Matrix Application in the Smith-Waterman Algorithm for DNA Local Alignment

Fityatul Haq Rosyidi - 13523116[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1] 13523116@std.stei.itb.ac.id FityatulHaqRosyidi25@gmail.com

***Abstract*— The Smith-Waterman algorithm is widely used in bioinformatics for local sequence alignment, aiming to find the most similar subsequences between two biological sequences, such as DNA, RNA, or protein. This paper discusses the application of matrices in the Smith-Waterman algorithm, focusing on how the scoring matrix is constructed and utilized to perform local alignments of DNA sequences. The matrix represents the dynamic programming approach where each cell contains a score reflecting the similarity or dissimilarity between sequence elements. The alignment is derived by tracing back from the highest score in the matrix, identifying the optimal local alignment. The paper further elaborates on the implementation of this algorithm using Python, emphasizing the use of matrices to manage large sequence data and calculate the best local alignment.**

***Keywords*— Local Alignment, Matrix, Smith-Waterman Algorithm**

I. Introduction

DNA Sequence Alignment is a process in bioinformatics aimed at identifying similarities or differences between two or more DNA sequences. The Smith-Waterman algorithm, first introduced in 1981, has become one of the primary methods for local sequence alignment. This algorithm uses a dynamic programming approach to build a scoring matrix that reflects the similarity between sequence pairs, with the goal of identifying the most similar segments of the sequences.

This paper discusses the application of matrix concepts in the Smith-Waterman algorithm for DNA sequence alignment. The main focus of this discussion is how matrices are used to reflect the relationships between nucleotide bases. Additionally, this paper will also explore how the Smith-Waterman algorithm is implemented using the Python programming language.

II. Theoretical Framework

A. DNA

DNA (*Deoxyribonucleic Acid*) is a biological molecule that serves as the repository of genetic information in nearly all living organisms, including humans. It is structured as a double-helix chain made up of nucleotides, with each nucleotide consisting of three key components: a phosphate group, deoxyribose sugar, and one of four nitrogenous bases—adenine, thymine, cytosine, or guanine. The double-helix structure of DNA was first discovered by James Watson and Francis Crick in 1953, demonstrating how this molecule can store and replicate genetic information with remarkable precision. The genetic information contained within DNA is organized as a sequence of nitrogenous bases, which is later translated into proteins through the processes of transcription and translation [1].

In modern biotechnology, DNA plays a crucial role in various important applications, ranging from genetic analysis and genetic engineering to forensics. Techniques such as Polymerase Chain Reaction (PCR) have enabled the amplification of small amounts of DNA to sufficient levels for analysis. The Short Tandem Repeat (STR) technique is widely used in forensic identification to recognize individuals through biological samples such as hair or saliva. Additionally, in the medical field, DNA sequencing technology has helped scientists understand genetic mutations that contribute to the development of diseases like cancer.

Beyond its applications in applied sciences, DNA is essential in understanding evolution and the genetic relationships between species. DNA isolation and cloning techniques have facilitated the reconstruction of phylogenetic trees that reveal the evolutionary connections between species. In Indonesia, DNA analysis technology has been used for identifying accident victims and providing evidence in criminal cases. As technology advances, DNA has become not only a tool for understanding the biology of life but also a key to the development of gene therapy and solutions to various global health challenges.

B. Nucleotide Bases

In molecular biology, nucleotides are the fundamental units that form the building blocks of nucleic acids such as DNA and RNA. Each nucleotide consists of three components: a phosphate group, a sugar molecule, and a nitrogenous base. There are four primary nitrogenous bases in DNA, each contributing to the genetic code. These bases are adenine (A), thymine (T), cytosine (C), and guanine (G) [2].

1) Adenine (A)

Adenine is a purine base, which means it consists of a two-ring structure. In the DNA double helix, adenine pairs with thymine through two hydrogen bonds. This base is essential in encoding genetic information and plays a role in energy transfer processes within the cell, such as ATP (adenosine triphosphate) functions.

2) Thymine (T)

Thymine is a pyrimidine base, with a single-ring structure. It is complementary to adenine, forming two hydrogen bonds in the DNA double helix. Thymine plays a crucial role in maintaining the integrity of the genetic information by participating in base pairing, ensuring accurate replication and transcription of DNA.

3) Cytosine (C)

Cytosine is another pyrimidine base, with a single-ring structure. It pairs with guanine through three hydrogen bonds in the DNA structure. Cytosine is involved in various cellular processes, including the regulation of gene expression and the repair of damaged DNA, helping to maintain the stability of the genome.

4) Guanine (G)

Guanine is a purine base, similar to adenine, with a two-ring structure. In the DNA double helix, guanine pairs with cytosine through three hydrogen bonds. Guanine plays a significant role in the stability of the DNA structure, as the G-C base pair is one of the most thermodynamically stable pairs. Additionally, guanine is involved in RNA and protein synthesis.

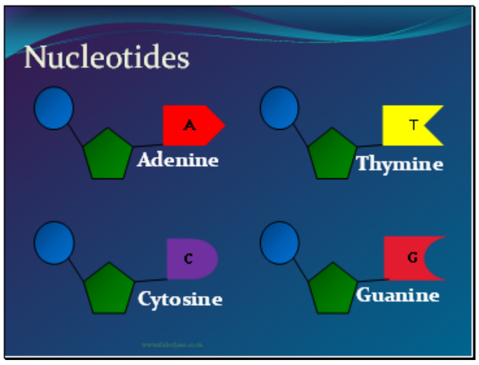


Figure 1. Nucleotide Bases (Source : DNA Structure - Presentation Genetics)

These four nucleotide bases are essential in forming the genetic code that governs cellular functions and inheritance. The specific sequence of these bases along a DNA strand encodes genetic information, which is used during processes like transcription and translation to produce proteins and other cellular components.

C. Matrix

Matrix is an arrangement of numbers in a rectangular format, organized into rows and columns, used to represent and manipulate data in various fields such as mathematics, physics, and engineering [3]. Matrices enable the simplification and solution of systems of linear equations, linear transformations, and the analysis of complex data. Basic operations on matrices include addition, subtraction, multiplication, and determinant calculation, all of which play a crucial role in numerical analysis and scientific computation.

$$A_{m\times n} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ \vdots & \ddots & \vdots \\ a_{m,1} & \cdots & a_{m,n} \end{bmatrix}$$

Figure 2. Matrix of dimension m×n (Source : writer's archive)

In the context of linear algebra, matrices are used to represent linear transformations, where an input vector is multiplied by a matrix to produce a transformed output vector. This enables the analysis of the properties of transformations such as rotation, scaling, and reflection in multidimensional spaces. Additionally, matrices are employed in modeling various phenomena in science and engineering, including signal processing, computer graphics, and network analysis.

A deep understanding of matrices and their operations forms the foundation for various numerical methods used to solve complex problems in science and engineering. For example, the Gaussian elimination method applies elementary row operations to matrices to solve systems of linear equations. Furthermore, the concepts of eigenvalues and eigenvectors of matrices are used in stability analysis and the dynamics of systems.

III. Smith-Waterman Algorithm

A. Definition

The Smith-Waterman algorithm is a local sequence alignment algorithm used to find similarities between two biological sequences, such as DNA, RNA, or proteins [4]. Developed by Temple Smith and Michael Waterman in 1981, this algorithm utilizes a dynamic programming approach to construct a scoring matrix that reflects how similar pairs of bases or amino acids are within the sequences being analyzed. This matrix is calculated by considering three potential sources of score values for each element in the matrix: match, mismatch, and gap. The algorithm is specifically optimized to find the most similar parts of sequences without requiring a full alignment of the entire sequence, distinguishing it from global alignment algorithms like Needleman-Wunsch.

The Needleman-Wunsch algorithm, introduced in 1970 by Saul Needleman and Christian Wunsch, is a global sequence alignment algorithm that aligns two sequences over their entire length. Unlike the Smith-Waterman algorithm, which is focused on local alignment, Needleman-Wunsch aims to find the best alignment between two sequences by aligning every part of the sequences from start to end. It uses dynamic programming to create a scoring matrix, similar to Smith-Waterman, but it penalizes gaps throughout the entire sequence, making it more suitable for aligning complete sequences. The primary difference between the two algorithms lies in their scope: Needleman-Wunsch is designed for global alignment, whereas Smith-Waterman is designed for local alignment, making Smith-Waterman more efficient when dealing with subsequences that may have a high degree of similarity, even if they don't align perfectly across the entire length of the sequences.

The Smith-Waterman algorithm was developed by Temple Smith and Michael Waterman in 1981 as a solution to the problem of local sequence alignment. Their work built upon earlier methods for sequence comparison and introduced dynamic programming to find the optimal local alignment between two sequences. The algorithm was groundbreaking because it allowed for the identification of highly similar regions between sequences, even when these regions were embedded within larger, unrelated parts of the sequences. This provided a powerful tool for understanding the evolutionary relationships between species and the functional roles of various genetic sequences. The introduction of the Smith-Waterman algorithm marked a major advancement in computational biology and bioinformatics, enabling the detailed comparison of genetic sequences that is now a fundamental aspect of genomics.

In bioinformatics, the Smith-Waterman algorithm plays a crucial role in many applications, particularly in the comparison and alignment of biological sequences such as DNA, RNA, and protein sequences. It is used extensively in genomic research, including the identification of gene sequences, the study of mutations, and the comparison of genetic diversity among different organisms. For example, Smith-Waterman is commonly used in sequence homology searches, such as in BLAST (Basic Local Alignment Search Tool), to find regions of local similarity between sequences in large biological databases. The algorithm is also instrumental in protein structure prediction, where identifying conserved regions in protein sequences can help infer the function of unknown proteins. Additionally, Smith-Waterman has applications in the detection of sequence motifs, which are short, recurring patterns that have biological significance. Overall, the algorithm's ability to find local alignments makes it an invaluable tool in advancing our understanding of genetics and molecular biology.

B. Flow

Here is the flow of the Smith-Waterman algorithm:

Step 1: Initialize the Scoring Matrix

First, create a scoring matrix, say H, with dimensions $(m + 1) \times (n + 1)$, where m is the length of the first sequence and n is the length of the second sequence.

Set all elements in the first row and the first column of matrix H to 0. This matrix will be used to calculate the score based on sequence comparisons.

The input consists of two nucleotide sequences, for example, Seq1 and Seq2.

Step 2 : Filling the Scoring Matrix

For each element H_{ij} starting from i = 1 to m and j = 1 to n:

- Check for match : if $Seq1_{i-1} = Seq2_{i-1}$, then the score is the match_score; oterwise, use the mismatch_penalty
- Calculate the score for each element:

$$H_{ij} = \max\,(0, score, H_{(i-1)j} - gap\ penalty, H_{i(j-1)} - gap\ penalty)$$

- Store the value of H_{ij} and the position (i, j) if H_{ij} is greter than the maximum value found so far.
- After filling the entire matrix, determine the position (i, j) with the highest value in matrix H. This position serves as the starting point for the traceback process.

Step 3 : Traceback

Start the traceback from the position with the highest score (i, j) until $H_{ij} = 0$ (a score of 0 indicates the end of the local alignment).

If the value of H_{ij} comes from diagonal (i.e., there was a match or mismatch) :

- Add Seq1[i-1] to the aligned sequence (aligned_seq1).
- Add Seq2[j-1] to the aligned sequence (aligned_seq2).
- Move to $i - 1$ and $j - 1$

If the value of H_{ij} comes from the top row (gap in Seq2):

- Add $Seq1_{i-1}$ to the aligned sequence (aligned_seq1).
- Add a "-" (gap) to aligned_seq2.
- Move to $i - 1$

If the value of H_{ij} comes from the left column (gap in Seq1):

- Add $Seq2_{j-1}$ to the aligned sequence (aligned_seq2).
- Add a "-" (gap) to aligned_seq1.
- Move to $j - 1$

Once the traceback is complete, the aligned sequences will be obtained in reverse order. Reverse aligned_seq1 and aligned_seq2 to obtain the final alignment result.

Return aligned_seq1 and aligned_seq2 as the result of the local alignment of the two sequences.

C. Complexity

The Smith-Waterman algorithm is used for local sequence alignment in bioinformatics and typically has a time complexity that is considered quadratic, i.e., $O(n^2)$, where n is the length of the sequences being compared. This quadratic complexity arises because the algorithm constructs a matrix of size $(m + 1) \times (n + 1)$ to calculate the local alignment, with m and n being the lengths of the two sequences. Each cell in the matrix is filled by considering the best score from neighboring cells, which results in a time complexity proportional to the product of the lengths of the sequences being compared.

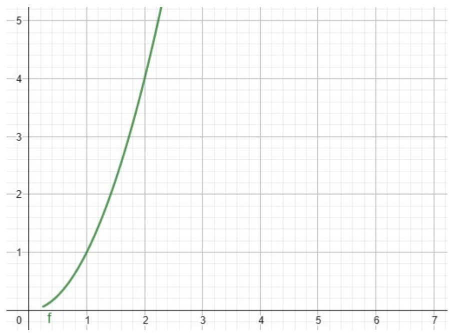


Figure 3. $O(n^2)$ Complexity Graph (Source : GeoGebra)

IV. Implementation

This algorithm is implemented using Python and utilizes the NumPy library for storing the matrix structure. The source code can be accessed through the GitHub repository link provided in the appendix.

A. Directory Structure

```
Smith-Waterman-Algorithm/
├── src/
│   ├── main.py
│   └── sample-generator.py
├── doc
├── test/
│   ├── DNA-sample-1
│   ├── DNA-sample-2
│   ├── DNA-sample-3
│   └── DNA-sample-4
│        ├── seq-10.txt
│        └── seq-50.txt
│
└── README.md
```

The project directory structure is organized as follows:

- `main.py` **:** Contains the source code for the implementation of the Smith-Waterman algorithm.
- `sample-generator.py` **:** Contains the source code for generating random nucleotide sequence samples, which are useful for testing the program.
- `doc` **:** Directory for documentation related to the project.
- `test` **:** Directory containing data samples
- `README.md` **:** Contains explanations and information about the project.

B. Data Samples

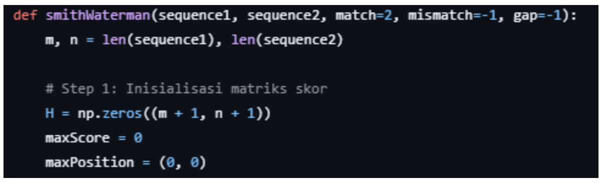
The DNA sequence samples are stored in the test/DNA-sample-<number> folder in the form of .txt files, where <number> ranges from 1 to 4. Each folder contains text file named seq-<length> where <lenght> refers to the length of the DNA sequence. Each text file containing a nucleotide sequence as shown below.

```
CTTAATAAGGCTCAGTCAAATCTACGTACAATAT
GGTTGGGCAAGCGGGA
```

C. Source Code

The main.py file contains two main functions: smithWaterman and main. However, the discussion focuses on the smithWaterman function, which is the core of this algorithm implementation.

Step 1: Initialize the Scoring Matrix

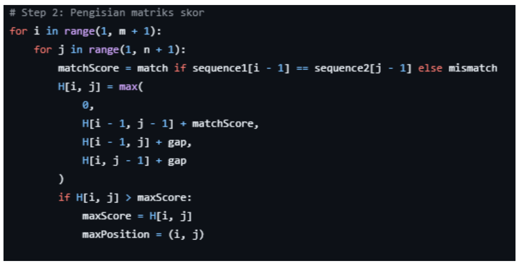


```python
def smithWaterman(sequence1, sequence2, match=2, mismatch=-1, gap=-1):
    m, n = len(sequence1), len(sequence2)

    # Step 1: Inisialisasi matriks skor
    H = np.zeros((m + 1, n + 1))
    maxScore = 0
    maxPosition = (0, 0)
```

Figure 4. Source code – step 1 (Source: writer's archive)

The code above initializes a scoring matrix called H. This matrix has dimensions of (m+1, n+1), where m and n are the lengths of the two sequences being compared. The matrix is filled with initial values of zero using np.zeros, ensuring that there are no initial scores in the calculation. Additionally, two important variables are also initialized: maxScore, which is used to track the highest score in the matrix during the matching process, and maxPosition, which stores the index position of that highest score.

Step 2 : Filling the Scoring Matrix

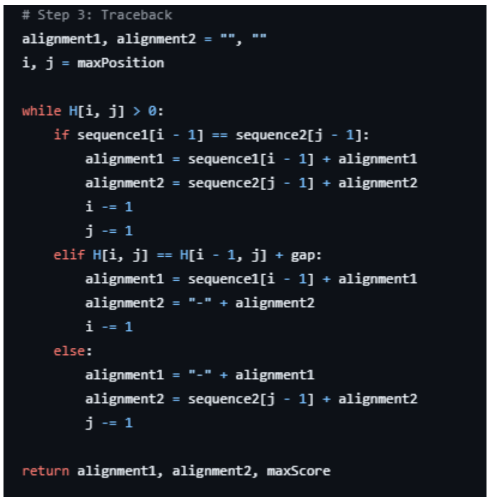


```python
# Step 2: Pengisian matriks skor
for i in range(1, m + 1):
    for j in range(1, n + 1):
        matchScore = match if sequence1[i - 1] == sequence2[j - 1] else mismatch
        H[i, j] = max(
            0,
            H[i - 1, j - 1] + matchScore,
            H[i - 1, j] + gap,
            H[i, j - 1] + gap
        )
        if H[i, j] > maxScore:
            maxScore = H[i, j]
            maxPosition = (i, j)
```

Figure 5. Source code – step 2 (Source: writer's archive)

In the second step, the scoring matrix H is filled with values based on the Smith-Waterman algorithm. This process is performed using double iteration. During each iteration, the matchScore is determined based on whether the elements from sequence1 and sequence2 at the respective indices match or not. If they match, the matchScore is set to the match value; if not, the mismatch value is used. The value for each element in the matrix H is then calculated as the maximum of four possible values. Finally, if the value of an element in the matrix H exceeds the current maxScore, both maxScore and the position of the maximum score (maxPosition) are updated accordingly.

Step 3 : Traceback



```python
# Step 3: Traceback
alignment1, alignment2 = "", ""
i, j = maxPosition

while H[i, j] > 0:
    if sequence1[i - 1] == sequence2[j - 1]:
        alignment1 = sequence1[i - 1] + alignment1
        alignment2 = sequence2[j - 1] + alignment2
        i -= 1
        j -= 1
    elif H[i, j] == H[i - 1, j] + gap:
        alignment1 = sequence1[i - 1] + alignment1
        alignment2 = "-" + alignment2
        i -= 1
    else:
        alignment1 = "-" + alignment1
        alignment2 = sequence2[j - 1] + alignment2
        j -= 1

return alignment1, alignment2, maxScore
```

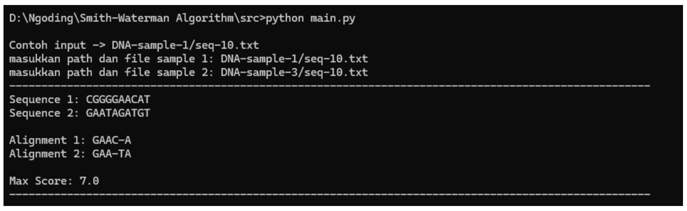
Figure 6. Source code – step 3 (Source: writer's archive)

In the third step, a traceback process is performed to construct the aligned sequences based on the maximum score calculated earlier. This starts from the position of the maximum score (maxPosition) in the matrix H. At this stage, alignment1 and alignment2 are initialized as empty strings. These two variables will later store the alignment results between the two input sequences. During the traceback, the algorithm moves through the matrix by following the path from the highest score until it reaches a value of 0, considering whether the value comes from a match, a mismatch, or a gap. As it moves through the matrix, the algorithm builds the aligned sequences by adding corresponding characters from sequence1 and sequence2 or gaps as needed.

Once the traceback process is completed, the function will return the aligned sequences alignment1 and alignment2, as well as the maxScore.

D. Testing

Here is an output for aligning a sample DNA sequence with a length of 10:

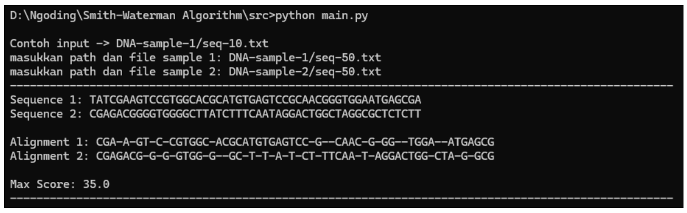


```
D:\Ngoding\Smith-Waterman Algorithm\src>python main.py

Contoh input -> DNA-sample-1/seq-10.txt
masukkan path dan file sample 1: DNA-sample-1/seq-10.txt
masukkan path dan file sample 2: DNA-sample-3/seq-10.txt
------------------------------------------------------------------------------------------------
Sequence 1: CGGGGAACAT
Sequence 2: GAATAGATGT

Alignment 1: GAAC-A
Alignment 2: GAA-TA

Max Score: 7.0
------------------------------------------------------------------------------------------------
```

Figure 7. Output for sample with length of 10 (Source: writer's archive)

As seen in the image above, alignment1 shows the first sequence with a gap at position 5 to align with T in the second sequence. Meanwhile, alignment2 shows a gap at position 4 to align with C in the first sequence.

Here is an output for aligning a sample DNA sequence with a length of 50:



```
D:\Ngoding\Smith-Waterman Algorithm\src>python main.py

Contoh input -> DNA-sample-1/seq-10.txt
masukkan path dan file sample 1: DNA-sample-1/seq-50.txt
masukkan path dan file sample 2: DNA-sample-2/seq-50.txt
------------------------------------------------------------------------------------------------
Sequence 1: TATCGAAGTCCGTGGCACGCATGTGAGTCCGCAACGGGTGGAATGAGCGA
Sequence 2: CGAGACGGGGTGGGGCTTATCTTTCAATAGGACTGGCTAGGCGCTCTCTT

Alignment 1: CGA-A-GT-C-CGTGGC-ACGCATGTGAGTCC-G--CAAC-G-GG--TGGA--ATGAGCG
Alignment 2: CGAGACG-G-G-GTGG-G--GC-T-T-A-T-CT-TTCAA-T-AGGACTGG-CTA-G-GCG

Max Score: 35.0
------------------------------------------------------------------------------------------------
```

Figure 8. Output for sample with length of 50 (Source: writer's archive)

For the sample data with a larger size of 50, the alignment result obtained is as shown above. This alignment demonstrates the best local similarity between the two DNA sequences, with minimal differences. The gaps inserted are used to maximize the matching between the sequences, highlighting the regions where the sequences are most similar to each other.

V. Conclusion

The Smith-Waterman algorithm is an essential tool in bioinformatics for performing local sequence alignment, enabling the identification of highly similar subsequences between DNA, RNA, or protein sequences. The use of matrices in this algorithm plays a critical role in organizing and calculating alignment scores through dynamic programming, making the process efficient and effective. By employing a scoring matrix, the algorithm identifies the optimal local alignment by considering matches, mismatches, and gaps. This paper demonstrates how matrices facilitate the comparison of biological sequences and provides an implementation of the algorithm using Python. With its proven accuracy in finding local similarities, the Smith-Waterman algorithm, enhanced by matrix-based computations, continues to be a valuable method in genomic analysis and bioinformatics research.

VI. Appendix

Github repository : [FityatulhaqRosyidi/Smith-Waterman-Algorithm]

VI. Acknowledgment

I would like to express my deepest gratitude to God Almighty, for His grace and blessings, allowing me to complete this paper. I also wish to sincerely thank all the individuals who have provided support and assistance in the preparation of this paper. Special thanks go to Dr. Ir. Rinaldi Munir, M.T., the lecturer for the IF2123 Linear Algebra and Geometry course. I also extend my gratitude and encouragement to my fellow students in the Department of Informatics Engineering, whose support has kept me motivated and made me feel supported throughout this process.

Lastly, I hope this paper can provide valuable insights to its readers and contribute to the advancement of knowledge, particularly in the fields of linear algebra and bioinformatics.

References

[1] J. D. Hartani, "Makalah Sel DNA," *Academia.edu*, [Online]. Available: https://www.academia.edu/7350185/Makalah_Sel_DNA. [Accessed: Jan. 1, 2025].

[2] D. Puspitaningrum, A. Pravitasari, and E. Ernawati, "Implementasi Algoritma Smith–Waterman Pada Local Alignment Dalam Pencarian Kesamaan Pensejajaran Barisan DNA (Studi Kasus: DNA Tumor Wilms)," *Pseudocode*, Universitas Bengkulu.

[3] Brijesh. S. Gupta, "Application of Matrices in Engineering:, International Journal of Science and Research (IJSR), vol. 13, no. 3, pp. 1514-1518, Mar. 2024.

[4] Bambang, Alif. 2017. Penerapan Algoritma Program Dinamis pada Penyejajaran Sekuens dengan Algoritma Smith–Waterman. Makalah IF2211 Strategi Algoritma, Bandung

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, 31 Desember 2024

[illegible]

Fityatul Haq Rosyidi 13523116