Applications and Implementations of Suffix Trees for Sequence and Structure Analysis in Bioinformatics

Asanilta Fahda 13513079
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
asanilta@students.itb.ac.id

The processing of sequences and structures is indispensable to the field of bioinformatics. Pattern detection or matching is often required for various purposes in the analysis of DNA, RNA, or protein, because it gives an insight as to how the sequences relate to one another, which can in turn lead to the progress of many biological discoveries and advancements. The suffix tree is a data structure that allows fast searching and comparison of strings, which can increase both speed and accuracy in certain processes. This paper gives a brief explanation of several examples of the multiple ways that the suffix tree can be used in making the process and analysis of bioinformatics data possible, and how they are implemented in doing so.

***Keywords*—suffix tree, bioinformatics, sequence, structure, computational biology**

I. Introduction

The mutual bond between biology and computers has significantly grown closer since the ever-growing advancement of algorithms. What used to take tremendous amounts of time in order to extract and process large amounts of biological data has now become increasingly more practical and feasible, thanks to bioinformatics.

Bioinformatics (sometimes also called *computational biology*, although distinctions between the two terms vary) is a field of science which focuses on developing computational methods for understanding the flood of data that biology is continuously providing. There are many areas which rely on these methods, with some of the major branches being Genomics, Proteomics, Computer-Aided Drug Design, Bio Data Bases & Data Mining, Molecular Phylogenetics, Microarray Informatics, and Systems Biology [1]. A few examples from the many existing algorithms currently applied in bioinformatics include:

- Baum-Welch algorithm (forward-backward algorithm) used to find the unknown parameters of a hidden Markov Model, applied for finding genes
- Needleman-Wunsch algorithm (dynamic programming algorithm) used for sequence-to-sequence alignment
- Kabsch algorithm used for calculating the optional rotation matrix that minimizes the root mean squared deviation between two paired sets of points, applied for comparing protein structures.

A big part, or perhaps the biggest part, that plays in bioinformatics is the study of biological sequences, i.e. DNA, RNA, and protein sequences. These sequences are usually treated as text or strings.

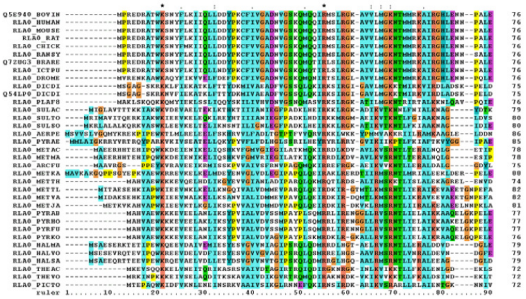

Fig 1.1. Multiple sequence alignment in bioinformatics [1]

There is a great deal of information stored in these sequences, but it would take a long while to search each of the sequences one by one. This is where the suffix tree comes in handy.

A suffix tree is one of the types of the tree data structure, which specialize in performing certain string operations quickly. It can easily find substrings, compare multiple strings together, detect or match patterns in the strings, and find plenty of other properties in each of the strings. However, recent progress in algorithms has also made way of implementing the suffix tree not only for string-like sequences but also more complex structures, especially protein. Thus, there are numerous ways that the suffix tree could be applicable in bioinformatics, some of which are further explained in this paper.

II. Related Theories

A. Tree

A tree is a connected undirected graph with no simple circuits [2]. In other words, an undirected graph is a tree if and only if there is a unique simple path between any two of its vertices. From Figure 2.1, we can conclude that G1

and G2 are trees. Meanwhile, G3 is not a tree because the nodes *e, b, a, d, e* form a circuit, and neither is G4 because it is not a connected graph.

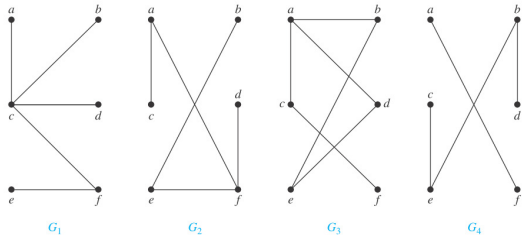

Fig 2.1. Examples of trees and graphs that are not trees[2]

A specific type of tree used in data structures are rooted trees, that is, a tree in which one vertex has been designated as the root and every edge is directed away from the root. As a convention, rooted trees may be drawn without direction as its direction is already implied.

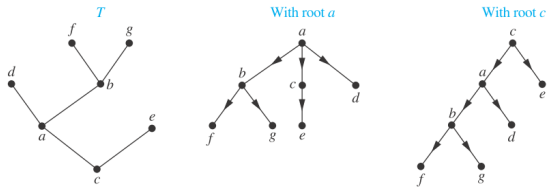

Fig 2.3. A tree and rooted trees formed by designated two different roots[2]

Some terminologies concerning trees:

- **Root:** the top node of a tree
- **Parent:** the node leading to a given node
- **Child:** each of the nodes one edge further from the given node
- **Siblings:** nodes with the same parent
- **Ancestors:** each of the nodes in the path from the root to a given node
- **Descendants:** each of the nodes with the given node as its ancestor
- **Leaf:** a node with no children
- **Internal vertices:** nodes with children
- **Subtree:** a tree created by taking a node from the original tree as the new root, consisting of all its descendants and the edges incident to these descendants
- **Forest:** a disjoint union of trees
- **Level:** 1 + the number of edges between the given node and the root
- **Height:** the number of edges from the longest path between the root and a leaf

B. Suffix Tree

One of the many types of tree data structures is the suffix tree, also called PAT tree, firstly introduced by Weiner [3]. It is defined as a compact representation of a trie corresponding to the suffixes of a given string where all nodes with one child are merged with their parents [4]. It is considered the fundamental data structure for combinatorial pattern matching because of its quick performance on several substring operations. An example of a suffix tree can be illustrated as following.

Let S denote a string of length n and $S[i,j]$ denote the substring of S from position i to j. The symbol $\$$ is concatenated to S. A suffix tree with n leaves has edges labeled by characters from the alphabet. The path from the root to the leaf j, denoted by $P(j)$, will have a path label which reads the suffix of $S[j,n]$ with a $\$$ sign.

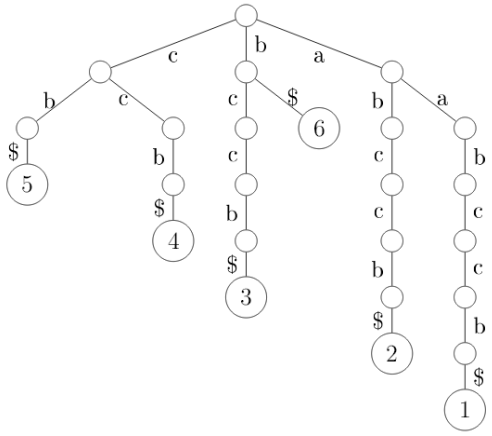

Fig 2.4. A suffix tree for the string 'aabccb'[5]

A generalized suffix tree is a suffix tree for a set of strings which represents all the prefixes in those strings. By taking m strings, $S_1,...,S_m$ with lengths $n_1,...,n_m$, respectively, and adding $\$_i$ *to* S_i *for* i-1,...,m, the generalized suffix tree will have leaves the amount of the total of the lengths of its strings, with each leaf labeled by the number j of the string and a number between 1 and n_j. The label of the path from the root to the leaf (i,j) represents the suffix $S_j[i,n_j]$ of the string j with a \$ sign.

Naive implementations of suffix tree constructions require a complexity of $O(n^2)$, where n indicates the length of the string. This made the use of suffix trees highly impractical for real-life applications in the past, due to the time and space requirements of its construction. In 1995, Ukkonen [6] proposed an a linear-time, online algorithm for constructing suffix trees which reduced the running time down to $O(n)$ or $O(n \log n)$ in its worst case. Afterwards, Farach [7] gave the first algorithm that is optimal for all alphabets, which has now become the basis for new algorithms.

Assuming that a suffix tree has been built for a string S with a length of n or that a generalised suffix tree has been built for the set of strings $D = \{S_1,S_2,...,S_k\}$ of total length $n = |n_1| + |n_2| + ... |n_k|$, the suffix tree will allow you to [8]:

- Check if a string P of length m is a substring in $O(m)$ time
- Find the first occurrence of the patterns *P1, ..., Pq* of the total length m as substrings in $O(m)$ time
- Find all z occurrences of the patterns *P1,...,Pq* of total length m as substrings in $O(m+z)$ time

- Search for a regular expression P in time expected sublinear in n
- Find each suffix of a pattern P, the length of the longest match between a prefix of $P[i...m]$ and a substring D in $\Theta(m)$ time
- Find the longest common substrings of the string Si and Sj in $\Theta(ni + nj)$ time
- Find all maximal pairs, maximal repeats or supermaximal repeats in $\Theta(n + z)$ time
- Find the Lempel-Ziv decomposition in $\Theta(n)$ time
- Find the longest repeated substrings in $\Theta(n)$ time
- Find the most frequently occurring substrings of a minimum length in $\Theta(n)$ time
- Find the shortest strings from $\sum$ that do not occur in D, in $O(n + z)$ time, if there are z such strings
- Find the shortest substrings occurring only once in $\Theta(n)$ time
- Find for each i, the shortest substrings of S_i not occurring elsewhere in D in $\Theta(n)$ time
- Find the lowest common ancester between nodes in $\Theta(n)$ time
- Find the longest common prefix between the suffixes $S_i[p..n_i]$ and $S_j[p..n_j]$ in $\Theta(1)$
- Search for pattern P of length m with at most k mismatches in $O(kn+z)$ time
- Find all z maximal palindromes in $\Theta(n)$, or $\Theta(gn)$ time if gaps of length g are allowed, ir $\Theta(kn)$ time if mismatches are allowed
- Find all z tandem repeats in $O(n \log n + z)$ and k-mismatch tandem repeats in $O(kn \log (n/k) + z)$
- Find the longest commons substrings to at least k strings in D for $k = 2,...,K$ in $\Theta(n)$ time
- Find the longest palindromic substring of a given string in linear time

C. Data in Bioinformatics

The data that we are dealing with in bioinformatics can be roughly grouped into four types, i.e. DNA Data, RNA Data, Protein Data, and Micro Array Image Data (traditional Digital Images) [1]. However, the use of suffix tree is currently only relevant to the first three.

- DNA

  DNA, or deoxyribonucleic acid, is a molecule which stores genetic information required by the cell to synthesize protein and replicate itself. It is made up of four nucleotide bases: Adenine (A), Guanine (G), Thymine (T), and Cytosine (C). Each base in a strand is paired with its complementary base on the opposite strand (A with T and vice-versa, G with C and vice-versa), with the two strands forming a double-helix structure. The information can be extracted and transformed into a string consisting of combinations of four letters, each letter corresponding to its appropriate nucleotide base, according to the sequence found inside the DNA.

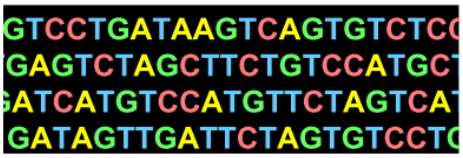

Fig 2.5. DNA sequence[1]

- RNA

  RNA, or ribonucleic acid, is a single-stranded molecule smaller than DNA which plays various roles in the synthesis of proteins (as mRNA, rRNA, or tRNA). The sequence of nucleotide bases in DNA is transferred to mRNA during transcription, in which Uracil binds to Adenine, thus replacing all occurrences of Thymine. The string depicting an RNA sequence also consists of combinations of four letters (A, U, G, and C).

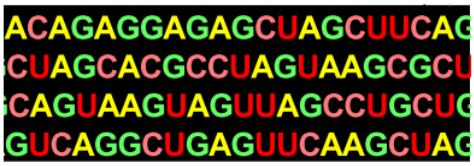

Fig 2.6. RNA sequence[1]

- Protein

  Protein sequences differ from the former two because of its wider variations. There are 20 different types of basic amino acids which can be found in a protein sequence, shown in the table below.

No.	Code	Name
1	A	Alanine
2	C	Cysteine
3	D	Aspartic Acid
4	E	Glutamic Acid
5	F	Phenylalanine
6	G	Glycine
7	H	Histidine
8	I	Isoleucine
9	K	Lysine
10	L	Leucine
11	M	Methionine
12	N	Asparagine
13	P	Proline
14	Q	Glutamine
15	R	Arginine
16	S	Serine
17	T	Threonine
18	V	Valine
19	W	Tryptophan
20	Y	Tyrosine

Table 1. Amino acid codes

Fig 2.7. Protein sequence[1]

III. Applications and Implementations of Suffix Trees

The applications of suffix trees and its variations in bioinformatics have grown wider and more diverse, along with the progress of the algorithms implementing it. Listed below are several examples of how the use of suffix trees could help the processes in bioinformatics.

A. Multiple Sequence Alignment

Sequence alignment, the most common task in bioinformatics, is a way of arranging DNA, RNA, or protein sequences for comparison in order to define functional, structural, or evolutionary relationships between the sequences. This is usually done by detecting certain similarities or patterns within the sequences. There exist already several systems implementing suffix trees for this function.

An example is MUMmer [9], a system for rapidly aligning entire genomes, which performs a maximal unique match (MUM) decomposition between the input genome sequences. The main biological features identified by the system are SNPs (Single Nucleotide Polymorphisms, which are single nucleotide 'mutations'), regions where the input sequences have diverged by more than an SNP, repeats that occur in one genome but not the other, and tandem repeats (regions of repeated DNA that occur when a pattern of two or more nucleotides is repeated directly adjacent to each other). It combines the use of suffix trees, the longest increasing subsequence (LIS), and Smith-Waterman alignment.

B. Recognizing DNA Contamination

Various laboratory processes could often cause the contamination of a DNA string by unwanted DNA or even protein. There are plenty of manners that contamination could occur. For example, it might come from the host DNA itself in the form of bacteria or yeast. It could also occur in the cloning process, originating from the cloning vectors used to incorporate the desired DNA in a host organism, or from PCR (polymerase chain reaction) primers. DNA contamination is a serious problem which could cause not only waste of time and effort but may also lead to wrong conclusions, which marks the importance of this checking process that can be computed as following:

Given a string S_1 (a newly isolated and sequenced DNA string) and a set of strings S (sources of possible contamination), find all substrings of S that occur in S_1 and are longer than some given length l.

This problem can be solved by the following process [8]:

1. *Build a generalized suffix tree T for $S \cup \{S_1\}$*
2. *Mark all internal nodes of T whose subtree contains leaves for both S_1 and some string of S*
3. *Traverse T; For any marked node v with string-depth $\geq l$ report L(v) as a suspicious substring*

The total time is $\Theta(m)$ where m is the total length of the strings.

C. Minimum Length Encoding

The Ziv-Lempel method is a universal lossless data compression algorithm used in various utilities, including text and image compression. Gusfield [8] presented an efficient variant of the algorithm:

For any position i in a string S of length m, define the substring $Prior_i$ to be the longest prefix of S[i..m] that also occurs as a substring of S[1..i – 1]. For any position i in S, define l_i as the length of $Prior_i$. For $l_i > 0$, define s_i as the starting position of the left-most copy of $Prior_i$.

The algorithm creates the compressed version of a string. For a small example, the string *abababababababababababababababab* can be described as *ab*(1,2)(1,4)(1,8)(1,16), which uses less symbols than the original. The algorithm's implementation using suffix trees allows it to run in $O(m)$ time.

Ziv-Lempel-like algorithms have been used in bioinformatics not for obtaining efficient storage but rather for measuring the "complexity" of strings, with significant strings having higher compression than random ones, studying the "relatedness" of two strings S_1 and S_2 by compressing S_2 using only the suffix tree for S_1 (or vice versa), and estimating the "entropy" of short strings to distinguish exons and introns in eukaryotic DNA.

D. Probe Selection

Microarrays are a common tool in measuring the expression levels of multiple genes simultaneously. To function, each array requires one specific oligonucleotide, called a probe, to attach to its surface. Ideally, this probe should be unique to bind only to the target sequence and avoid cross-hybridization. Kaderali and Schlied [10] presented an algorithm based on the combination of suffix trees and dynamic algorithm to design and select the proper signatures for these arrays. The algorithm builds a generalized suffix tree from the reverse complement of all the candidate probe sequences then rejects non-unique probes which form perfect duplexes with more than one target by identifying all common substrings.

The preselection of probes is done by taking account of probe length, unique probes, and probe melting temperature. A generalized suffix tree containing the complements of all target sequences is then generated and all infeasible probes are removed. Furthermore, the tree is then used to group probes with common prefixes together, which speeds up calculation.

E. Protein Indexing

Determining similar proteins used to be done by finding amino acid sequences that are similar to one another. However, it has been discovered that structurally-similar proteins also have similarities in functions even if they differ in their sequences. This is why protein structure analysis, including classification and prediction, requires an index structure that enables fast and scalable searching. There are several methods that are used for indexing protein structures. Reference [11] uses suffix trees in order to retrieve the maximal matches for all query segments, which are then chained to obtain alignments with database proteins. Similar proteins are selected by their alignment score against the query.

Other variations include the use of the probabilistic suffix tree and the geometric suffix tree. The probabilistic suffix tree stores conditional probabilities associated with subsequences, with two approaches for local prediction (multiple-domain prediction and best-domain prediction), both achieved in $O(Lm)$ time where L is the depth bound of the tree and m is the size of the query sequence [12]. Meanwhile, the geometric suffix tree is a new structure for indexing 3D structures of proteins [13]. While the edges in a traditional suffix tree represent substrings of text, the edges in this geometric suffix tree represent 3D substructures. The tree can be stored in $O(N)$ space where N is the sum of the lengths of the proteins in the database and constructed in $O(N^2)$ time.

F. RNA Structural Pattern Matching

As with protein structures, RNA structures also tend to point out the similarities of certain properties. Although RNA sequences can be compared easily in a string sense using traditional suffix trees, the use of structural suffix tree (s-suffix tree) can also determine their relationship in a structural sense, as proposed by Shibuya [14]. This is done by comparing s-strings, which are a generalization of parameterized strings (a parameterized string is a string over the union of two alphabets $\sum$ and $\prod$, where $\sum$ is an ordinary alphabet and $\prod$ is a set of parameters). The s-suffix tree can be constructed in $O(n(\log|\sum|+\log|\prod|))$ time. An example of an s-suffix tree is given below.

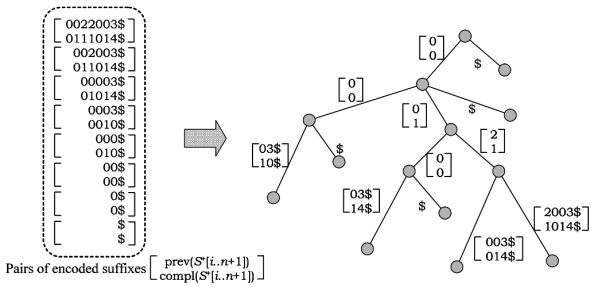

Fig 3.1. The s-suffix tree for an RNA string S = "AUAUCGU" [14]

The comparison between two structures is done by checking if the s-strings are an s-match. Two s-strings S and S' are considered an s-match if and only if $prev(S) = prev(S')$ and $compl(S) = compl(S')$. The computation for *prev* and *compl* are given in the algorithm below:

Let last_position(p) = 0 for all the parameters p at first. For $1 \le i \le n$, let prev(S)[i] = compl(S)[i] = S[i] if S[i] is an ordinary alphabet character, otherwise do the following.

1. *Set 0 to prev(S)[i] if last_position(prev(S)[i]) = 0. Otherwise set i – last_position(prev(S)[i]) to prev(S)[i].*
2. *Similarly, set 0 to compl(S)[i] if last_position(complement(prev(S)[i])) = 0. Otherwise set i – last_position(complement(prev(S)[i])) to compl(S)[i].*
3. *Set i to last_position(S[i]).*

Further algorithms are also given to increase the speed of this process.

IV. The Space Problem of Suffix Tree Implementations

The options for suffix tree implementations are vast, but the problem with suffix trees typically lies in their space requirements. The construction of a suffix tree typically calls for a complexity of O(m), which doesn't sound so terrible. However, it is the constant in front of the m that limits much of the real-life applications it could theoretically serve. This constant factor varies with each implementation.

MUMmer uses approximately 17 bytes for each basepair in the reference sequence [9]. It is able to build a whole genome alignment for an *E.coli* in 77 MB and plenty of other types of species. A human chromosome, however, could take up almost 4 GB, and the total space required for a whole human genome is approximated to be about 47 GB. Compressed suffix trees succeeded in bringing the size of the human genome down to 3GB, but their performance became 90 times slower than the regular suffix tree. The high space requirement has led to the frequent use of disk-based construction methods for very large sequences, and the development of algorithms focused on space-efficiency. An example of an algorithm that allows inputs larger than the main memory is B^2ST, which claims to build a suffix tree for 12 GB of real DNA sequences in 26 hours on a single machine with 2 GB of RAM [15]. A few problems still continue to exist but new algorithm designs also continue to appear as an attempt to solve them.

Some have decided to search for another alternative. A powerful substitute to the suffix tree is the suffix array, which stores all the suffixes of a string inside an array. In general, searching processes with suffix arrays are not performed as fast as with suffix trees, but they do serve an advantage in lower memory consumption. Enhanced suffix arrays [16], on the other hand, are even said to also perform faster than suffix trees, which basically gives them the best of both worlds. Unfortunately, not so many programs for ESA construction or specific algorithms for bioinformatics implementing ESA are currently available, but several have already started to arise in the recent years.

V. CONCLUSION

In conclusion, suffix trees are a helpful data structure to use for comparisons and pattern detection, which makes it a versatile tool that can be applied to almost any kind of process in bioinformatics. It can determine the similarities between sequences and structures, find certain properties in a sequence, and can be combined with other algorithms to create new specific algorithms that are more efficient. On the other hand, the suffix tree still suffers from a major drawback in its memory requirement and must therefore be either improved or replaced when used for very large inputs.

REFERENCES

[1] Achuthsankar S. Nair, "Computational Biology & Bioinformatics: A Gentle Overview". *Communications of the Computer Society* of *India,* January 2007.

[2] Kenneth H. Rosen, *Discrete Mathematics and Its Applications 7th ed.,* McGraw-Hill, 2012.

[3] P. Weiner, "Linear pattern matching algorithm," *Proc.14 IEEE Symposium on Switching and Automata Theory*, 1973.

[4] Stefan Edelkamp, "suffix tree", in *Dictionary of Algorithms and Data Structures* [online]. Available: http://www.nist.gov/dads/HTML/suffixtree.html. [Accessed: Feb. 10, 2014].

[5] Bálint Márk Vásárhelyi, "Suffix Trees and Their Applications", MSc Thesis, Faculty of Science, Eötvös Loránd University, Budapest, 2013.

[6] E. Ukkonen, "On-line construction of suffix trees," *Algorithmica,* vol. 14, pp. 249-260, 1995.

[7] M. Farach, "Optimal suffix tree construction with large alphabets," *Proceedings of the 38th Annual Symposium on Foundations of Computer Science,* 1997.

[8] D. Gusfield, *Algorithms on Strings, Trees and Sequences,* Cambridge University Press, 1997.

[9] *The MUMmer Home Page* [online]. Available: http://mummer.sourceforge.net [Accessed: Feb. 10, 2014].

[10] Lars Kaderali and Alexander Schliep, "Selecting signature oligonucleotides to identify organisms using DNA arrays", *Bioinformatics,* vol. 18, pp 1340-1349, 2002.

[11] Feng Gao and Mohammed J. Zaki, "PSIST: Indexing Protein Structures using Suffix Trees," *Proc. IEEE Computational Systems Bioinformatics Conference (CSB),* 2005.

[12] Zhaohui Sun, "Local Prediction Approach for Protein Classification Using Probabilistic Suffix Trees," *Proc of The Second Asia-Pacific Bioinformatics Conference*, vol. 29, 2004.

[13] T. Shibuya, "Geometric suffix tree: A new index structure for protein 3D structures", *Proc. Combinatorial Pattern Matching, LNCS 4009*, 2006

[14] T. Shibuya, "Generalization of a suffix tree for RNA structural pattern matching", *Algorithmica*, vol. 39 no. 1, 2004.

[15] M. Barsky, U. Stege, A. Thomo, and C. Upton, "Suffix trees for very large genomic sequences," *Proceedings of the 18th ACM conference on Information and knowledge management*, ser. CIKM '09, 2009.

[16] M. I. Abouelhoda, S. Kurtz, et al. "Replacing suffix trees with enhanced suffix arrays," *Journal of Discrete Algorithms*, 2004.

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, 11 Desember 2014

Asanilta Fahda 13513079