International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 11, No.1 (Jan-2022)

https://dx.doi.org/10.12785/ijcds/110148

Method of Adjacent Distance Array Outperforms Conventional Huffman Codes to Decode Bengali Transliterated Text Swiftly

Pranta Sarker[1] **and Mir Lutfur Rahman**[2]

[1]*Department of Computer Science and Engineering, North East University Bangladesh, Sylhet, Bangladesh*
[2]*Department of Computer Science, University of Hertfordshire, Hertfordshire, United Kingdom*

Received 10 Jun. 2021, Revised 26 Nov. 2021, Accepted 4 Jan. 2022, Published 20 Jan. 2022

Abstract: This research works on high symbolic Bengali text and transforms it into corresponding less symbolic English complying with the transliteration method. The Huffman-based approaches serve to compress retaining the original quality of the data. On the other hand, faster encoding and decoding is the most sophisticated sphere in data compression. We propose an adjacent distance array, a novel data structure based on the Huffman principle for encoding and decoding the character of transliterated text. The encoding and decoding algorithms have been explained for the introduced modus operandi and juxtaposed with conventional Huffman-based algorithms. Our research is outdoing than any regular Huffman-based algorithms, concentrating on the speed of the encoding and decoding manner discovered after estimating all decisions.

Keywords: Huffman coding, Bengali transliterated text, Adjacent distance array, Data compression, Decoding time

1. Introduction

The consequence of compression data is to linger the quality by lessening the original file's size. However, that compression might happen on either lossy or lossless [1]. Usually, the research area of data compression emphasizes lossless compression and its techniques, and the Huffman principle [2] is considered an asset in this field. The Huffman encoding or related arithmetic coding can construct a lumpy frequency distribution, mainly relying on the symbols' neighboring context [3]. Another form of compression algorithms is based on the dictionary [4]. These algorithms are in the form of Ziv-Lempel [5], and they are capable of replacing a string to a first appearance with a pointer of the identical string [6]. The compressions based on the dictionary technique are now implementing new quaternary methods instead of traditional binary codes [7]. At present, Huffman coding is efficiently running on data mining [8] and wireless sensor networks to compress data [9], [10].

Some word-based compression exploits each word considering a basic unit [4]. Using Huffman coding, word-based text compression has already raised a standard [11]. Many natural language text compressors used data compression techniques for general purposes by performing the character-level compression [12]. A natural language might have either more symbols (such as Chinese, and Bengali) or less symbols (such as Arabic, and English) of the alphabet to represent. If we want a more symbolic natural language in terms of a less symbolic one, the occurred frequency of some symbols increases [3]. Thus, we can get a peak compression ratio according to the minimum redundancy-coding scheme of the Huffman principle [2]. This research used to translate each alphabet of a high emblematic language Bengali to English, which is less symbolic of language. It availed the technique of adjacent distance array with the Huffman principle on that transliterated text to acquire a significant decoding time while decoding process.

The rest of this research paper has been arranged into several sections. The related works of our study are summarized in Section 2. Section 3 described the objective of our research. The details of our two desired languages have presented in Section 4. The architecture for our introduced system has been developed in Section 5. It is analyzed and implemented in Section 6 and Section 7. Finally, Section 9 concluded the research by discussing all experimental data and their outcomes in Section 8.

2. Related Work

The research on Huffman-based data compression began in 1952 when [2] introduced a Binary tree-based variable code length. Under the scheme of minimum redundancy code, that research produced a small code word for the most occurred and the longest code word for the minor occurrence of symbols with an individual prefix for neglecting the ambiguity in the text. Later on, many works have been done to improve space in Huffman decoding. Reference [13] presented a data structure based on an array that requires $2n - 3$ memory for a Huffman tree, where n is the number of symbols. The data structure is further updated and reduced memory to $\lceil \frac{3n}{2} \rceil + \lceil (\frac{n}{2}) \log n \rceil + 1$ by [14]. Hashemian [15] introduced

E-mail address: psarker@neub.edu.bd, mirlutfur.rahman@gmail.com

ordering and clustering way to diminish the extra memory requirement from $O\big((n+d)\lceil \log 2n \rceil\big)$ to $O\big(2d\lceil \log 2n \rceil\big)$ bits. For the single side grown Huffman tree, the research gained $O(d)$ decoding time to remove the effect of sparsity. Lin and Chung [16] were later able to improve an ample memory less than [15]. Reference [17] got an astral memory reduced to $\lceil \frac{3n}{2} \rceil$ by introducing the leaf of circular nodes concept to define a Huffman tree. The ternary Huffman tree presented to enhance the searching process to find a symbol on the tree also alleviates the space [18].

There has been a plethora of research that emphasized the decoding performance of Huffman coding. Reference [19] implemented a recursive Huffman tree that found the decoding competence by decoding more than one symbol at a time for some small size blocks. Reference [20] provided a quaternary Huffman tree method expressed by `00` (left-branch), `01` (left-mid), `10` (right-mid), and `11` (right-branch) instead of a traditional binary tree that has `0` and `1` for both left or right child or branch. It can generate an optimum codeword so the tree can follow the 2-bit process that helps to perform the decoding on $O(\log_4 n)$ in the opposite of $O(\log_2 n)$ for traditional binary Huffman. Habib et al., [21] further upgraded their research and proposed Octanary (`tribit`-based) and Hexanary (`quadbit`-based) approaches that outperform both encoding and decoding time for some existing methods Zopfli and quaternary.

The research history of Bengali data compression is still not rich enough. One of the Bengali corpus, Ekushe-Khul, was considered by [22] to perform compression with the Type to Token Ratio and Compression Ratio concept for all large and mid-length lists. Implementing a static compression method [23] kept the Bengali short text message system stability between decoding time and compression for the devices with low processing power and storage. The first traditional Huffman-based transliteration approach has presented by [3]. They have taken a Bengali text, a more symbolic language transformed into English, a less emblematic of natural language, and applied traditional Huffman to get an optimum compression ratio for the Bengali text.

Furthermore, they increased the compression performance complying with the same method [4]. The adjacent distance array based on the Huffman technique was first applied to English text and attained an outstanding decoding time prospect performance [1]. Only for the compression purpose, they introduce the system of the adjacent distance array for the transliterated Bengali text afterward [24].

3. Research Objective

The fundamental goal of our research is to translate a more symbolic language into a less symbolic one, which is known as the transliteration process. Then we have performed the encoding and decoding process with Huffman-based novel modus operandi of the adjacent distance array on that transliterated text. We concentrated on the compression-decompression time that attained significantly compared to various and usual Huffman-based algorithms for the transliterated text. The alphabet of a natural language which is high and low symbolic, we have taken our consideration as follows —

- Bengali —as a language of more symbolic, and
- English —as a language of less symbolic.

4. Language Interpretation

A. Representation of Bengali Alphabet

The language of Bengali has a proud history. The only language in the world for which people had to give their life in 1952. The writing structure for the Bengali characters can be [3], [4]:

- বাংলা লিপি (Bangla lipi), and
- বাংলা হরফ (Bangla horof)

The vowels (e.g., অ, আ, ই, ঈ, উ, ঊ, ঋ, এ, ঐ, ও, ঔ) of the Bengali language are known as "shorborno", comprised of 11 characters. The consonants (e.g., ক, খ, গ, ঘ, ঙ, চ, ছ, জ, ঝ, ঞ, ট, ঠ, ড, ঢ, ণ, ত, থ, দ, ধ, ন, প, ফ, ব, ভ, ম, য, র, ল, শ, ষ, স, হ, ড়, ঢ়, য়, ৎ, ং, ঃ, ঁ) are called "byanjonborno", which consists of 39 characters. Besides, the Bengali alphabet has modifiers for ten vowels (e.g., া, ি, ী, ু, ূ, ৃ, ে, ৈ, ো, ৌ) and five consonants (e.g., ব-ব ফলা, -য ফলা, -র ফলা, -রেফ, -হসন্ত) that call "kar" and "fola", respectively. In the Bengali language, there also has some conjunct or joint symbols (e.g., ক্ষ, ঙ্ক, ঙ্গ, ঞ্চ, ঞ্ছ, ঞ্জ, ত্ত, ঙ্ঘ, হ্ম, ও), and often may different characters can bound unitedly like ডাক্তার, অষ্ট, আস্তে. The numeric symbols in the Bengali language are dissimilar from English (0 – 9) as those are represented by (e.g., ০, ১, ২, ৩, ৪, ৫, ৬, ৭, ৮, ৯). ৯, ২, ঋ, ৯, ৲ are Unicode characters rarely used in the Bengali language. However, Dari (।), comma (,), semicolon (;), full stop (.), colon (:), hyphen (-) and space ( ) are remaining the same as English in Bengali to manifest. In addition, the Bengali language has no case sensitivity issue, and this language considers the uppercase and lowercase English letters as equivalent. There are 16 bits of Unicode required to express each Bengali character on the computer. Figure 1 is showing all Bengali alphabets Unicode presentations.

B. Representation of English Alphabet

The English language consists of 26 alphabets (e.g., a, b, c, d, e, ..., z, A, B, C, D, E, ..., Z). There have five vowels (e.g., `a` or `A`, `e` or `E`, `i` or `I`, `o` or `O`, `u` or `U`) and the rest of consonants (e.g., `b` or `B`, `c` or `C`, `d` or `D`, `f` or `F`, `g` or `G`, `h` or `H`, `j` or `J`, ..., `n` or `N`, `p` or `P`, ..., `t` or `T`, `v` or `V`, ..., `z` or `Z`) among them. The alphabets of the English language are case sensitive, which means the lowercase alphabets (e.g., a, b, c, d, e, ..., z) are not as equivalent as uppercase (e.g., A, B, C, D, E, ..., Z). Since the ASCII character set can express the 8 bits character of the English language, the uppercase letter of combination always gives a different value with lowercase letters unless defined for a distinct goal [25].

5. Architecture

A. Transliteration Procedure

Transliteration is the process used to transform one language or script into another. However, translation is

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
U+098x	ঀ	ঁ	ং	ঃ		অ	আ	ই	ঈ	উ	ঊ	ঋ	ঌ			এ
U+099x	ঐ			ও	ঔ	ক	খ	গ	ঘ	ঙ	চ	ছ	জ	ঝ	ঞ	ট
U+09Ax	ঠ	ড	ঢ	ণ	ত	থ	দ	ধ	ন		প	ফ	ব	ভ	ম	য
U+09Bx	র		ল				শ	ষ	স	হ			়	ঽ	া	ি
U+09Cx	ী	ু	ূ	ৃ	ৄ			ে	ৈ			ো	ৌ	্	ৎ	
U+09Dx								ৗ					ড়	ঢ়		য়
U+09Ex	ৠ	ৡ	ৢ	ৣ			০	১	২	৩	৪	৫	৬	৭	৮	৯
U+09Fx	ৰ	ৱ	৲	৳	৴	৵	৶	৷	৸	৹	৺	৻	□	□	□	

Figure 1. Unicode representation of Bengali characters

another approach that differs from the transliteration system. The transliteration approach generally does not rely upon to express the novelty of linguistics. It allows the Graphemic conversion of the alphabet. Here offered some different transliteration of vowels Graphemes in Figure 2.

Systematic transliteration happens letter by letter, or even enumerate with mapping from one system to write another that a scholar can restore the genuine spelling [26]. This research is about Bengali characters used to express the Graphemic conversion of English. Since the transliteration process allows transformation, in that case, we can represent a high symbolic with a less symbolic language. Therefore, we might need the following actions in Table I to present a Bengali character. This process increases the compression ratio significantly found on [3], [4] as the Bengali text transforms corresponding 39 English or ASCII set of characters according to Avro Phonetic Layout in Figure 3. Moreover, 16 bits of Bengali Unicode characters are demoted to 8 bits ASCII characters despite the occurred frequency for each English symbol increases.

Concerning 65 Bengali characters, the necessities of the bit are determined by the modus operandi of the data structure book as follows [27]:

$$N_T = N_I + N_E = 64 + 65 = 129 \tag{1}$$

Here, N_T calculating the number of total nodes by the summation of both internal $N_I = 64$ and external $N_E = 65$ nodes. Therefore, the height or depth of the tree is:

$$\text{Depth} = \text{Floor}\,(\log_2 N_T + 1)$$

$$\text{Depth} = \text{Floor}\,(\log_2 129 + 1) = \text{Floor}\,(7.01 + 1) = 8$$

Hence, the number of bits that are wanted to represent 65 Bengali symbols is:

$$N_B = 65 \times \text{Largest level} = 65 \times (\text{Depth} - 1) \tag{2}$$

$$N_B = 65 \times (8 - 1) = 65 \times 7 = 455 \text{ bits}$$

Accordingly, for the representation of 39 English symbols, we have needed total bits are:

$$N_T = N_I + N_E = 38 + 39 = 77$$

$$\text{Depth} = \text{Floor}\,(\log_2 N_T + 1)$$

$$\text{Depth} = \text{Floor}\,(\log_2 77 + 1) = \text{Floor}\,(6.26 + 1) = 7$$

$$N_E = 39 \times \text{Largest level} = 39 \times (\text{Depth} - 1) \tag{3}$$

$$N_E = 39 \times (7 - 1) = 39 \times 6 = 234 \text{ bits}$$

Therefore, the ultimate compression ratio is:

$$r = F(N_B, N_E) \tag{4}$$

$$r = \left(\frac{N_B - N_E}{N_B}\right) \times 100\%$$

$$r = \left(\frac{455 - 234}{455}\right) \times 100\% = 48.57\%$$

As we know, the Huffman tree can not generate a complete binary tree, and it is preferably relaying on a 2-tree or an Extended binary tree. Thence the number of symbols is diminished by 26 (65 − 39 = 26) certainly to represent the corresponding Bengali text in English.

B. Compression using Adjacent Distance Array

This research adopted a Huffman-based compression mode with an adjacent distance array applied in [1]. Then, it has been initiated by transliterating the Bengali text into English [24]. The conventional Huffman forms the lowest code for the most frequent symbols. However, the codeword formed of minimum recurrent symbols grows sequentially and would be immense. It might take a much time to read data during the decoding process from the

Oṛiyā	ଅ	ଆ	ଇ	ଈ	ଉ	ଊ	ଋ	ଏ	ଐ	ଓ	ଔ
Devanāgarī	अ	आ	इ	ई	उ	ऊ	ऋ	ए	ऐ	ओ	औ
Bengali	অ	আ	ই	ঈ	উ	ঊ	ঋ	এ	ঐ	ও	ঔ
Telugu	అ	ఆ	ఇ	ఈ	ఉ	ఊ	ఋ	ఎ	ఐ	ఒ	ఔ
Translit.	a	ā	i	ī	u	ū	ṛ	e	ai	o	au

Figure 2. Transliteration for some different vowels Graphemes

TABLE I. Expressing Bengali Letters into Corresponding English

Bengali letter	**Corresponding letter(s) requires to represent in English**	
	Letter	**Type**
অ	o	Small letter
ঈ	I	Capital letter
ং	ng	Composing of small letters only
ঔ	OU	Composing of capital letters only
ঠ	Th	Composing of both small and capital letters
ঁ	^	ASCII character of the set

encoded file, denoted by *encoded*. The modus operandi in this research has strived a modification here. It has used a threshold value, denoted by T, which helps to identify the adjacent symbols. An array is preserving the distances of ASCII values among all adjacent characters that data structure we are calling the adjacent distance array, denoted by *adjacent*. For instance, the distance between '`A`' and '`C`' is `2` in ASCII Character set if we view whatever '`A`' to '`C`' $\rightarrow$ `(-2)` or '`C`' to '`A`' $\rightarrow$ `(-2)`. The distances can obtain the position of a symbol. Now, if the distance of ASCII values between two adjacent characters S_i and S_{i+1} is lesser or equal to the threshold value, T, in that case, the adjacent symbols would have $S_{i+1} + S_{i+2} + S_{i+3} + \ldots + S_{i+m}$ for a distinct character S_i.

Therefore,

$$| S_i - S_{i+1} | \leq T \tag{5}$$

If (5) is fulfilled by any distinct symbol S_i there would be happened following operations:

- The *encoded* would save the Huffman formed code, and
- The *adjacent* would keep the ASCII value distances among all satisfied symbols.

What if dissatisfaction of the condition in this case,

- S_i would recognize as a new symbol for the *encoded*.

Moreover, this procedure recommences until we get to the end of the text.

In *encoded*, '0' has been used as a "separator bit" to create a separation for each distinct symbol. In *adjacent*, the ASCII value distances of adjacent symbols hold as a unique coding scheme that relies on a threshold value, T. Thus, two kinds of a scheme of bits would generate below:

- The first kind of design for `2`-bit patterns
  - The first bit contains '1', which indicates the beginning of the code.
  - The second bit might hold any one of the following:
    - '`0`' specifies the positive value of distance, and
    - '1' indicates the non-positive value of distance.
- The second sort is the binary interpretation of distances.

That binary interpreted distances have an equal unit of bits. Again, that length is determined by how the threshold value, T can represent the maximum bits. From this concept, we can compute it as:

$$T = 2^x + 1 \tag{6}$$

Here, x is the highest amount of bits demanded to design the threshold value, T, and it is always performed accurately for this method. Therefore, the *encoded* has a memory that computed by:

$$\sigma_1 = \sum_{i=1}^{N} (F_i - A_i) \times C_i \tag{7}$$

Figure 3. Avro Phonetic Layout

In (7),
F_i = total frequency of occurred symbol.
A_i = the frequency miniaturized from the *encoded* into *adjacent*.
C_i = code bits produced by the Huffman coding scheme.
N = cumulative symbols in the *encoded*.

Again, the memory for the *adjacent* is computed by:

$$\sigma_2 = \sum_{i=1}^{M} (A_i \times x) \tag{8}$$

In the (8),
x = amount of bits is expected to represent the distance.
M = cumulative distances for the adjacent symbol.

Consequently, we have obliged to compute the whole memory for putting the complete message is:

$$\sigma = \sigma_1 + \sigma_2 + H_T + S_N \tag{9}$$

In the (9),
H_T = the header of Huffman tree.
S_N = cumulative separators in *adjacent* for each distinct symbol.

In the decoding manner, our approach is used to decode the first symbol (e.g., S_i) from the *encoded*. The *adjacent* would help to decode the following adjacent characters (e.g., S_{i+1}, S_{i+2}, S_{i+3}, …, S_{i+m}) by calculating the ASCII value distances unless any separator bit '0' is found or unless the *adjacent* has reached the end. Thus, the process keeps traversing away from the whole code list of the Huffman for adjacent characters that certainly enhance the decoding time.

6. Analysis

A. Data Analysis

We have taken some Bengali text as a sample for our consideration. Then, we have transliterated it into corresponding English text employing the Avro Phonetic Layout shown in Figure 3. In that case, we have included the frequencies of symbols for both Bengali and English text concerning their number of unique and total characters shown in Table II.

B. Decoding Time Complexity for Regular Huffman

As we know, the traditional Huffman can construct a binary tree based on the frequency of the symbols shown in Figure 4, but the tree is not balanced [21], [28] though. Meanwhile, consider the first sample text for the transliterated English from Table II.

For each character, Table III presents the ASCII values and their occurrences. That's also included the Huffman produced code bits from Figure 4 and resembling Bengali probable characters would be decoded during the decoding process.

Regarding the time complexity, for traversing a binary tree, there is a required $O(\log_2 m)$ if we consider m, the number of nodes on the tree [29]. For the time being, if we see Figure 4, we must traverse the whole Huffman binary tree at least for twelve leaf nodes during the decoding manner to get the corresponding Bengali character manifested in Table III. Therefore, on average, if a tree has k nodes for n length of the encoded string, that case, the time complexity $O(\log_2 k)$ is required for visiting the entire tree, and $O(n \log_2 k)$ works as an overall time complexity for decoding a character [30].

C. Decoding Time Complexity for Our Approach

Our adjacent distance array technique based on the Huffman principle utilizes the encoded file that saves the binary representation of each distinct symbol. Considering the *encoded*, the decoding process endeavors to decode a symbol by computing the ASCII value distances with caring desired *adjacent* that keeps all distances for each character. Hence, this process needs $O(1)$ time complexity because only an arithmetic function (subtraction) has issued exactly from keeping the whole Huffman binary tree away to traverse. In that case, our decoding

TABLE II. Some Transliterated Specimen Strings From Bengali to English

SN.	Language	Specimen texts	Total symbols	Unique symbols
1.	Bengali	ও আমার দেশের মাটি, আমি তোমায় ভালবাসি।	38	20
	English	o amar deSer maTi, ami tomay valobasi.	38	18
2.	Bengali	নাগরিকের মধ্যে সম্পদের সুষম বন্টন নিশ্চিত করা এবং রাষ্ট্রের সকল অর্থনৈতিক উন্নয়নে সম্পদ অর্জনের উদ্দেশ্যে সুষম সুযোগ-সুবিধা প্রধান করা রাষ্ট্রের অন্যতম দায়িত্ব। তাই সর্বত্র নাগরিকের মাঝে সুযোগের সমতা সৃষ্টি করা গেলেই কেবল নাগরিকের সেবা নিশ্চিত হবে; ইতিমধ্যে বাংলাদেশ একটি স্বাধীন গণতান্ত্রিক রাষ্ট্র হিসেবে বিশ্বসভায় পরিচিতি লাভ করেছে।	335	43
	English	nagoriker moddhe sompoder suShom bon-Ton nishcit kora ebong raShTrer sokol or-rthonOItik unnoyone sompod orrjoner ud-deShZe suShom suzOg-subidha prodhan kora raShTrer onZotom dayitwo. tai sorrbotro nagoriker majhe suzOger somota srriShTi kora gelei kebol nagoriker seba nishcit hobe; itimodhZe bangladesh ekTi swadhIn goNo-tantrik raShTro hisebe bishwosovay poriciti lav koreche.	375	33
3.	Bengali	মোদের গরব, মোদের আশা আ-মরি বাংলা ভাষা। মাগো তোমার কোলে, তোমার বোলে কতই শান্তি ভালবাসা। কি যাদু বাংলা গানে, গান গেয়ে দাঁড় মাঝি টানে... গেয়ে গান নাচে বাউল, গান গেয়ে ধান কাটে চাষা। আ-মরি বাংলা ভাষা। বিদ্যাপতি, চন্ডী-গোবিন, হেম-মধু, বঙ্কিম-নবীন... ঐ ফুলেরই মধুর রসে, বাঁধলো সুখে মধুর বাসা। বাজিয়ে রবি তোমার বীণে, আনলো মালা জগৎ জিনে। তোমার চরণ-তীর্থে মাগো আজি জগৎ করে যাওয়া-আসা। আমি ঐ ভাষাতেই প্রথম বোলে, ডাকনু মায়ে 'মা' 'মা' বলে। আমি ঐ ভাষাতেই বলবো হরি সাঙ্গ হলে কাঁদা হাসা। মাগো তোমার কোলে, তোমার বোলে কতই শান্তি ভালবাসা। মোদের গরব, মোদের আশা আ-মরি বাংলা ভাষা...	585	52
	English	mOder gorob, mOder asha a-mori bangla vaSha. magO tOmar kOle, tOmar bOle kotoi shanti valobaSha. ki zadu bangla gane, gan geye daˆR majhi Tane... geye gan nace baul, gan geye dhan kaTe caSha. a-mori bangla vaSha. bidZapoti, conDI-gObin, hem-modhu, boNgkim-nobIn... OI fuleroi modhur rose, baˆdhlO sukhe mod-hur basa. bajiye robi tOmar bINe, anolO mala jogot“ jine. tOmar coroN-tIrrthe magO aji jogot“ kore zaOya-asa. ami OI vaShatei pro-thom bOle, Dakonu maye 'ma' 'ma' bole. ami OI vaShatei bolobO hori saNgg hole kaˆda hasa. magO tOmar kOle, tOmar bOle kotoi shanti valobaSha. mOder gorob, mOder asha a-mori bangla vaSha...	625	39

TABLE III. Occurred Frequency and ASCII Values for Sample String – 1 (Transliterated English)

Serial no.	Characters	ASCII (decimal)	Frequency	Huffman generated bits	Corresponding Bengali character while decoding (except any modifiers* and conjunct* symbols)
0.	‘*a*’	97	7	00	‘আ’
1.	‘ ’	32	6	110	' '
2.	‘*m*’	109	4	100	‘ম’
3.	‘*i*’	105	3	1011	‘ই’
4.	‘*o*’	111	3	1110	‘অ’
5.	‘*r*’	114	2	11111	‘র’
6.	‘*e*’	101	2	11110	‘এ’
7.	‘.’	46	1	10100	‘।’
8.	‘*S*’	83	1	01110	‘শ’
9.	‘*T*’	84	1	01111	‘ট’
10.	‘*b*’	98	1	01010	‘ব’
11.	‘*d*’	100	1	01011	‘দ’
12.	‘*l*’	108	1	01000	‘ল’
13.	‘*s*’	115	1	01100	‘স’
14.	‘*v*’	118	1	01101	‘ভ’
15.	‘*y*’	121	1	01001	‘য়’
16.	‘,’	44	1	101010	‘,’
17.	‘*t*’	116	1	101011	‘ত’

* All modifiers and conjunct symbols make use of vowels and consonants for the Bengali language.

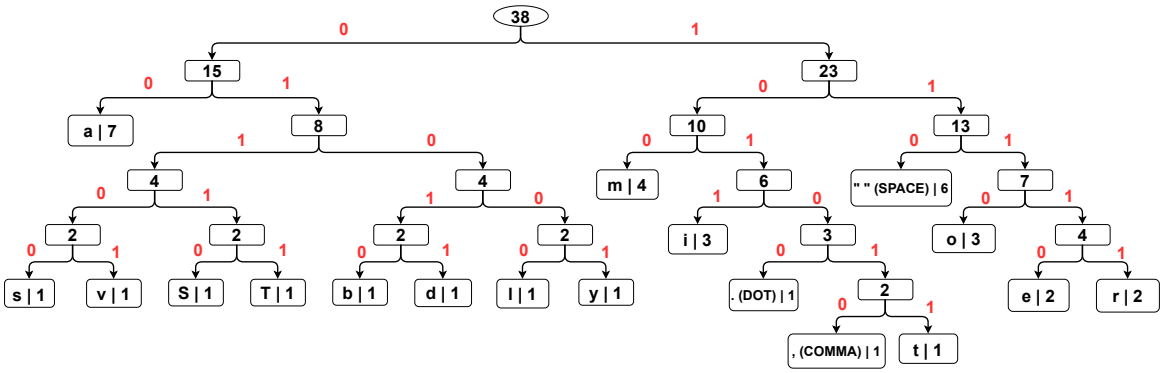

Figure 4. Structure of regular Huffman tree for sample text – 1 (transliterated English)

process demands $O\left(\left(\left(n-a\right)\log_2 n\right)+a\right)$ time complexity, which is a significant depletion from the traditional Huffman method, decodes the entire text of characters. Here, ‘*a*’ considers a code width of overall adjacent symbols. The decoding process has been described thoroughly in Section 7-B.

7. Implementation

Heretofore, we have provided a thorough analysis of the transliteration process in Section 5-A. Our novel compression proposal employs the adjacent distance array with the Huffman principle in Section 5-B. The transliteration approach defines the transformation of Bengali, identified as a high emblematic to the less emblematic English language. Moreover, the *adjacent* technique can get a significant time to decode characters while decoding, explained in Section 6-C. Hereafter, we would describe the implementation of our method. With the reduction of encoding symbols, this process can decode each character preserving adjacent symbols respecting the threshold value. Thus, there is no requirement to bother any tree or code list. The implementation of our proposed system follows two procedures:

- The Encoding procedure, and

- The Decoding procedure.

A. Elaboration of Encoding Procedure

At the beginning of the encoding process, the introduced technique takes the Bengali text (B) and transforms it into analogous Graphemic English text (E), allowing the transliteration process. It can offer a specific threshold value, T and approach to a Huffman-produced code list (L) of characters. The Huffman-generated codeword for each distinct symbol would save on the *encoded* from the transliterated English text. In the meantime, the encoding process of the proposed technique would scrutinize two several conditions as follows:

i. The resultant distance between adjacent symbols (e.g., S_{i+1}, S_{i+2}, S_{i+3}, ..., S_{i+m}) from S_i should be lesser or equal to the threshold value, T, and

ii. Each adjacent distance's code length should also be lesser or equal to the length of Huffman generated code bit.

The proposed technique offers the *adjacent* to keep distances if both (i, and ii) are true. It otherwise allows the *encoded* for considering the adjacent symbol as a new one and continues the procedure. Our proposed approach uses '0' as a separator bit for a distinct symbol in the *adjacent* after storing the distances of all adjacent characters from the *encoded*. The whole encoding process will last until the range of the transliterated English text finishes. Algorithm 1 describes the entire process in pseudocode form. However, two coded files can generate from a complete encoding process –

- The file of encoded codes, and
- The file of codes for the *adjacent*.

Algorithm 1 Encoding Algorithm

Input: Bengali text (B)
Output: *adjacent*, and *encoded*

Take the Bengali text (B) as an input
`Transliterate` the corresponding Bengali into English text (E)
Set threshold value, T, and generate the Huffman code list (L)
for $i \leftarrow 0$ to $size(E)$ **do**
  $S_i \leftarrow E[i]$
  if $i = 0$ **then**
    $encoded \leftarrow encoded \cup \{S_i\}$
    $previous \leftarrow S_i$
  else
    if $distance(S_i, previous) \leq T$ **and** $size(L_i) \geq size(T)$ **then**
      $adjacent \leftarrow adjacent \cup \left\{\left(distance\left(S_i, previous\right)\right)\right\}$
    else
      $encoded \leftarrow encoded \cup \{S_i\}$
      $previous \leftarrow S_i$
    end if
  end if
end for
return

B. Elaboration of Decoding Procedure

The decoding manner of the proposed method considers the final encoded file (*encoded*), and also Huffman formed a code list (L) of characters after completion of the encoding stage. According to the decoding procedure, the first character can decode from the *encoded*. Then it would focus on the adjacent distance array (*adjacent*) utilized in the encoding process for decoding the rest of the adjacent characters computing the distance of ASCII values. The decoding process would run until it finds a separator bit '0' or reach the ends of the *adjacent* and so on illustrated as pseudocode format in Algorithm 2. Lastly, the decoded English text converts into corresponding Graphemic Bengali following the transliteration system, indicating the decoding process's termination.

Algorithm 2 Decoding Algorithm

Input: *encoded*, *adjacent*, and Huffman code list (L)
Output: English text (E) converts into Bengali text (B) allowing the `Transliteration` system

Initialization : $e \leftarrow null$; $decoded \leftarrow \emptyset$
for $i \leftarrow 0$ to $size(encoded)$ **do**
  $e \leftarrow encoded[i]$
  if $e \in L$ **then**
    $decoded \leftarrow decoded \cup \{L.\text{symbol}\}$
    for $j \leftarrow 0$ to $size(adjacent)$ **do**
      if $adjacent[j] = 0$ **then**
        break
      else
        $\text{adj}_s \leftarrow \text{adj}_s \cup \left\{(char)\ distance \left(L.\text{symbol}, adjacent[j]\right)\right\}$
        $decoded \leftarrow decoded \cup \{\text{adj}_s\}$
      end if
    end for
  else
    $i \leftarrow i + 1$
    $e \leftarrow e.\texttt{concate}(encoded[i])$
  end if
end for
return

8. Result and Discussion

The observation intends to assess the overall performance of our proposed method juxtaposing the conventional Huffman-based algorithm to attain a significant time in the decoding manner. Accordingly, we have considered regular Huffman [31] and Zopfli [32], [33], one of the most prosperous binary Huffman-based compression mechanisms by Google Inc. Not only the Zopfli has a tremendous compression ratio, however, but it also has an enormous prospect to replace Gzip, an efficient file format employed on Internet [34]. The proposed approach of our research follows the transliteration process from Bengali to corresponding English text. Additionally, in decoding the symbols, the method has to utilize English and transliterate it into original Bengali text. In this case, we have taken into account some eminence and examined corpus data as transliterated English text condensed in Table IV to evaluate the encoding-decoding or compression-decompression performance contrasted to the traditional Huffman and Zopfli. The appraisal ran on a Linux-based `64`-bit Operating System (Ubuntu `14.04` LTS), including the hardware like `7.7` GiB Primary memory, Intel(R) Core (TM) `i5-6500` CPU `@ 3.20` GHz × `4` -Processor, and Gallium `0.4` on llvmpipe (LLVM `3.4`, `256` bits) -Graphics. The codecs are all compiled on the same GCC `4.9.2` compiler.

According to our approach, we have conceded two different threshold values to estimate the performance for each corpus, threshold values $T = 7$ and $T = 15$. Also, we have implemented the Huffman codes, and Zopfli [35] in our environment and juxtaposed the time of decoding manner with the ultimate compression method of adjacent distance array. Note that the compression-decompression time is determined by the `second` unit (in short, `S`). Also, we would utilize some notations for each corpus to compute at our convenience as follows in Table V.

A. Performance Analysis for the Canterbury Corpus

The Canterbury Corpus [36] is all about 1158.08 `Kilobytes` (in short, `KB`) in size, and it consists total 87 of discrete characters. Table VI is proffering the exploratory outcome for that corpus, which indicates the Zopfli, and regular Huffman performs very poorly than our proposed algorithm in terms of the decoding time. Though Zopfli is capable of procuring a high compression ratio, it is not our concerning point. With $T = 15$, our adjacent distance array technique outperformed any traditional Huffman-based algorithms, even the Zopfli, for the compression-decompression time. It is around 63% of time improvement on average shown in Table VII.

B. Performance Analysis for the Brown Corpus

The renowned Brown Corpus [37] is a file of 6040.63 `Kilobytes`. The number of distinct characters for this corpus is 95, so Table VIII shows a comparison study. The consequence indicates the best performance of our proposed method while the traditional Huffman-based algorithms are consuming significantly compression-decompression time. As displayed in Table IX, our algorithm with $T = 15$ performs better for achieving an average of around 59% enhancement of compression and decompression time though $T = 7$ has a good compression ratio than $T = 15$.

C. Performance Analysis for the Supara Corpus

The analyzing study for the Supara Corpus [38] has manifested in Table X. Our proposed method is usually performed up to the mark for this corpus than any binary Huffman-based algorithms. On the other hand, $T = 7$ achieved a relatively better compression ratio though this is not significant if we consider the result of $T = 15$ by concerning the compression-decompression time. Averagely, the time accomplished almost 62% improvement than Zopfli, and regular Huffman algorithm manifested in Table XI. The file size of the Supara Corpus is 1464.21 `Kilobytes`, and it is composed of 106 distinct symbols.

Summing up all of the experimental effects in a graph with Figure 5, we can say that the method of adjacent distance array can accomplish a significant compression and decompression time when the value of the threshold is rising than regular Huffman-based algorithms. Moreover, the contrast is almost half of Zopfli.

However, it can get a comparatively much compression ratio for the minimum threshold value if we take many, which happens to cause fewer adjacent characters. Again, the size of the encoded file can be huge for the fewer adjacent characters that consume much time. Hence, the threshold value is such a parameter that can regulate the overall execution of our technique. Figure 6 is appeared with a comprehensive chart for each corpus to demonstrate the time enhancement ratio concerning the Huffman-based traditional algorithms.

9. Conclusions and Future Work

This research has focused on the lossless data compression methodology. We have executed a novel modus

TABLE IV. Different Data Set has been considered for the experiment

Serial no.	Name of data set	Description	Size of the data set (in Kilobytes)
1.	Canterbury Corpus	The corpus has been designed for lossless data compression purposes. However, the improved version of this corpora is known as the Calgary Corpus.	1158.08
2.	Brown Corpus	This corpus considers the leading major formed corpora with various genres. It is not used only the compression but also for different purposes.	6040.63
3.	Supara Corpus	This corpus consists of several customized data classes such as Literature, Politics, News, and other aspects of Bangladesh in a translated copy from the Bengali language. The creators have designed it for data compression purposes only.	1464.21

TABLE V. The Table of Notations

Notations	Description
TT_n	The compression-decompression time for the threshold value is either $n = 7$ or $n = 15$.
T_{RH}	Indicates the compression-decompression time for the regular Huffman algorithm.
T_{ZP}	Indicates the compression-decompression time for the algorithm of Zopfli.
IT_{RH}	The overall improved time we compared to get the regular Huffman algorithm.
IT_{ZP}	The overall improved time we compared get to the algorithm of Zopfli.

TABLE VI. Comparison of Performance for the Canterbury Corpus (1158.08 KB)

Algorithm/Approach	Compressed file size (in Kilobytes)	Time of compression-decompression (in seconds)
Method of this research with $T = 7$ ($TT_{n=7}$)	788.18	1.449
Method of this research with $T = 15$ ($TT_{n=15}$)	796.97	1.433
Regular Huffman (TT_{RH})	681.48	4.318
Zopfli (T_{ZP})	418.50	3.431

TABLE VII. Time Improvement (in %) for Canterbury Corpus (1158.08 KB)

Algorithm/Approach	Time improvement with compared to regular Huffman, $IT_{RH} = \left(100 - \left(\frac{TT_n}{T_{RH}}\right) \times 100\right)\%$	Time improvement with compared to Zopfli, $IT_{ZP} = \left(100 - \left(\frac{TT_n}{T_{ZP}}\right) \times 100\right)\%$	Average time improvement in terms of regular Huffman and Zopfli, $\left(\frac{IT_{RH} + IT_{ZP}}{2}\right)\%$
Method of this research with $T = 7$, $n = 7$	66.44	57.77	62.11
Method of this research with $T = 15$, $n = 15$	66.81	58.23	62.52

TABLE VIII. Comparison of Performance for the Brown Corpus (6040.63 KB)

Algorithm/Approach	**Compressed file size (in Kilobytes)**	**Time of compression-decompression (in seconds)**
Method of this research with $T = 7$ ($TT_{n=7}$)	4037.61	7.064
Method of this research with $T = 15$ ($TT_{n=15}$)	4093.39	7.037
Regular Huffman (TT_{RH})	3470.66	22.353
Zopfli (T_{ZP})	2230.37	13.574

TABLE IX. Time Improvement (in %) for Brown Corpus (6040.63 KB)

Algorithm/Approach	**Time improvement with compared to regular Huffman**, $IT_{RH} = \left(100 - \left(\frac{TT_n}{T_{RH}}\right) \times 100\right)\%$	**Time improvement with compared to Zopfli**, $IT_{ZP} = \left(100 - \left(\frac{TT_n}{T_{ZP}}\right) \times 100\right)\%$	**Average time improvement in terms of regular Huffman and Zopfli**, $\left(\frac{IT_{RH} + IT_{ZP}}{2}\right)\%$
Method of this research with $T = 7$, $n = 7$	68.40	47.96	58.18
Method of this research with $T = 15$, $n = 15$	68.52	48.16	58.34

TABLE X. Comparison of Performance for the Supara Corpus (1464.21 KB)

Algorithm/Approach	**Compressed file size (in Kilobytes)**	**Time of compression-decompression (in seconds)**
Method of this research with $T = 7$ ($TT_{n=7}$)	989.04	1.824
Method of this research with $T = 15$ ($TT_{n=15}$)	1002.35	1.807
Regular Huffman (TT_{RH})	856.28	6.169
Zopfli (T_{ZP})	442.00	3.789

operandi based on the Huffman principle advancing a data structure name adjacent distance array. The data structure performs efficiently regarding the compression and decompression time by putting the distances of characters instead of vising the whole binary Huffman tree. We appropriated this theory for the transliteration approach. Moreover, we have exerted the more symbolic Bengali language, transforming it into a less symbolic English language. Subsequently, we have put the conjecture of the adjacent distance array technique in the transliterated English text. We analogized our empirical data with the traditional Huffman approaches and obtained a significant enhancement of compression-decompression time. If we infer the worst-case scenario, the time is around half lesser than the conventional Huffman algorithms acknowledged in this research. We have studied such binary Huffman algorithms that perform fitter to compress data; however, growing separator bits for the several adjacent symbols is the drawback for this purpose. Therefore, mitigating the separator bits would be the future aspect of research in our case. Besides, the study might have an example of the transliteration process that practiced only Bengali texts concerning the overall encoding-decoding time aspect. Nevertheless, the fundamental goal of this research can be investing in other languages, which have more symbols than the English language.

Acknowledgment

The authors are grateful for the help of the Robotics Lab under the Department of Computer Science and Engineering of North East University Bangladesh for the settlement of testing data.

TABLE XI. Time Improvement (in %) for Supara Corpus (1464.21 KB)

Algorithm/Approach	**Time improvement with compared to regular Huffman,** $IT_{RH} = \left(100 - \left(\frac{TT_n}{T_{RH}}\right) \times 100\right)\%$	**Time improvement with compared to Zopfli,** $IT_{ZP} = \left(100 - \left(\frac{TT_n}{T_{ZP}}\right) \times 100\right)\%$	**Average time improvement in terms of regular Huffman and Zopfli,** $\left(\frac{IT_{RH} + IT_{ZP}}{2}\right)\%$
Method of this research with $T = 7$, $n = 7$	70.43	51.86	61.15
Method of this research with $T = 15$, $n = 15$	70.71	52.31	61.51

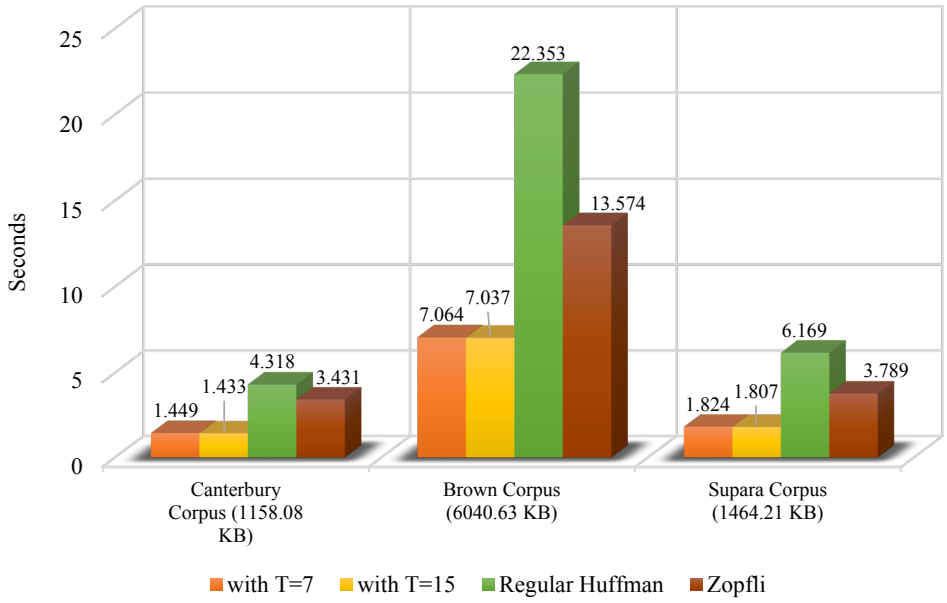

Figure 5. Summary of compression-decompression time for different corpus

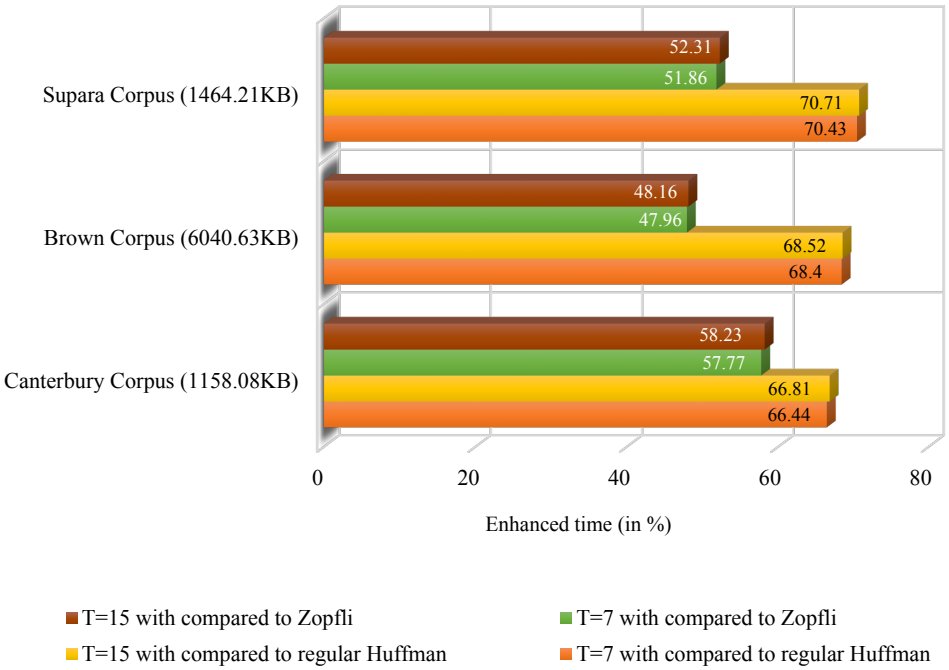

Figure 6. Time enhancement ratio for each corpus for regular Huffman and Zopfli

References

[1] M. L. Rahman, P. Sarker, and A. Habib, "A Faster Decoding Technique for Huffman Codes Using Adjacent Distance Array," in *Proceedings of International Joint Conference on Computational Intelligence*, M. S. Uddin and J. C. Bansal, Eds. Singapore: Springer Singapore, 2020, pp. 309–316.

[2] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[3] M. M. Hossain, A. Habib, and M. S. Rahman, "Transliteration Based Bengali Text Compression using Huffman principle," in *2014 International Conference on Informatics, Electronics & Vision (ICIEV)*, 2014, pp. 1–6.

[4] ——, "Performance Improvement Of Bengali Text Compression Using Transliteration And Huffman Principle," *Int. Journal of Engineering Research and Application*, vol. 6, no. 9, pp. 88–97, 2016.

[5] P. Fenwick, "Differential Ziv-Lempel Text Compression," in *J.UCS The Journal of Universal Computer Science: Annual Print and CD-ROM Archive Edition Volume 1 {\textbullet} 1995*, H. Maurer, C. Calude, and A. Salomaa, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 591–602. [Online]. Available: https://doi.org/10.1007/978-3-642-80350-5_49

[6] D. A. Lelewer and D. S. Hirschberg, "Data Compression," *ACM Comput. Surv.*, vol. 19, no. 3, p. 261296, 9 1987. [Online]. Available: https://doi.org/10.1145/45072.45074

[7] A. Habib, M. J. Islam, and M. S. Rahman, "A dictionary-based text compression technique using quaternary code," *Iran Journal of Computer Science*, vol. 3, no. 3, pp. 127–136, 2020. [Online]. Available: https://doi.org/10.1007/s42044-019-00047-w

[8] C. Oswald, A. I. Ghosh, and B. Sivaselvan, "An Efficient Text Compression Algorithm - Data Mining Perspective," in *Mining Intelligence and Knowledge Exploration*, R. Prasath, A. K. Vuppala, and T. Kathirvalavakumar, Eds. Cham: Springer International Publishing, 2015, pp. 563–575.

[9] S. Renugadevi and P. S. N. Darisini, "Huffman and Lempel-Ziv based data compression algorithms for wireless sensor networks," in *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering*, 2013, pp. 461–463.

[10] D. I. Săcăleanu, R. Stoian, and D. M. Ofrim, "An adaptive Huffman algorithm for data compression in wireless sensor networks," in *ISSCS 2011 - International Symposium on Signals, Circuits and Systems*, 2011, pp. 1–4.

[11] A. Sinaga, Adiwijaya, and H. Nugroho, "Development of word-based text compression algorithm for Indonesian language document," in *2015 3rd International Conference on Information and Communication Technology (ICoICT)*, 2015, pp. 450–454.

[12] P. M. Long, A. I. Natsev, and J. S. Vitter, "Text compression via alphabet re-representation," in *Proceedings DCC '97. Data Compression Conference*, 1997, pp. 161–170.

[13] K.-L. Chung and Y.-K. Lin, "A novel memory-efficient Huffman decoding algorithm and its implementation," *Signal Processing*, vol. 62, no. 2, pp. 207–213, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0165168497001254

[14] H.-C. Chen, Y.-L. Wang, and Y.-F. Lan, "A Memory-Efficient and Fast Huffman Decoding Algorithm." *Inf. Process. Lett.*, vol. 69, pp. 119–122, 2 1999.

[15] R. Hashemian, "Memory efficient and high-speed search Huffman coding," *IEEE Transactions on Communications*, vol. 43, no. 10, pp. 2576–2581, 1995.

[16] Y.-K. Lin and K.-L. Chung, "A space-efficient Huffman decoding algorithm and its parallelism," *Theoretical Computer Science*, vol. 246, no. 1, pp. 227–238, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0304397599000808

[17] R. Chowdhury, M. Kaykobad, and I. King, "An Efficient Decoding Technique for Huffman Codes," *Information Processing Letters*, vol. 81, 9 2001.

[18] P. Suri and M. Goel, "Ternary Tree and Memory-Efficient Huffman Decoding Algorithm," *International Journal of Computer Science Issues*, vol. 8, 1 2011.

[19] Y.-K. Lin, S.-C. Huang, and C.-H. Yang, "A fast algorithm for Huffman decoding based on a recursion Huffman tree," *Journal of Systems and Software*, vol. 85, no. 4, pp. 974–980, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121211002925

[20] A. Habib and M. S. Rahman, "Balancing decoding speed and memory usage for Huffman codes using quaternary tree," *Applied Informatics*, vol. 4, no. 1, p. 5, 2017. [Online]. Available: https://doi.org/10.1186/s40535-016-0032-z

[21] A. Habib, M. J. Islam, and M. Rahman, "Huffman Based Code Generation Algorithms: Data Compression Perspectives," *Journal of Computer Science*, vol. 14, pp. 1599–1610, 12 2018.

[22] M. R. Islam and S. A. RAJON, *On the Design of an Effective Corpus for Evaluation of Bengali Text Compression Schemes*, 1 2009.

[23] A. S. Mohammad Arif, M. Asif, and I. Rashedul, "An Enhanced Static Data Compression Scheme Of Bengali Short Message," *International Journal of Computer Science and Information Security*, vol. 4, 9 2009.

[24] P. Sarker and M. L. Rahman, "Introduction to Adjacent Distance Array with Huffman Principle: A New Encoding and Decoding Technique for Transliteration Based Bengali Text Compression," in *Progress in Advanced Computing and Intelligent Engineering*, C. R. Panigrahi, B. Pati, B. K. Pattanayak, S. Amic, and K.-C. Li, Eds. Singapore: Springer Singapore, 2021, pp. 543–555.

[25] B. S. Gottfried and . K. Chhabra, "The ASCII Character Set," in *Programming With C*, 3rd ed. Tata Mcgraw Hill, 0, ch. Two, p. 2.17.

[26] N. Kharusi and A. Salman, "The English Transliteration of Place Names in Oman," *Journal of Academic and Applied Studies*, vol. 1, pp. 1–27, 10 2011.

[27] S. Lipschutz, *Data Structures with C*, 1st ed. New York, USA: McGraw-Hill, 2010.

[28] K. K. Rajput, "Are Huffman trees balanced?" 2017. [Online]. Available: https://www.quora.com/Are-Huffman-trees-balanced?q=AreHuffmantreesbalanced%3F

[29] T. H. Cormen, C. E. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. United States: MIT Press, 1989.

[30] The University of Auckland, "Huffman Encoding." [Online]. Available: https://www.cs.auckland.ac.nz/software/AlgAnim/huffman.html#:

[31] "The source code of Regular Huffman," 10 2021. [Online]. Available: https://www.geeksforgeeks.org/huffman-coding-greedy-algo-3/

[32] J. Alakuijala, E. Kliuchnikov, Z. Szabadka, and L. Vandevenne, "Comparison of Brotli, Deflate, Zopfli, LZMA, LZHAM and Bzip2 Compression Algorithms."

[33] J. Alakuijala and L. Vandevenne, "Data compression using Zopfli." [Online]. Available: http://fh7922mg.bget.ru/articles/compression/data-compression-using-zopfli.html

[34] Z. Syed and T. Soomro, "Compression Algorithms: Brotli, Gzip and Zopfli Perspective," *Indian Journal of Science and Technology*, vol. 11, pp. 1–4, 2018.

[35] Google Inc., "The source code of Zopfli." [Online]. Available: https://github.com/google/zopfli

[36] R. Arnold, T. Bell, and M. Powell, "The Canterbury Corpus," 2001. [Online]. Available: http://corpus.canterbury.ac.nz/resources/cantrbry.zip

[37] H. Kučera and W. N. Francis, "The Brown Corpus," 1961. [Online]. Available: https://ia800306.us.archive.org/21/items/BrownCorpus/brown.zip%0A

[38] A. Habib, M. L. Rahman, and P. Sarker, "The Supara Corpus," 2017. [Online]. Available: https://github.com/shidhu/Supara-Corpus

Pranta Sarker Pranta Sarker completed his Bachelor of Science in Engineering degree in Computer Science and Engineering from North East University Bangladesh. After graduation, he joined as a Faculty member in the Department of Computer Science and Engineering at the same University. The journey of his teaching is three years now, and that might be quite long. Currently, he is pursuing a Master of Science in Thesis degree under the department of Computer Science and Engineering from Shahjalal University of Science and Technology. His research works usually focus on Data Compression, Blockchain, Trust, Reputation, and Federated Identity Management.

Mir Lutfur Rahman Mir Lutfur Rahman completed his Bachelor of Science in Engineering degree in Computer Science and Engineering from North East University Bangladesh. He joined as a Faculty member in the Department of Computer Science and Engineering at the same University after graduation. Now, his MSc degree is ongoing in Advanced Computer Science at the University of Hertfordshire. He has two years of teaching experience. His research works usually focus on Data Compression, Computer Vision, and Machine Learning.