An Empirical Analysis Between Run-length and Dictionary Compression Technique

***Abstract*—Data compression is a technique of representing data in a compact form. It is a process that mitigates the data size, redundancy, and truncate some extra information. A diminutive of an actual data is always propitious as it takes less cost in every way like space, time, computation, bandwidth, and many more. Recent researches lack in providing correct use of compression in different situation. For say, before encrypting a Cryptographic Hash value, Run-length may work better than a Dictionary algorithm to minimize the size and length of the data. With this in mind this article explores two of the lossless compression technique: Run-length and Dictionary compression algorithm. We have implemented both of the code and for comparison the values have been collected for specific reasoning. Lastly, the favorable states for which these two compression algorithms can be used is stated according to the acquired result.**

***Index Terms*—Run-length, Dictionary, Compression, Lossless, Lossy, Compression ratio, Compression rate**

I. Introduction

Not very long ago, transferring deluge of data was cumbersome and time consuming. But, with the advent of enabling compression methodologies in computerized applications, it brings a revolution in digital world [1]. Each day humans generate and capture more than 2.5 quintillion bytes of data [2]. Additionally, more than 90% of the data generated in recorded human history was created in the last few years and it was estimated that this amount will exceed 40 Zettabytes or 40 trillion gigabytes by 2020 [3]. So, it will not be very long that we will need much more sophisticated compression techniques at specific purposes.

The primary aspect of compression is reducing the size of data [4]. This is really convenient while transferring a large data. After using the technique, there should be an appreciable difference between the actual and compressed data [5]. In data transmission applications, achieving speed is the primary purpose for the compression, on the other hand, for storage devices, the degree of compressing is the main concern [6]. Usually lossless compression is used for preserving computer executable file, medical images, and for many other important reasons [7]. To closely understand the usefulness and amenities of Run-length and Dictionary Algorithm a myriad of researches have been going on [8].

In this article, We compare and analysis between Run-length and Dictionary compression algorithm of the lossless compression technique. We have started the research work by depicting some of the related works in Section II. Next, we present some exegesis review of Run-length and Dictionary Algorithm in Section III. Then, Section IV outlines the implementation software and tools. After that, we present the evaluation and the result in Section V. Next, the discussion is presented in Section VI Lastly, we conclude the work in Section VII.

II. Related works

To increase the efficiency of different compression technique, a lot of researches are going on. Among some significant researches, [9] is a notable work where Brisaboa et al. have proposed a way to shrink the compressed dictionary data for faster services. Brisaboa et al. [10] also have done a great job recently by using longest common prefix. They have designed a new data structure for achieving faster storage and query time for WEB and URL dictionaries. Kempa et al. in their research [11] have showed how their universal data structure works for all the types of dictionary compression algorithms. It also supports random access in any dictionary scheme.

There are an indefinite number of researches on Run-length algorithm. Uthayakumar et al. [12] have presented a comparison between some of the lossless compression technique for satellite images. According to their research the Lempel–Ziv–Markov chain algorithm have efficient result than others. Moreover, a notable comparison work have been done by Srinivas et al. [13]. The findings of their research work is appreciable. With all these compression techniques, which should be used for which purposes is an important decision [14]. For this reason, our research work is concentrated on the comparison between Run-length and Dictionary algorithm in respect of compression and decompression. Next, we present some background information of Run-length and Dictionary algorithm.

III. Preliminaries

Let us have some insights regarding the algorithms.

A. Run-length Algorithm

Run-length algorithm is one of the most simple compression algorithm. It is solely developed for repeated character. For example: a string ”aaabbbbdd” of 9 bytes can be compressed as ”a3b4d2” which is of only 6 bytes. The worst case scenario can happen that the compressed data may be twice the size of the actual data [15].

B. Dictionary Algorithm

Dictionary compression technique rely upon the observation that there are correlations between parts of data. The basic idea is to replace those repetitions by references to a ”dictionary” containing the original [16].

The kind of Dictionary algorithm, we have used is the Lempel Ziv Algorithm. It is not a single algorithm, but a whole family of algorithms, stemming from the two algorithms proposed by Jacob Ziv and Abraham Lempel in their landmark papers in 1977 and 1978 [17]. An important step of this algorithm is that, all the ASCII code will be stored before, the new made up characters will be stored in with the earlier stored characters, if they are not extant by the canon. This process will be going on recursively until the whole String will be parsed. The selected characters' ASCII code will be stored separately. So, while decompressing the compressed data, the system must need that definite updated dictionary and stored ASCII codes.

IV. Implementation

For showcasing and evaluating the output, we implements the algorithms in Java. For Java programming, we have used NetBeans IDE for developing the implementation [18]. The experiment has been carried out in a PC with a Windows 10 Pro 64-bit 20H2 OS and hardware configurations of Intel(R) Core(TM) i5- 7200U @2.50GHz CPU, 8 GB DDR4 RAM, 100 GB SSD, 2 TB HDD and Intel(R) HD Graphics 620 GPU.

A. Data Structure and necessary built-in tools

1) Run-length: To implement Run-length algorithm, with the help of String class all the necessary calculations have been completed. For calculating execution time, we have used "System.nanoTime()", as the conventional "System.currentTimeMillis()" which provides the current system time in milliseconds, can not perceive the time accurately. Because sometimes for small Strings it gives 0 second as output. By taking current time before the start of execution and again after the end of execution, we have gotten the execution time by taking a difference between the two reading of specific segment of code. For calculating size, we have used a String class method, "String.getBytes("UTF-16BE")", which returns a byte array consisting the byte values of each character of the String. After that, by multiplying 8, the bit value for specific string can be accumulated.

2) Dictionary: To implement Dictionary algorithm, we have used two Map objects. One for storing all the ASCII code as a dictionary and another for all the new dictionary entries. We have used Map key as String and value as ASCII value, the query processing gets faster in this way. we have used an Integer type List object to get the compressed data which is composed of just the dictionary ASCII values. For calculating time, same scheme has been used as we had used for Run-length. As Dictionary returns an integer representing list, for this reason we have multiplied the list size with 8 to get the size in bits.

V. Evaluation and Result

To compare and analysis the two algorithm, we have used 4 types of data. Two of them are very small: one having recurring string called "Small2" and another have normal arbitrary string called "Small1". For rest of the two, "Incom" is consist of a set of words which is String type data and "Decimal" has a set of decimal values. The Table I depicts the necessary information regarding the specified data.

TABLE I: Data Lists

Data file name	Memory space in the computer	String length
Small1	384	24
Small2	272	17
Incom	441072	27567
Decimal	80384	5024

The "Small1" is a String without any space between characters. This String does not prejudice any of the two algorithms. "Small2" is a string of recurring characters without any space which is highly favorable to Run-length algorithm. We can inspect how Dictionary algorithm works on this. The "Incom" is a set of words, string type having 27567 length. Compressing this string will be a challenge for both of the algorithms. Then the Decimal is a set of decimal numbers of 5 MB size. The data is available at [18]. The factors we are going to consider for this research work are compression speed, decompression speed, saving percentage, and memory space. And all the inputs for the algorithms have been used as String object. The compression ratio is calculated using (1).

$$CompressionRatio = \frac{CompressedSize}{ActualSize} \tag{1}$$

But the saving percentage will give better influence than compression ratio. The saving percentage can be calculated using (2).

$$SavingPercentage = \frac{ActualSize - CompressedSize}{ActualSize} \tag{2}$$

A. Experiment

1) Small1: After executing the string "Small1" for both of the algorithm, the result depicts that Run-length algorithm has done a bad job and Dictionary algorithm has done a moderate job which can be seen from Fig. 1. The compressed size for Run-length algorithm is about double of the actual data size. On the contrary dictionary algorithm have taken more time but saved memory space. The results have been illustrated in Table II and Table III.

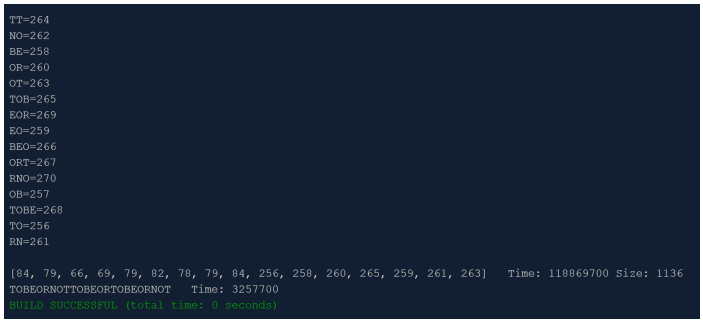

Fig. 1: The output of Dictionary Algorithm for Small1 String

2) Small2: The "Small2" String is actually favorable for Run-length Algorithm as same characters are coming repetitively. But, still Dictionary algorithm did a much better job in terms of size which can be found in Table III. One noticeable thing is Run-length algorithm's saving percentage was 17.64, on the contrary Dictionary algorithm's was 64.70. But, Run-length has compressed the data faster than Dictionary algorithm which is illustrated in Fig. 2.

```
w4a3d1e1x6y1w1    Time: 604001
Size: 224 bits
wwwwaaadexxxxxxyw    Time: 330300
BUILD SUCCESSFUL (total time: 0 seconds)
```

Fig. 2: The output of Run-length Algorithm for Small2 String

3) Incom: "Incom" String has 27567 character, in this term Run-length has shown its worst outcome which is depicted in Fig. 3. It has used about 96.61% extra memory to compress according to Table II.

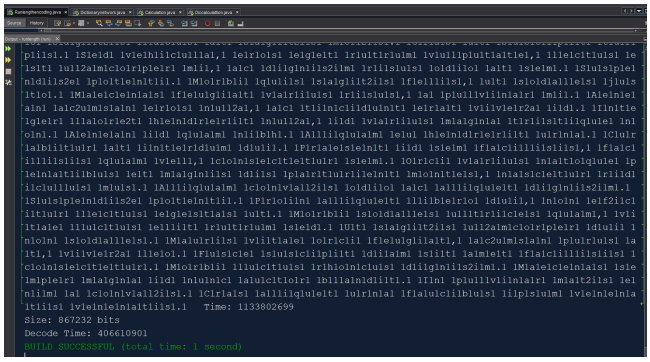

Fig. 3: The partial output of Run-length Algorithm for Incom string

The relieving point is that Dictionary algorithm has saved about 87.92% of actual data according to Table III. It compresses the actual 441072 bits of data into a 53288 bits compressed data as presented in Fig. 4. Even for compression and decompression time factor, it is faster than Run-length algorithm, illustrated in Table III and Table II respectively.

4) Decimal: "Decimal" String has 5024 length of decimal values. For these characters, Run-length fails provide propitious result as can be seen from Fig. 5, whereas Dictionary algorithm again compressed the strings with a saving percentage 79.98 as stated in Table III.

The Dictionary algorithm compressed the data in 9614501 nanosecond on the other hand it took 47525900 nanosecond to compress the data for Run-length algorithm, can be perceived from Fig. 6.

B. Result

Summarizing all this experimented data, Table II and Table III can be concocted for Run-length and Dictionary respectively.

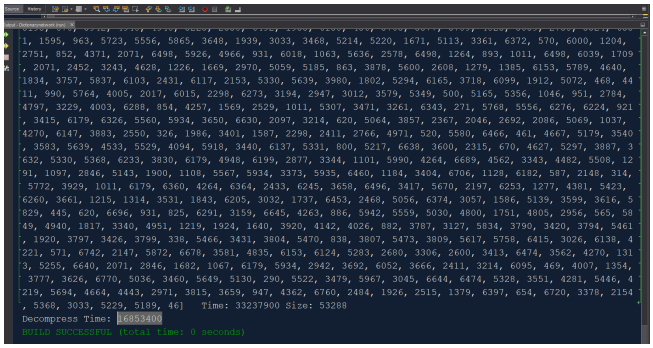

Fig. 4: The partial output of Dictionary Algorithm for Incom.txt file's string

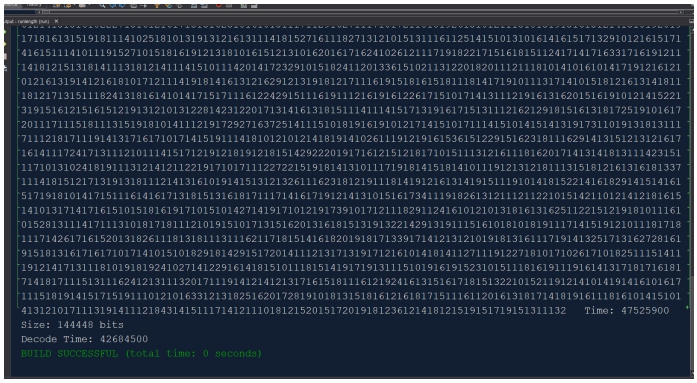

Fig. 5: The partial output of Run-length Algorithm for Decimal string

TABLE II: Run-length algorithm's performance for the chosen data

Data file name	Compressed size	saving percentage(%)	Compression speed	Decompression speed
Small1	736	-91.66	260760	470359
Small2	224	17.64	72979	172340
Incom	867232	-96.61	1133802699	406610901
Decimal	144448	-79.70	47525900	42684500

TABLE III: Dictionary algorithm's performance for the chosen data

Data file name	Compressed size	saving percentage(%)	Compression speed	Decompression speed
Small1	128	66.66	1308000	944520
Small2	96	64.70	913360	965860
Incom	53288	87.92	33237900	16853400
Decimal	16096	79.98	9614501	7160100

VI. Discussion

It can be stated from the results illustrated in Section V, Run-length algorithm is highly efficient only when there is a repetitive pattern. Consequently, with normal book or files

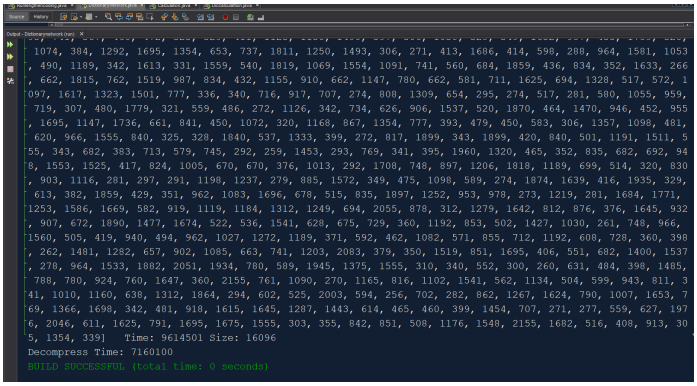

Fig. 6: The partial output of Dictionary Algorithm for Decimal string

it will be very costly. On the contrary, Dictionary algorithm works better when file size is larger. At first, there will be many types of pattern to store in dictionary, but later it will usually truncate the actual string efficiently.

In terms for the factors of compression and decompression speed, Dictionary gives an average performance. In fact, it will take a considerable amount time for larger data. But, for Run-length, without recurrent pattern its results have been on the lower side.

Aberrant outputs corresponding to the inputs really prove that Dictionary algorithm is better. For the factor of "Saving percentage", without Run-length's favorable characters, its performance was very inefficient. One thing, can be stated that when the data has only 0/1, Run-length's performance is incomparably better. So, from this work, it can be concluded that without recurrent pattern Dictionary algorithm is better than Run-length algorithm in every other aspects.

VII. Conclusions

In this article, we have presented a comparison between Run-length and Dictionary Algorithm empirically. For this work, we have considered few factors which are compressed size, compression speed, decompression speed, and saving percentage. In each factor, Run-length algorithm has been inferior to Dictionary algorithm. Only at the stage when the input is recurring characters, Run-length algorithm shows better than its other results. Both of the algorithms have been tested on different pattern and size of data. After a long analysis and consideration, Dictionary algorithm is better than Run-length algorithm.

References

[1] S. Shahriyar, M. Murshed, M. Ali, and M. Paul, "Cuboid coding of depth motion vectors using binary tree based decomposition," in *2015 Data Compression Conference*. IEEE, 2015, pp. 469–469.

[2] C. Dobre and F. Xhafa, "Intelligent services for big data science," *Future generation computer systems*, vol. 37, pp. 267–281, 2014.

[3] U. Sivarajah, M. M. Kamal, Z. Irani, and V. Weerakkody, "Critical analysis of big data challenges and analytical methods," *Journal of Business Research*, vol. 70, pp. 263–286, 2017.

[4] M. Javed and P. Nagabhushan, "Automatic page segmentation without decompressing the run-length compressed text documents," *arXiv preprint arXiv:2007.01142*, 2020.

[5] B. Zhao, W. Huang, H. H. Wang, and Z. Liu, "Image de-noising algorithm based on image reconstruction and compression perception," in *2017 International Conference on Inventive Computing and Informatics (ICICI)*. IEEE, 2017, pp. 532–535.

[6] S. Kodituwakku and U. Amarasinghe, "Comparison of lossless data compression algorithms for text data," *Indian journal of computer science and engineering*, vol. 1, no. 4, pp. 416–425, 2010.

[7] S. Porwal, Y. Chaudhary, J. Joshi, and M. Jain, "Data compression methodologies for lossless data and comparison between algorithms," *International Journal of Engineering Science and Innovative Technology (IJESIT) Volume*, vol. 2, pp. 142–147, 2013.

[8] A. Terekhov and A. Korchagina, "Reference signal compression in intrusive methods for assessing the quality of speech transmission using the unitary numbering system," in *2021 Systems of Signals Generating and Processing in the Field of on Board Communications*. IEEE, 2021, pp. 1–4.

[9] N. R. Brisaboa, R. Cánovas, F. Claude, M. A. Martínez-Prieto, and G. Navarro, "Compressed string dictionaries," in *International Symposium on Experimental Algorithms*. Springer, 2011, pp. 136–147.

[10] N. R. Brisaboa, A. Cerdeira-Pena, G. de Bernardo, and G. Navarro, "Improved compressed string dictionaries," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 29–38.

[11] D. Kempa and N. Prezza, "At the roots of dictionary compression: String attractors," in *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, 2018, pp. 827–840.

[12] J. Uthayakumar and T. Vengattaraman, "Performance evaluation of lossless compression techniques: An application of satellite images," in *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*. IEEE, 2018, pp. 750–754.

[13] S. Bachu and K. M. Achari, "Motion estimation in video compression and comparision between lsk, huffman & runlength coding," in *2015 International Conference on Electrical, Electronics, Signals, Communication and Optimization (EESCO)*.

[14] H. Daou and F. Labeau, "Real-time compression of intra-cerebral eeg using eigendecomposition with dynamic dictionary," in *2013 Data Compression Conference*. IEEE, 2013, pp. 486–486.

[15] R. B. Patil and K. Kulat, "Audio compression using dynamic huffman and rle coding," in *2017 2nd International Conference on Communication and Electronics Systems (ICCES)*. IEEE, 2017, pp. 160–162.

[16] S. Shanmugasundaram and R. Lourdusamy, "A comparative study of text compression algorithms," *International Journal of Wisdom Based Computing*, vol. 1, no. 3, pp. 68–76, 2011.

[17] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on information theory*, vol. 23, no. 3, pp. 337–343, 1977.

[18] S. S. Akash. Implementation documentation. (2021). [Online]. Available: https://github.com/ssakash6611/An-empirical-analysis-of-compression-technique.git