Proposed Data Integrity Algorithm

Safaa O. Mahdi
Department of Computer Science, University of Babylon

Abstract

In this paper, we introduce a new algorithm for detecting changes that happen in files. The proposed algorithm uses an array of sixteen locations that is contains the unique information about file as proved practically. It can be used for error detection but not correction. It can specify the type of alteration such as modify, addition or losing.

Terms: Error detection, Integrity.

الخلاصة

في هذا البحث، قمنا باقتراح خوارزمية جديدة لاكتشاف التغييرات التي تحصل في الملفات. الخوارزمية المقترحة تستخدم مصفوفة ذات ستة عشر عنصر للاحتفاظ بمعلومات وحيدة من الملف كما ثبت ذلك عملياً. وهي تستخدم لاكتشاف الأخطاء وليس تصحيحها وتستطيع أن تميز نوع التلاعب أو الخطأ الذي حصل في الملف كالإضافة أو الحذف أو التحديث، ولها القدرة على اكتشاف الخطأ ولو كان بتاً واحداً فقط.

Introduction

Data can be corrupted during transmission. For reliable communication, errors must be detected and corrected.

The concept of including extra information in the transmission solely for the purposes of comparison is a good one. But instead of repeating the entire data stream, a shorter group of bits may be appended to the end of each unit. This technique is called ***redundancy*** because the extra bits are redundant to the information; they are discarded as soon as the accuracy of the transmission has been determined. Error detection uses the concept of redundancy, which means adding extra bits for detecting errors at the destination (Forouzan, 1998).

Once the data stream has been generated, it passes through a device that analyzes it and adds on an appropriately coded redundancy check. The data unit, now enlarged by several bits, travels over the link to the receiver. The receiver puts the entire stream through a checking function. If the received bit stream passes the checking criteria, the data portion of the data unit is accepted and the redundant bits are discarded (Forouzan, 1998).

While error correction can be handled in two ways. In one, when an error is discovered, the receiver can have the sender retransmit the entire data unit. In the other, a receiver can use an error correction code, which automatically corrects certain errors (Forouzan, 1998).

Proposed Algorithm

In this section, we state our algorithm that is used to detect the errors in files or check the integrity of the files. The algorithm computes and stores only array (16 values) that is stating the information about the file. But Why The Array Length Is 16? Because We Take the Length Of Segment From File Four Bits And The Combination Of Four Bits Is 16 Value.

We assume that the target file is circular by linking the end of file with the beginning of it, as shown in figure (1). This is because the stop condition of the algorithm is to reach the end of file and pass it to the first three bits from the beginning of file. When The Window Shifts And Reach The End Of File, The Window Length Become Three ,So We Take First Bit From Beginning Of File To Make Window Length Four And Repeating This Process Three Times (Three Bits).

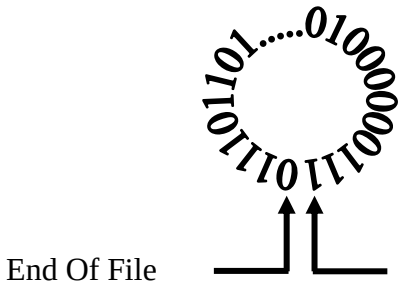

Figure (1): Circular File

The Flowchart of the algorithm is shown in figure (2). The algorithm can be used for error detection but not correction.

All our information about file is stored in the C array, so after declaring it, we open the target file and read a segment (window) of length four bits and convert them to integer digit (Z), after that we increment its index in C array (C[Z]). We can imagine Z as window of size four bits and move it toward the end of file, this can be done by shifting the window to right one bit and make new value of Z. This process is repeated until we reach the end of file and loop back to the beginning of file and take three bits from beginning of file only. The suggested algorithm can detect the type of alteration such as "*modify*", "*add*" or "*delete*" by compute the length of file from the following equation:

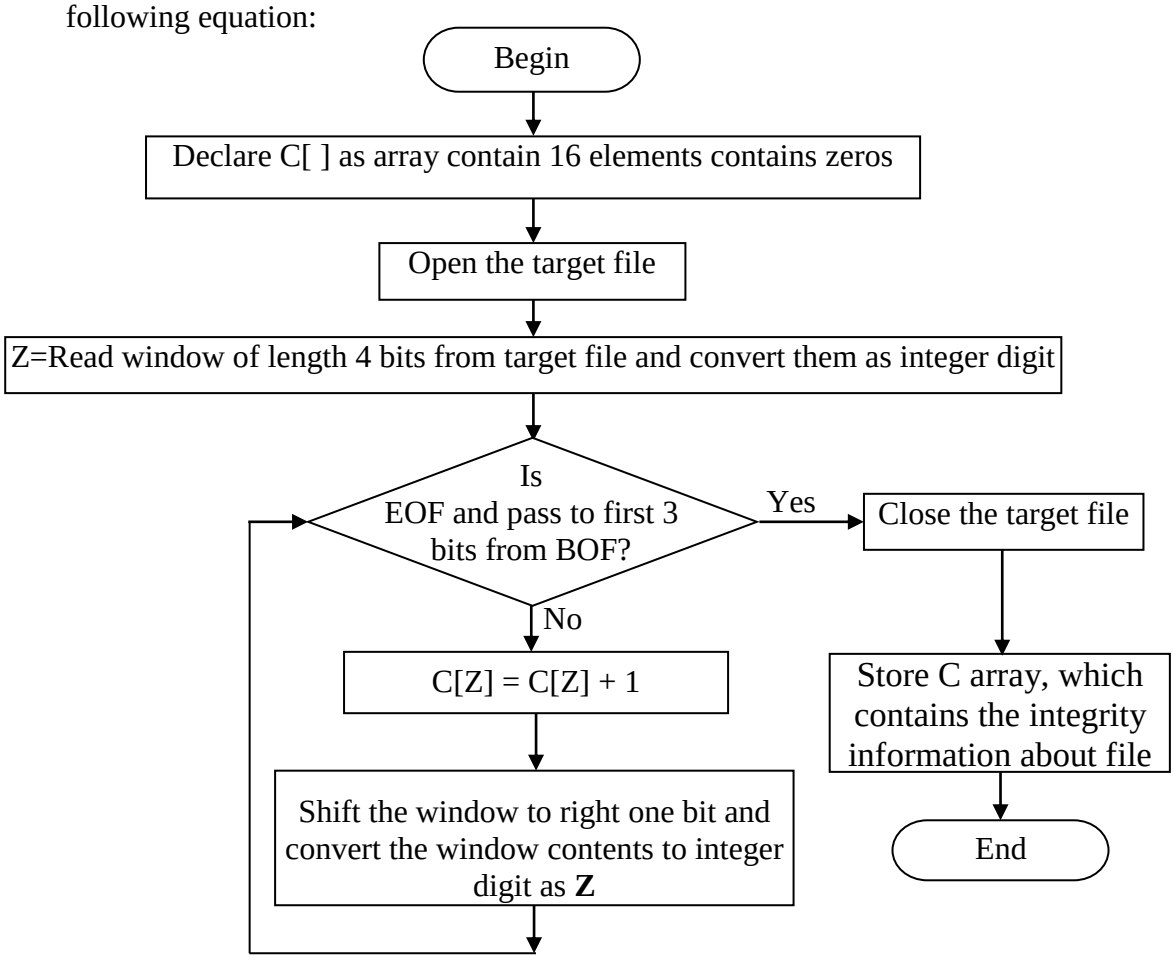

Figure (1) The proposed algorithm for Alteration Detection

$$Length\_Of\_File = \frac{(\sum_{i=0}^{15} C[i])}{8} \quad \ldots(1)$$

The resulting value from equation (length of file in byte) is compared with the computed length of file stored previously. If they are equal, then there is no adding to the file or deletion from it and we must test the values for checking alteration, otherwise there is an adding or deleting operation from the file and there is no need to test the values, this process is described in figure (3).

The algorithm uses an array of sixteen elements only (C array) and it can detect if at least one bit is changed. The time complexity of the algorithm is shown in figure (4), and it has linear complexity. Where the X-axis is the Size in Kilobyte and the Y-axis is the Time in second.

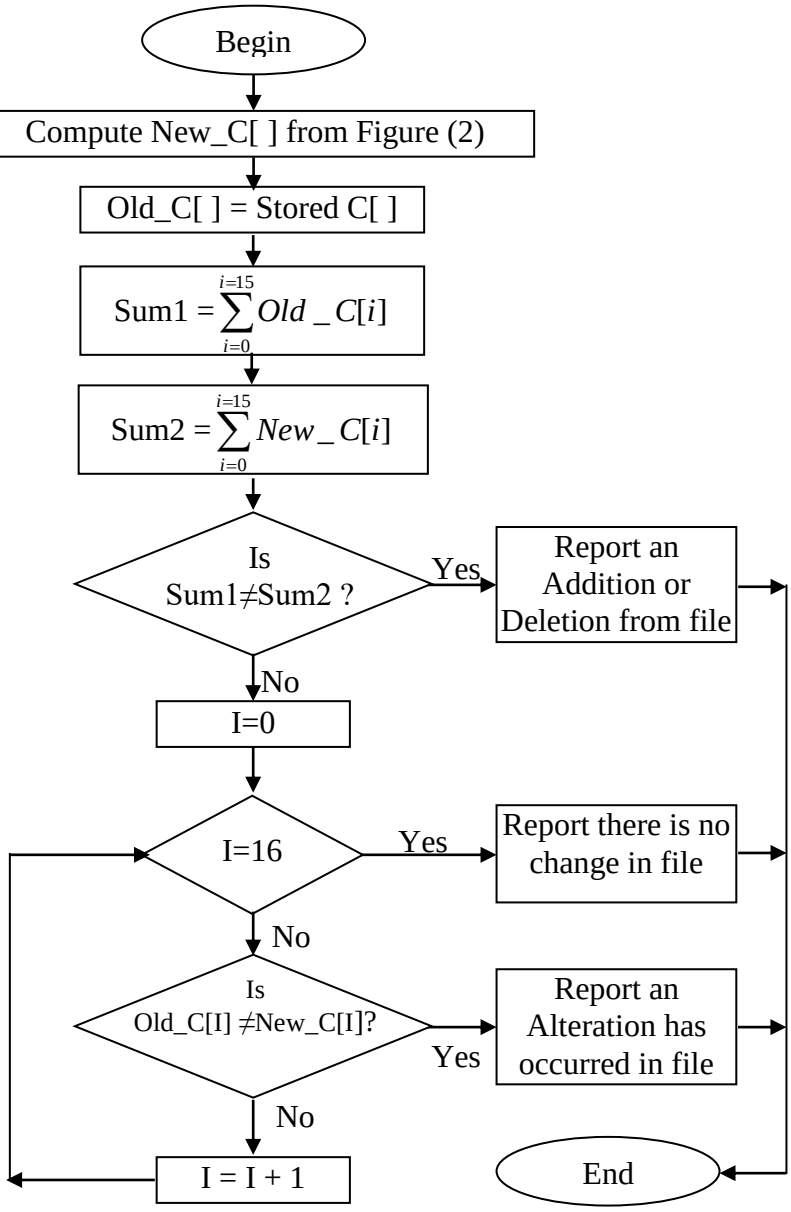

Figure (3) The process of verifying the change in Contents of files

Example:

If we take the content of a file in binary form (has length 3 bytes) like below:
(1 0 1 1 0 1 1 1 0 0 0 0 1 1 1 1 1 0 1 0 1 0 0 0)
by applying the suggested algorithm, we obtain the following results:

Index	0	1	2	3	4	5	6	7	8	9	1	1	1	1	1	1
Value	1	2	1	1	1	2	1	2	2	0	2	2	1	2	2	2

This result is store in the database to use it in another time for checking purposes.

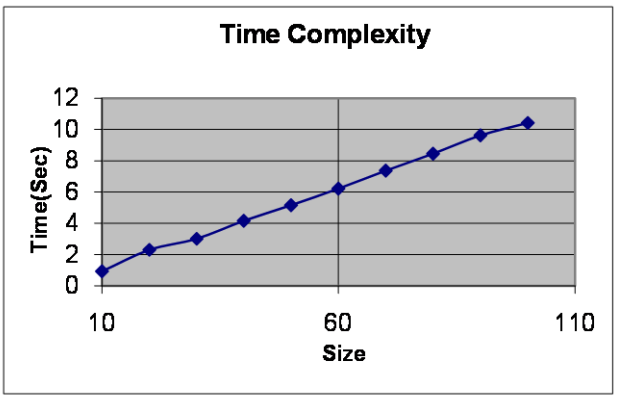

Figure (4): Time Complexity of The Proposed Algorithm

If one bit has been changed by any way as shown below:
(1 0 1 1 0 1 1 1 0 0 0 0 1 1 **0** 1 1 0 1 0 1 0 0 0)
so, when applying the suggested algorithm, we obtain the following results:

Index	0	1	2	3	4	5	6	7	8	9	1	1	1	1	14	15
Value	1	2	1	1	1	2	3	1	2	0	2	3	1	3	1	0

As we mention, when we apply equation (1) we obtain the same length of file,

$$Length\_Of\_File = \frac{24}{8} = 3\,(Bytes)$$

so there is no addition operation to the file or deletion from it. Then we compare the value of C array computed with the stored one, thus we note that there is a difference between them (6 values changed), so the file was altered.

If two bits has been changed, as shown below:
(1 0 1 **0** 0 1 1 1 0 **1** 0 0 1 1 1 1 1 0 1 0 1 0 0 0)
By applying the suggested algorithm, we obtain the following results:

Index	0	1	2	3	4	5	6	7	8	9	1	1	1	1	1	1
Value	0	1	1	2	3	2	0	2	1	2	4	0	0	2	2	2

We note that 10 values is changed, so the file was altered.

Conclusion

The efficiency of the algorithm is accepted because it is applied to about 40 files and get good results in detect errors in files such as adding, deleting and modification. We implement a program in C++ language to check the algorithm, so we prove that there is not found two files having the same array values because we test it in different file lengths.

References

Forouzan, B. (1998). "***Introduction To Data Communications And Networking***", McGraw-Hill Companies.