Testing the Hypothesis : Infinite Storage Inside of Pi Using String Matching Algorithm

Abdul Rafi Radityo Hutomo - 13522089
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail : abdulrafi623@gmail.com

***Abstract*— The seemingly infinite nature of pi has captivated mathematicians for centuries. Recently, the hypothesis that pi might harbor all possible data strings within its digits has gained traction. This paper investigates this proposition by employing string matching algorithms to systematically search for predefined data patterns within pi's digits. We explore the theoretical foundation of normal numbers, which relates to the uniform distribution of digits within a number's representation. Leveraging the Bailey-Borwein-Plouffe (BBP) formula for digit-level exploration, we present a methodology for data storage within pi using minimal metadata and dynamic text generation. This approach necessitates efficient string matching within a vast, dynamic text string (pi's digits). We address this challenge by employing a "chunking" strategy, dividing the data into smaller segments for optimized searching. By analyzing the success rate and efficiency of these searches under different chunking granularities (per byte, per 4 bits), we aim to assess the validity of the hypothesis and explore the potential implications for data storage and information retrieval.**

Keywords—pi; string matching; storage

I. INTRODUCTION

Pi (π) is a well-known mathematical constant representing the ratio of a circle's circumference to its diameter. However, beyond its role in geometry, Pi has the characteristics of containing every single possible string of numbers in its decimal expansion. This gives rise to a hypothetical theory to use Pi as a storage media able to store infinite information, supposedly solving the problem of storage once and for all. This is theoretically done by calculating the offset at which a certain piece of information encoded in the forms of hexadecimal numbers appear among the decimal expansion of Pi. This procedure would be done by generating the decimal expansion of Pi and using string matching algorithms to find a certain piece of information which we want to store.

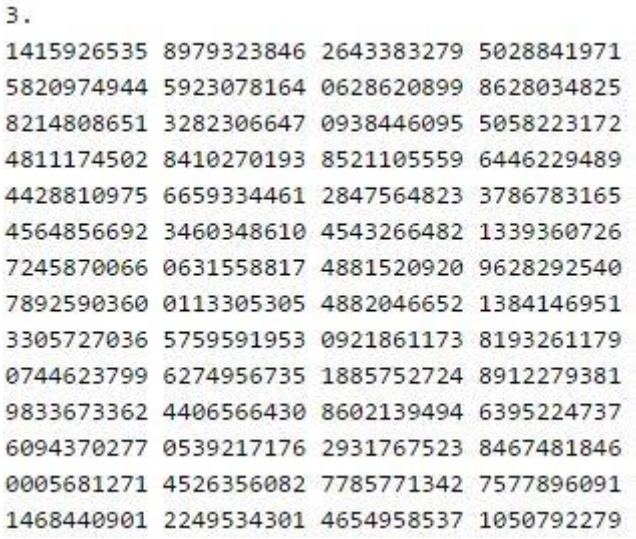

Figure 1 : The Infinite Decimal Expansion of Pi (Source : wfaa.com)

II. THEORETICAL FOUNDATIONS

A. Normal Numbers

In the realm of number theory, a special class of real numbers exists known as "normal numbers." These numbers possess a property related to their digit representation in any chosen base (b, where b is an integer greater than 1). A real number is considered normal in base-b if, when expressed in that base, all digit sequences of equal length appear with equal probability as the number of digits approaches infinity. Imagine an infinitely long decimal representation, where each digit has an equal chance of being 0, 1, 2, and so on, regardless of its position within the sequence. This concept extends to any base system, not just base-10 (decimals). The theoretical significance of normal numbers lies in their inherent randomness and uniformity. They lack any discernible patterns or biases in their digit distribution.

This concept becomes particularly relevant when studying the properties of sequences embedded within a number's representation. As the concept of uniformity within normal numbers and given their infinite property implies that any sequence of number in the respective base has equal

probabilities to appear in the decimal expansion of such number.

The normality of number, however, is not easily provable. So far, only a few numbers have been proven to be normal, namely the uncomputable Chaitlin's Constant. While the computable numbers such as √2, π, or e is widely believed to be normal, but remains unproven. For the remainder of this paper, we will assume that π is a normal number.

B. Bailey-Borwein-Plouffe Formula

Bailey-Borwein-Plouffe (BBP) formula is a formula to obtain the value of pi using the given summation :

$$\sum_{k=0}^{\infty}\left[\frac{1}{16^k}\left(\frac{4}{8k+1}-\frac{2}{8k+4}-\frac{1}{8k+5}-\frac{1}{8k+6}\right)\right]$$

Now, using this formula, it possible to derive a formula to calculate the nth digit of Pi in base 16 without calculating the preceding digits of Pi in the following way:

$$\Sigma(n,j)=\sum_{k=0}^{n}\frac{16^{n-k}\bmod(8k+j)}{8k+j}+\sum_{k=n+1}^{\infty}\frac{16^{n-k}}{8k+j}$$

$$\pi(n)=16\left\{4\Sigma(n,1)-2\Sigma(n,4)-\Sigma(n,5)-\Sigma(n,6)\right\}$$

where the curly brace ({}) signify fractional part.

C. String Matching

String matching algorithms are a cornerstone of various computational tasks involving text processing and information retrieval. These algorithms efficiently locate occurrences of a specific "pattern" string (containing predefined characters) within a larger "text" string. Their applications range from searching for keywords in documents to identifying gene sequences in biological data.

There exist numerous string matching algorithms, each with its own strengths and weaknesses in terms of efficiency and suitability for different scenarios. Two well-established algorithms commonly employed in such tasks are the Knuth-Morris-Pratt (KMP) algorithm and the Boyer-Moore algorithm.

C1. The Knuth-Morris-Pratt (KMP) Algorithm

This algorithm operates by pre-processing the pattern string to construct a "Longest Prefix Suffix" (LPS) Table. This table pre-computes the maximum length of a prefix of the pattern that is also a suffix of a proper substring of the pattern. During the search process, the KMP algorithm iteratively compares the pattern with the text string one character at a time. In the event of a mismatch, the LPS allows the algorithm to efficiently shift the pattern string, avoiding unnecessary character-by-character comparisons. This pre-processing step makes the KMP algorithm particularly efficient for patterns containing internal repetitions.

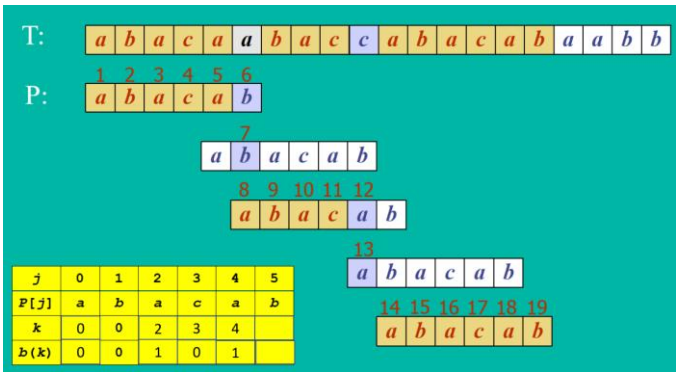

Figure 2 : Illustration of KMP Algorithm (source : informatika.stei.itb.ac.d)

C2. The Boyer-Moore Algorithm

This algorithm adopts a different approach using a combination of two techniques.

a. The looking-glass technique
   Scan the text for the pattern by moving backward through the pattern
b. The character-jump technique
   In the case of a mismatch a character jump is employed described by one of these 3 cases
   Case 1: If the pattern contains the mismatched character in the text, shift the pattern right to align the last occurrence of the mismatched character in the text with the occurrence of that character in the pattern
   Case 2: If the pattern contains the mismatched character in the text but a shift right to the last occurrence is not possible, than shift the pattern right by one step.
   Case 3: If cases 1 and 2 do not apply than shift the pattern to align the start of the pattern with the character after the mismatch

To speed up the searching, Boyer Moore algorithm uses preprocessing of the pattern to create a Last Occurrence Table containing the last occurrence of every character in the alphabet within the pattern.

The Boyer-Moore algorithm achieves efficiency using the two methods by making character jumps, reducing the number of characters needed to be matched within the text.

The choice of the most suitable string matching algorithm depends on several factors. However, one of the contributing factors is the size of the alphabets used within the text and the pattern. The KMP Algorithm excels on text with smaller alphabets thus creating more internal repetitions within the pattern while the BM Algorithm excels on text with bigger alphabets allowing the algorithm to do bigger jumps.

D. Computer Storage

At the heart of any data storage system lies its ability to reliably manage the reading and writing of data, along with the crucial task of maintaining associated metadata. This functionality is facilitated by a combination of hardware and software components working together.

On the hardware side, storage media serves as the physical repository for data. Traditional options include hard disk drives

(HDDs) for bulk storage and solid-state drives (SSDs) for faster access times. Newer technologies, such as optical storage and magnetic tape libraries, offer additional options for long-term archival or disaster recovery purposes.

Software plays a crucial role in managing data access, organization, and metadata within the storage system. File systems provide a hierarchical structure for storing and organizing data into files and folders, facilitating user interaction with the data. Additionally, storage controllers function as the brains of the system, handling data transfer requests, managing read/write operations, and ensuring data integrity.

Aside from the data stored, another crucial piece of information is metadata. Metadata acts as a descriptive layer for the stored data, providing essential information such as file size, creation date, modification history, and access permissions. Metadata management systems allow users to efficiently search for and retrieve specific data based on these attributes.

III. Methods

A. File Representation

In the realm of digital storage, all files, regardless of their perceived format (text documents, images, videos, etc.), are ultimately represented as a collection of bytes. A byte, the fundamental unit of data storage in computers, typically consists of eight bits (binary digits), each capable of holding a value of either 0 or 1. This binary sequence within each byte allows for the representation of various data types, including characters, numbers, and instructions.

Within the context of this paper, the "content" we aim to store does not correspond to the human-readable representation of a file (e.g., text characters in a document). Instead, we focus on the underlying byte representation of this content. By treating the data as an array of bytes, we essentially disregard any inherent structure or meaning associated with the original file format. This approach allows us to explore the possibility of storing any arbitrary data sequence within the seemingly random sequence of pi's digits, irrespective of the original file type. This agnostic approach to data representation forms the foundation for our investigation into the potential for pi to harbor all possible data.

In file representation, we deviate from conventional storage practices by not physically writing the data itself within our “media instead the storage solution within pi relies solely on minimal metadata.

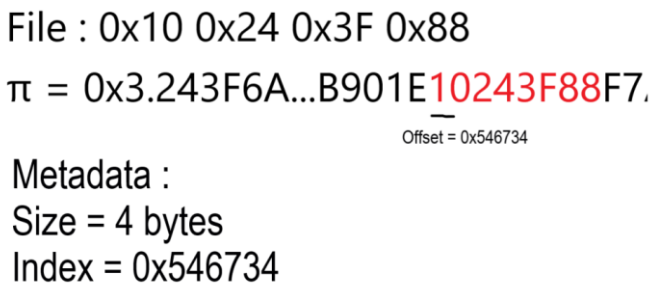

Figure 3 : Ilustration of File Representation (Author's Notes)

This metadata serves as a map to the hidden data, stored not within pi itself, but potentially retrieved from its digits on demand. At its most basic level, this metadata would consist of two crucial pieces of information:

1. Data Length: This value would represent the total number of bytes constituting the original data we intend to "store" within pi. This information is essential for accurately retrieving the data later.
2. Index in Pi: This value would pinpoint the specific location within the vast sequence of pi's digits where the data can be reconstructed. This precise indexing is critical for efficiently locating the data when needed.

By storing only this minimal metadata, we investigate the feasibility of encoding the existence and location of data within pi, without physically embedding the data inside the media itself. This minimalist approach allows us to focus on the theoretical implications of pi's information capacity, exploring the possibility of using its seemingly random sequence to represent the existence and retrievability of any data stream.

B. String Matching Elements

The problem tackled in this paper presents a unique application of string-matching algorithms for a data storage exploration scenario. Unlike traditional string matching where both the pattern and text strings are predetermined, this problem necessitates a dynamic text element.

Here, the "pattern" represents the data to be retrieved, which translates to a byte array corresponding to the original file. However, the "text" string deviates from the static nature typically encountered in string matching. In this case, the text string embodies the infinite decimal representation of pi (π).

A crucial aspect of this approach lies in the dynamic generation of the text. The specific portion of pi's digits relevant to data retrieval is not pre-computed or stored beforehand. Instead, it is dynamically generated upon accessing a specific index within the pi sequence. This index is a critical component of the minimal metadata associated with the data, as described earlier.

C. File Chunking

The BBP Formula, while able to directly calculate values at certain index in pi, requires greater computation time for higher index. In the context of this problem, where the "pattern" represents the data to be retrieved (potentially a large file),

directly searching for the entire data sequence within pi's digits could be time consuming. To address this challenge a technique to chunk the data is emplyed.

Chunking involves dividing the data (byte array) into smaller, more manageable segments. This approach significantly reduces the search time required by the string-matching algorithm. There are various chunking strategies, each with its own advantages and considerations.

In this investigation, we explore two primary chunking options:

Chunking per Byte: Here, the data is divided into segments of one byte each, possibly the most natural representation of a file itself.

Chunking per 4 Bits: This approach segments the data into units of four bits (half a byte). This allows the algorithm to find an earlier index for which the same data might be located at (look at illutration).

IV. Results

A. Experimentation Data

Table 1 : Comparison of Different Algorithms and Chunking

Text	Algorithm	Chuking	Encode Time (s)	Decode Time (s)
"Hello World" (panjang 11)	KMP	Byte	2.338	2.344
		4 bits	0.001	0.002
	BM	Byte	2.374	2.379
		4 bits	0.002	0.002
"Strategi Algoritma Itu Menyenangkan" (panjang 34)	KMP	Byte	4.539	4.551
		4 bits	0.003	0.003
	BM	Byte	4.630	4.642
		4 bits	0.003	0.005
"Lorem Ipsum Dolor si amet…" (hingga 842 karakter)	KMP	Byte	175.6	175.955
		4 bits	0.057	0.073
	BM	Byte	154.773	154.122
		4 bits	0.057	0.075
Beberapa Algoritma String Matching Adalah Knut-Morris-Pratt (KMP) dan Boyer Moore Algorithm (BM) keduanya memiliki kelebihan dan kekurangannya masing-masing (Panjang 156)	KMP	Byte	27.817	27.902
		4 bits	0.029	0.036
	BM	Byte	32.418	32.486
		4 bits	0.025	0.033

Table 2 : Bit Length Required to store index for Byte Encoded

Text	Byte	Avera	Maximum Bit
	Encoded Result	ge Bit Length Requir ed	Length
"Hello World"	139 143 43 43 749 609 138 749 403 43 472	8.181818	10
"Strategi Algoritma Itu Menyenangkan"	163 126 403 443 126 143 94 76 609 155 43 94 749 403 76 126 66 443 609 133 126 505 609 58 143 381 64 143 381 443 381 94 96 443 381	0.085714	10
"Lorem Ipsum Dolor si amet…" (hingga 842 karakter)	Omitted(In Appendix)	3.386445	6
"Beberapa Algoritma String Matching Adalah Knut-Morris-Pratt (KMP) dan Boyer Moore Algorithm (BM) keduanya memiliki kelebihan dan kekurangannya masing-masing" (Panjang 156)	Omitted(in Appendix)	8.294872	10

Text	4 Bits Encoded Result	Average Bit Length Required	Maximum Bit Length
"Hello World"	2 7 5 10 5 58 5 58 5 4 1 13 10 27 5 4 27 1 5 58 5 2	3.454545	6
"Strategi Algoritma Itu Menyenangkan"	10 0 27 2 27 1 5 17 27 2 5 10 5 27 5 20 1 13 2	3.642857	7

	17 5 58 5 27 5 4 27 1 5 20 27 2 5 15 5		
“Lorem Ipsum Dolor si amet…” (hingga 842 karakter)	Omitted (in Appendix)	8.325803	10
“Beberapa Algoritma String Matching Adalah Knut-Morris-Pratt (KMP) dan Boyer Moore Algorithm (BM) keduanya memiliki kelebihan dan kekurangannya masing-masing” (Panjang 156)	Omitted (in Appendix)	3.445513	7

V. Analyis

From the experimentation data, it could be seen that the KMP Algorithm is faster than BM algorithm. This is most likely due to KMP algorithms making big character jumps because of the length of the pattern that tends to be long, so a mistake far into the pattern will cause the KMP Algorithm to be more efficient, while in BM algorithm, the character jump wouldn’t be as apparent.

It can also be inferred that chunking of 4 bit is significantly faster than byte chunking, this is most likely due to the extra time needed to generate more digits of Pi and further into Pi aswell, meaning the time needed for byte chunking is longer because of the quantity and also the amount of time needed to generate digits of pi further into the pattern is also increasing.

Finally, to store the index of digits of Pi, using byte chunking you need more than one byte, while for half byte chunking you need about 3.445, more than the traditional average, 3. This means that in order to store a file inside of Pi, you will need more bytes to store the index of the file in Pi meaning it would require more storage than the actual file.

VI. Conclusion

In Conclusion, it is not possible to use Pi to store infinite storage. However, if one wishe to do so, the preferred string matching algorithms would be Knuth-Morris-Pratt Algorithm instead of Boyer Moore

Acknowledgments

The author would like to express their gratitude to Allah SWT for the ease and success in completing this paper on time. Their sincere appreciation is also extended to the lecturers of the IF2211 Algorithm Strategies program, especially Dr. Ir. Rinaldi Munir, M.T., who taught in Class K1 and provided the knowledge that enabled the author to complete this assignment. The author would also like to thank their family and friends for their motivation and support during the paper writing process. Despite being aware of the shortcomings of this paper, the author hopes that this work can be beneficial to the wider community.

References

[1] R. Munir, ‘IF2211 Strategi Algoritma - Semester II Tahun 2023/2024’,https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

[2] Giordano, Mose ‘Computing the hexadecimal value of pi’ https://giordano.github.io/blog/2017-11-21-hexadecimal-pi/

[3] geeksforgeeks.org, ‘KMP Algorithm for Pattern Searching’,https://www.geeksforgeeks.org/kmp-algorithm-for-pattern-searching/ accessed on 12 June 2024.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Juni 2024

Abdul Rafi Radityo Hutomo - 13522089

Appendix

A. Result of encoding of large body of text

Original Text	Byte Encoded	Half Byte Encoded
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Curabitur et volutpat massa, tincidunt varius nulla. Morbi posuere, nulla at pharetra congue, felis lorem condimentum risus, non blandit elit massa sed enim. Aliquam accumsan rhoncus lectus a lobortis. Vivamus pretium mauris sit amet rutrum congue. Duis ac consequat quam. Suspendisse potenti. Duis porttitor egestas pellentesque. Integer eu tempus ex, sit amet faucibus ipsum. Morbi sodales pharetra finibus. Nullam elit velit, mattis vel congue vitae, tincidunt in erat. Phasellus feugiat lectus sit amet rhoncus ullamcorper. Integer varius congue magna, eget varius enim sodales vel. Nam luctus neque odio, nec faucibus neque vulputate non. Etiam a odio volutpat, auctor magna nec, aliquam ligula. Vivamus at ipsum cursus, bibendum ipsum ac, efficitur turpis. Fusce at varius ante.	241 749 403 143 66 609 76 62 296 505 66 609 472 749 43 749 403 609 296 76 126 609 443 66 143 126 507 609 36 749 381 296 143 36 126 143 126 505 403 609 443 472 76 62 76 296 36 76 381 94 609 143 43 76 126 115 609 1 505 403 443 309 76 126 505 403 609 143 126 609 529 749 43 505 126 62 443 126 609 66 443 296 296 443 507 609 126 76 381 36 76 472 505 381 126 609 529 443 403 76 505 296 609 381 505 43 43 443 115 609 58 749 403 309 76 609 62 749 296 505 143 403 143 507 609 381 505 43 43 443 609 443 126 609 62 708 443 403 143 126 403 443 609 36 749 381 94 505 143 507 609 120 143 43 76 296 609 43 749 403 143 66 609 36 749 381 472 76 66 143 381 126 505 66 609 403 76 296 505 296 507 609 381 749 381 609 309 43 443 381 472 76 126 609 143 43 76 126 609 66 443 296 296 443 609 296 143 472 609 143 381 76 66 115 609 155 43 76 201 505 443 66 609 443 36 36 505 66 296 443 381 609 403 708 749 381 36 505 296 609 43 143 36 126 505 296 609 443 609 43 749 309 749 403 126 76 296 115 609 213 76 529 443 66 505 296 609 62 403 143 126 76 505 66 609 66 443 505 403 76 296 609 296 76 126 609 443 66 143 126 609 403 505 126 403 505 66 609 36 749 381 94 505 143 115 609 231 505 76 296 609 443 36 609 36 749 381 296 143 201 505 443 126 609 201 505 443 66 115 609 163 505 296 62 143 381 472 76 296 296 143 609 62 749 126 143 381 126 76 115 609 231 505 76 296 609 62 749 403 126 126 76 126 749 403 609 143 94 143 296 126 443 296 609 62 143 43 43 143 381 126 143 296 201 505 143 115 609 133 381 126 143 94 143 403 609 143 505 609 126 143 66 62 505 296 609 143 824 507 609 296 76 126 609 443 66 143 126 609 120 443 505 36 76 309 505 296 609 76 62 296 505 66 115 609 58 749 403 309 76 609 296 749 472 443 43 143 296 609 62 708 443 403 143 126 403 443 609 120 76 381 76 309 505 296 115 609 45 505 43 43 443 66 609 143 43 76 126 609 529 143 43 76 126 507 609 66 443 126 126 76 296 609 529 143 43 609 36 749 381 94 505 143 609 529 76 126 443 143 507 609 126 76 381 36 76 472 505 381 126 609 76 381 609 143 403 443 126 115 609 352 708 443 296 143 43 43 505 296 609 120 143 505 94 76 443 126 609 43 143 36 126 505 296 609 296 76 126 609 443 66 143 126 609 403 708 749 381 36 505 296 609 505 43 43 443 66 36 749 403 62 143 403 115 609 133 381 126 143 94 143 403 609 529 443 403 76 505 296 609 36 749 381 94 505 143 609 66 443 94 381 443 507 609 143 94 143 126 609 529 443 403 76 505 296 609 143 381 76 66 609 296 749 472 443 43 143 296 609 529 143 43 115 609 45 443 66 609 43 505 36 126 505 296 609 381 143 201 505 143 609 749 472 76 749 507 609 381 143 36 609 120 443 505 36 76 309 505 296 609 381 143 201 505 143 609 529 505 43 62 505 126 443 126 143 609 381 749 381 115 609 129 126 76 443 66 609 443 609 749 472 76 749 609 529 749 43 505 126 62 443 126 507 609 443 505 36 126 749 403 609 66 443 94 381 443 609 381 143 36 507 609 443 43 76 201 505 443 66 609 43 76 94 505 43 443 115 609 213 76 529 443 66 505 296 609 443 126 609 76 62 296 505 66 609 36 505 403 296 505 296 507 609 309 76 309 143 381 472 505 66 609 76 62 296 505 66 609 443 36 507 609 143 120 120 76 36 76 126 505 403 609 126 505 403 62 76 296 115 609 42 505 296 36 143 609 443 126 609 529 443 403 76 505	2 58 5 4 27 1 5 10 5 15 1 13 5 20 27 13 27 0 27 10 5 15 1 13 5 2 5 4 5 58 5 4 27 1 1 13 27 0 5 20 27 2 1 13 5 17 5 15 5 10 27 2 1 58 1 13 5 0 5 4 5 24 27 0 5 10 5 0 27 2 5 10 27 2 27 10 27 1 1 13 5 17 5 2 5 20 27 13 5 20 27 0 5 0 5 20 5 24 5 27 1 13 5 10 5 58 5 20 27 2 1 24 1 13 2 0 27 10 27 1 5 17 5 1 5 20 27 2 27 10 27 1 1 13 5 10 27 2 1 13 27 5 5 4 5 58 27 10 27 2 27 13 5 17 27 2 1 13 5 15 5 17 27 0 27 0 5 17 1 58 1 13 27 2 5 20 5 24 5 0 5 20 5 2 27 10 5 24 27 2 1 13 27 5 5 17 27 1 5 20 27 10 27 0 1 13 5 24 27 10 5 58 5 58 5 17 1 24 1 13 2 15 5 4 27 1 5 1 5 20 1 13 27 13 5 4 27 0 27 10 5 10 27 1 5 10 1 58 1 13 5 24 27 10 5 58 5 58 5 17 1 13 5 17 27 2 1 13 27 13 5 7 5 17 27 1 5 10 27 2 27 1 5 17 1 13 5 0 5 4 5 24 5 27 27 10 5 10 1 58 1 13 5 5 5 10 5 58 5 20 27 0 1 13 5 58 5 4 27 1 5 10 5 15 1 13 5 0 5 4 5 24 5 2 5 20 5 15 5 10 5 24 27 2 27 10 5 15 1 13 27 1 5 20 27 0 27 10 27 0 1 58 1 13 5 24 5 4 5 24 1 13 5 1 5 58 5 17 5 24 5 2 5 20 27 2 1 13 5 10 5 58 5 20 27 2 1 13 5 15 5 17 27 0 27 0 5 17 1 13 27 0 5 10 5 2 1 13 5 10 5 24 5 20 5 15 1 24 1 13 2 17 5 58 5 20 27 17 27 10 5 17 5 15 1 13 5 17 5 0 5 0 27 10 5 15 27 0 5 17 5 24 1 13 27 1 5 7 5 4 5 24 5 0 27 10 27 0 1 13 5 58 5 10 5 0 27 2 27 10 27 0 1 13 5 17 1 13 5 58 5 4 5 1 5 4 27 1 27 2 5 20 27 0 1 24 1 13 10 5 5 20 27 5 5 17 5 15 27 10 27 0 1 13 27 13 27 1 5 10 27 2 5 20 27 10 5 15 1 13 5 15 5 17 27 10 27 1 5 20 27 0 1 13 27 0 5 20 27 2 1 13 5 17 5 15 5 10 27 2 1 13 27 1 27 10 27 2 27 1 27 10 5 15 1 13 5 0 5 4 5 24 5 27 27 10 5 10 1 24 1 13 2 2 27 10 5 20 27 0 1 13 5 17 5 0 1 13 5 0 5 4 5 24 27 0 5 10 27 17 27 10 5 17 27 2 1 13 27 17 27 10 5 17 5 15 1 24 1 13 10 0 27 10 27 0 27 13 5 10 5 24 5 2 5 20 27 0 27 0 5 10 1 13 27 13 5 4 27 2 5 10 5 24 27 2 5 20 1 24 1 13 2 2 27 10 5 20 27 0 1 13 27 13 5 4 27 1 27 2 27 2 5 20 27 2 5 4 27 1 1 13 5 10 5 27 5 10 27 0 27 2 5 17 27 0 1 13 27 13 5 10 5 58 5 58 5 10 5 24 27 2 5 10 27 0 27 17 27 10 5 10 1 24 1 13 2 20 5 24 27 2 5 10 5 27 5 10 27 1 1 13 5 10 27 10 1 13 27 2 5 10 5 15 27 13 27 10 27 0 1 13 5 10 27 7 1 58 1 13 27 0 5 20 27 2 1 13 5 17 5 15 5 10 27 2 1 13 5 5 5

	296 609 443 381 126 143 115	
Beberapa Algoritma String Matching Adalah Knut-Morris-Pratt (KMP) dan Boyer Moore Algorithm (BM) keduanya memiliki kelebihan dan kekurangannya masing-masing	366 143 309 143 403 443 62 443 609 155 43 94 749 403 76 126 66 443 609 163 126 403 76 381 94 609 58 443 126 36 708 76 381 94 609 155 472 443 43 443 708 609 305 381 505 126 83 58 749 403 403 76 296 83 352 403 443 126 126 609 116 305 58 352 485 609 472 443 381 609 366 749 64 143 403 609 58 749 749 403 143 609 155 43 94 749 403 76 126 708 66 609 116 366 58 485 609 96 143 472 505 443 381 64 443 609 66 143 66 76 43 76 96 76 609 96 143 43 143 309 76 708 443 381 609 472 443 381 609 96 143 96 505 403 443 381 94 443 381 381 64 443 609 66 443 296 76 381 94 83 66 443 296 76 381 94	1 5 10 5 1 5 10 27 1 5 17 27 13 5 17 1 13 2 17 5 58 5 27 5 4 27 1 5 20 27 2 5 15 5 17 1 13 10 0 27 2 27 1 5 20 5 24 5 27 1 13 2 15 5 17 27 2 5 0 5 7 5 20 5 24 5 27 1 13 2 17 5 2 5 17 5 58 5 17 5 7 1 13 2 81 5 24 27 10 27 2 1 15 2 15 5 4 27 1 27 1 5 20 27 0 1 15 10 13 27 1 5 17 27 2 27 2 1 13 1 7 2 81 2 15 10 13 1 20 1 13 5 2 5 17 5 24 1 13 2 1 5 4 27 20 5 10 27 1 1 13 2 15 5 4