IOSR Journal of Mathematics (IOSR-JM)
e-ISSN: 2278-5728, p-ISSN:2319-765X. Volume 9, Issue 2 (Nov. – Dec. 2013), PP 33-37
www.iosrjournals.org

The use of Algorithmic Method of Hamming Code Techniques for the Detection and Correction of Computational Errors in a Binary Coded Data: Analysis on an Integer Sequence *A119626*.

Afolabi Godfrey[1] & Ibrahim A.A[2]
[1]*Mathematics Department, Joda International School, P.M.B 1031, Nigeria.*
[2]*Department of Mathematics, Usmanu Danfodio University, Nigeria.*

Abstract: *This paper presents a review of the Algorithmic method of Hamming codes techniques for detection and correction of computational errors in a binary coded data, analysis in an integer sequence A119626. This integer sequence is obtained from computing the difference in the number of inverted pairs of the first and the last cycles which in turn is obtained from a special (123)-avoiding permutation pattern. The computation in this paper was restricted to values of n = 1, 2, 3, 4, and 5 respectively. This paper simply considered the execution time (T) and rate (R) for any given time t of the algorithmic method of analysis based on the number of iterations (steps) involved in the general procedure of encoding, detection and correction of errors for all the values of n.*

Key words*: Algorithmic method, binary coded data, execution time (T) and rate (R), Hamming codes and integer sequence A119626.*

I. Introduction

It has been generally observed that in communication, which is the process of transmission and reception of data from one place to another at some distance or in computational analysis involving integer sequences are not without error(s). There are a number of reliable codes that can be used to encode data, that is, adding parity code to the original result (data) obtained so that any error(s) that may occur can be detected and corrected. Until this is done, communication or computational results obtained will not be efficient talk more of been reliable. According to [1], the binary coded data (binary number system) is widely used in digital communication systems. Although the binary number system has many practical advantages and it is widely used in digital computer, in many cases, it is convenient to work with the decimal number system, especially when the interaction between human beings and machine is extensive. This is because, most numerical data generated by humans is basically in terms of decimal numbers. Thus, to simplify the challenges encountered in communication (interaction) or computation between human and machine, several codes have been devised in which decimal digits are represented by sequences of binary digits that is, in binary coded data form.

Hamming code is one of such codes. It is a set of error-correction codes that can be used to detect and correct bit errors that can occur when data (information or results obtained) is transmitted, stored or computed [2]. Like other error-correction code, Hamming code makes use of the concept of parity and parity bits, which are bits that are added to data (say, *k* parity bits are added to an *n*-bit data to form a new word of $n + k$ bits), so that the validity and reliability of the data are being checked when it is read or after it has been transmitted or computed and received as data. Using more than one parity bit, an error-correction code can not only identify a single bit error in the data unit, but also its position in that data unit [2]. In a Hamming encoder, parity bits are decided so as to place a fixed parity on different combinations, which are then checked for. In the decoder, parity bits are set for the fixed parity combinations which were checked. The binary equivalence of this combination decides the position of the error. Then that particular bit is flipped (interchanged from either '0' to '1' or '1' to '0') to correct the erroneous bit. Hamming code is a single error correction code, that is, it can detect a single error and proceed to correct it. It can also detect multiple (double) errors only on the condition that no correction is attempted. Error correction may be avoided at certain cases where retransmission of the computed or sent information (data) is feasible and effective. But, error detection on the other hand, is a must in all cases. Once there is a deviation from the original information (data) computed or transmitted, the cause (error) of such deviation must be detected and corrected as the case may be. Error control therefore, is the major concern of this paper.

Hence, the analysis of Algorithmic method of Hamming code techniques for detection and correction of computational errors in a binary coded data in an integer sequence *A119626* shall be discussed in this paper. This special integer sequence *A119626* is obtained from computing the difference in the number of inverted pairs of the first and the last cycles which in turn is obtained from a special (123)-avoiding permutation pattern. They are: 3, 6, 12, 30, 84, 246, 732, 2790, 8364, 25086, etc generated by the formula $3 + 3^n$, with n taking

values from 0, 1, 2, 3… , [3]. The computation in this paper shall be restricted to values of n = 1, 2, 3, 4, and 5 respectively and the results shall in no small measure contribute to the knowledge of error control. According to [4], the Algorithmic method involves the following steps:

i. Number the bits starting from 1: bit 1**, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25 etc.**

ii. Write the binary equivalence of the bits numbered in (1) above: **1, 10, 11, 100, 101, 110, 111, 1000, etc.**

iii. Mark all bit positions that are powers of two as parity bits: **1, 2, 4, 8, 16, 32, 64, 128, etc.**

iv. Mark all other bit positions other than powers of two as data to be encoded: **3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 33 etc.**

v. For each parity bit calculates the parity for some of the bits in the code word. The position of the parity bit determines the sequence of bits that it alternately checks and skips:

a. Parity position 1: start with 1, check 1 bit, skip 1 bit, etc (**1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, etc**).

b. Parity position 2: start with 2, check 2 bits, skip 2 bits, etc (2**, 3, 6, 7, 10, 11, 14, 15, 18, 19, 22, 23, 26, 27, etc**).

c. Parity position 4: start with 4, check 4 bits, skip 4 bits, etc (**4, 5, 6, 7, 12,13,14,15, 20, 21, 22, 23, etc**).

d. Parity position 8: start with 8, check 8 bits, skip 8 bits, etc (**8-15, 24-31, 40-47, 56-63, etc**).

e. Parity position 16: start with 16, check 16 bits, skip 16 bits, etc (**16-31, 48-63, 80-95, 112-127, 144-159, etc**).

f. Parity position 32: start with 32, check 32 bits, skip 32 bits, etc (**32-63, 96-127, 160-191, 224-255, 288-319, etc**), etc.

vi. Set a parity bit to '**1**' if the total number of 1's in the positions it checks is odd. Otherwise, set a parity bit to '**0**' if the total number of 1's in the position it checks is even. Thus, the parity code is obtained. An encoded data is formed when the parity codes are placed in their respective positions.

These encoded data therefore are either transmitted or stored. The comparison between the parity codes of the computed result or transmitted data with that of the received data will indicate whether an error has occurred or not. If the parity codes are found to be the same, that is result all 0's then, no error has occurred but if otherwise, that is any difference, then an error has occurred and practical steps are therefore taken to indentify the erroneous bit and flip it (that is interchange the bit from '0' to '1' or from '1' to '0' as the case may be) to correct it. By the results from this paper therefore, the knowledge of error control by the use of Algorithmic method of Hamming code techniques for detection and correction of errors shall be greatly improved upon.

II. Definition of Terms used

Throughout this paper, the following terms as used are defined as follows:

i. Algorithmic method: The algorithmic method is the procedure involving the following steps: (a) number the bits starting from 1 (b) write their respective binary equivalences (c) mark all bit positions that are powers of two (2^n, $n = 0, 1, \ldots$) as parity bits (d) mark all other bit positions other than the powers of two as data (e) for each parity bit position, calculate the parity code, the position of the parity bit determine the sequence of bits that it alternately checks and skips and (f) set parity bit to '1' if the total numbers of 1's in the positions it checks is odd, otherwise set it to '0' if even. Thus, this gives the parity code.

ii. Binary coded data: According to [5], in most digital and communication systems, information is transmitted in form of binary coded data that is, messages are in the form of the symbols '0' and '1'. Communication is the process of transmitting information [5]. This transmission can either be made between two distinct places, say a telephone call or between two points in time. As an example, the writing of this paper so that it could be read latter is a form of communication.

iii. Binary coded decimal: Binary coded decimals are those codes in which error detection and correction is done in binary information (bits). Hence, after the error is detected or located, correction means only flipping the bit found erroneous [5]. E.G: The decimal number in the complete five-bit reverse (that is, parity position) binary code is shown in the Table below: (DN= Decimal number)

DN	$2^0 = 1$	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$
0	0	0	0	0	0
1	1	0	0	0	0
2	0	1	0	0	0
3	1	1	0	0	0
4	0	0	1	0	0
5	1	0	1	0	0
6	0	1	1	0	0
7	1	1	1	0	0
8	0	0	0	1	0
9	1	0	0	1	0

10	0	1	0	1	0
11	1	1	0	1	0
12	0	0	1	1	0
13	1	0	1	1	0
14	0	1	1	1	0
15	1	1	1	1	0
16	0	0	0	0	1
17	1	0	0	0	1
18	0	1	0	0	1
19	1	1	0	0	1
20	0	0	1	0	1

iv. Parity operator: This is the process of reversing parity bit set. E.G, the even parity (XOR) operator is the reversing of parity bit sets of the comparison between the parity code of the encoded (sent) data and that of the received (re-calculated) data when detecting an even data error location. While the odd parity (XNOR) operator is the reversing of both the parity bit sets of the received data positions having '1' and the comparison of the encoded (sent) parity code with that of the received (re-calculated) parity code when detecting an odd data error location [4].

III. Data Presentation and Analysis

The results obtained from the computation of the integer sequence *A119626* generated by the formula $3 + 3^n$, with *n* taking values from 1-5, along with their respective binary coded data (binary equivalences), parity codes and encoded data are shown in the table below:

Table 3.0 (Results obtained from the computation of the integer sequence *A119626*)

n	$3 + 3^n$	Binary coded data	Parity code	Encoded data
1	6	110	**011**	**0**1**1**10
2	12	1100	**011**	**0**1**1**100
3	30	11110	**1110**	**1**11111**0**0
4	84	1010100	**0011**	**00**110**1**0**1**100
5	246	11110110	**0110**	**0**1111110**0**110
…	…	…	**…**	…

Source: Researcher's calculation

From TABLE 3.0 above, the computational results of the integer sequence *A119626* using the formula $3 + 3^n$, with $n = 1$-5, were obtained as follows:

When $n = 1$; $3 + 3^1 = 6$, when $n = 2$; $3 + 3^2 = 12$, when $n = 3$; $3 + 3^3 = 30$, when $n = 4$; $3 + 3^4 = 84$, when $n = 5$; $3 + 3^5 = 246$.

Their respective parity codes and encoded data were obtained by the analysis of the Algorithmic method as follow:

6: [110]: x x 1 x 1 0, position 1: check bits 1, 3, 5: we have, ! x 1 x 1 0: even parity so set position 1 to a '**0**'. Position 2: check bits 2, 3, 6: we have, **0** ! 1 x 1 0: odd parity so set position 2 to a '**1**'. Position 4: check bits 4, 5. 6: we have, **0 1** 1 ! 1 0: odd parity so set position 4 to a '**1**'. The parity code is **0 1 1**, thus the encoded data: **0 1** 1 **1** 1 0.

12: [1100]: x x 1 x 1 0 0, position 1: check bits 1, 3, 5, 7: we have, ! x 1 x 1 0 0: even parity so set position 1 to a '**0**'. Position 2: check 2, 3, 6, 7: we have, **0** ! 1 x 1 0 0: odd parity so set position 2 to a '**1**'. Position 4: check 4, 5, 6, 7: we have, **0 1** 1 ! 1 0 0: odd parity so set position 4 to a '**1**'. The parity code is **0 1 1**, thus the encoded data: **0 1** 1 **1** 1 0 0.

30: [11110]: x x 1 x 1 1 1 x 0, position 1 check bits 1, 3, 5, 7, 9: we have, ! x 1 x 1 1 1 x 0: odd parity so set position 1 to a '**1**'. Position 2: check bits 2, 3, 6, 7: we have, **1** ! 1 x 1 1 1 x 0: odd parity so set position 2 to a '**1**'. Position 4: check bits 4, 5, 6, 7: we have, **1 1** 1 ! 1 1 1 x 0: odd parity so set position 4 to a '**1**'. Position 8: check bits 8-15: we have, **1 1** 1 **1** 1 1 1 ! 0: even parity so set position 8 to a '**0'.** The parity code is **1 1 1 0**, and the encoded data is: **1 1** 1 **1** 1 1 1 **0** 0.

84: [1010100]: x x 1 x 0 1 0 x 1 0 0. Position 1: check bits 1, 3, 5, 7, 9, 11: we have, ! x 1 x 0 1 0 x 1 0 0: even parity so set position 1 to a '**0**'. Position 2: check bits 2, 3, 6, 7, 10, 11: we have, **0** ! 1 x 0 1 0 x 1 0 0: even parity so set position 2 to a '**0**'. Position 4: check bits 4, 5, 6, 7: we have, **0 0** 1 ! 0 1 0 x 1 0 0: odd parity so set position 4 to a '**1**'. Position 8: check bits 8-15: we have, **0 0** 1 **1** 0 1 0 ! 1 0 0: odd parity so set position 8 to a '**1**'. The parity code is **0 0 1 1**. Thus, the encoded data is: **0 0** 1 **1** 0 1 0 **1** 1 0 0.

246: [11110110]: x x 1 x 1 1 1 x 0 1 1 0. Position 1: check bits 1, 3, 5, 7, 9, 11, 13: we have, ! x 1 x 1 1 1 x 0 1 1 0: even parity so set position 1 to a '**0**'. Position 2: check bits 2, 3, 6, 7, 10, 11: we have, **0** ! 1 x 1 1 1 x 0 1 1 0: odd parity so set position 2 to a '**1**'. Position 4: check bits 4, 5, 6, 7, 10, 11: we have, **0 1** 1 ! 1 1 1 x 0 1 1 0:

odd parity so set position 4 to a '**1**'. Position 8: check bits 8-15: we have, **0 1** 1 **1** 1 1 1 ! 0 1 1 0: even parity so set position 8 to a '**0**'. The parity code is **0 1 1 0**, and the encoded data is: **0 1** 1 **1** 1 1 1 **0** 0 1 1 0.

3.1 The Analysis of the Algorithmic method of Hamming code techniques for detection and correction of computational errors in a binary coded data of an integer sequence *A119626*

The results obtained from the computation of integer sequence *A119626* generated by the formula $3 + 3^n$, with *n* taking values from 1-5, their respective encoded (computed) data, erroneous results (data) obtained instead with their binary coded data (binary equivalences) and the positions of the erroneous data (DEP) are shown in the table below:

Table 3.1(Results of error detection and correction obtained from the computation of the integer sequence *A119626*)

n	$3 + 3^n$	Binary Equivalence	Parity Code	Encoded Data	$3 + 3^n$ error	Binary Equivalence	Parity Code	Encoded Data	DEP
1	6	110	**011**	**011**10	7	111	**000**	**00**1**0**11	6
2	12	1100	**011**	**011**1100	13	1101	**100**	**1**0**1**0101	7
3	30	11110	**1110**	**111**1111**0**0	22	10110	**0100**	**011**0011**0**0	5
4	84	1010100	**0011**	**00**110101100	85	1010101	**1110**	**111**0100**0**101	11
5	246	11110110	**0110**	**011**1111**0**0110	118	01110110	**1010**	**100**1111**0**0110	3
…	…	…	…	…	…	…	…	…	…

Source: Researcher's calculation

From the TABLE 3.1 above, the parity code, encoded data and the data error positions of the respective erroneous results (data) obtained in the process of computation are shown using the analysis of algorithmic method as follows:

7: [111]: x x 1 x 1 1, position 1: check bits 1, 3, 5: we have, ! x 1 x 1 1: even parity so set position 1 to a '**0**'. Position 2: check bits 2, 3, 6: we have, **0** ! 1 x 1 1: even parity so set position 2 to a '**0**'. Position 4: check bits 4, 5, 6: we have, **0 0** 1 ! 1 1: even parity so set position 4 to a '**0**'. The parity code is **0 0 0**, thus the encoded data: **0 0** 1 **0** 1 1. Original (sent) parity code **0 1 1** differs from the received parity code **0 0 0** in positions 2 and 4 respectively, their addition will give the data error position. Thus, 2 + 4 = 6. Therefore, data position 6 is erroneous. To correct it, flip it from '1' to '0'.

13: [1101]: x x 1 x 1 0 1, position 1: check bits 1, 3, 5, 7: we have, ! x 1 x 1 0 1: odd parity so set position 1 to a '**1**'. Position 2: check 2, 3, 6, 7: we have, **1** ! 1 x 1 0 1: even parity so set position 2 to a '**0**'. Position 4: check 4, 5, 6, 7: we have, **1 0** 1 ! 1 0 1: even parity so set position 4 to a '**0**'. The parity code is **1 0 0**, thus the encoded data: **1 0** 1 **0** 1 0 1. Original (sent) parity code **0 1 1** differs from the received parity code **1 0 0** in positions 1, 2 and 4 respectively, their addition will give the data error position. Thus, 1+ 2 + 4 = 7. Therefore, data position 7 is erroneous. To correct it, flip it from '1' to '0'.

22: [10110]: x x 1 x 0 1 1 x 0, position 1 check bits 1, 3, 5, 7, 9: we have, ! x 1 x 0 1 1 x 0: even parity so set position 1 to a '**0**'. Position 2: check bits 2, 3, 6, 7: we have, **0** ! 1 x 0 1 1 x 0: odd parity so set position 2 to a '**1**'. Position 4: check bits 4, 5, 6, 7: we have, **0 1** 1 ! 0 1 1 x 0: even parity so set position 4 to a '**0**'. Position 8: check bits 8-15: we have, **0 1** 1 **0** 0 1 1 ! 0: even parity so set position 8 to a '**0'.** The parity code is **0 1 0 0**, and the encoded data is: **0 1** 1 **0** 0 1 1 **0** 0. The original (sent) parity code **1 1 1 0** differs from the received parity code **0 1 0 0** in positions 1 and 4 respectively, their addition will give the data error position. Thus, 1 + 4 = 5. Therefore, data position 5 is erroneous. To correct it, flip it from '0' to '1'.

85: [1010101]: x x 1 x 0 1 0 x 1 0 1. Position 1: check bits 1, 3, 5, 7, 9, 11: we have, ! x 1 x 0 1 0 x 1 0 1: odd parity so set position 1 to a '**1**'. Position 2: check bits 2, 3, 6, 7, 10, 11: we have, **1** ! 1 x 0 1 0 x 1 0 1: odd parity so set position 2 to a '**1**'. Position 4: check bits 4, 5, 6, 7: we have, **1 1** 1 ! 0 1 0 x 1 0 1: odd parity so set position 4 to a '**1**'. Position 8: check bits 8-15: we have, **1 1** 1 **1** 0 1 0 ! 1 0 1: even parity so set position 8 to a '**0**'. The parity code is **1 1 1 0**. Thus, the encoded data is: **1 1** 1 **1** 0 1 0 **0** 1 0 1. The original (sent) parity code **0 0 1 1** differs from the received parity code **1 1 1 0** in positions 1, 2 and 8 respectively, their addition will give the data error position. Thus, 1 + 2 + 8 = 11. Therefore, data position 11 is erroneous. To correct it, flip it from '1' to '0'.

118: [01110110]: x x 0 x 1 1 1 x 0 1 1 0. Position 1: check bits 1, 3, 5, 7, 9, 11, 13: we have, ! x 0 x 1 1 1 x 0 1 1 0: odd parity so set position 1 to a '**1**'. Position 2: check bits 2, 3, 6, 7, 10, 11: we have, **1** ! 0 x 1 1 1 x 0 1 1 0: even parity so set position 2 to a '**0**'. Position 4: check bits 4, 5, 6, 7, 10, 11: we have, **1 0** 0 ! 1 1 1 x 0 1 1 0: odd parity so set position 4 to a '**1**'. Position 8: check bits 8-15: we have, **1 0** 0 **1** 1 1 1 ! 0 1 1 0: even parity so set position 8 to a '**0**'. The parity code is **1 0 1 0**, and the encoded data is: **1 0** 0 **1** 1 1 1 **0** 0 1 1 0. The original (sent) parity code **0 1 1 0** differs from the received parity code **1 0 1 0** in positions 1and 2 respectively, their

addition will give the data error location. Thus, 1 + 2 = 3. Therefore, data position 3 is erroneous. To correct it, flip it from '0' to '1'.

From the analysis above, the total number of steps (iterations) and their respective total execution time (*T*) and rate (*R*), for any given time *t*, involved in the general analysis of encoding, detection and correction of computational error(s) using the Algorithmic method is summarized in the table below. Where the time of execution (*T*) and rate (*R*) are given as follows: $T = i \times t$ (that is, number of iteration *i* multiplied by the time *t* taken) and $R = i / t$ (that is the number of iteration *i* divided by the time *t* taken).

Table 3.2 (Results of the total number of steps, execution time and rate)

n	$3+3^n$	AME	AMEDC	T	AMET	AMER
1	6	3	5	8	8 x *t*	8 \ *t*
2	12	3	5	8	8 x *t*	8 \ *t*
3	30	4	6	10	10 x *t*	10 \ *t*
4	84	4	6	10	10 x *t*	10 \ *t*
5	246	4	6	10	10 x *t*	10 \ *t*
	Total	18	28	46	46 x *t*	46 \ *t*

Source: Researcher's calculation

(Where: AME = algorithmic method of encoding, AMEDC = algorithmic method of error detection and correction, T = total, AMET = algorithmic method of execution time, AMER = algorithmic method of execution rate, *t* = time given)

IV. Conclusion

From the analysis of the Algorithmic method of Hamming code techniques for encoding, detection and correction of computational error(s) in a binary coded data of an integer sequence *A119626* discussed in this paper, the total number of steps (iterations) for values of n = 1-5, was obtained to be forty six (46). The execution time (*T*) and rate (*R*) for any given time *t*, of the analysis would be obtained by multiplying and dividing *T* and *R* by *t* respectively. The results from this paper therefore will help to ascertain the total number of steps (iterations) involved in any given computation, the execution time (*T*) and rate (*R*) for any given time *t* and the efficiency. Although the computation was restricted to values of n = 1-5 of the integer sequence *A119626*, however, the results obtained can be applied generally on computation involving binary coded data.

References

[1] I. Koren, *Computer arithmetic algorithms* (Natrick MA): A. K. Peters, 2002.
[2] R. W. Hamming, Bell system Technology, *Journal of Error Detecting and Correcting Codes vol. 29, April, 1950, pp. 147-160.*
[3] A. A. Ibrahim, Mathematics Association of Nigeria, *the journal On the Combinatorics of A Five-element sample Abacus of vol. 32, 2005, No. 2B: 410-415.*
[4] Moon & K. Todd, *Error correction coding* (http://www.neng.usu.edu/ece/faculty/tmoon/eccbook/book.html). (New Jersey: John Wiley, 2005 and sons ISBN 978-0-471-64800-0).
[5] B. Sklar, Digital communication: *fundamentals and applications* (Second Edition: Prentice-Hall, 2001).