Finding Similarity of Two Organisms' DNAs Using Longest Common Subsequence

Dicky Novanto 13515134[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13515134@std.stei.itb.ac.id

***Abstract*—Biology subject, particularly in genetics, are continuously growing and developing fast. The biologists taking part in this subject are constantly discovering new species. Each of species has its unique genetic code that differs them from the other species. But, knowing the information about the genetic code of a certain species is not sufficient to classify each new species. Finding the relation of each new species with the other species based on their genetic code is absolutely a tedious job. Therefore, they absolutely need a program to find the relation of each species, particularly a program to find the longest common subsequence of two genetic codes. In this paper, it will discuss about basic theorem of dynamic programming, longest common subsequence algorithm, and experiment to find the longest common subsequence of a certain genetic codes and the other organism's genetic codes, along with its analysis.**

Keywords—genetic codes, dynamic programming, longest common subsequence, tables.

I. Introduction

The importance of biological discoveries about how closely related two organisms using DNA as its indicator is one of the very important subjects in Biology. Through the discoveries, the biologists can find many fresh knowledge that is beneficial in human life, for instance, the fact that the discoveries the similarity of DNA from human and one of primates, chimpanzee, that is reaching 70 % can be concluded that human and chimpanzees are closely related and thus believed that the 2 species came from a certain species that is common ancestor of humans and chimpanzees. This fact surely encourages biologists to find the common ancestor of the two species and can lead to find a massive discovery about the common ancestor.

Furthermore, after determining the most closely related organism, it can continue to classification of each new organism in order to include new organism to a certain group which has similarities with the new organism. There are still millions of species to discover in earth and the longest common subsequence algorithm is a helpful tool to find the similarities between 2 genetic codes of 2 organisms. With the algorithm, it will able to show how similar are the 2 genetic codes. Moreover, the new species can be classified to a group which has the most similarities with it.

II. Basic Theorem of Dynamic Programming

Dynamic Programming is a general algorithm design technique for optimizing a solution of a multistage problem. Dynamic Programming technique can be used for solving problem that has overlapping sub-problems [1]. Overlapping sub-problems is a smaller part of a whole problem and the solution of the sub-problems can be used more than once to construct a solution for a bigger sub-problem. Hence, it is better to solve the overlapping sub-problems once and the solution can be used to construct a solution for a bigger sub-problem rather than repeatedly searching the solution for the sub-problems over and over again. That is obviously the main idea of dynamic programming technique, to prevent computing the same thing twice or more. Computation for the solutions of sub-problems and certainly, the main solution, is using tables. Using tables to record the solution is also called memoization.

According to Richard Bellman, the inventor of Dynamic Programming, there is principle of dynamic programming in order to obtain an optimal solution. It is called principle of optimality. It has meaning that a solution of a certain problem is optimal, then the solutions of all sub-problems is optimal. When we are reconstructing a solution of a certain state, it will use the previous optimal solution of a sub-problems without searching the solution of the sub-problem from the start state again [2].

There are several characteristics of a problem to be considered as solvable using dynamic programming technique [2]:

1. The problem can be divided into sub-problems and for each of them can be found only one solution.
2. A state is constructed from some states (or only one states) which is related to the state.
3. Solution of a certain state can be used for the next state.
4. Cost of state is increasing steadily as the step of finding solutions is increasing.
5. Cost of a certain state depends on cost of past steps and cost in the current step.
6. The solution of a certain step is independent from the solution of past steps.

7. There exists a recurrence relation of a certain step of which the state's solution is optimal and can construct next state's optimal solution.

8. Principle of optimality holds.

There are two approach of finding optimal solution using dynamic programming technique, that is top-down approach and bottom up approach. Top-down approach usually using the recurrence relation, basis of relation, and memoization technique. Memoization is a step to write an optimal solution of a certain state so that when a certain state's solution is required to form the next solution of a state, the written solution can be directly used without computing the same solution again. This approach only computes some needed sub-problems and this could be faster as we don't need to write all solutions of sub-problems. Nevertheless, the top-down approach can be slower if many sub-problems are visited due to function call overhead [3].

On the other hand, bottom up approach is a technique to fill up the solution of all the sub-problems in a certain problem and each state is only computed once. This technique has a drawback; it is often that the solution of a sub-problem is not used to form a solution of a certain problem, thus it requires more memory to save the solutions. But sometimes, the bottom up approach is faster than the top-down approach as the top-down creates function call overhead from recursive calls and when many sub-problems are revisited to construct a greater sub-problem's solution [3]. Hence, it is important to use a correct technique to solve dynamic programming problems.

These are general steps to use dynamic programming algorithm [2]:

1. Create optimal solution structure.
2. Define recurrence relation to find optimal solution.
3. Use one of two dynamic programming technique: top-down or bottom up approach.
4. Construct optimal solution.

III. Longest Common Subsequence Algorithm

Longest Common Subsequence problem is a problem to find the maximum length of common subsequence (can be a list of characters) of 2 strings. For this time, we limit the discussion of longest common subsequence for only 2 strings with no spaces within the strings (since there can be finding longest common subsequence for 2 sentences and finding the length of longest common subsequence, that is words). For example, if I have strings:

A: DICKYNOVANTOINFORMATIKA

B: STRATEGIALGORITMA

Therefore, the longest common subsequence of both string is: ATIORMA. Because the longest common subsequence of the 2 strings appears in the made bold characters below:

A: DICKYNOV**ANTO**INF**ORMA**TIKA

B: STR**ATE**G**I**ALG**OR**IT**MA.**

In this case there are no subsequence of the 2 strings that is longer than "ATIORMA".

In bottom up, firstly, we need to define a matrix of integer with size ((length of string 1) + 1) times ((length of string 2) + 1) to save the solution of each state. We need the addition 1 of each string length because we need to define the base cases of longest common subsequence solution and to define it, we need the added length. The base cases of this solution is when there is one empty string of the 2 strings, the longest common subsequence length of both string is obviously 0, because there is no same subsequence of two strings. The general formula of the length longest common subsequence considered two cases:

1. If the two characters of certain indices of two strings that is being checked is same, then the new length of longest common subsequence is the previous length of longest common subsequence (excluding the last characters checked on both strings) added by 1.

2. If the two characters of certain indices of two strings checked is different, then the current length of longest common subsequence is the maximum of longest common subsequence length if the last character of string 1 checked is removed and longest common subsequence length if the last character of string 2 checked is removed. It means that it is finding the maximum of solution from previous length produced.

Here is the pseudocode of finding the length of longest common subsequence using bottom up technique (n is length of string 1 and m is the length of string 2, and a[i][j] is the solution when the state is i and j):

```
for i=0 to n {
  a[i][0] = 0;//base case
}
for j=0 to m {
  a[0][j] = 0;//base case
}
for i=1 to n {
  for j=1 to m {
    if(word1[i-1] = word2[j-1]) then
      a[i][j] = a[i-1][j-1] + 1; //same
characters
    } else {
      a[i][j] = max (a[i-1][j], a[i][j-1]);
      //different characters
    }
  }
}
print a[n][m]; //printing the answer
```

The top-down formula is almost the same as the bottom up formula. The only difference is that top-down approach utilizes recursive function and of course a table to save solution of each state.

Here is the LCS function with top-down approach returning the length of longest common subsequence of the two strings:

```
   function lcs(int n, int m) → integer {
     if n or m is 0 then return 0;
     else if (a[n][m] is not empty) then return
a[n][m];
     else if(word1[n-1] == word2[m-1]) then
return a[n][m] = lcs(n-1, m-1) + 1;
//memoization
     else {
        return a[n][m] = max(lcs(n-1, m),
lcs(n,m-1)); //memoization
     }
   }
```

Notice that there is memoization in table a and this is the most important part in speeding up the recursive program. Stated in the code that when a[n][m] is not empty, which means that the solution in state n and m has already computed and written in the table a, the code will just return the solution written in a[n][m] without searching the solution in this state again.

Both complexity in top-down and bottom up approach is clearly O(n.m). The complexity is obvious in the implementation of bottom-up approach since there are 2 nested loops, loop (m times for n times). If we implemented a naïve recursive algorithm (without memoization), than we can see that there are overlapping sub-problems that is computed several times and this is time consuming as the time complexity in worst case is O(2^n). This could happen if there are no matching between 2 strings and thus will execute max (lcs(n-1, m), lcs(n, m-1)), which means that it keeps dividing into 2 new recursion function until n or m is 0. This is the reason why the worst case complexity is O(2^n).

Here will be explained how the bottom up approach works to solve the longest common subsequence problem. We take an example of two strings, that is:

Word1: ACAATCC

Word2: AGCATGC

Here is the base case of finding length of longest common subsequence from the two strings:

	Word2		**A**	**G**	**C**	**A**	**T**	**G**	**C**
Word1	**Index**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
	0	0	0	0	0	0	0	0	0
A	**1**	0							
C	**2**	0							
A	**3**	0							
A	**4**	0							
T	**5**	0							
C	**6**	0							
C	**7**	0							

Table 1: The base case tables formed.

Index 0 indicates the case if a certain string is an empty string. It is obvious that the length of longest common subsequence of empty string and a (non) empty string is 0 since there are no subsequence that is common between two strings.

	Word2		**A**	**G**	**C**	**A**	**T**	**G**	**C**
Word1	**Index**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
	0	0	0	0	0	0	0	0	0
A	**1**	0	1	1	1	1	1	1	1
C	**2**	0							
A	**3**	0							
A	**4**	0							
T	**5**	0							
C	**6**	0							
C	**7**	0							

Table 2: Result of longest common subsequence involving the first characters of word1.

We can see that in index (1,1) (order of index: (row, column)), the characters of both strings is same, so using the solution of index (0,0) and then added by 1 as ‘A’ has to be added to be the part of longest common subsequence and the longest common subsequence of string “A” and “A” is ‘A’ itself. Then the index (1,2) is 1 because of gaining the maximum value from a[1][1] and a[0][2] and the length of longest common subsequence of string “AG” and “A” is 1, and the common subsequence is surely ‘A’. In the final column, a[1][7] is still 1 as we can see that the longest common subsequence of string “A” and “AGCATGC” is ‘A’ and ‘A’ is only 1 character, hence the answer is 1.

	Word2		**A**	**G**	**C**	**A**	**T**	**G**	**C**
Word1	**Index**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
	0	0	0	0	0	0	0	0	0
A	**1**	0	1	1	1	1	1	1	1
C	**2**	0	1	1	2	2	2	2	2
A	**3**	0	1	1	2	3	3	3	3
A	**4**	0							
T	**5**	0							
C	**6**	0							
C	**7**	0							

Table 3: Computing length of longest common subsequence until the 4th row.

Table 3 shows that there are changes in table content compared to table 2. The value a[2][3] has changed to 2 due to the same characters checked, that is ‘C’, so the longest common subsequence of string “AC” and “AGC” is “AC” that has length of 2. This also happens in index (3,4) where the longest common subsequence of string “ACA” and “AGCA” is

"ACA" with length of 3. This continuously happens until the last row and column of table.

	Word2		A	G	C	A	T	G	C
Word1	**Index**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
	0	0	0	0	0	0	0	0	0
A	**1**	0	1	1	1	1	1	1	1
C	**2**	0	1	1	2	2	2	2	2
A	**3**	0	1	1	2	3	3	3	3
A	**4**	0	1	1	2	3	3	3	3
T	**5**	0	1	1	2	3	4	4	4
C	**6**	0	1	1	2	3	4	4	5
C	**7**	0	1	1	2	3	4	4	5

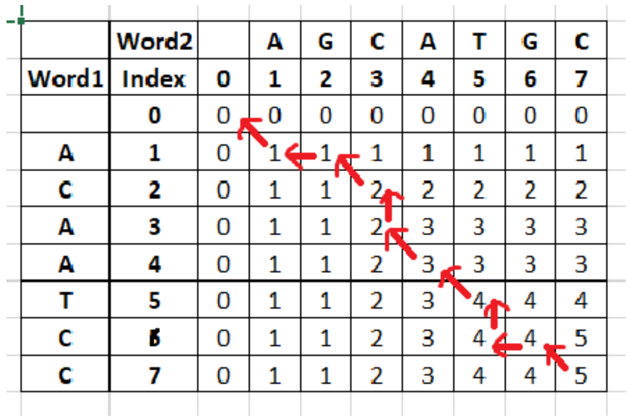
Table 4: Finished computing length of longest common subsequence.

From table 4, we can conclude that the length of longest common subsequence of string "ACAATCC" and "AGCATGC" is 5. This value can be picked up from the last index of row and column, that is a[7][7].

After computing the solution and formed a table containing solution of the solution and its sub-problems, the last step needed is to find the longest common subsequence itself. There is algorithm to find it. Here is the pseudocode to find the longest common subsequence of 2 strings.

```
idx = a[m][n];
i = m; j = n;
while(i and j are not 0) {
  if(word1[i-1]==word2[j-1]) then //same char
    idx = idx - 1; i = i - 1; j = j - 1;
    ans[idx] = word1[i-1];
  else {
    if(a[i-1][j] > a[i][j-1]) then i = i - 1;
    else {
      j = j - 1;
    }
  }
}
for i = 0 to a[m][n] {
  print ans[i];
}
```

The main idea of finding solution is that we are tracing back from the solution starting from the last index of row and column all the way down until index (0,0). Tracing back means that a certain solution must be clear where it come from, and the characters that are found same on a certain index written down as the solution.



	Word2		A	G	C	A	T	G	C
Word1	**Index**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
	0	0	0	0	0	0	0	0	0
A	**1**	0	1	1	1	1	1	1	1
C	**2**	0	1	1	2	2	2	2	2
A	**3**	0	1	1	2	3	3	3	3
A	**4**	0	1	1	2	3	3	3	3
T	**5**	0	1	1	2	3	4	4	4
C	**6**	0	1	1	2	3	4	4	5
C	**7**	0	1	1	2	3	4	4	5

Figure 1: Tracing back from table to get the longest common ancestor

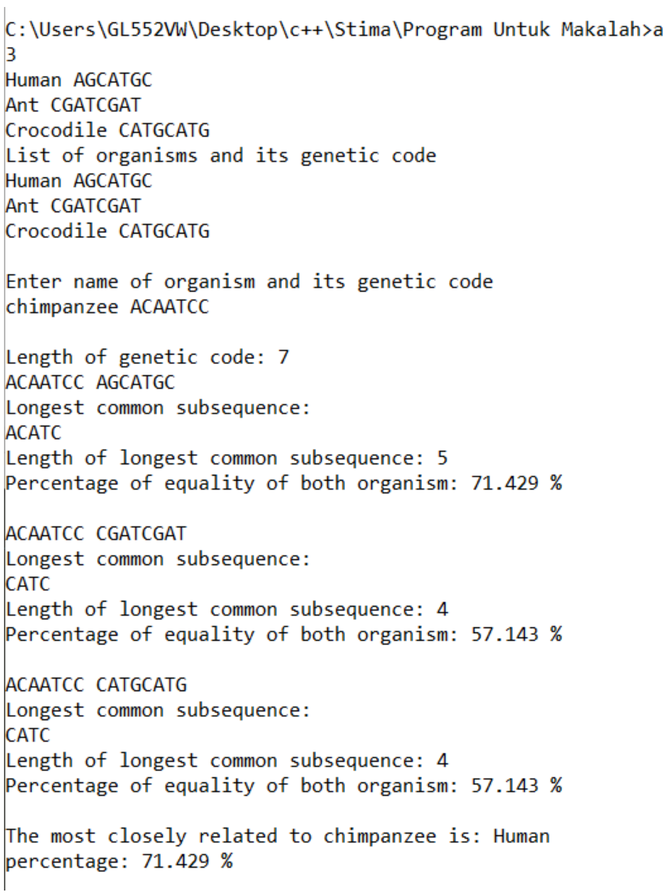
From figure 1, we can get the information that we tracing back from the last index to the first index of tables. In index (7,7), the character from both string at the index is same, that is 'C', so 'C' is written as a solution, and can subtract all the indices. In index (6,6), we can see that character 'C' is not the same as 'G', so we need to find the maximum value of a[i-1][j] and a[i][j-1], we can pick one of them (shown in the figure, it goes to the left). In index (6,5) the characters do not match, so we are going upwards because the maximum value of 3 and 4 is 4. Then this time, the character match (character T) and we save the T character as a solution. Just do this along the path and we got the longest common subsequence: "ACATC".

Finding the longest common subsequence of two strings have the complexity of O(n + m). It is because it traverses down from the last index to the index (0,0) in the solution table and this requires the n + m times in the worst case make the way to the first index.

IV. Application and Experiments with Analysis

Longest common subsequence algorithm has many applications in life. One of them is to determine the similarities between 2 organisms. Suppose that we have a set of organisms along with its genetic code sample, and we have a certain genetic code of an organism that has not been in the dictionary of organisms, and we have curiosity to know what is the closest related from the new organism to the organism that has been recorded. The closest related organisms are the one that longest common subsequence of genetic codes of both organisms, which means has the biggest percentage of genetic code similarity. The percentage of similarity can be obtained by dividing the length of longest common subsequence and the length of genetic code times 100 %.

Here will be shown the example of finding the similarities of two organisms, that is chimpanzee and spider and the other organism in the dictionary (human, ant, and crocodile). The genetic codes in the figures below are just an example, not the real genetic codes of each species.



```
C:\Users\GL552VW\Desktop\c++\Stima\Program Untuk Makalah>a
3
Human AGCATGC
Ant CGATCGAT
Crocodile CATGCATG
List of organisms and its genetic code
Human AGCATGC
Ant CGATCGAT
Crocodile CATGCATG

Enter name of organism and its genetic code
chimpanzee ACAATCC

Length of genetic code: 7
ACAATCC AGCATGC
Longest common subsequence:
ACATC
Length of longest common subsequence: 5
Percentage of equality of both organism: 71.429 %

ACAATCC CGATCGAT
Longest common subsequence:
CATC
Length of longest common subsequence: 4
Percentage of equality of both organism: 57.143 %

ACAATCC CATGCATG
Longest common subsequence:
CATC
Length of longest common subsequence: 4
Percentage of equality of both organism: 57.143 %

The most closely related to chimpanzee is: Human
percentage: 71.429 %
```

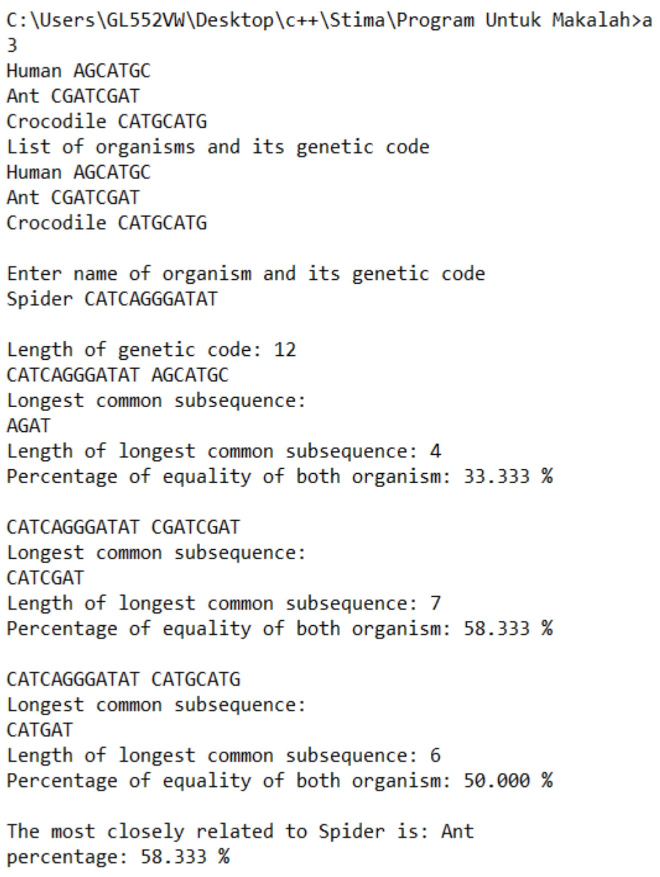
Figure 2: Longest common subsequence of chimpanzee's genetic code and the rest of genetic codes.

From figure 2, we get the information that the most closely related to chimpanzee is human with the percentage of similarity is 71.429 %. This comes from the length of longest common subsequence of genetic code of human: "AGCATGC" and genetic code of chimpanzee: "ACAATCC" is "ACATC" with the length of 5 and (5/7)* 100% = 71.429 %. This percentage is obviously higher than the other percentage of just 57.143 % similarity between chimpanzee's genetic code and ant and crocodile's genetic code.

Here are two genetic codes of human and chimpanzee with its longest common subsequence characters made bold:

Human	: **A**G**CAT**G**C**
Chimpanzee	: **ACA**A**TC**C

This example has already been discussed in the previous section (Longest common subsequence algorithm).



```
C:\Users\GL552VW\Desktop\c++\Stima\Program Untuk Makalah>a
3
Human AGCATGC
Ant CGATCGAT
Crocodile CATGCATG
List of organisms and its genetic code
Human AGCATGC
Ant CGATCGAT
Crocodile CATGCATG

Enter name of organism and its genetic code
Spider CATCAGGGATAT

Length of genetic code: 12
CATCAGGGATAT AGCATGC
Longest common subsequence:
AGAT
Length of longest common subsequence: 4
Percentage of equality of both organism: 33.333 %

CATCAGGGATAT CGATCGAT
Longest common subsequence:
CATCGAT
Length of longest common subsequence: 7
Percentage of equality of both organism: 58.333 %

CATCAGGGATAT CATGCATG
Longest common subsequence:
CATGAT
Length of longest common subsequence: 6
Percentage of equality of both organism: 50.000 %

The most closely related to Spider is: Ant
percentage: 58.333 %
```

Figure 3: Experiment with input the new organism is Spider and its genetic code.

We can conclude from figure 3, that the mostly related organism to spider is ant with the percentage of similarity is 58.333 %. The value is the highest among the 3 percentage. This value is obtained by dividing the length of longest common subsequence and the length of the spider's genetic code, that is (7 / 12) * 100% = 58.333 %.

Here are the two words, code genetics of spider and ant, with the longest common subsequence characters made bold:

Spider	: **CATC**A**G**GG**AT**AT
Ant	: **CGATCGAT**

V. Conclusion

In conclusion, dynamic programming technique is a tool to solve some problems that has overlapping sub-problems within in order to prevent solution re-calculation on a same sub-problem. Dynamic programming technique can be divided into 2 approaches, that is top-down and bottom up approaches. While top-down utilize recurrence relation and memoization once the solution is found, bottom up approach usually searching the solution of all the sub-problems.

One of dynamic programming algorithm is longest common subsequence. Longest common subsequence algorithm is an algorithm to find the maximum length of common subsequence of 2 strings. The algorithm is very useful in real life, particularly in Biology subject. In Biology, longest common subsequence can be used to determine how close an organism to the other organism using their genetic codes. The more percentage of similarity of both codes, the more similar the new organism checked to organism in the data.

Finding length of longest common subsequence of 2 strings with naïve recursive function (top-down approach without memoization) has the time complexity O(2^n). It is much slower compared to both top-down and bottom up approach owing to re-computation in naïve solution.

ACKNOWLEDGMENT

Firstly, the author would like to thank God for His blessing given to the author, so that the author has power and strength to make this paper wholeheartedly. Without his blessing, the author will not finish this paper well.

Secondly, the author would give thanks to Dr. Ir. Rinaldi Munir, MT; Dr. Nur Ulfa Maulidevi, ST., M.Sc; and Dr. Masayu Leylia Khodra, ST., MT for guiding the author and the students of IF2211: Strategi Algoritma course. Without their willing to share the knowledge from this course, the author and the students will not be able to make this paper.

Lastly, the author also thanks the author's parents and author's friends for supporting the author to attend the course well so that the author gained much knowledge from the course, hence being able to make this paper well also.

REFERENCES

[1] Levitin, Anany. *Introduction to The Design & Analysis of Algorithms*, 3rd ed. Villanova University, Pennsylvania: 2012, Pearson.

[2] Munir, Rinaldi. *Slide of IF2211: Strategi Algoritma, Program Dinamis* (2015).

[3] Halim, Steven and Felix Halim. Competitive Programming 3: Increasing the Lower Bound of Programming Contests. Singapore: 2013, National University of Singapore.

[4] https://www.ics.uci.edu/~eppstein/161/960229.html, accessed on May 16, 2017 at 02.24 A.M.

[5] http://www.livescience.com/topics/newfound-species, accessed on May 17, 2017 at 01.13 P.M.

DECLARATION

I hereby certify that this paper is my own writing, neither a copy nor from another paper, and not an act of plagiarism.

Bandung, May 17, 2017

[illegible]

Dicky Novanto / 13515134