SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

FALL 2025

Common Subsequences in Genome Sequences: An Algorithm Using Binary Search Trees

By Jocelyn Wang

AUTHOR BIOGRAPHY

Jocelyn is a junior at Minnetonka High School in Minnesota. She is interested in applied mathematics, data science, and engineering and wishes to conduct further research and study these fields in the future.

ABSTRACT

Deoxyribonucleic acid (DNA), plays a crucial role in the field of biology. Identifying similar DNA sequences across species is critical for understanding genomics and evolutionary biology, due to DNA's role as the universal genetic code. The identification of similar sequences serves many purposes, some of which include the tracing of evolutionary relationships, studying hereditary diseases, and supporting forensic analysis amongst different species. In this paper, we focus on the development of an algorithm that successfully outputs the shared subsequences between two sequences and a specified length. Our primary goal is to successfully and efficiently run this algorithm, that produces quick and reliable results. To achieve this goal, we discuss and highlight the usage of a Binary Search Tree (BST), a data structure, due to its ability to efficiently organize and search data, and its ability to swiftly retrieve data. Providing three inputs, which consist of two sequences and one specified length, the algorithm will successfully output the shared values in both sequences. The outcomes of this research demonstrate the usefulness of algorithms and different data structures in different practical applications, such as advancements in efficient data processing and technology. Ultimately, this research shows the potential and value of computational approaches in addressing real-world challenges related to genetic research.

Keywords: DNA, Genetic Traits, Binary Search Tree, Sequence Analysis, Shared Subsequences, Genomes, Algorithm Development, Optimization, Data Structures

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

FALL 2025

INTRODUCTION

DNA and Nucleotides

DNA, or deoxyribonucleic acid, is present in nearly all cells of living organisms. The presence of DNA not only determines the genetic abilities and characteristics of an organism, but it also can predict existing relations among different species and organisms that would not be confirmed without the knowledge of its genetic makeup. Genome, or DNA sequencing, has become a more prominent topic in molecular and evolutionary biology in the past few decades due to the availability of new technology. Specifically, finding biologically functional, or congruent sections of DNA, has been particularly useful to scientists due to its functions in predicting genetic behavior in organisms with similar strands of DNA.

DNA is made up of nucleotides, specifically consisting of a sugar molecule (deoxyribose), a phosphate group, and one of four nitrogenous bases: adenine (A), cytosine (C), guanine (G), and thymine (T) (National Cancer Institute, 2011). These nucleotides form the commonly seen double helix structure of DNA. With these four nitrogenous bases (A, C, G, and T), similarities and congruences in DNA can be found through simple observation. But with the average length of the human genome being 3 billion base pairs (National Committee on Mapping and Sequencing the Human Genome, 1988), it becomes extremely difficult to interpret common sections of nucleotides or bases in DNA. Combined with the discovery of new organisms, it has become crucial to discover the relation between different strands of DNA in those organisms and how they relate to each other (Brown, 2018). This development can lead to an efficient understanding of new organisms in fields of molecular and evolutionary biology.

Binary Search Trees

In this paper, we use a data structure, commonly used in computer science, called the Binary Search Tree (BST). This data structure has multiple uses, some of which include spell checking and organizing web pages in search engines. Binary search trees help us organize and store the data to develop an algorithm to find the similar sections present in the given strings. There are other algorithms not involving the use of BSTs that can accomplish this task in a similar manner, but the use of BSTs creates a more intuitive algorithm that can be applied to different situations. As the sequences get longer and the substring lengths increase, the algorithm becomes more complex, which causes a slower algorithmic computation and production of the results. For this reason, we choose to use BSTs in this algorithm. Binary search trees create an efficient way to add a new substring to the tree and it leads to quicker searches in the tree to find similar substrings, as opposed to using the naive approach of comparing each string with every other string. In addition, the binary search tree algorithm, in comparison with other data structures, is effective in this scenario due to its ability to insert new substrings into the tree while keeping the tree sorted. The hierarchical format of the tree introduces a quicker sorting method such that each branch can be searched much quicker than the naive approach.

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

FALL 2025

In this paper, we use the BST to search for shared subsequences of nucleotides, of length k, given sequences of nucleotides. A binary tree is when every node in the left subtree is smaller than the root, and every node in the right subtree is greater than the root (Nelson-Fromm, 2023). A binary search tree is a special version of a binary tree that allows for more efficient searching and recovering. Binary search trees can allow the user to quickly sort lists and determine the order of the numbers in the list. We use binary search trees in this paper to find common subsequences of letters, through a similar approach. In a binary search tree, the depth increases as more nodes are added, which leads to the need to rebalance the tree to decrease the depth. In this paper, we do not discuss the rebalancing of the tree and continue without the rebalancing. This does not interfere with how successful the algorithm is in finding common sequences, but it does slightly hinder the length of time the algorithm will take to run.

METHODS

To fully understand the problem and the algorithm by which it is resolved, the methods are broken down into subsections as follows. First, the manner in which the dataset was found is explained, after which the reasons for choosing the binary search tree are clarified. Finally, the approach to the problem is outlined, along with the identification of specific parts of the binary search tree that are highlighted by the algorithm.

Dataset

To obtain data to test this algorithm, we used a "Random DNA Sequence Generator" (Bioinformatics.org, 2023). This paper uses a random number generator to determine the length of the first and second sequences to randomize the data we are inputting (Mads Haahr, 2019).

Two-Step Approach

The two-step approach that is used in this algorithm implements two separate steps to sort the sequences and create a final output. First, the algorithm stores the substrings of the first sequence. The algorithm runs through all substrings of the given length in the sequence and stores them into the tree in a hierarchical manner. It then extracts the substrings of a given length in the second sequence, and checks whether that substring is present in the first sequence, following the hierarchical format of the binary search tree, going from top to bottom. If a match is found, the algorithm stores the match until the final output is called.

ALGORITHM EXPLANATION

The algorithm begins by taking all possible substrings of a given length in the first sequence. For example, if our first sequence were to be "CGTTAGAG", and the length is 4, the substrings would be "CGTT", "GTTA", "TTAG", "TAGA", and "AGAG". These substrings are then added to the binary search tree (Fig. 1) which makes it possible for them to be accessed later in the algorithm. Next, we build the tree based on the substrings that we obtained from the first sequence. When a new substring is inserted into the tree, it is

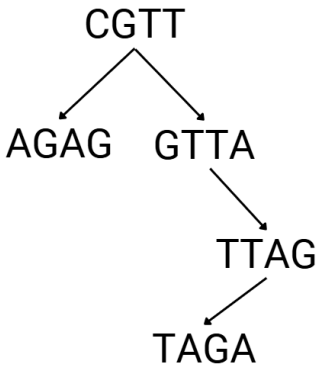
SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

FALL 2025

examined alphabetically. If a new substring is sorted earlier alphabetically, it is put into the left subtree. If it is sorted later alphabetically, it is inserted into the right subtree. This becomes a recursive process until an empty position on the tree is found (i.e., no child nodes), and thus the new substring from the first sequence can be stored. Next, the algorithm uses a similar method to extract all possible substrings of a given length from the second sequence. For each substring, the algorithm begins the recursive process to search through the tree, traversing alphabetically until a substring is found, or a new node is achieved, meaning the substring is not present in the tree. An advantage of using the BST enables the algorithm to eliminate half of the remaining strings in each possible step. Finally, after all possible substrings of the second sequence are checked against the first one, the algorithm returns a set of common substrings as the final result.

Figure 1
Example of binary search tree for a DNA sequence



Note. A binary search tree constructed from the string "CGTTAGAG" using substring length k = 4. Nodes represent unique subsequences of length 4, demonstrating efficacy in sequence comparison

RESULTS

After successfully developing our algorithm, we were able to process a variety of different sequences. We first ran this algorithm by using two strings consisting of 100 characters. Our algorithm successfully ran and produced a result in under 0.5 seconds.

The complexity of this algorithm varies based on whether we choose to rebalance the tree. If we choose not to rebalance the tree, the complexity changes, which can lead to a faster computation, given larger sequences of letters. It was expected that this algorithm would successfully find the common subsequences between two sequences provided a length. The length of the computation and the output of the algorithm proved this to be true in the trial period of this algorithm. In addition, the strings of letters can easily be substituted for actual sentences, including spaces and numbers, and would run the same exact way. The longer the sequences were, the longer the time it took to run the algorithm, which also depended on the given length of subsequences. The

SCHOLARLY REVIEW

SCHOLARLY REVIEW JOURNAL

E-ISSN: 2996-8380
Published by Leadership & Innovation Lab

FALL 2025

algorithm successfully ran in an efficient manner in under one second with strings of lengths up to 70,000 characters.

Tying this back to our original problem, with an algorithm like this, we now have the ability to swiftly examine two sequences of DNA and successfully compare them in terms of the common subsequences found between them. This algorithm can assist in creating advancements in research due to its ability of similar sequences in DNA to show potential similarities between species or even in relating to two different species. In our algorithm, we programmed the algorithm in a specific way, but there are alternate approaches to program the algorithm leading to variations in complexity and processing times based on the lengths of the sequences.

DISCUSSION

In this study, we developed and tested an algorithm that uses binary search trees to find common subsequences of a specified length between two DNA sequences. Our results showed that this is an effective method and can handle sequences of longer lengths, producing quick and efficient results even for strings up to 70,000 characters. This is an improvement in comparison to other approaches such as the dynamic programming method for the longest common subsequence (LCS), which is known to have a higher computational time, especially when sequence lengths increase (Hirschberg, 1975). While this program is more simple to understand, the time complexity makes it an impractical approach for larger datasets like those found in genomics.

Compared to other modern methods, such as algorithms that use k-mer frequencies (Vinga & Almeida, 2003), the BST approach offers a balance between speed and specificity. Alignment-free methods such as the one used by Vinga & Almeida (2003) are fast and can handle large datasets, but sometimes they miss exact matches or struggle with repetition. Newer algorithms, like GSAlign (Lin & Hsu, 2020) have also advanced the field of genomics by making alignment faster and more accurate, but they rely on more complex data structures or require more computer memory. The BST method in this paper is simpler to implement and provides a more efficient searching method by using the tree's hierarchical structure to reduce unnecessary comparisons.

Overall, the BST algorithm offers a simple and unique solution for the goal of finding fixed-length common subsequences. While it may not capture more complex relationships between sequences, such as insertions or deletions, it still serves as a practical tool for sequence comparison tasks. Future work could combine this approach with alignment-free or other methods to improve both sensitivity and speed for even larger or more complex genomic analyses.

CONCLUSION

In this work, we provided a straightforward algorithm to search for similar subsequences of a specified length between two sequences using the Binary Search Tree data structure, with the intention of using this algorithm to be applied to the genetic field. This study shows potential applications of this problem, such as

looking for common strings of nitrogenous bases to show evolutionary relationships between species and the potential of finding common traits displayed through similarities in DNA. Using random strings and string lengths, the algorithm was run multiple times with varying data inputs.

This paper does not discuss the potential of using a different approach to achieve the same results and focuses on using the binary search tree data structure as the main component to the algorithm. Using the two-step approach to the algorithm by storing and searching for the subsequences, we were able to efficiently develop this program and replicate it through the general algorithm. Additionally, further research could be completed using this algorithm to examine longer sequences or be used along with find and replace functions.

REFERENCES

Bioinformatics.org. (2023). *Random DNA Sequence*. Bioinformatics.org. https://www.bioinformatics.org/sms2/random_dna.html

Hirschberg, D. S. (1975). A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, *18*(6), 341–343. https://doi.org/10.1145/360825.360861

Lin, H.-N., & Hsu, W.-L. (2020). GSAlign: an efficient sequence alignment tool for intra-species genomes. *BMC Genomics*, *21*(1). https://doi.org/10.1186/s12864-020-6569-1

Mads Haahr. (2019). *RANDOM.ORG - Integer Generator*. Random.org; RANDOM.ORG. https://www.random.org/integers/

National Cancer Institute. (2011, February 2). *https://www.cancer.gov/publications/dictionaries/cancer-terms/def/base-pair*. Www.cancer.gov. https://www.cancer.gov/publications/dictionaries/cancer-terms/def/base-pair

National Committee on Mapping and Sequencing the Human Genome. (1988). *Mapping and Sequencing the Human Genome*. National Academies Press. https://doi.org/10.17226/1097

Nelson-Fromm, T. (2023). *CS 225 | Binary Search Trees*. Illinois.edu. https://courses.grainger.illinois.edu/cs225/fa2023/resources/bst/

Rubin, J. (2011, November 1). *MIT School of Engineering |» Can a computer generate a truly random number?* Mit Engineering. https://engineering.mit.edu/engage/ask-an-engineer/can-a-computer-generate-a-truly-random-number/

Vinga, S., & Almeida, J. (2003). Alignment-free sequence comparison--a review. *Bioinformatics*, *19*(4), 513–523. https://doi.org/10.1093/bioinformatics/btg005