

**Batch: A-1** **Roll No.: 1711007**

**Experiment No. 9**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

****

**Title:** Lossless Compression using Run Length Encoding (study experiment)

**Objective:** To familiarize the beginner to MATLAB by introducing the basic featuresand commands of the program.

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
|  |  |
| **CO1** | Identify various discrete time signals and systems and perform signal |
| manipulation |
|  |
|  |  |

**Books/ Journals/ Websites referred:**

1. http://www.mathworks.com/support/
2. www.math.mtu.edu/~msgocken/intro/intro.html
3. www.mccormick.northwestern.edu/docs/efirst/matlab.pdf
4. A.Nagoor Kani “Digital Signal Processing”, 2nd Edition, TMH Education.

**Introduction:**

### Lossless Compression

*Lossless compression* is a method of [data compression](https://www.sciencedirect.com/topics/computer-science/data-compression) in which the size of the file is reduced without sacrificing image quality. Unlike lossy compression, no data is lost when this method is used. Because the data is preserved, the technique will decompress the data and restore it exactly to its original state. Because no data is removed from the file, it can't be compressed to the same degree as lossy compression. On average, files are reduced by a maximum of 50 percent, whereas lossy compression can surpass this and provide a greater reduction in file size. To achieve compression without losing data, several methods can be used:

1. **Adaptive dictionary algorithms** Dictionary-based algorithms shrink files by finding common elements that match a catalog of data. One method that is used by GIFs and TIFFs is LZW. When the [compression algorithm](https://www.sciencedirect.com/topics/computer-science/compression-algorithm) is run, it will compare data in the file to data that is stored in the dictionary, and then replace values in the image with those in the dictionary. To illustrate this with a simple example, let's say the dictionary contained the values 1-my, 2-is, 3-stupid. If the file contained the words “my job is stupid,” it could replace the words with values from the dictionary index, making it “1 job 2 3” in the file. To decompress the file, it would then use the dictionary to revert the compressed data back to its original form. The more words that are used in the file that will match those in the dictionary, the greater the reduction in size when the file is saved.
2. **Deflation** This method uses a combination of [Huffman Coding](https://www.sciencedirect.com/topics/computer-science/huffman-coding) and LZ77 (Lempel-Ziv) coding algorithms. Deflation is used by PNG, MNG, and TIFF formats as well as by compression programs such as ZIP, PKZIP, and GZIPLempel-Ziv coding works by finding sequences of data that have been repeated, and then replaces the repeated data with information regarding where to find it before. A “sliding window” is used to keep track of data that appeared previously in a file so that a repetition of the data can be replaced with information concerning how far back that sequence resides in the data, and the length of the repeated data.
3. **Entropy encoding** This method compresses images by looking for patterns in the data, and replacing those that occur frequently in the image with codes that are smaller in size. Huffman Coding is an example of entropy encoding. It assigns variable-length codes to a known set of values. Values that occur the most frequently in a file are assigned the shortest codes, and patterns that occur infrequently are assigned longer codes. In doing so, the bulk of data is replaced by the shortest possible code, which means less data is used to create the file.
4. **Run-length encoding** Data will often contain sequences of the same data occurring repeatedly in a file; run-length encoding finds these repetitive patterns and stores them as a count of how often the pattern repeats and the value. For example, if we looked at the line AAAABBC, we would see that it consists of four A’s, two B’s and one C. Using run-length encoding, we could convert this to 4A2BC.

For lossless [compression algorithms](https://www.sciencedirect.com/topics/computer-science/compression-algorithm), we measure the compression effect by the amount of shrinkage of the source file in comparison to the size of the compressed version. Following this idea, several approaches can be easily understood by the definitions below:

* **Compression ratio** This is simply the ratio of the output to the input file size of a [compression algorithm](https://www.sciencedirect.com/topics/computer-science/compression-algorithm), i.e. the compressed file size after the compression to the source file size before the compression.

Compression ratio= (size after Compression)/ (size before Compression)

* **Compression factor:** This is the reverse of [*compression ratio*](https://www.sciencedirect.com/topics/mathematics/compression-ratio).

Compression factor= (size before Compression)/ (size after Compression)

* **Saving percentage:** This shows the shrinkage as a percentage.

Saving percentage=100 \* ((size before Compression) – (size after Compression))/ (size before Compression)

**Huffman Compression:**

An [algorithm](https://www.webopedia.com/TERM/A/algorithm.html) for the [lossless compression](https://www.webopedia.com/TERM/L/lossless_compression.html) of files based on the frequency of occurrence of a symbol in the file that is being compressed. The Huffman algorithm is based on statistical coding, which means that the probability of a symbol has a direct bearing on the length of its representation. The more probable the occurrence of a symbol is, the shorter will be its [bit](https://www.webopedia.com/TERM/B/bit.html)-size representation. In any file, certain characters are used more than others. Using [binary](https://www.webopedia.com/TERM/B/binary.html) representation, the number of bits required to represent each character depends upon the number of characters that have to be represented. Using one bit we can represent two characters, i.e., 0 represents the first character and 1 represents the second character. Using two bits we can represent four characters, and so on.

Unlike [ASCII](https://www.webopedia.com/TERM/A/ASCII.html) code, which is a fixed-length code using seven bits per character, Huffman compression is a variable-length coding system that assigns smaller codes for more frequently used characters and larger codes for less frequently used characters in order to reduce the size of files being compressed and transferred.

For example, in a file with the following data:

XXXXXXYYYYZZ

The frequency of "X" is 6, the frequency of "Y" is 4, and the frequency of "Z" is 2. If each character is represented using a fixed-length code of two bits, then the number of bits required to store this file would be 24, i.e., (2 x 6) + (2x 4) + (2x 2) = 24.

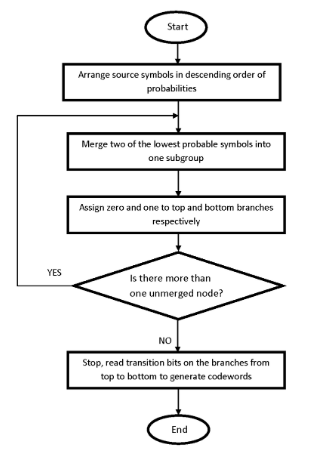
If the above data were compressed using Huffman compression, the more frequently occurring numbers would be represented by smaller bits, such as:

X by the code 0 (1 bit)  
Y by the code 10 (2 bits)  
Z by the code 11 (2 bits)

Therefore the size of the file becomes 18, i.e., (1x 6) + (2 x 4) + (2 x 2) = 18.

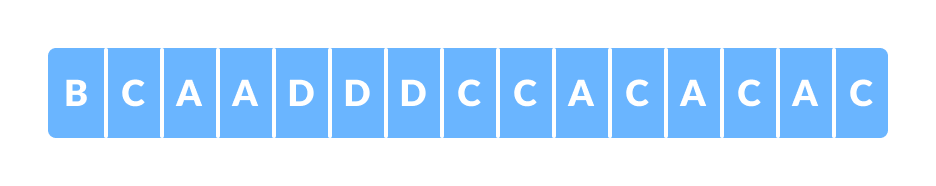
In the above example, more frequently occurring characters are assigned smaller codes, resulting in a smaller number of bits in the final compressed file.

**Flowchart (Huffman Coding):**



**Example** (Huffman Compression):

Suppose the string below is to be sent over a network.



Each character occupies 8 bits. There are a total of 15 characters in the above string. Thus, a total of 8 \* 15 = 120 bits are required to send this string.

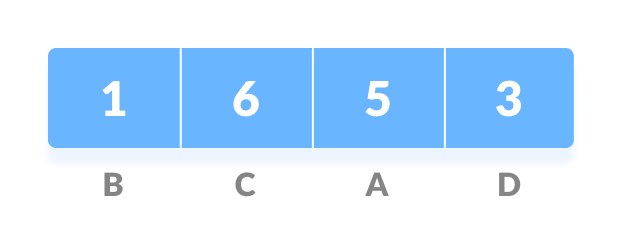
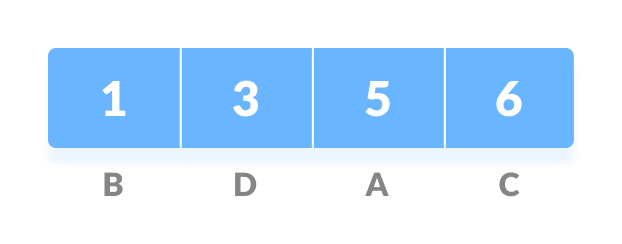
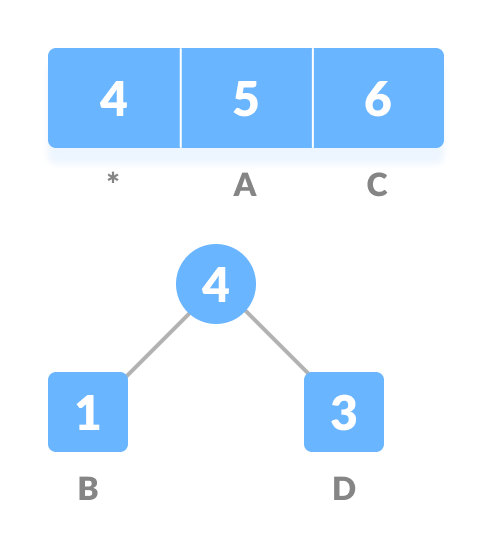
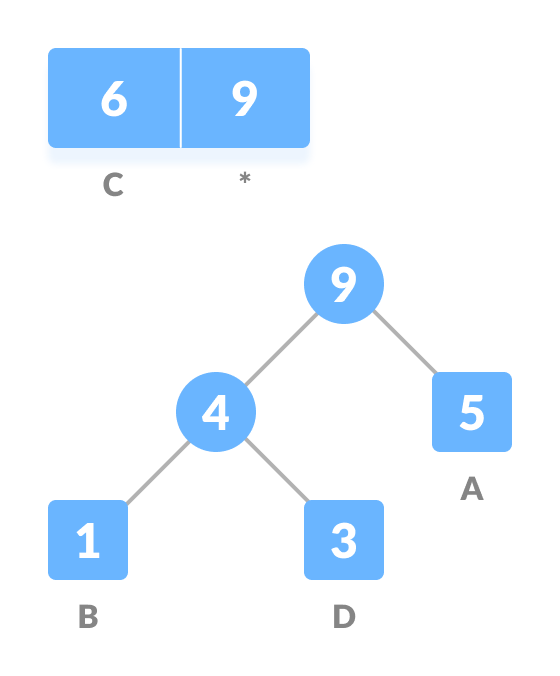
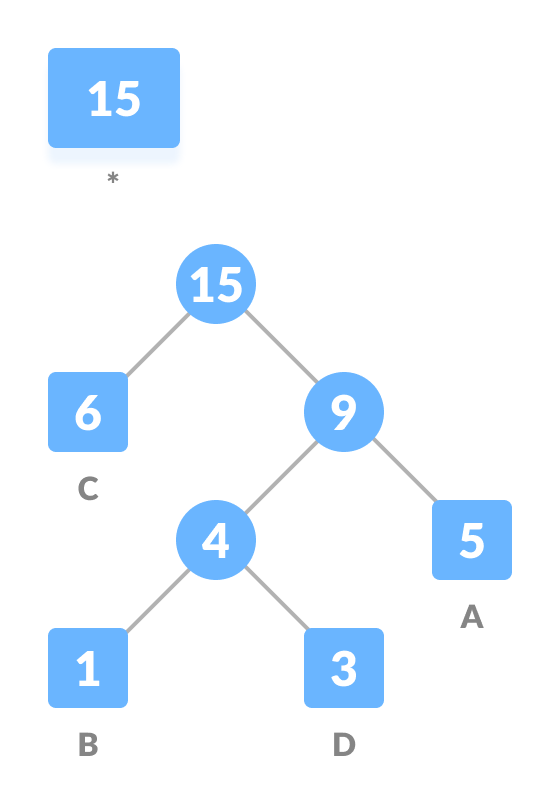
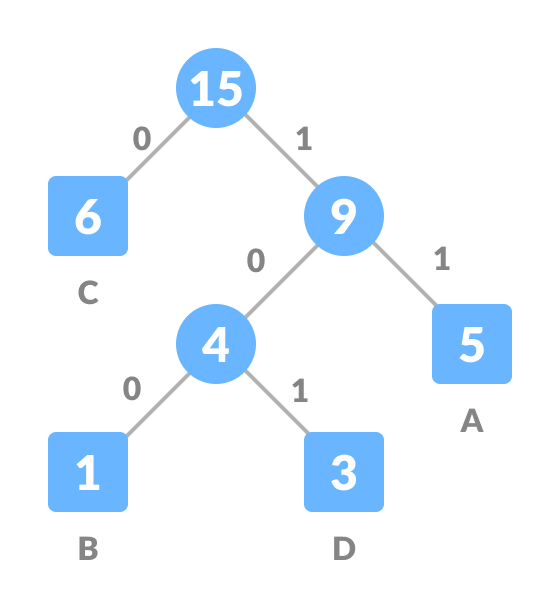
Using the Huffman Coding technique, we can compress the string to a smaller size.

Huffman coding first creates a tree using the frequencies of the character and then generates code for each character.

Once the data is encoded, it has to be decoded. Decoding is done using the same tree.

Huffman Coding prevents any ambiguity in the decoding process using the concept of **prefix code** ie. a code associated with a character should not be present in the prefix of any other code. The tree created above helps in maintaining the property.

Huffman coding is done with the help of the following steps.

1. Calculate the frequency of each character in the string.  
     
   
2. Sort the characters in increasing order of the frequency. These are stored in a priority queue Q.  
     
   
3. Make each unique character as a leaf node.
4. Create an empty node z. Assign the minimum frequency to the left child of z and assign the second minimum frequency to the right child of z. Set the value of the z as the sum of the above two minimum frequencies.  
     
   
5. Remove these two minimum frequencies from Q and add the sum into the list of frequencies (\* denote the internal nodes in the figure above).
6. Insert node z into the tree.
7. Repeat steps 3 to 5 for all the characters.  
     
   
8. For each non-leaf node, assign 0 to the left edge and 1 to the right edge.  
     
   

For sending the above string over a network, we have to send the tree as well as the above compressed-code. The total size is given by the table below.

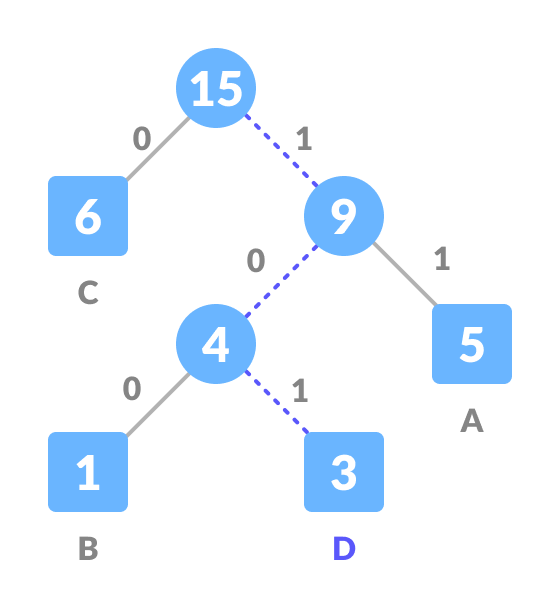
|  |  |  |  |
| --- | --- | --- | --- |
| Character | Frequency | Code | Size |
| A | 5 | 11 | 5\*2 = 10 |
| B | 1 | 100 | 1\*3 = 3 |
| C | 6 | 0 | 6\*1 = 6 |
| D | 3 | 101 | 3\*3 = 9 |
| 4 \* 8 = 32 bits | 15 bits |  | 28 bits |

Without encoding, the total size of the string was 120 bits. After encoding the size is reduced to 32 + 15 + 28 = 75.

Decoding the code:

For decoding the code, we can take the code and traverse through the tree to find the character.

Let ### is to be decoded, we can traverse from the root as in the figure below.



**Algorithm (Huffman Coding)**

create a priority queue Q consisting of each unique character.

sort then in ascending order of their frequencies.

for all the unique characters:

create a newNode

extract minimum value from Q and assign it to leftChild of newNode

extract minimum value from Q and assign it to rightChild of newNode

calculate the sum of these two minimum values and assign it to the value of newNode

insert this newNode into the tree

return rootNode

**Run Length Encoding:**

Run-length encoding (RLE) is a very simple form of data compression in which a stream of data is given as the input (i.e. "AAABBCCCC") and the output is a sequence of counts of consecutive data values in a row (i.e. "3A2B4C"). This type of data compression is lossless, meaning that when decompressed, all of the original data will be recovered when decoded. Its simplicity in both the encoding (compression) and decoding (decompression) is one of the most attractive features of the algorithm.

Here you can see a simple example of a stream ("run") of data in its original form and encoded form:

**EXAMPLE:**

**Input data**:

AAAAAAFDDCCCCCCCAEEEEEEEEEEEEEEEEE

**Output data**:

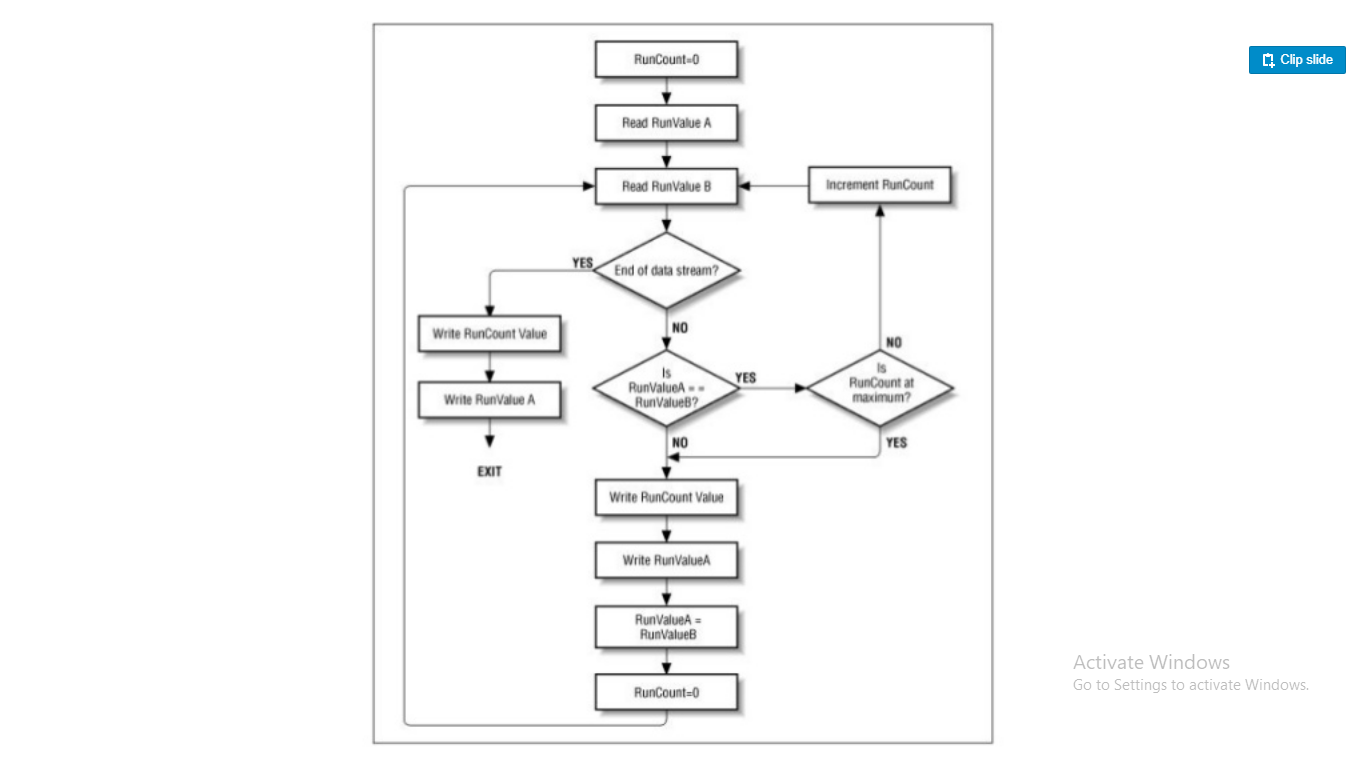
6A1F2D7C1A17E

In this example we were able to compress the data from 34 characters down to 13.

As you may have noticed, the more consecutive values in a row, the more space we save in the resulting compression. On the other hand, if you have a sequence of data that frequently changes between values (i.e. "BEFEFADED") then we won't save much space at all. In fact, we could even increase the size of our data since a single instance of a character results in 2 characters (i.e. "A" becomes "1A") in the output of the encoding.

Because of this, RLE is only good for certain types of data and applications. For example, the [Pixy camera](https://pixycam.com/pixy-cmucam5/), which is a robotics camera that helps you easily track objects, uses RLE to compress labeled video data before transferring it from the embedded camera device to an external application. Each pixel is given a label of "no object", "object 1", "object 2", etc. This is the perfect encoding for this application because of its simplicity, speed, and ability to compress the low-entropy label data.

**Flowchart (RLE):**



**Conclusion:**

Hence, a comprehensive study was done on lossless compression and its different types like Huffman coding and RLE.