CQUPT – University at Albany
Computer Science – International College

# CSI 403 --- Algorithms and Data Structures

## Project 2 --- Spring 2025

**Assigned: Tuesday, March 25th, 2025.**
**Due Date: Your co-instructor will inform you the date this exercise is to be submitted.**

## Purpose of the Project

The primary goal of this exercise is to develop a simple tool to both compress and decompress files. Its secondary goal is to develop skills for working with heaps, priority queues, and Huffman trees.

## To do

Your solution will use the Huffman encoding algorithm to both compress and decompress files. For compression it will create an encoded version of the input file by using the ASCII characters for 0 and 1. This means your compressed output will be larger than the original. The decompressing process will be based on the Huffman encoding shared between the two programs. The idea here is to have a working Huffman algorithm without being concerned with the real low-level implementation of it.

### 1. Huffman tree

The algorithm below is taken from your textbook (page 434) and must be used for the construction of your Huffman tree. You must also use the code from your textbook and the lecture notes for constructing and updating your Priority Queue by means of the Min-Heap algorithm.

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4      allocate a new node z
5      x = EXTRACT-MIN(Q)
6      y = EXTRACT-MIN(Q)
7      z.left = x
8      z.right = y
9      z.freq = x.freq + y.freq
10     INSERT(Q, z)
11 return EXTRACT-MIN(Q)
```

### 2. The Transmitter

You are to create files of bits (‘0’ and ‘1’ characters) for the input files supplied. Two types of input files are to be created for each file to be compressed. One of such files is for the *data* and the other for the *encoding* used. Details of both files are provided below.

### 2.1 The Data file

The data stored in this file will be structured as a set of data blocks. Each data block will consist of 2 SYN characters, ASCII 22, and a control character indicating the length of the message followed by a maximum of 16 data characters. Every data block must contain 16 message encoded characters

except, possibly, the case where the remainder of the file cannot fill the buffer. Consider, for instance, a data file containing the string *BCCABBDDAECCBBAEDDCC.* The transmitter will place the first 16 characters of this file in a data block. The table below illustrates the resulting structure populated with characters instead of the Huffman encodings obtained for each data character.

| | | |
|---|---|---|
| 2222 | 16 | BCCABBDDAECCBBAE |

The final data to be transmitted for the message above will be a sequence of 1s and 0s obtained from the Huffman frequencies for each character included in the data file. The table below illustrates the binary sequence 101111011 which is the Huffman encoding for the substring *BCCA.* The … is to indicate the characters that follow the *BCCA* substring are omitted.

| | | |
|---|---|---|
| 2222 | 16 | 101111011 … |

**2.2 The Encoding file**

The information stored in the *encoding* file follows the structure used for the *data* file except that 2 SOH (Start of Heading), ASCII 1, characters are used as a replacement for the SYN characters. The contents of this file are organized as a set of character/encoding pairs. The example that follows illustrates the contents of one of such files for the data block in the previous subsection. The **Huffman encoding** table displays the number of times each character is present in the referred message. Assume the data is initially available sorted by alphabetical order (A B C D E) and not by the frequency count. This means you are required to call your priority queue code to obtain the character with the smallest frequency, fix your Min-Heap and call your priority queue code a second time to obtain the next smallest frequency again. Your solution must call your priority queue every time during the construction of your Huffman tree. Please note that solutions that use a linked list to construct the priority queue will not be considered.

| | | |
|---|---|---|
| 11 | 17 | A011B10C11D00E001 |

| **Huffman encoding** | | |
|---|---|---|
| Character | Count | Code |
| A | 3 | 011 |
| B | 5 | 10 |
| C | 6 | 11 |
| D | 4 | 00 |
| E | 2 | 001 |

**3. The Receiver**

This program should take the files created by the Transmitter as input, decode the stream of bits and print out the original transmitted message on the output file.

Both programs should have three distinct layers: **layer1**, **layer2** and **layer3**. The **layer1** should contain routines to handle tasks such as converting a character into a binary bit pattern and writing bits into an output file. It should also include routines for reading bits from a file and converting bits into characters. The **layer2** should contain routines for framing (putting two SYN/SOH characters, a LENGTH character, and data into a frame), and deframing (separating those control characters from data characters). The **layer3** should contain routines to handle tasks such as reading data to be transmitted from input data file in the Transmitter and writing received data into the output data file in the Receiver, as well as any other routines that you feel necessary.

**4. Huffman encoding and decoding functions**

Your Huffman encoding / decoding solution must include the following methods:

1. **initializeFromFile(String FileName)**
   Takes the supplied file name and builds Huffman tree using a Priority Queue as discussed during the lectures. It must also print the number of bytes read. It may also use this function to compute and print the character frequency counts.

2. **encodeFile(String InFile, String OutFile)**
   Takes the supplied file names and encodes the InFile, placing the result in the OutFile. It also must check to make sure initializeFromFile has been executed. It must also print both input and output byte count and compute the size of the ENCODED file as a percentage of the size of the ORIGINAL file (level of compression).

3. **decodeFile(String InFile, String OutFile)**
   Takes the supplied file names and decodes the InFile, placing the result in the OutFile. It also checks to make sure initializeFromFile has been executed. It must also print the number of bytes received as well as the number of bytes resulting from the decoding.

You must include the following Code Structure in your solution.

```
int huffmanCode(String args)
{
        Huffman T;
        T.initializeFromFile("c:\\FileToBeProcessed.txt");
        T.EncodeFile("c:\\FileToBeProcessed.txt", "c:\\FileToBeProcessed.enc");
        T.DecodeFile("c:\\FileToBeProcessed.enc", "c:\\FileToBeProcessed.dec");
        return 0;
}
```

Your solution must be based on the following algorithms that are available on both your textbook as well as on the set of lecture notes:

Min-Heapify(A, i)
Build-Min-Heap(A, n)
Min-Heap-Minimum(A)
Min-Heap-Extract-Min(A)
Min-Heap-Increase-Key(A, x, k)

Min-Heap-Insert(A, x, n)

**5. Checking your decompressed file with the original one.**

From the command line window, you are to use **fc /b** to compare the two files.

prompt> **fc /b C:\ FileToBeProcessed.txt C:\ FileToBeProcessed.dec**
should show "**no differences found**" if both your encoder and decoder work well from end-to-end.

Your program must be developed using the C++ programming language. It should be layered, modularized, and well commented on. The following is a tentative marking scheme and what is expected to be submitted for this assignment:

1. External Documentation (as many pages necessary to fulfill the requirements listed below.) including the following:
    a. Title page.
    b. A table of contents.
    c. [5%] System documentation.
        i. A high-level data flow diagram for the system.
        ii. A list of routines and their brief descriptions.
        iii. Implementation details.
    d. [25%] Test documentation.
        i. [5%] How you tested your program.
        ii. [20%] Testing outputs. (You may use the example presented in this document as well as the data below for testing your initial prototype. However, you are encouraged to submit as many test sets as you like.)
            1) *project2-test1-input.txt*
            *BCCABBDDAECCBBAEDDCCBCCABBDDAECCBBAEDDCC*
            2) *project2-test2-input.txt*
            *SISSY SEES THE SEA-SHE SELLS SEA-SHELLS*
            3) *project2-test3-input.txt*
            *NIYON MH MONANOYIN. NIYON HMATAMHMONAN. OYIN*
    e. [5%] User documentation.
        i. How to run your program.
        ii. Describe parameter (if any).

2. Source Code
    a. [60% total] Correctness.
        i. [20%] *project2-test1-input.txt*
        *BCCABBDDAECCBBAEDDCCBCCABBDDAECCBBAEDDCC*
        ii. [20%] *project2-test2-input.txt*
        *SISSY SEES THE SEA-SHE SELLS SEA-SHELLS*
        iii. [20%] *project2-test3-input.txt*
        *NIYON MH MONANOYIN. NIYON HMATAMHMONAN. OYIN*
    b. [5%] Programming style.
        i. Layering.
        ii. Readability.

iii. Comments.
iv. Efficiency.

## How to Submit your Work

- Your solutions must be submitted according to the information provided by your co-instructor.
- Your files for *Project-2* must include the source code for all modules used for the exercise as well as an executable file to allow your solution to be tested. You must include all your source files in a Compressed (zipped) folder (.zip). Your (.zip) folder as well as the subject of your message must follow the format: *Project-2 Your Name*. Marks will be deducted if you do not follow this requirement.