

# Assignment

## ELL-786 Multimedia Systems

Submitted By:

Kartik Gupta (2018JTM2250)

&

Maj Vineet Sangwan (2019EET2594)

on

*Implementation of an Efficient CABAC  
encoder*



Bharti School Of  
Telecommunication Technology and Management  
IIT Delhi  
India  
June 28, 2020

# Contents

<b>1</b>	<b>CABAC</b>	<b>3</b>
<b>2</b>	<b>Algorithm</b>	<b>3</b>
<b>3</b>	<b>Problem Statement</b>	<b>6</b>
3.1	Implementation of Code . . . . .	7
3.2	Preprocessing on Files . . . . .	7
3.3	Results . . . . .	9
<b>4</b>	<b>Appendix</b>	<b>12</b>
4.1	Main Code . . . . .	12
4.2	Input and Output File Code . . . . .	17
4.3	Code for Preprocessing . . . . .	21
4.4	Code for QM coder . . . . .	22

## List of Figures

1	CABAC method of entropy encoding used within H264 video compression standard . . . . .	3
2	Plot . . . . .	5
3	Compression Ratio for different context values (N) . . . . .	9
4	Text file compression with $N = 2$ . . . . .	10
5	Image file compression with $N = 2$ . . . . .	10
6	Audio file compression with $N = 2$ . . . . .	11
7	Binary file compression with $N = 2$ . . . . .	11

# 1 CABAC

Context-adaptive binary arithmetic coding (CABAC) is a form of entropy encoding used in the H.264/MPEG-4 AVC and High Efficiency Video Coding (HEVC) standards. It is a lossless compression technique, although the video coding standards in which it is used are typically for lossy compression applications. CABAC is notable for providing much better compression than most other entropy encoding algorithms used in video encoding, and it is one of the key elements that provides the H.264/AVC encoding scheme with better compression capability than its predecessors.

CABAC has been adopted as a normative part of the H.264/AVC standard; it is one of two alternative methods of entropy coding in the new video coding standard. The other method specified in H.264/AVC is a low-complexity entropy-coding technique based on the usage of context-adaptively switched sets of variable-length codes, so-called Context-Adaptive Variable-Length Coding (CAVLC). Compared to CABAC, CAVLC offers reduced implementation costs at the price of lower compression efficiency. For TV signals in standard- or high-definition resolution, CABAC typically provides bit-rate savings of 10-20% relative to CAVLC at the same objective video quality.

## 2 Algorithm

The design of CABAC involves the key elements of binarization, context modeling, and binary arithmetic coding. These elements are illustrated as the main algorithmic building blocks of the CABAC encoding block diagram, as shown below.

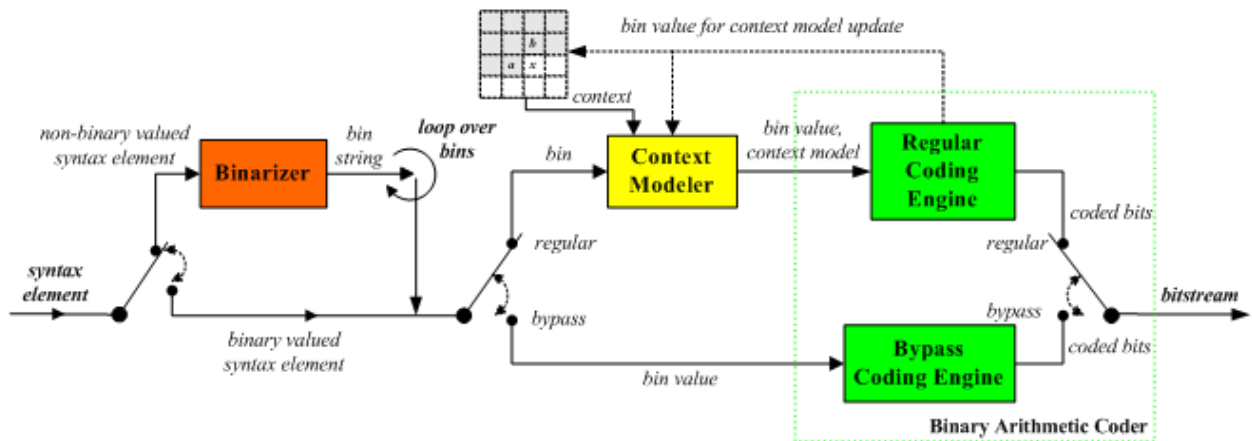


Figure 1: CABAC method of entropy encoding used within H264 video compression standard

- **Binarization :** The coding strategy of CABAC is based on the finding that a very efficient coding of syntax-element values in a hybrid block-based video coder, like components of motion vector differences or transform-coefficient level values, can be achieved by employing a binarization scheme as a kind of preprocessing unit for the subsequent stages of context modeling and binary arithmetic coding. In general, a binarization scheme defines a unique mapping of syntax element values to sequences of binary decisions, so-called bins, which can also be interpreted in terms of a binary code tree. The design of binarization schemes in CABAC is based on a few elementary prototypes whose structure enables simple online calculation and which are adapted to some suitable model-probability distributions.
- **Coding-Mode Decision and Context Modeling :** By decomposing each syntax element value into a sequence of bins, further processing of each bin value in CABAC depends on the associated coding-mode decision which can be either chosen as the regular or the bypass mode. The latter is chosen for bins related to the sign information or for lower significant bins which are assumed to be uniformly distributed and for which, consequently, the whole regular binary arithmetic encoding process is simply bypassed. In the regular coding mode, each bin value is encoded by using the regular binary arithmetic-coding engine, where the associated probability model is either determined by a fixed choice, without any context modeling, or adaptively chosen depending on the related context model. As an important design decision, the latter case is generally applied to the most frequently observed bins only, whereas the other, usually less frequently observed bins, will be treated using a joint, typically zero-order probability model. In this way, CABAC enables selective context modeling on a sub-symbol level, and hence, provides an efficient instrument for exploiting inter-symbol redundancies at significantly reduced overall modeling or learning costs. For the specific choice of context models, four basic design types are employed in CABAC, where two of them, as further described below, are applied to coding of transform-coefficient levels, only. The design of these four prototypes is based on a priori knowledge about the typical characteristics of the source data to be modeled and it reflects the aim to find a good compromise between the conflicting objectives of avoiding unnecessary modeling-cost overhead and exploiting the statistical dependencies to a large extent.
- **Pre-Coding of Transform-Coefficient Levels :** Coding of residual data in CABAC involves specifically designed syntax elements that are different from those used in the traditional run-length pre-coding approach. For each block with at least one nonzero quantized transform coefficient, a sequence of binary significance flags, indicating the position of significant (i.e., nonzero) coefficient levels within the scanning path, is produced in a first step. Interleaved with these significance flags, a sequence of so-called last flags (one for each significant coefficient level) is generated for signaling the position of the last significant level within the scanning path. This so-called significance information is transmitted as a preamble of the regarded transform block followed by

the magnitude and sign information of nonzero levels in reverse scanning order. The context-modeling specifications for coding of binarized level magnitudes are based on the number of previously transmitted level magnitudes greater or equal to 1 within the reverse scanning path, which is motivated by the observation that levels with magnitude equal to 1 are statistical dominant at the end of the scanning path. For context-based coding of the significance information, each significance / last flag is conditioned on its position within the scanning path which can be interpreted as a frequency-dependent context modeling. Furthermore, for each of the different transforms (16x16, 4x4 and 2x2) in H.264/AVC (Version 1) as well as for luma and chroma component, a different set of contexts denoted as context category is employed. This allows the discrimination of statistically different sources with the result of a significantly better adaptation to the individual statistical characteristics.

- Probability Estimation and Binary Arithmetic Coding :** On the lowest level of processing in CABAC, each bin value enters the binary arithmetic encoder, either in regular or bypass coding mode. For the latter, a fast branch of the coding engine with a considerably reduced complexity is used while for the former coding mode, encoding of the given bin value depends on the actual state of the associated adaptive probability model that is passed along with the bin value to the M coder - a term that has been chosen for the novel table-based binary arithmetic coding engine in CABAC. The specific features and the underlying design principles of the M coder can be found here. In the following, we will present some important aspects of probability estimation in CABAC that are not intimately tied to the M coder design.

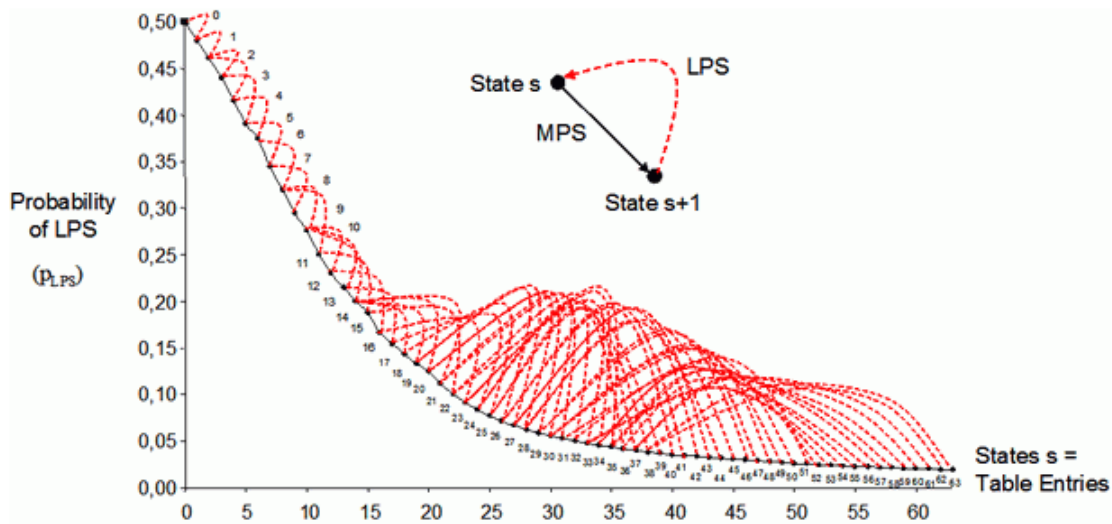


Figure 2: Plot

Probability estimation in CABAC is based on a table-driven estimator using a finite-state machine (FSM) approach with transition rules as illustrated above. Each probability model in CABAC can take one out of 128 different states with associated probability values  $p$  ranging in the interval  $[0.01875, 0.98125]$ . Note that with the distinction between the least probable symbol (LPS) and the most probable symbol (MPS), it is sufficient to specify each state by means of the corresponding LPS-related probability  $p_{LPS} \in [0.01875, 0.5]$ . The method for designing the FSM transition rules was borrowed from Howaro and Vittere using a model of "exponential aging" with the following transition rule from time instance  $t$  to  $t+1$

$$p^{(t+1)}_{LPS} = \begin{cases} a \cdot p^{(t)}_{LPS} & \text{if an MPS occurs} \\ a \cdot p^{(t)}_{LPS} + (1-a) & \text{if an LPS occurs} \end{cases}$$

According to this relationship, the scaling factor  $\alpha$  can be determined by  $p_{\min} = 0.5a^{N-1}$ , with the choice of  $p_{\min} = 0.01875$  and  $N = 128/2 = 64$ . Note however that the actual transition rules, as tabulated in CABAC and as shown in the graph above, were determined to be only approximately equal to those derived by this exponential aging rule.

Typically, without any prior knowledge of the statistical nature of the source, each model would be initialized with the state corresponding to a uniform distribution ( $p = 0.5$ ). However, in cases where the amount of data in the process of adapting to the true underlying statistics is comparably small, it is useful to provide some more appropriate initialization values for each probability model in order to better reflect its typically skewed nature. This is the purpose of the initialization process for context models in CABAC which operates on two levels. On the lower level, there is the quantization-parameter dependent initialization which is invoked at the beginning of each slice. It generates an initial state value depending on the given slicedependent quantization parameter  $SilceQP$  using a pair of so-called initialization parameters for each model which describes a modeled linear relationship between the  $SilceQP$  and the model probability  $p$ . As an extension of this low-level pre-adaptation of probability models, CABAC provides two additional pairs of initialization parameters for each model that is used in predictive (P) or bi-predictive (B) slices. since the encoder can choose between the corresponding three tables of initialization parameters and signal its choice to the decoder, an additional degree of pre-adaptation is achieved, especially in the case of using small slices at low to medium bit rates.

### 3 Problem Statement

The goal of this assignment is to implement an efficient CABAC (Context-adaptive binary arithmetic coding) encoder. You will implement a CABAC encoder and include as many proposed techniques as possible to improve its efficiency in terms of bit rate.

### 3.1 Implementation of Code

- **File Read :** The data files are read 1 byte at a time and converted into bit stream. C++ I/O (ifstream, ofstream) routines have been used for file reading/writing
- **Main Code :** The source code for CABAC is used which does the encoding given the 'context' and the 'current symbol'.  
Following routines (in red) of the source code provided are called:

```
FILE *fp;
fp=fopen("CABACencoded.dat", "w+"); //Stores the encoded data in this
//file
//calling the parameterized constructor for CABAC with the file pointer
QM obj(fp);

//Initialize the encoder parameters
obj.StartQM("encode");

//Encoding begins
WHILE Last Symbol read
    IF current symbol = 0
        gc = get Context based on previous n bits
        obj.encode(0,gc)
    ELSE IF current symbol = 1
        gc = get Context based on previous n bits
        obj.encode(0,gc)
    END IF
END WHILE

//Flush the remaining contents
obj.Flush();
```

- **File Write :** File Writing is done by the provided source code itself.

### 3.2 Preprocessing on Files

The preprocessing is done in the following way:

**Step 1.** The whole files is read and stored as bytes.

**Step 2.** First the MSB of each byte is read and stored, then the 2<sup>nd</sup>MSB is read and stored and eventually the *LSB* is read and stored.

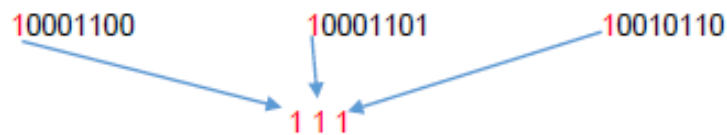
E.g. If the file contains 3 bytes '140, 141, 150 ' then the preprocessing is done as follows:



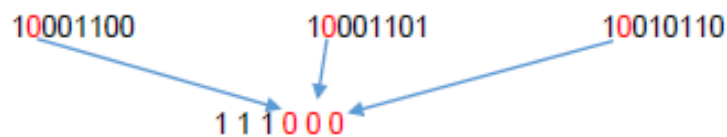
Bytes	140	141	150
-------	-----	-----	-----

Bit pattern	10001100	10001101	10010110
-------------	----------	----------	----------

Step 1. Read the MSB



Step 2. Read the 2<sup>nd</sup> MSB



Step 8. Read the LSB



Final bit stream is '11100000001110111001010'

This algorithm is chosen because:

- There is no drastic change between neighboring pixels/data in image/audio file. So the bytes are more or less same in neighboring areas.
- QM coder gives best compression when it gets a long stream of MPS.
- The above algorithm gives long stream of 0's and make the data amenable for compression using QM coding.

### 3.3 Results

The original files are read directly bit by bit and the CABAC encoder is used with Preprocessing on files for different values of context i.e.  $N = 2, 3, 4$

		<b>N = 2</b>	
<b>Input File</b>	<b>Original File Size (bytes)</b>	<b>Output File Size (bytes)</b>	<b>Compression Ratio (%)</b>
<b>text.dat</b>	8358	6101	136.99
<b>Image.dat</b>	65536	57162	114.64
<b>audio.dat</b>	65536	45261	144.79
<b>binary.dat</b>	65536	7565	866.30

		<b>N = 3</b>	
<b>Input File</b>	<b>Original File Size (bytes)</b>	<b>Output File Size (bytes)</b>	<b>Compression Ratio (%)</b>
<b>text.dat</b>	8358	6107	136.85
<b>Image.dat</b>	65536	56985	115.00
<b>audio.dat</b>	65536	45146	145.16
<b>binary.dat</b>	65536	7505	873.23

		<b>N = 4</b>	
<b>Input File</b>	<b>Original File Size (bytes)</b>	<b>Output File Size (bytes)</b>	<b>Compression Ratio (%)</b>
<b>text.dat</b>	8358	6107	136.85
<b>Image.dat</b>	65536	56937	115.10
<b>audio.dat</b>	65536	45101	145.30
<b>binary.dat</b>	65536	7481	876.03

Figure 3: Compression Ratio for different context values (N)

```

rollo@rollo-Dell-System-Inspiron-N4110: ~/Desktop/MM FINAL/Multimedia_Assignment3-master/CABAC/Source Codes
-----
WELCOME
-----
Enter the file to be encoded
/home/rollo/Desktop/MM FINAL/Multimedia_Assignment3-master/CABAC/Compressed Files/Text/text.dat
File has successfully uploaded
-----
Enter the number of context
You can select only 1,2,3,4
2
-----
Select the option by entering 1 or 2:
1. Do bitMap Preprocessing
2. Don't do Preprocessing
1
You have chosen option : 1
File content has successfully placed in memory
Preprocessed File stored in *PreProcessed.dat*
File content has successfully placed in memory
Done Encoding :)
-----
INPUT Information
-----
Input File Name      : /home/rollo/Desktop/MM FINAL/Multimedia_Assignment3-master/CABAC/Compressed Files/Text/text.dat
Context Dependency   : 2
PreProcessing        : YES
Original File size   : 8358 bytes
-----
OUTPUT Information
-----
Output File Name     : CABACencoded.dat
Compressed File size  : 6072 bytes
Compression Ratio     : 137.648%
-----
(base) rollo@rollo-Dell-System-Inspiron-N4110:~/Desktop/MM FINAL/Multimedia_Assignment3-master/CABAC/Source Codes$

```

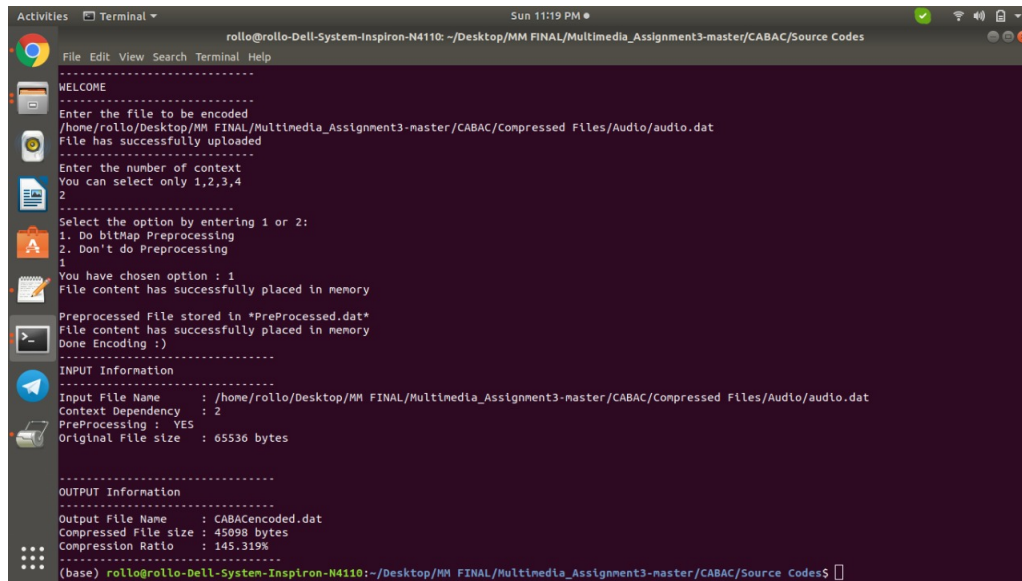
Figure 4: Text file compression with  $N = 2$ 

```

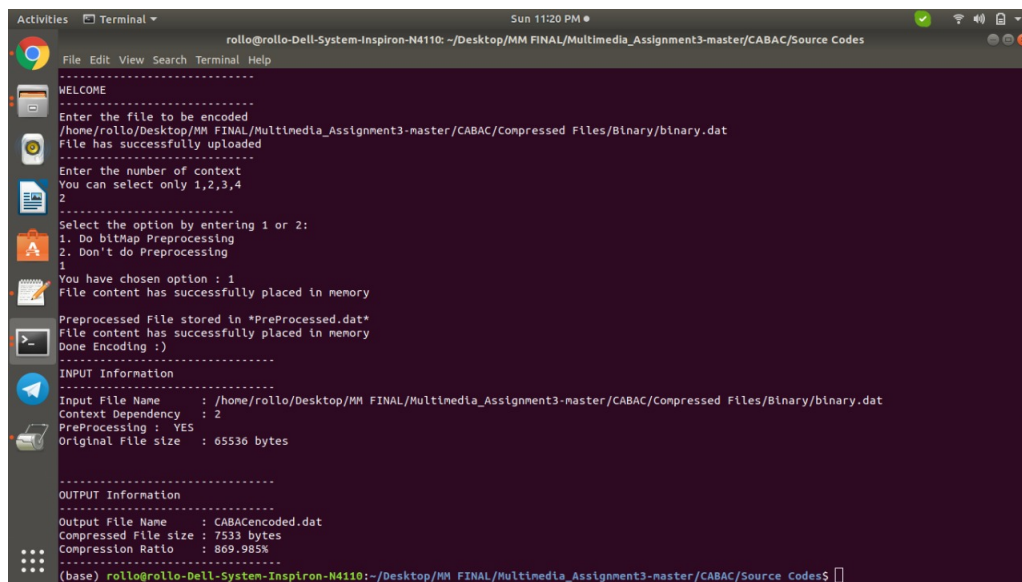
rollo@rollo-Dell-System-Inspiron-N4110: ~/Desktop/MM FINAL/Multimedia_Assignment3-master/CABAC/Source Codes
-----
WELCOME
-----
Enter the file to be encoded
/home/rollo/Desktop/MM FINAL/Multimedia_Assignment3-master/CABAC/Compressed Files/Image/image.dat
File has successfully uploaded
-----
Enter the number of context
You can select only 1,2,3,4
2
-----
Select the option by entering 1 or 2:
1. Do bitMap Preprocessing
2. Don't do Preprocessing
1
You have chosen option : 1
File content has successfully placed in memory
Preprocessed File stored in *PreProcessed.dat*
File content has successfully placed in memory
Done Encoding :)
-----
INPUT Information
-----
Input File Name      : /home/rollo/Desktop/MM FINAL/Multimedia_Assignment3-master/CABAC/Compressed Files/Image/image.dat
Context Dependency   : 2
PreProcessing        : YES
Original File size   : 65536 bytes
-----
OUTPUT Information
-----
Output File Name     : CABACencoded.dat
Compressed File size  : 56934 bytes
Compression Ratio     : 115.169%
-----
(base) rollo@rollo-Dell-System-Inspiron-N4110:~/Desktop/MM FINAL/Multimedia_Assignment3-master/CABAC/Source Codes$

```

Figure 5: Image file compression with  $N = 2$



```
rollo@rollo-Dell-System-Inspiron-N4110: ~/Desktop/MM FINAL/Multimedia_Assignment3-master/CABAC/Source Codes
-----
WELCOME
-----
Enter the file to be encoded
/home/rollo/Desktop/MM FINAL/Multimedia_Assignment3-master/CABAC/Compressed Files/Audio/audio.dat
File has successfully uploaded
-----
Enter the number of context
You can select only 1,2,3,4
2
-----
Select the option by entering 1 or 2:
1. Do bitMap Preprocessing
2. Don't do Preprocessing
1
You have chosen option : 1
File content has successfully placed in memory
Preprocessed File stored in *PreProcessed.dat*
File content has successfully placed in memory
Done Encoding :)
-----
INPUT Information
-----
Input File Name      : /home/rollo/Desktop/MM FINAL/Multimedia_Assignment3-master/CABAC/Compressed Files/Audio/audio.dat
Context Dependency   : 2
PreProcessing        : YES
Original File size   : 65536 bytes
-----
OUTPUT Information
-----
Output File Name     : CABACencoded.dat
Compressed File size  : 45098 bytes
Compression Ratio     : 145.319%
-----
(base) rollo@rollo-Dell-System-Inspiron-N4110:~/Desktop/MM FINAL/Multimedia_Assignment3-master/CABAC/Source Codes$
```

Figure 6: Audio file compression with  $N = 2$ 

```
rollo@rollo-Dell-System-Inspiron-N4110: ~/Desktop/MM FINAL/Multimedia_Assignment3-master/CABAC/Source Codes
-----
WELCOME
-----
Enter the file to be encoded
/home/rollo/Desktop/MM FINAL/Multimedia_Assignment3-master/CABAC/Compressed Files/Binary/binary.dat
File has successfully uploaded
-----
Enter the number of context
You can select only 1,2,3,4
2
-----
Select the option by entering 1 or 2:
1. Do bitMap Preprocessing
2. Don't do Preprocessing
1
You have chosen option : 1
File content has successfully placed in memory
Preprocessed File stored in *PreProcessed.dat*
File content has successfully placed in memory
Done Encoding :)
-----
INPUT Information
-----
Input File Name      : /home/rollo/Desktop/MM FINAL/Multimedia_Assignment3-master/CABAC/Compressed Files/Binary/binary.dat
Context Dependency   : 2
PreProcessing        : YES
Original File size   : 65536 bytes
-----
OUTPUT Information
-----
Output File Name     : CABACencoded.dat
Compressed File size  : 7533 bytes
Compression Ratio     : 869.985%
-----
(base) rollo@rollo-Dell-System-Inspiron-N4110:~/Desktop/MM FINAL/Multimedia_Assignment3-master/CABAC/Source Codes$
```

Figure 7: Binary file compression with  $N = 2$ 

All the Codes, Results and Input-Output files are available on GitHub.

Click the below link :

[https://github.com/2018JTM2250/Multimedia\\_Assignment3.git](https://github.com/2018JTM2250/Multimedia_Assignment3.git)

## 4 Appendix

### 4.1 Main Code

```

1 //=====
2 // Name      : CABAC.cpp
3 // Description : CABAC main code
4 //=====
5
6
7 // including libraries
8 #include <iostream>
9 #include <math.h>
10 #include "QmCoder/qmcode.h"
11 #include "FileIO/fileIO.h"
12 #include "PreProcessing/bitPlaneMap.h"
13
14
15 // defining functions
16 void printFileSize(int, char[], char[], int, int, int);
17 void shiftBit(bool *array, bool nextBit);
18 void printBoolArray(bool *array);
19 int getContext(bool *array);
20
21 using namespace std;
22
23 //Global Variables initialization
24 char *memblock;
25 int n;
26 int preprocess;
27
28 // Main program //
29 int main() {
30
31     int flag1 = 0;
32     int flag2 = 0;
33
34     //User Interaction to enter file_name
35     cout << "_____ " << endl;
36     cout << "WELCOME" << endl;
37     cout << "_____ " << endl;
38     cout << "Enter the file to be encoded " << endl;
39     char filename[1024];
40     cin.getline(filename, 1024);
41     cout << "File has successfully uploaded" << endl;
42
43     //User Interaction to enter the context
44     while(flag1 == 0){
45         cout << "_____ " << endl;

```

```

46     cout << "Enter the number of context" << endl;
47     cout << "You can select only 1,2,3,4" << endl;
48     cin >> n;
49     //Enter the # of context dependency
50     if(n == 1 || n == 2 || n == 3 || n == 4 )
51     {
52         flag1 = 1;
53     }
54     else{
55         cout << "Invalid Entry ..!!!" << endl;
56         cout << "Enter either 1,2,3 or 4" << endl;
57         cout << "_____ " << endl;
58     }
59 }
60
61 //User interaction to do preprocessing
62
63 while(flag2 == 0){
64     cout << "_____ " << endl;
65     cout << "Select the option by entering 1 or 2: " << endl;
66     cout << "1. Do bitMap Preprocessing " << endl;
67     cout << "2. Don't do Preprocessing " << endl;
68     cin >> preprocess;
69
70     if(preprocess == 1 || preprocess == 2)
71     {
72         cout << "You have chosen option : " << preprocess << endl;
73         flag2 = 1;
74     }
75     else{
76         cout << "Invalid Entry ..!!!" << endl;
77         cout << "Enter either 1 or 2" << endl;
78         cout << "_____ " << endl;
79     }
80 }
81
82
83 // _____//
84 //-----Main code begins-----//
85
86 //Array to store the previous (memory) for context bits
87 bool *array = new bool[n];
88 int i=0;
89 while (i < n)
90 {
91     array[i] = 0;
92     i++;
93 }

```

```

94
95 // Read the Input file
96 // char filename[] = "resources/image.dat";
97 //read the contents of the file and store it in memblock
98 memblock = readFileByBytes(filename);
99 int originalFileSize = FileSizeinBytes;
100
101 //Do PreProcessing
102 //This will store the preprocessed file in *PreProcessed.dat* file
103 if(preprocess == 1){
104     bitmapProcessing(memblock, originalFileSize);
105     //Pre-processed Input filename
106     char filename1[] = "PreProcessed.dat";
107     //read the contents of the file and store it in memblock
108     memblock = readFileByBytes(filename1);
109 }
110
111 //output the data to this file
112 FILE *fp;
113 fp=fopen("CABACencoded.dat", "w+");
114 //calling the parameterized constructor for CABAC
115 QM obj(fp);
116 //Initialize the encoder parameters
117 obj.StartQM("encode");
118
119 //Read the file till EOF
120 while (!checkEOF()) {
121     if (ReadBit() == 0){
122         int gc = getContext(array);
123         obj.encode(0,getContext(array));
124         shiftBit(array,0);
125     }
126     else{
127         int gc = getContext(array);
128         obj.encode(1,getContext(array));
129         shiftBit(array,1);
130     }
131 }
132
133 //Flush the remaining contents
134 obj.Flush();
135 cout << "Done Encoding :);";
136 fclose(fp);
137
138 //Output file Size
139 char outputFilename[] = "CABACencoded.dat";
140 char *opFilename = outputFilename;
141 int encodedFileSize = getFileSize(opFilename);
142
143 //Print the result

```

```
144     printFileSize(n, filename , outputFilename , encodedFileSize , originalFileSize ,
145                 preprocess);
146 //     cout << "Done/!!";
147 //return 0;
148 }
149
150
151
152
153
154
155 /**
156  * This function changes the context when a
157  * new symbol is read.
158  * Basically it shifts the array to the right.
159  * In the bool array, position 0 is the most recent position.
160  */
161
162
163 void shiftBit(bool *array, bool nextBit){
164     int i = n-1;
165     while (i >= 0)
166     {
167         if(i == 0){
168             array[i] = nextBit;
169         }
170
171         else{
172             array[i] = array[i-1];
173         }
174
175         i = i-1;
176     }
177 }
178
179
180 void printBoolArray(bool *array){
181     int i = 0;
182     while (i < n)
183     {
184         cout << array[i] << " ";
185         i++;
186     }
187     cout << endl;
188 }
189
190
191 /**
192  * THis function computes the context based
```



```

193 * on the bool array.
194 * e.g. if the bool array has entries {0,1,1},
195 * then the context is  $2^{(0)}*0 + 2^{(1)}*1 + 2^{(2)}*1 = 5$ 
196 */
197 int getContext(bool *array){
198     int context = 0;
199     int i = 0;
200
201     while (i < n)
202     {
203         context += pow(2,i)*array[i];
204         i++;
205     }
206
207     return context;
208 }
209
210 /**
211 * This function prints the input and output info.
212 */
213 void printFileSize(int n, char inputFilename[], char outputFilename[], int
    encodedFileSize, int originalFileSize, int preprocess){
214
215     // Displaying input information
216     cout << "_____ " << endl;
217     cout << "INPUT Information" << endl ;
218     cout << "_____ " << endl ;
219
220
221     cout << "Input File Name      : " << inputFilename << endl;
222     cout << "Context Dependency      : " << n << endl;
223
224     if(preprocess == 1){
225         cout << "PreProcessing : YES" << endl;
226     }
227     else{
228         cout << "PreProcessing : NO " << endl;
229     }
230
231     cout << "Original File size      : " << originalFileSize << " bytes" << endl;
232
233
234     // Displaying output information
235     cout << "_____ " << endl;
236     cout << "OUTPUT Information" << endl ;
237     cout << "_____ " << endl;
238
239
240     cout << "Output File Name      : " << outputFilename << endl;
241     cout << "Compressed File size : " << encodedFileSize << " bytes" << endl;

```

```

242 cout << "Compression Ratio : " << (float)encodedFileSize/originalFileSize
    *100 << "%" << endl << endl;
243 cout << "_____ " << endl;
244 }

```

## 4.2 Input and Output File Code

```

1 //=====
2 // Name      : fileIO.cpp
3 // Description : Reading and Writing files
4 //=====
5
6
7 // including header file "fileIO.h"
8 #include "fileIO.h"
9
10 // declaring a global variable for storing the size of the file in Bytes
11 int FileSizeinBytes;
12
13 // Stream class to write on files
14 std::ofstream myFile;
15
16 using namespace std;
17
18 // function to get size of the file
19 int getFileSize(char *filename){
20
21     streampos size; // stream position type
22
23     //Stream class to read from files
24     // ios::in – open a file for reading
25     // ios::ate–Open a file for output and move the read/write control to the
        end of the file.
26     ifstream file (filename, ios::in|ios::binary|ios::ate);
27
28     if(file.is_open()) {
29         //cout << "Reading size of the file" << endl;
30         size = file.tellg();
31         file.close();
32     }
33     else{
34         cout << "opening desired file for getting size of the file is unsuccessful
            " << endl;
35     }
36     return size;
37 }
38
39 // function to read content of the file
40 char* readFileByBytes(char* fileName){
41

```

```

42     streampos size;    // stream position type
43     char * memblock;
44
45     //Stream class to read from files
46     // ios::in – open a file for reading
47     // ios::ate–Open a file for output and move the read/write control to the
48     // end of the file .
49     ifstream file (fileName , ios::in|ios::binary|ios::ate);
50
51     if (file.is_open())
52     {
53         //cout << "Reading data of the file" << endl;
54         size = file.tellg();    // returns the current      get      position of
55         // the pointer in the stream
56         memblock = new char [size];
57         file.seekg (0, ios::beg);
58         file.read (memblock, size);
59         file.close();
60
61         cout << "File content has successfully placed in memory" <<endl;
62         FileSizeinBytes = size;
63
64     }
65
66     else {
67         cout << "Opening file for reading content is unsuccessful" << endl ;
68     }
69
70     return memblock;
71 }
72
73 void WriteByte(unsigned char byte){
74     cout << " " << (int)byte ;
75 }
76
77 /**
78  * write the file .
79  * @param –
80  * 1. char data : data you want to write in the file .
81  */
82 void writeFileByBytes(unsigned char data){
83     //cout << "data written in file" << endl;
84     myFile << data;
85 }
86
87 namespace Wr{
88     unsigned char b;
89     int s;
90 }

```

```

90
91 /**
92  * This function write the bits given from MSB-> LSB
93  * The first bit is written to the MSB, and so on the
94  * 8th bit written to the LSB.
95  */
96
97 void WriteBit(bool x)
98 {
99     Wr::b |= (x ? 1 : 0) << (7-Wr::s);
100    Wr::s++;
101
102    if (Wr::s == 8)
103    {
104        writeFileByBytes(Wr::b);
105        Wr::b = 0;
106        Wr::s = 0;
107    }
108 }
109
110
111 void writeSingleCode(unsigned long code, char size) {
112
113     for(int i = 0; i < size; i++){
114         bool x = ((code & (1 << i)) ? 1:0);
115         WriteBit(x);
116     }
117 }
118
119 /**
120  * Check the last bits status
121  */
122 void checkStatusOfLastBit(){
123     if(Wr::s <= 8){
124         for (int i = 0; i < Wr::s; i++)
125             Wr::b |= 0 << Wr::s;
126         writeFileByBytes(Wr::b);
127     }
128 }
129
130 /**
131  * Open the file once.
132  * @param-
133  * 1. fileName : name of the file in which you
134  * want to write
135  */
136 void writePrepare(char *fileName){
137     //ios::app-> append to end of file.//No need to append
138     //ios::binary-> file is binary not text.
139     //ios::out -> write to the file

```

```

140     myFile.open(fileName, ios::out | ios::binary);
141 }
142
143 //for read bits
144 namespace RB {
145     int pointer;
146     unsigned char b1;
147     int s1;
148 }
149
150 /**
151  * This function reads the file bit by bit (starting from MSB)
152  * with the help of 3 global variables.
153  * pointer -> stores the current location in the memory block.
154  * b1 -> stores the current symbol.
155  * s1 -> stores the current count from the 8 bits.
156  */
157 bool ReadBit() {
158     if (RB::s1 == 0) {
159         RB::b1 = memblock[RB::pointer++];
160         cout << (char)b1 << endl;
161         RB::s1 = 8;
162     }
163     cout << "s: " << RB::s1 << endl;
164
165     bool bit = (RB::b1 >> (RB::s1 - 1)) & 1;
166     RB::s1--;
167     return bit;
168 }
169
170 /**
171  * This function supports the read function and helps
172  * find out the EOF
173  */
174 bool checkEOF() {
175     bool bit = 0;
176     //IF the count is equal to FileSize and s1-> points to LSB.
177     if (RB::pointer == (FileSizeinBytes) && RB::s1 == 0)
178         bit = 1;
179     cout << "pointer: " << RB::pointer << endl;
180     return bit;
181 }
182
183
184
185
186 /**
187  * Close the file once all write operations done.
188  */
189 void closeFile() {

```

```

190     myFile.close();
191 }

```

### 4.3 Code for Preprocessing

```

1  //=====
2  // Name      : bitPlaneMap.cpp
3  // Description : Bitmap Preprocessing
4  //=====
5
6
7
8  #include "bitPlaneMap.h"
9
10
11 /**
12  * This function creates a mask for reading bits
13  * starting from MSB -> LSB
14  */
15 void createMask(vector<int> &mask){
16     mask.push_back(0x80);
17     mask.push_back(0x40);
18     mask.push_back(0x20);
19     mask.push_back(0x10);
20     mask.push_back(0x08);
21     mask.push_back(0x04);
22     mask.push_back(0x02);
23     mask.push_back(0x01);
24 }
25
26
27 /**
28  * This function does the bitmap processing.
29  * Read MSB's of all the 8-bit blocks, store it as stream,
30  * then read 2nd MSB, store it in stream and so on.
31  * e.g.
32  * If the memory block has 2 elements:
33  * memblock[0] = 01100010;
34  * memblock[1] = 01110001;
35  *
36  * output bit stream: 0011110100001010
37  */
38 void preprocessFile(char *memblock, int fileSize){
39
40     vector<int> mask;
41     createMask(mask);
42
43     for(int i = 0; i < 8; i++){
44         for(int j = 0; j < fileSize; j++){
45             if((memblock[j] & mask.at(i)) != 0){

```

```

46 //         cout << 1 << " ";
47         WriteBit(1);
48     }
49     else{
50 //         cout << 0 <<" ";
51         WriteBit(0);
52     }
53     }//end of main if
54
55 }
56 cout << endl;
57 }
58
59
60 /**
61  * This function takes the input data read, does the bitmap
62  * processing and stores the result in *PreProcessed.dat* file.
63  * @param:
64  * 1. char *memblock – Memory read
65  * 2. int fileSize – size of the input file in bytes
66  */
67
68 void bitmapProcessing(char *memblock, int fileSize){
69     //Write prepare the output file.
70     char opFile[] = "PreProcessed.dat";
71     char *outputFile = opFile;
72     writePrepare(outputFile);
73
74     //pre-process the file
75     preprocessFile(memblock, fileSize);
76     cout << "Preprocessed File stored in *PreProcessed.dat*" << endl;
77     closeFile();
78
79 }

```

#### 4.4 Code for QM coder

```

1 //=====
2 // Name       : qmcoder.cpp
3 // Description : QM coder
4 //=====
5
6
7
8
9 #include <stdlib.h>
10 #include "qmcoder.h"
11
12 #define QMputc(BP, m_File)    if (bFirst) {fputc(BP, m_File);} else {bFirst =
    1;};

```

```

13
14 using namespace std;
15
16 int lsz[256]= {
17     0x5a1d, 0x2586, 0x1114, 0x080b, 0x03d8,
18     0x01da, 0x0015, 0x006f, 0x0036, 0x001a,
19     0x000d, 0x0006, 0x0003, 0x0001, 0x5a7f,
20     0x3f25, 0x2cf2, 0x207c, 0x17b9, 0x1182,
21     0x0cef, 0x09a1, 0x072f, 0x055c, 0x0406,
22     0x0303, 0x0240, 0x01b1, 0x0144, 0x00f5,
23     0x00b7, 0x008a, 0x0068, 0x004e, 0x003b,
24     0x002c, 0x5ae1, 0x484c, 0x3a0d, 0x2ef1,
25     0x261f, 0x1f33, 0x19a8, 0x1518, 0x1177,
26     0x0e74, 0x0bfb, 0x09f8, 0x0861, 0x0706,
27     0x05cd, 0x04de, 0x040f, 0x0363, 0x02d4,
28     0x025c, 0x01f8, 0x01a4, 0x0160, 0x0125,
29     0x00f6, 0x00cb, 0x00ab, 0x008f, 0x5b12,
30     0x4d04, 0x412c, 0x37d8, 0x2fe8, 0x293c,
31     0x2379, 0x1edf, 0x1aa9, 0x174e, 0x1424,
32     0x119c, 0x0f6b, 0x0d51, 0x0bb6, 0x0a40,
33     0x5832, 0x4d1c, 0x438e, 0x3bdd, 0x34ee,
34     0x2eae, 0x299a, 0x2516, 0x5570, 0x4ca9,
35     0x44d9, 0x3e22, 0x3824, 0x32b4, 0x2e17,
36     0x56a8, 0x4f46, 0x47e5, 0x41cf, 0x3c3d,
37     0x375e, 0x5231, 0x4c0f, 0x4639, 0x415e,
38     0x5627, 0x50e7, 0x4b85, 0x5597, 0x504f,
39     0x5a10, 0x5522, 0x59eb
40 };
41
42 int nlps[256]= {
43     1, 14, 16, 18, 20,    23, 25, 28, 30, 33,
44     35, 9, 10, 12, 15,    36, 38, 39, 40, 42,
45     43, 45, 46, 48, 49,    51, 52, 54, 56, 57,
46     59, 60, 62, 63, 32,    33, 37, 64, 65, 67,
47     68, 69, 70, 72, 73,    74, 75, 77, 78, 79,
48     48, 50, 50, 51, 52,    53, 54, 55, 56, 57,
49     58, 59, 61, 61, 65,    80, 81, 82, 83, 84,
50     86, 87, 87, 72, 72,    74, 74, 75, 77, 77,
51     80, 88, 89, 90, 91,    92, 93, 86, 88, 95,
52     96, 97, 99, 99, 93,    95, 101, 102, 103, 104,
53     99, 105, 106, 107, 103, 105, 108, 109, 110, 111,
54     110, 112, 112
55 };
56
57 int nmpps[256]= {
58     1, 2, 3, 4, 5,    6, 7, 8, 9, 10,
59     11, 12, 13, 13, 15,    16, 17, 18, 19, 20,
60     21, 22, 23, 24, 25,    26, 27, 28, 29, 30,
61     31, 32, 33, 34, 35,    9, 37, 38, 39, 40,
62     41, 42, 43, 44, 45,    46, 47, 48, 49, 50,

```



```

63 51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
64 61, 62, 63, 32, 65, 66, 67, 68, 69, 70,
65 71, 72, 73, 74, 75, 76, 77, 78, 79, 48,
66 81, 82, 83, 84, 85, 86, 87, 71, 89, 90,
67 91, 92, 93, 94, 86, 96, 97, 98, 99,100,
68 93,102,103,104, 99, 106,107,103,109,107,
69 111,109,111
70 };
71
72 int swit[256]= {
73 1,0,0,0,0, 0,0,0,0,0,
74 0,0,0,0,1, 0,0,0,0,0,
75 0,0,0,0,0, 0,0,0,0,0,
76 0,0,0,0,0, 0,1,0,0,0,
77 0,0,0,0,0, 0,0,0,0,0,
78 0,0,0,0,0, 0,0,0,0,0,
79 0,0,0,0,1, 0,0,0,0,0,
80 0,0,0,0,0, 0,0,0,0,0,
81 1,0,0,0,0, 0,0,0,1,0,
82 0,0,0,0,0, 1,0,0,0,0,
83 0,0,0,0,0, 1,0,0,0,0,
84 1,0,1
85 };
86
87 QM::QM(FILE *FP)
88 {
89 m_File = FP;
90 max_context = 4096;
91 //max_context = 3;
92 st_table = (unsigned char *) calloc(max_context, sizeof(unsigned char));
93 mps_table= (unsigned char *) calloc(max_context, sizeof(unsigned char));
94 }
95
96 void QM::StartQM(const char *direction)
97 {
98 if (! strcmp(direction, "encode"))
99 {
100 sc = 0;
101 A_interval = 0x10000;
102 C_register = 0;
103 ct = 11;
104
105 count = -1;
106 debug = 0;
107 BP = 0;
108 bFirst = 0;
109 }
110 else if (! strcmp(direction, "decode"))
111 {
112 count = 0;

```

```

113     MPS = 0;
114     A_interval = 0x10000 ;
115     C_register = 0 ;
116     bEnd = 0;
117
118     Byte_in( );
119     C_register <<= 8 ;
120     Byte_in( );
121     C_register <<= 8 ;
122     Cx = ( unsigned ) (( C_register & 0xffff0000) >> 16) ;
123
124     ct = 0;
125     debug = 0;
126 }
127 else {
128     cout << "Command " << direction << " cannot be recognized , please use
129     encode/decode only." << endl;
130 }
131 }
132
133
134 QM::~QM()
135 {
136     free(st_table);
137     free(mps_table);
138 }
139
140
141 void
142 QM::reset()
143 {
144     for (int i = 0; i < max_context; i++)
145     {
146         st_table[i] = 0;
147         mps_table[i] = 0;
148     }
149 }
150
151
152 void
153 QM::encode(unsigned char symbol, int context )
154 {
155     if (this->debug) cout <<(char) (symbol+'0') << " " << context << endl;
156
157     if (context >= max_context)
158     {
159         unsigned char *new_st , *new_mps;
160         new_st = (unsigned char *) calloc(max_context*2, sizeof(unsigned char));
161         new_mps= (unsigned char *) calloc(max_context*2, sizeof(unsigned char));

```

```

162     memcpy(new_st, st_table, max_context*sizeof(unsigned char));
163     memcpy(new_mps, mps_table, max_context*sizeof(unsigned char));
164     max_context *= 2;
165     free(st_table);
166     free(mps_table);
167     st_table = new_st;
168     mps_table = new_mps;
169 }
170
171 next_st = cur_st = st_table[context];
172 next_MPS = MPS = mps_table[context];
173 Qe = lsz[ st_table[context] ];
174
175 if (MPS == symbol)
176     Code_MPS( );
177 else
178     Code_LPS( );
179
180 st_table[context] = next_st;
181 mps_table[context] = next_MPS;
182 };
183
184
185 void
186 QM::encode(unsigned char symbol, int prob, int mps_symbol )
187 {
188     if (this->debug) cout <<(char) (symbol+'0') << " " << prob << endl;
189
190     next_st = cur_st = 0;
191     next_MPS = MPS = mps_symbol;
192     Qe = prob;
193
194     if (MPS == symbol)
195         Code_MPS();
196     else
197         Code_LPS();
198 };
199
200
201 void
202 QM::Flush()
203 {
204     Clear_final_bits();
205     C_register <<= ct ;
206     Byte_out();
207     C_register <<= 8;
208     Byte_out();
209     QMputc(BP, m_File);
210     QMputc(0xff, m_File);count++;
211     QMputc(0xff, m_File);count++;

```

```

212 }
213
214
215 unsigned char
216 QM::decode(int context)
217 {
218     if (context >= max_context)
219     {
220         unsigned char *new_st, *new_mps;
221         new_st = (unsigned char *) calloc(max_context*2, sizeof(unsigned char));
222         new_mps = (unsigned char *) calloc(max_context*2, sizeof(unsigned char));
223         memcpy(new_st, st_table, max_context*sizeof(unsigned char));
224         memcpy(new_mps, mps_table, max_context*sizeof(unsigned char));
225         max_context *= 2;
226         free(st_table);
227         free(mps_table);
228         st_table = new_st;
229         mps_table = new_mps;
230     }
231     next_st = cur_st = st_table[context];
232     next_MPS = MPS = mps_table[context];
233     Qe = lsz[st_table[context]];
234     unsigned char ret_val = AM_decode_Symbol();
235     st_table[context] = next_st;
236     mps_table[context] = next_MPS;
237
238     if (this->debug) cout <<(char) (ret_val+'0') << " " << context << endl;
239     return ret_val;
240 };
241
242
243 unsigned char
244 QM::decode(int prob, int mps_symbol)
245 {
246     next_st = cur_st = 0;
247     next_MPS = MPS = mps_symbol;
248     Qe = prob;
249     unsigned char ret_val = AM_decode_Symbol();
250
251     if (this->debug) cout <<(char) (ret_val+'0') << " " << prob << endl;
252     return ret_val;
253 };
254
255
256 void
257 QM::Code_LPS()
258 {
259     A_interval -= Qe ;
260
261     if (!(A_interval < Qe))

```

```

262 {
263     C_register += A_interval ;
264     A_interval = Qe ;
265 }
266
267 if (swit[cur_st] == 1)
268 {
269     next_MPS = 1 - MPS;
270 }
271 next_st = nlps[cur_st];
272
273 Renorm_e();
274 };
275
276
277 void
278 QM::Code_MPS()
279 {
280     A_interval -= Qe ;
281
282     if (A_interval < 0x8000)
283     {
284         if (A_interval < Qe)
285         {
286             C_register += A_interval ;
287             A_interval = Qe ;
288         }
289         next_st = nmps[cur_st];
290         Renorm_e();
291     }
292 }
293
294
295 void
296 QM::Renorm_e()
297 {
298     while (A_interval < 0x8000)
299     {
300         A_interval <<= 1 ;
301         C_register <<= 1 ;
302         ct--;
303
304         if (ct == 0)
305         {
306             Byte_out();
307             ct = 8 ;
308         }
309     }
310 }
311

```

```

312
313 void
314 QM::Byte_out()
315 {
316     unsigned t = C_register >> 19;
317
318     if (t > 0xff)
319     {
320         BP++;
321         Stuff_0();
322         Output_stacked_zeros();
323         QMputc(BP, m_File); count++;
324         BP = t;
325     }
326     else
327     {
328         if (t == 0xff)
329         {
330             sc++;
331         }
332         else
333         {
334             Output_stacked_0xffs();
335             QMputc(BP, m_File); count++;
336             BP = t;
337         }
338     }
339     C_register &= 0x7ffff;
340 }
341
342
343 void
344 QM::Output_stacked_zeros()
345 {
346     while (sc > 0)
347     {
348         QMputc(BP, m_File); count++;
349         BP = 0;
350         sc--;
351     }
352 }
353
354
355 void
356 QM::Output_stacked_0xffs()
357 {
358     while (sc > 0)
359     {
360         QMputc(BP, m_File); count++;
361         BP = 0xff;

```

```

362     QMputc(BP, m_File);count++;
363     BP = 0 ;
364     sc--;
365 }
366 }
367
368
369 void
370 QM::Stuff_0()
371 {
372     if (BP == 0xff)
373     {
374         QMputc(BP, m_File);count++;
375         BP = 0;
376     }
377 }
378
379
380 void
381 QM::Clear_final_bits()
382 {
383     unsigned long t;
384     t = C_register + A_interval - 1 ;
385     t &= 0xffff0000 ;
386
387     if (t < C_register) t += 0x8000 ;
388
389     C_register = t ;
390 }
391
392
393 unsigned char
394 QM::AM_decode_Symbol()
395 {
396     unsigned char D;
397
398     A_interval -= Qe ;
399
400     if (Cx < A_interval)
401     {
402         if (A_interval < 0x8000)
403         {
404             D = Cond_MPS_exchange();
405             Renorm_d();
406         }
407         else
408             D = MPS;
409     }
410     else
411     {

```

```

412     D = Cond_LPS_exchange();
413     Renorm_d();
414 }
415
416 return D;
417 }
418
419
420 unsigned char
421 QM::Cond_LPS_exchange()
422 {
423     unsigned char D;
424     unsigned C_low;
425
426     if (A_interval < Qe)
427     {
428         D = MPS;
429         Cx -= A_interval;
430         C_low = C_register & 0x0000ffff;
431
432         C_register = ((unsigned long)Cx<<16) + (unsigned long)C_low;
433         A_interval = Qe;
434         next_st = nmps[ cur_st ];
435     }
436     else
437     {
438         D = 1 - MPS;
439         Cx -= A_interval;
440         C_low = C_register & 0x0000ffff;
441         C_register = ((unsigned long)Cx << 16) + (unsigned long)C_low;
442         A_interval = Qe;
443
444         if ( swit[ cur_st ]==1 )
445         {
446             next_MPS = 1-MPS;
447         }
448         next_st = nlps[ cur_st ];
449     }
450 }
451
452 return D;
453 }
454
455
456 unsigned char
457 QM::Cond_MPS_exchange( )
458 {
459     unsigned char D;
460
461     if (A_interval < Qe)

```



```

462 {
463     D = 1 - MPS;
464     if (swit[cur_st] == 1)
465     {
466         next_MPS = 1 - MPS;
467     }
468     next_st = nlps[cur_st];
469 }
470 else
471 {
472     D = MPS;
473     next_st = nmps[cur_st];
474 }
475
476 return D;
477 }
478
479
480 void
481 QM::Renorm_d( )
482 {
483     while (A_interval < 0x8000)
484     {
485         if (ct == 0)
486         {
487             if (bEnd == 0) Byte_in();
488             ct = 8;
489         }
490         A_interval <<= 1;
491         C_register <<= 1;
492         ct--;
493     }
494
495     Cx = (unsigned) ((C_register & 0xffff0000) >> 16);
496 };
497
498
499 void
500 QM::Byte_in( )
501 {
502     unsigned char B;
503     B = fgetc(m_File), count++;
504
505     if (B == 0xff)
506     {
507         Unstuff_0();
508     }
509     else
510     {
511         C_register += (unsigned) B << 8;

```

```
512     }
513 };
514
515
516 void
517 QM::Unstuff_0( )
518 {
519     unsigned char B;
520     B = fgetc(m_File), count++;
521
522     if( B == 0 )
523     {
524         C_register |= 0xff00 ;
525     }
526     else
527     {
528         if( B == 0xff )
529         {
530             //cerr << "\nEnd marker has been met!\n";
531             bEnd = 1;
532         }
533     }
534 }
535
536
537 int QM::Counting()
538 {
539     if (ct == 0)
540     {
541         return count*8;
542     }
543     else
544     {
545         return count*8+8-ct;
546     }
547 }
```

## References

- [1] Jigar. *CABAC*. GitHub. <https://github.com/jigar23/CABAC>.
- [2] J.D. Bruguera R.R. Osorio. *Arithmetic coding architecture for H.264/AVC CABAC compression system*. IEEE. <https://ieeexplore.ieee.org/document/1333259>.
- [3] VCodex. *H.264/AVC Context Adaptive Binary Arithmetic Coding (CABAC)*. <https://www.vcodex.com/h264avc-context-adaptive-binary-arithmetic-coding-cabac/>.
- [4] Wikipedia. *Context-adaptive binary arithmetic coding*. [https://en.m.wikipedia.org/wiki/Context-adaptive\\_binary\\_arithmetic\\_coding](https://en.m.wikipedia.org/wiki/Context-adaptive_binary_arithmetic_coding).