte(count));

                                    }
                                    count=1;
                                    temp=cText;


                              }
                              else if(count<255)
                              {
                                    count+=1;
                              }
                              else
                              {
                                    bw.Write(Convert.ToByte(27));
                                    bw.Write(temp);
                                    bw.Write(Convert.ToByte(count));
                                    count=1;
                                    temp=cText;

                              }
                        }
                  }
                  if(count<4)
                  {
                        for(int i=0;i<count;i++)
                              bw.Write(temp);

                  }
                  else
                  {

                        bw.Write(Convert.ToByte(27));
                        bw.Write(temp);
                        bw.Write(Convert.ToByte(count));

                  }
                  sr.Close();
                  bw.Close();
                  fs.Close();
            }
      }
}
```

RLE Text Decompression Class

```
using System;
using System.IO;
using System.Windows.Forms;

namespace DataComprDecompr
{
	/// <summary>
	/// Summary description for RLETextDecomp.
	/// </summary>
	public class CRLETextDecomp
	{
		public CRLETextDecomp()
		{
			char cText;
			char temp;
			char cCount;
			int count;
			byte sChar=27;
			String CompFileName=frmText.textFileName;
			String CompFileDirect=frmText.TCompDirect;
			String DecompFileDirect=frmText.TDecompDirect;

			FileStream fs;
			BinaryReader br;
			StreamWriter sw;
			try
			{
				fs=new
FileStream(CompFileDirect+"\\"+CompFileName+".data",FileMode.Open,FileA
ccess.Read);
				br=new BinaryReader(fs);
				sw=new
StreamWriter(DecompFileDirect+"\\"+CompFileName+".txt");
				while(br.PeekChar()!=-1)
				{
					cText=Convert.ToChar(br.Read());
					if(cText==Convert.ToChar(sChar))
					{
						temp=Convert.ToChar(br.Read());
						cCount=Convert.ToChar(br.Read());
						count=Convert.ToInt16 (cCount);
						//Console.WriteLine(count);
						for(int i=0;i<count;i++)
```

```
                                    {
                                          sw.Write(temp);
                                    }
                              }

                              else
                              {
                                    sw.Write(cText);
                              }
                        }
                        sw.Close();

                  }
                  catch(System.IO.FileNotFoundException  exFNF)
                  {
                        MessageBox.Show(exFNF.Message,"File Not
Found",MessageBoxButtons.OK,MessageBoxIcon.Error);
                  }

            }
      }
}
```

RLE Image Compression Class

```
using System;
using System.IO;
using System.Drawing;
using System.Drawing.Imaging;

namespace DataComprDecompr
{
      /// <summary>
      /// Summary description for CRLEImageComp.
      /// </summary>
      public class CRLEImageComp
      {
            public CRLEImageComp()
            {

                  int i;
                  String ImageFilePath=frmImage.imagePath;
                  String RLEIDirect=frmImage.ImgCompDirect;
                  String comprFileName=frmImage.imageFileName;

                  FileStream fs=new
FileStream(RLEIDirect+"\\"+comprFileName+".data",FileMode.OpenOrCreate,
FileAccess.Write);
                  BinaryWriter bw=new BinaryWriter(fs);
```

```
byte []IArray;
String pixelStr="";
String temp="";
int count=0;
byte special=27;
byte alter=26;

//Read the content of image file as array of bytes
Image BImage=new Bitmap(ImageFilePath);
MemoryStream ms=new MemoryStream();
BImage.Save(ms,ImageFormat.Bmp);
IArray=ms.GetBuffer();
frmImage.imageFileSize=IArray.Length;

//Write the content of header file to compressed file
for(i=0;i<54;i++)
{
      bw.Write(Convert.ToByte(IArray[i]));
}
for(;i<IArray.Length;)
{
      if(count==0)
      {
            for(int j=0;j<3 && i<IArray.Length;j++)
            {
                  if(IArray[i]==special)
                  {
                        IArray[i]=alter;
                  }
                  pixelStr+=IArray[i];
                  i++;
            }
            temp=pixelStr;
            count+=1;

      }
      else
      {
            pixelStr="";
            for(int j=0;j<3 && i<IArray.Length;j++)
            {
                  if(IArray[i]==special)
                  {
                        IArray[i]=alter;
                  }
                  pixelStr+=IArray[i];
                  i++;
            }
```

```
                        if(temp!=pixelStr)
                        {
                              if(count<2)
                              {
                                    i-=6;
                                    for(int x=0;x<3;x++,i++)
                                    {

bw.Write(Convert.ToByte(IArray[i]));
                                    }
                              }
                              else if(count<=255)
                              {

bw.Write(Convert.ToByte(special));
                                    i-=6;
                                    for(int x=0;x<3;x++,i++)
                                    {

bw.Write(Convert.ToByte(IArray[i]) );
                                    }

bw.Write(Convert.ToByte(count));
                              }
                              count=1;
                              temp=pixelStr;
                              i+=3;
                        }
                        else if(count<255)
                        {
                              count+=1;
                        }
                        else
                        {
                              bw.Write(Convert.ToByte(special));
                              i-=6;
                              for(int x=0;x<3;x++,i++)
                              {

bw.Write(Convert.ToByte(IArray[i]) );
                              }
                              bw.Write(Convert.ToByte(count));
                              count=1;
                              temp=pixelStr;
                              i+=3;

                        }
                  }
            }
```

```
                    if(count<2)
                    {
                          i-=3;
                          for(int x=0;x<3;x++,i++)
                          {
                                bw.Write(Convert.ToByte(IArray[i]) );

                          }

                    }
                    else if(count<=255)
                    {
                          bw.Write(Convert.ToByte(special));
                          i-=3;
                          for(int x=0;x<3;x++,i++)
                          {
                                bw.Write(Convert.ToByte(IArray[i]) );
                          }
                          bw.Write(Convert.ToByte(count));
                    }

                    bw.Close();

              }
              }
        }
```

RLE Image Decompression Class

```
using System;
using System.IO;
using System.Drawing;
using System.Collections;
using System.Drawing.Imaging;

namespace DataComprDecompr
{
      /// <summary>
      /// Summary description for CRLEImageDecomp.
      /// </summary>
      public class CRLEImageDecomp
      {
            public CRLEImageDecomp()
            {
                  char cPixelCount;
                  int pixelCount;
                  byte special=27;
```

```
                    byte pByte;
                    byte [] element=new byte[3];

                    String IExtension=frmImage.imageFileExtension;
                    String DecompFileDirect=frmImage.ImgDecompDirect;
                    String CompFileName=frmImage.imageFileName;
                    String CompFileDirect=frmImage.ImgCompDirect;


                    FileStream fs=new FileStream(CompFileDirect+ "\\"+
CompFileName + ".data",FileMode.Open,FileAccess.Read );
                    BinaryReader br=new BinaryReader(fs);
                    ArrayList al=new ArrayList();

                    //Add image header to arrayList
                    for(int w=0;w<54;w++)
                    {
                          al.Add(Convert.ToByte(br.ReadByte()));
                    }
                    while(br.PeekChar()!=-1)
                    {

                          pByte=Convert.ToByte(br.ReadByte());
                          if(pByte==special)
                          {
                                for(int j=0;j<3 && br.PeekChar()!=-1;j++)
                                {

      element[j]=Convert.ToByte(br.ReadByte());
                                }


      cPixelCount=Convert.ToChar(br.ReadByte());
                                pixelCount=Convert.ToInt16 (cPixelCount);

                                for(int i=0;i<pixelCount;i++)
                                {
                                      al.Add(Convert.ToByte(element[0]));
                                      al.Add(Convert.ToByte(element[1]));
                                      al.Add(Convert.ToByte(element[2]));
                                }
                          }
                          else
                          {
                                al.Add(Convert.ToByte(pByte));
                                for(int j=0;j<2 && br.PeekChar()!=-1;j++)

                                {
```

```
      al.Add(Convert.ToByte(br.ReadByte()));

                              }
                        }

                  }
                  byte [] test=new byte[al.Count];
                  for(int z=0;z<al.Count;z++)
                  {
                        test[z]=Convert.ToByte(al[z]);

                  }

                  MemoryStream ms=new MemoryStream();
                  ms.Write(test,0,test.Length );
                  Image IBitmap=new Bitmap(ms);
                  if(IExtension==".bmp")
                  {

      IBitmap.Save(DecompFileDirect+"\\"+CompFileName+IExtension,ImageF
ormat.Bmp);
                  }
                  else if(IExtension==".jpg")
                  {

      IBitmap.Save(DecompFileDirect+"\\"+CompFileName+IExtension,ImageF
ormat.Jpeg);
                  }
                  else if(IExtension==".gif")
                  {

      IBitmap.Save(DecompFileDirect+"\\"+CompFileName+IExtension,ImageF
ormat.Gif);
                  }
                  else
                  {

      IBitmap.Save(DecompFileDirect+"\\"+CompFileName+".bmp",ImageForma
t.Bmp);
                  }
                  br.Close();


            }
      }
}
```

Huffman Text Compression Class

```
using System;
using System.IO;
using System.Collections;

namespace DataComprDecompr
{
	/// <summary>
	/// Summary description for CHaffmanTextComp.
	/// </summary>
	public class CHaffmanTextComp
	{
		public static int cCount=0;
	//Number of characters in the text file
		public static String textFilePath;
		public static String HuffTDirect;
		public static String comprFileName;
		public static char [] charArray;
		public static cell [] TempTable;
		public static FileStream fs=new
FileStream("test.data",FileMode.OpenOrCreate,FileAccess.Write);

		public static BinaryWriter bw=new BinaryWriter(fs);

		public static byte [] codeByte;


		// cell represent a node of tree
		public struct cell
		{
			public String cText;
			public int freq;
			public String code;
		}
		public CHaffmanTextComp()
		{
			//set the values of compressed
file(name,directory,path)
			textFilePath=frmText.textPath;
			HuffTDirect=frmText.TCompDirect;
			comprFileName=frmText.textFileName;
			//fs=new
FileStream(HuffTDirect+"\\"+comprFileName+".data",FileMode.OpenOrCreate
,FileAccess.Write);
			//bw=new BinaryWriter(fs);;

			wCount();
```

```
                  LookupTable();
                  OverHead();
                  huffCoding();
            }
            // Sort method is used to sort array content in ascending
order
            public static void Sort(cell [] TableCell , int n)
            {
                  cell temp;
                  for(int i=0;i<n;i++)
                  {
                        for(int j=i;j<n;j++)
                        {
                              if(TableCell[i].freq > TableCell[j].freq)
                              {
                                    temp=TableCell[i];
                                    TableCell[i]=TableCell[j];
                                    TableCell[j]=temp;

                              }
                        }
                  }
            }

            // wCount method is used to count the number of charater in
text file
            public static void wCount()
            {
                  StreamReader sr=new StreamReader(textFilePath);
                  while(sr.Peek()!=-1)
                  {
                        sr.Read();
                        cCount+=1;
                  }
                  sr.Close();
            }

            public static void OverHead()
            {
                  byte special=27;
                  for(int g=0;g<TempTable.Length;g++)
                  {
                        char
cTextHead=Convert.ToChar(TempTable[g].cText);

                        if(TempTable[g].freq<=255)
                        {
                              bw.Write(Convert.ToByte(cTextHead));
```

```
bw.Write(Convert.ToByte(TempTable[g].freq));

                  }
                  else
                  {
                        int numBytes=TempTable[g].freq/255;
                        int  remain=TempTable[g].freq%255;
                        for(int z=0;z<numBytes;z++)
                        {

bw.Write(Convert.ToByte(cTextHead));
                              bw.Write(Convert.ToByte(255));

                        }
                        bw.Write(Convert.ToByte(cTextHead));

                        bw.Write(Convert.ToByte(remain));
                  }


            }
            bw.Write(Convert.ToByte(special));

      }


      public static void LookupTable()
      {
            int i=0;
            int count=0;
            bool exist;
            int accum;
            char ctxt,temp;
            String concatChar;
            cell concatCell=new cell();

            charArray=new char[cCount];
            StreamReader strR=new StreamReader(textFilePath);

            while(strR.Peek()!=-1)
            {
                  charArray[i]=Convert.ToChar(strR.Read());
                  i++;
            }
            strR.Close();

            //adding different elements of the text file to array
list
```

```
                    ArrayList AL=new ArrayList();
                    if(charArray.Length!=0)
                    {
                          AL.Add(charArray[0]);
                          for(int j=0;j<charArray.Length;j++)
                          {
                                exist=false;
                                for(int k=0;k<AL.Count &&
exist==false;k++)
                                {

      if(charArray[j]==Convert.ToChar(AL[k]))
                                      {
                                            exist=true;
                                      }

                                }
                                if(exist==false)
                                {
                                      AL.Add(charArray[j]);
                                }
                          }

                    }
                    cell [] nd=new cell[AL.Count];
                    for(int k=0;k<AL.Count;k++)
                    {
                          nd[k].cText =Convert.ToString(AL[k]);
                    }

                    for(int j=0;j<nd.Length;j++)
                    {
                          temp=Convert.ToChar(nd[j].cText);
                          for(int w=0;w<charArray.Length;w++)
                          {
                                ctxt=charArray[w];
                                if(ctxt==temp)
                                {
                                      count+=1;
                                }
                          }
                          nd[j].freq=count;
                          count=0;
                    }


                    cell [] TableCell=new cell[nd.Length];
                    TempTable=new cell[nd.Length];
                    cell [][] jagTable=new cell[nd.Length][];
```

```
                  if(nd.Length!=1)
                  {
                        for(int e=0;e<nd.Length;e++)
                        {
                              TableCell[e]=nd[e];
                        }

                        jagTable[0]=new cell[TableCell.Length];
                        for(int d=0;d<TableCell.Length;d++)
                        {
                              jagTable[0][d]=nd[d];
                              TempTable[d]=nd[d];
                        }
                        Sort(TempTable,TempTable.Length);
                        for(int
z=1,aLen=TableCell.Length;z<TableCell.Length;z++,aLen--)
                        {
                              Sort(jagTable[z-1],aLen);
                              accum=jagTable[z-1][0].freq+jagTable[z-
1][1].freq;
                              concatChar=jagTable[z-1][0].cText +
jagTable[z-1][1].cText;
                              concatCell.cText=concatChar;
                              concatCell.freq=accum;
                              jagTable[z]=new cell [aLen-1];
                              jagTable[z][0]=concatCell;

                              for(int j=2;j<jagTable[z-1].Length ;j++)
                              {
                                    jagTable[z][j-1]=jagTable[z-1][j];
                              }
                        }

                        for(int x=TempTable.Length-2;x>=0;x--)
                        {
                              for(int j=0;j<jagTable[x].Length;j++)
                              {
                                    if(j%2==0)
                                    {
                                          jagTable[x][j].code="1";
                                    }
                                    else
                                    {
                                          jagTable[x][j].code="0";
                                    }
                              }
                        }
```

```
                        // assignment of code for each character in the
file
                        // assigning 1 to the smallest value and 0 to
the second smallest one
                        bool stop=false;
                        String cTemp="";

                        for(int s=0;s<jagTable[0].Length;s++)
                        {
                              stop=false;
                              cTemp="";
                              for(int j=TempTable.Length-2;j>=0 &&
stop==false ;j--)
                              {
                                    for(int k=jagTable[j].Length-1;k>=0
&& stop==false ;k--)
                                    {

      if(jagTable[0][s].cText==jagTable[j][k].cText )
                                          {

      TempTable[s].code+=jagTable[j][k].code ;
                                                stop=true;
                                          }
                                          else
if(jagTable[j][k].cText.IndexOf(jagTable[0][s].cText,0)!=-1 &&
cTemp!=jagTable[j][k].cText)
                                          {

      TempTable[s].code+=jagTable[j][k].code;

      cTemp=jagTable[j][k].cText;

                                          }
                                    }
                              }
                        }


                  }
                  else
                  {
                        TempTable[0].cText=nd[0].cText;
                        TempTable[0].freq=nd[0].freq;
                        TempTable[0].code="1";
                  }

            }
```

```
            public static void huffCoding()
            {
                  byte trueByte=1;
                  byte falseByte=0;
                  String codeString="";

                  for(int f=0;f<charArray.Length ;f++)
                  {
                        for(int d=0;d<TempTable.Length;d++)
                        {

      if(charArray[f]==Convert.ToChar(TempTable[d].cText) )
                              {
                                    codeString+=TempTable[d].code;
                              }
                        }
                  }
                  codeByte=new byte[codeString.Length];
                  byte [] blockByte=new byte[8];
                  byte bitRepres;
                  for(int d=0;d<codeString.Length;d++)
                  {
                        if(codeString[d]=='0')
                              codeByte[d]=falseByte;
                        else
                              codeByte[d]=trueByte;
                  }
                  int Quat=codeByte.Length/8;
                  int lRemain=codeByte.Length %8;
                  if(lRemain==0)
                  {
                        int r=0;
                        int w=0;

                        for(w=0;w<codeByte.Length;)
                        {
                              for(r=0;r<8;r++,w++)
                              {
                                    blockByte[r]=codeByte[w];
                              }

      bitRepres=Convert.ToByte(blockByte[0]|(blockByte[1]<<1)|(blockByt
e[2]<<2)|(blockByte[3]<<3)|(blockByte[4]<<4)|(blockByte[5]<<5)|(blockBy
te[6]<<6)|(blockByte[7]<<7));
                              bw.Write(bitRepres);
                        }
                  }
                  else
```

```
                  {
                        int r=0;
                        int w=0;

                        for(w=0;w<Quat*8;)
                        {
                              for(r=0;r<8;r++,w++)
                              {
                                    blockByte[r]=codeByte[w];
                              }

      bitRepres=Convert.ToByte(blockByte[0]|(blockByte[1]<<1)|(blockByt
e[2]<<2)|(blockByte[3]<<3)|(blockByte[4]<<4)|(blockByte[5]<<5)|(blockBy
te[6]<<6)|(blockByte[7]<<7));
                              bw.Write(bitRepres);
                        }
                        for(r=0,w=Quat*8;w<codeByte.Length;w++,r++)
                        {
                              blockByte[r]=codeByte[w];
                        }
                        for(w=codeByte.Length;w<Quat*8;w++,r++)
                        {
                              blockByte[r]=falseByte;
                        }

      bitRepres=Convert.ToByte(blockByte[0]|(blockByte[1]<<1)|(blockByt
e[2]<<2)|(blockByte[3]<<3)|(blockByte[4]<<4)|(blockByte[5]<<5)|(blockBy
te[6]<<6)|(blockByte[7]<<7));
                        bw.Write(bitRepres);
                  }
                  bw.Close();


            }

      }
}
```

Huffman Text Decompression Class

```
using System;
using System.IO;
using System.Collections;

namespace DataComprDecompr
{
```

```
        /// <summary>
        /// Summary description for CHaffmanTextDecomp.
        /// </summary>
        public class CHaffmanTextDecomp
        {
              public static byte special=27;
              public static String codeString="";
              public static cell [] tableCell;
              public static cell [] TempTable;
              public static FileStream fs;
              public static BinaryReader br;
              public static StreamWriter sw;


              public struct cell
              {
                    public String cText;
                    public int freq;
                    public String code;
              }

              public CHaffmanTextDecomp()
              {
                    String CompHuffTDirect=frmText.TCompDirect;
                    String DecompHuffTDirect=frmText.TDecompDirect ;
                    String comprFileName=frmText.textFileName;

                    fs=new
FileStream(CompHuffTDirect+"\\"+comprFileName+".data",FileMode.Open,Fil
eAccess.Read);
                    BinaryReader br=new BinaryReader(fs);
                    StreamWriter sw=new
StreamWriter(DecompHuffTDirect+"\\"+comprFileName+".txt");

                    ReadOverHead();
                    LookupTable();
                    ReadData();
                    HuffmanDecoding();

              }

              public static void ReadOverHead()
              {

                    ArrayList ALchar=new ArrayList();
                    ArrayList ALcoun=new ArrayList();

                    while(br.PeekChar()!=-1 && br.PeekChar()!=special)
                    {
```

```
                        ALchar.Add(Convert.ToChar(br.ReadByte()));
                        ALcoun.Add(Convert.ToInt32(br.ReadByte()));

                        tableCell=new cell[ALchar.Count];

                        for(int k=0;k<ALchar.Count;k++)
                        {
                              tableCell[k].cText
=Convert.ToString(ALchar[k]);

      tableCell[k].freq=Convert.ToInt32(ALcoun[k]);
                        }
                  }

            }



            public static void Sort(cell [] TableCell , int n)
            {
                  cell temp;
                  for(int i=0;i<n;i++)
                  {
                        for(int j=i;j<n;j++)
                        {
                              if(TableCell[i].freq > TableCell[j].freq)
                              {
                                    temp=TableCell[i];
                                    TableCell[i]=TableCell[j];
                                    TableCell[j]=temp;

                              }
                        }
                  }
            }


            public static void LookupTable()
            {
                  int accum;
                  String concatChar;
                  cell concatCell=new cell();


                  TempTable=new cell[tableCell.Length];
                  cell [][] jagTable=new cell[tableCell.Length][];


                  if(tableCell.Length!=1)
```

```
                  {

                        jagTable[0]=new cell[tableCell.Length];
                        for(int d=0;d<tableCell.Length;d++)
                        {
                              jagTable[0][d]=tableCell[d];
                              TempTable[d]=tableCell[d];
                        }
                        Sort(TempTable,TempTable.Length);
                        for(int
z=1,aLen=tableCell.Length;z<tableCell.Length;z++,aLen--)
                        {
                              Sort(jagTable[z-1],aLen);
                              accum=jagTable[z-1][0].freq+jagTable[z-
1][1].freq;
                              concatChar=jagTable[z-1][0].cText +
jagTable[z-1][1].cText;
                              concatCell.cText=concatChar;
                              concatCell.freq=accum;
                              jagTable[z]=new cell [aLen-1];
                              jagTable[z][0]=concatCell;

                              for(int j=2;j<jagTable[z-1].Length ;j++)
                              {
                                    jagTable[z][j-1]=jagTable[z-1][j];
                              }
                        }

                        for(int x=TempTable.Length-2;x>=0;x--)
                        {
                              for(int j=0;j<jagTable[x].Length;j++)
                              {
                                    if(j%2==0)
                                    {
                                          jagTable[x][j].code="1";
                                    }
                                    else
                                    {
                                          jagTable[x][j].code="0";
                                    }
                              }
                        }

                        // assignment of code for each character in the
file
                        // assigning 1 to the smallest value and 0 to
the second smallest one
                        bool stop=false;
                        String cTemp="";
```

```
                        for(int s=0;s<jagTable[0].Length;s++)
                        {
                              stop=false;
                              for(int j=TempTable.Length-2;j>=0 &&
stop==false ;j--)
                              {
                                    for(int k=jagTable[j].Length-1;k>=0
&& stop==false ;k--)
                                    {

      if(jagTable[0][s].cText==jagTable[j][k].cText )
                                          {

      TempTable[s].code+=jagTable[j][k].code ;
                                                stop=true;
                                          }
                                          else
if(jagTable[j][k].cText.IndexOf(jagTable[0][s].cText,0)!=-1 &&
cTemp!=jagTable[j][k].cText)
                                          {

      TempTable[s].code+=jagTable[j][k].code;

      cTemp=jagTable[j][k].cText;

                                          }
                                    }
                              }
                        }


                  }
                  else
                  {
                        TempTable[0].cText=tableCell[0].cText;
                        TempTable[0].freq=tableCell[0].freq;
                        TempTable[0].code="1";
                  }

            }


            public static void ReadData()
            {
                  ArrayList AL=new ArrayList();


                  if(br.ReadByte()==special)
                  {
```

```
                        while(br.PeekChar()!=-1)
                        {
                              AL.Add(Convert.ToByte(br.ReadByte()));
                        }
                  }
                  byte [] dataByte=new byte[AL.Count];
                  byte [] y=new byte[8];

                  for(int j=0;j<AL.Count;j++)
                  {
                        dataByte[j]=Convert.ToByte(AL[j]);
                  }

                  for(int i=0;i<dataByte.Length;i++)
                  {
                        for(int j=0;j<8;j++)
                        {
                              y[j]=Convert.ToByte(dataByte[i] &
Convert.ToByte(Math.Pow(2,j)));
                              if(y[j]==0)
                              {
                                    codeString+="0";
                              }
                              else
                              {
                                    codeString+="1";
                              }

                        }
                  }
                  int totFreq=0;
                  for(int i=0;i<TempTable.Length;i++)
                  {

      totFreq+=TempTable[i].freq*TempTable[i].code.Length;

                  }
                  codeString=codeString.Substring(0,totFreq);


            }


            public static void HuffmanDecoding()
            {
                  String sTemp="";
                  for(int i=0;i<codeString.Length;i++)
                  {
                        sTemp+=codeString[i];
```

```
                        for(int j=0;j<TempTable.Length;j++)
                        {

                              if(sTemp==TempTable[j].code)
                              {
                                    sw.Write(TempTable[j].cText);
                                    sTemp="";
                              }

                        }
                  }
                  sw.Close();
            }

      }
}
```

Adaptive Huffman Text Compression Class

```
using System;
using System.IO;
using System.Collections ;
namespace AdpComp
{
      /// <summary>
      /// Summary description for Class1.
      /// </summary>


      class Class1
      {
            public static int cCount=0;
      //Number of characters in the text file
            public static ArrayList textArray;
            public static ArrayList charArray;
            public static cell [] node;
            public static char currentChar;
            public static cell [] TempTable;
            public static cell [] TableCell;
            public static cell [][] jagTable;
            public static StreamReader sr;
            public static FileStream fs;
            public static BinaryWriter bw;
            public static byte [] codeByte;
            public static String codeString;
            public static ArrayList outputArray;
```

```
        public struct cell
        {
              public String cText;
              public int freq;
              public String code;
        }

        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        public static void wCount()
        {
              char fText;
              String path=@"C:\Documents and
Settings\Wael\Desktop\input.txt";
              textArray=new ArrayList();
              sr=new StreamReader(path);
              while(sr.Peek()!=-1)
              {
                    fText=Convert.ToChar(sr.Read());
                    textArray.Add(fText);
                    cCount+=1;
              }
              sr.Close();
        }

        public static void Sort(cell [] TableCell , int n)
        {
              cell temp;
              for(int i=0;i<n;i++)
              {
                    for(int j=i;j<n;j++)
                    {
                          if(TableCell[i].freq > TableCell[j].freq)
                          {
                                temp=TableCell[i];
                                TableCell[i]=TableCell[j];
                                TableCell[j]=temp;

                          }
                    }
              }
        }

        public static void Initial()
        {
              node=new cell[charArray.Count ];
```

```
                for(int i=0;i<node.Length;i++)
                {
                        node[i].cText=Convert.ToString(charArray[i]);
                        node[i].freq=1;
                }
        }

        public static void LookupTable()
        {

                int accum;
                String concatChar;
                cell concatCell=new cell();

                TableCell=new cell[node.Length];
                TempTable=new cell[node.Length];
                jagTable=new cell[node.Length][];


                if(node.Length!=1)
                {
                        for(int e=0;e<node.Length;e++)
                        {
                                TableCell[e]=node[e];
                        }

                        jagTable[0]=new cell[TableCell.Length];
                        for(int d=0;d<TableCell.Length;d++)
                        {
                                jagTable[0][d]=node[d];
                                TempTable[d]=node[d];
                        }

                        Sort(TempTable,TempTable.Length);
                        for(int
z=1,aLen=TableCell.Length;z<TableCell.Length;z++,aLen--)
                        {
                                Sort(jagTable[z-1],aLen);
                                accum=jagTable[z-1][0].freq+jagTable[z-
1][1].freq;
                                concatChar=jagTable[z-1][0].cText +
jagTable[z-1][1].cText;
                                concatCell.cText=concatChar;
                                concatCell.freq=accum;
                                jagTable[z]=new cell [aLen-1];
                                jagTable[z][0]=concatCell;

                                for(int j=2;j<jagTable[z-1].Length ;j++)
                                {
```

```
                        jagTable[z][j-1]=jagTable[z-1][j];
                  }
            }

            for(int x=TempTable.Length-2;x>=0;x--)
            {
                  for(int j=0;j<jagTable[x].Length;j++)
                  {
                        if(j%2==0)
                        {
                              jagTable[x][j].code="1";
                        }
                        else
                        {
                              jagTable[x][j].code="0";
                        }
                  }
            }



      }
      else
      {
            TempTable[0].cText=node[0].cText;
            TempTable[0].freq=1;
            TempTable[0].code="1";
            jagTable[0]=new cell[TableCell.Length];
            jagTable[0][0]=TempTable[0];

      }

}

public static void AdaptiveCoding()
{
      for(int i=0;i<textArray.Count;i++)
      {
            currentChar=Convert.ToChar(textArray[i]);
            LookupTable();
            AssignCode();
            UpdateTable();

      }
}
public static void UpdateTable()
{
      for(int i=0;i<node.Length;i++)
      {
```

```
                        if(currentChar==Convert.ToChar(node[i].cText))
                        {
                              node[i].freq+=1;
                        }
                  }
            }
            public static void AssignCode()
            {
                  // assignment of code for each character in the file
                  // assigning 1 to the smallest value anode 0 to the
seconode smallest one
                  bool stop=false;
                  String cTemp="";
                  for(int s=0;s<jagTable[0].Length;s++)
                  {
                        stop=false;
                        cTemp="";
                        for(int j=TempTable.Length-2;j>=0 &&
stop==false ;j--)
                        {
                              for(int k=jagTable[j].Length-1;k>=0 &&
stop==false ;k--)
                              {

      if(jagTable[0][s].cText==jagTable[j][k].cText )
                                    {

      TempTable[s].code+=jagTable[j][k].code ;
                                          stop=true;
                                    }
                                    else
if(jagTable[j][k].cText.IndexOf(jagTable[0][s].cText,0)!=-1 &&
cTemp!=jagTable[j][k].cText)
                                    {

      TempTable[s].code+=jagTable[j][k].code;
                                          cTemp=jagTable[j][k].cText;
                                    }
                              }
                        }
                  }
                  for(int i=0;i<TempTable.Length;i++)
                  {

      if(Convert.ToChar(TempTable[i].cText)==currentChar)
                        {
                              codeString+=TempTable[i].code;
                        }
                  }
```

```
            }

      public static void GenerateCompresedData()
      {
                  byte trueByte=1;
                  byte falseByte=0;



                  codeByte=new byte[codeString.Length];
                  byte [] blockByte=new byte[8];
                  byte bitRepres;
                  for(int d=0;d<codeString.Length;d++)
                  {
                        if(codeString[d]=='0')
                              codeByte[d]=falseByte;
                        else
                              codeByte[d]=trueByte;
                  }
                  int Quat=codeByte.Length/8;
                  int lRemain=codeByte.Length %8;
                  if(lRemain==0)
                  {
                        int r=0;
                        int w=0;

                        for(w=0;w<codeByte.Length;)
                        {
                              for(r=0;r<8;r++,w++)
                              {
                                    blockByte[r]=codeByte[w];
                              }

      bitRepres=Convert.ToByte(blockByte[0]|(blockByte[1]<<1)|(blockByt
e[2]<<2)|(blockByte[3]<<3)|(blockByte[4]<<4)|(blockByte[5]<<5)|(blockBy
te[6]<<6)|(blockByte[7]<<7));
                              bw.Write(bitRepres);

      outputArray.Add(Convert.ToByte(bitRepres));
                        }
                  }
                  else
                  {
                        int r=0;
                        int w=0;

                        for(w=0;w<Quat*8;)
                        {
                              for(r=0;r<8;r++,w++)
```

```
                              {
                                    blockByte[r]=codeByte[w];
                              }

      bitRepres=Convert.ToByte(blockByte[0]|(blockByte[1]<<1)|(blockByt
e[2]<<2)|(blockByte[3]<<3)|(blockByte[4]<<4)|(blockByte[5]<<5)|(blockBy
te[6]<<6)|(blockByte[7]<<7));

      outputArray.Add(Convert.ToByte(bitRepres));
                              bw.Write(bitRepres);
                        }
                        for(r=0,w=Quat*8;w<codeByte.Length;w++,r++)
                        {
                              blockByte[r]=codeByte[w];
                        }
                        for(w=codeByte.Length;w<Quat*8;w++,r++)
                        {
                              blockByte[r]=falseByte;
                        }

      bitRepres=Convert.ToByte(blockByte[0]|(blockByte[1]<<1)|(blockByt
e[2]<<2)|(blockByte[3]<<3)|(blockByte[4]<<4)|(blockByte[5]<<5)|(blockBy
te[6]<<6)|(blockByte[7]<<7));
                        outputArray.Add(Convert.ToByte(bitRepres));
                        bw.Write(bitRepres);

                  }

                  bw.Close();
            }


      public static void TArray()
      {
            charArray=new ArrayList();
            bool exist;
            if(textArray.Count!=0)
            {
                  charArray.Add(textArray[0]);
                  for(int j=1;j<textArray.Count;j++)
                  {
                        exist=false;
                        for(int k=0;k<charArray.Count &&
exist==false;k++)
                        {

      if(Convert.ToChar(textArray[j])==Convert.ToChar(charArray[k]))
                              {
                                    exist=true;
```

```
                              }

                        }
                        if(exist==false)
                        {
                              charArray.Add(textArray[j]);
                        }
                  }
            }
      }
      public static void OverHead()
      {
            byte special=27;
            char [] cArray=new char[charArray.Count];
            for(int i=0;i<cArray.Length;i++)
            {
                  cArray[i]=Convert.ToChar(charArray[i]);
            }
            for(int i=0;i<cArray.Length ;i++)
            {
                  bw.Write(cArray[i]);
                  outputArray.Add(Convert.ToByte(cArray[i]));

      }

            bw.Write(Convert.ToByte(special));
            outputArray.Add(Convert.ToByte(special));
      }
      public static void WriteData()
      {



            byte [] outputBytes=new byte[outputArray.Count];
            for(int i=0;i<outputArray.Count;i++)
            {
                  Console.WriteLine(outputArray[i]);
                  outputBytes[i]=Convert.ToByte(outputArray[i]);

            }
            Console.WriteLine("Length: "+outputArray.Count);
//          for(int i=0;i<outputBytes.Length;i++)
//          {
//                bw.Write(outputBytes[i]);
//          }


      }
```

```
            static void Main(string[] args)
      {

                  fs=new
FileStream("test.data",FileMode.OpenOrCreate,FileAccess.Write);
                  bw=new BinaryWriter(fs);
                  outputArray=new ArrayList();
                  wCount();
                  TArray();
                  Initial();
                  OverHead();
                  AdaptiveCoding();
                  GenerateCompresedData();
                  WriteData();



            }
      }
}
```

Adaptive Huffman Text Decompression Class

```
using System;
using System.IO;
using System.Collections;

namespace AdpDecomp
{
	/// <summary>
	/// Summary description for Class1.
	/// </summary>
	class Class1
	{
		public static StreamWriter sw;
		public static ArrayList charArray;
		public static cell [] node;
		public static cell [] TableCell;
		public static cell [] TempTable;
		public static cell [][] jagTable;
		public static String codeString;
		public static byte special=27;

		public struct cell
		{
			public String cText;
			public int freq;
			public String code;
		}
		/// <summary>
		/// The main entry point for the application.
		/// </summary>
		[STAThread]

		public static void Initial()
		{
			node=new cell[charArray.Count ];
			for(int i=0;i<node.Length;i++)
			{
				node[i].cText=Convert.ToString(charArray[i]);
				node[i].freq=1;
			}
		}

		public static void ReadHeader()
		{
			charArray=new ArrayList();
			while(br.PeekChar()!=27)
			{
				charArray.Add(Convert.ToChar(br.ReadByte()));
```

```
		}
	}

	public static void LookupTable()
	{

		int accum;
		String concatChar;
		cell concatCell=new cell();
		TableCell=new cell[node.Length];
		TempTable=new cell[node.Length];
		jagTable=new cell[node.Length][];

		if(node.Length!=1)
		{
			for(int e=0;e<node.Length;e++)
			{
				TableCell[e]=node[e];
			}
			jagTable[0]=new cell[TableCell.Length];
			for(int d=0;d<TableCell.Length;d++)
			{
				jagTable[0][d]=node[d];
				TempTable[d]=node[d];
			}

			Sort(TempTable,TempTable.Length);
			for(int
z=1,aLen=TableCell.Length;z<TableCell.Length;z++,aLen--)
			{
				Sort(jagTable[z-1],aLen);
				accum=jagTable[z-1][0].freq+jagTable[z-
1][1].freq;
				concatChar=jagTable[z-1][0].cText +
jagTable[z-1][1].cText;
				concatCell.cText=concatChar;
				concatCell.freq=accum;
				jagTable[z]=new cell [aLen-1];
				jagTable[z][0]=concatCell;

				for(int j=2;j<jagTable[z-1].Length ;j++)
				{
					jagTable[z][j-1]=jagTable[z-1][j];
				}
			}

			for(int x=TempTable.Length-2;x>=0;x--)
			{
```

```
                              for(int j=0;j<jagTable[x].Length;j++)
                              {
                                    if(j%2==0)
                                    {
                                          jagTable[x][j].code="1";
                                    }
                                    else
                                    {
                                          jagTable[x][j].code="0";
                                    }
                              }
                        }

                  }
                  else
                  {
                        TempTable[0].cText=node[0].cText;
                        TempTable[0].freq=node[0].freq;
                        TempTable[0].code="1";
                  }
            }

            public static void AssignCode()
            {
                  // assignment of code for each character in the file
                  // assigning 1 to the smallest value anode 0 to the
seconode smallest one
                  bool stop=false;
                  String cTemp="";
                  for(int s=0;s<jagTable[0].Length;s++)
                  {
                        stop=false;
                        cTemp="";
                        for(int j=TempTable.Length-2;j>=0 &&
stop==false ;j--)
                        {
                              for(int k=jagTable[j].Length-1;k>=0 &&
stop==false ;k--)
                              {

      if(jagTable[0][s].cText==jagTable[j][k].cText )
                                    {

      TempTable[s].code+=jagTable[j][k].code ;
                                          stop=true;
                                    }
                                    else
if(jagTable[j][k].cText.IndexOf(jagTable[0][s].cText,0)!=-1 &&
cTemp!=jagTable[j][k].cText)
```

```
                                    {
TempTable[s].code+=jagTable[j][k].code;
                                          cTemp=jagTable[j][k].cText;
                                    }
                              }
                        }
                  }
            }

            public static void Sort(cell [] TableCell , int n)
            {
                  cell temp;
                  for(int i=0;i<n;i++)
                  {
                        for(int j=i;j<n;j++)
                        {
                              if(TableCell[i].freq > TableCell[j].freq)
                              {
                                    temp=TableCell[i];
                                    TableCell[i]=TableCell[j];
                                    TableCell[j]=temp;

                              }
                        }
                  }
            }


            public static void ReadCompData()
            {
                  codeString="";
                  ArrayList BArray=new ArrayList();
                  while(br.PeekChar()!=-1)
                  {
                        if(br.ReadByte()==Convert.ToByte(""))
                        {
                              br.ReadByte();
                        }
                        else
                        {
BArray.Add(Convert.ToByte(br.ReadByte()));
                        }
                  }

                  byte [] dataByte=new byte[BArray.Count];
                  byte [] y=new byte[8];
```

```
                  for(int j=0;j<BArray.Count;j++)
                  {
                        dataByte[j]=Convert.ToByte(BArray[j]);
                  }

                  for(int i=0;i<dataByte.Length;i++)
                  {
                        Console.WriteLine(dataByte[i]);
                        for(int j=0;j<8;j++)
                        {
                              y[j]=Convert.ToByte(dataByte[i] &
Convert.ToByte(Math.Pow(2,j)));
                              if(y[j]==0)
                              {
                                    codeString+="0";
                              }
                              else
                              {
                                    codeString+="1";
                              }

                        }
                  }
            }

            public static void AdaptiveDecoding()
            {
                  bool stop=false;
                  String currentCode=Convert.ToString(codeString[0]);
                  LookupTable();
                  AssignCode();


                  for(int i=0;i<codeString.Length-1;)
                  {
                        stop=false;

                        for(int j=0;j<TempTable.Length &&
stop==false;j++)
                        {
                              if(TempTable[j].code ==currentCode &&
i<codeString.Length-1)
                              {
                                    sw.Write(TempTable[j].cText);
                                    i++;

      currentCode=Convert.ToString(codeString[i]);
                                    for(int k=0;k<node.Length;k++)
                                    {
```

```
                                          if(node[k].cText ==
TempTable[j].cText)
                                          {
                                                node[k].freq+=1;
                                          }
                                    }
                                    stop=true;
                                    LookupTable();
                                    AssignCode();
                              }
                        }
                        if(stop==false && i<codeString.Length-1)
                        {
                              i++;
                              currentCode+=codeString[i];
                        }
                  }
                  sw.Close();
            }


            static void Main(string[] args)
            {
                  sw=new StreamWriter("input.txt");
                  FileStream fs=new
FileStream("test.data",FileMode.OpenOrCreate ,FileAccess.Read);
                  BinaryReader br=new BinaryReader(fs);

                  while(br.PeekChar()!=-1)
                  {
                        Console.WriteLine(br.ReadByte());

                  }
                  ReadHeader();
                  Initial();
                  ReadCompData();
                  AdaptiveDecoding();
                  br.Close();

            }
      }
}
```