

NR - ISSN 0001-3099



abacus

THE JOURNAL OF THE MATHEMATICAL
ASSOCIATION OF NIGERIA

VOLUME 35, NUMBER 2
MATHEMATICS SERIES
AUGUST 2008

DR (Mrs.) OLADELE
ABACUS

**THE JOURNAL OF
MATHEMATICAL ASSOCIATION
OF NIGERIA (M.A.N)**

NR-ISSN-3099

**EDITOR-IN-CHIEF
PROFESSOR J. S. SADIKU**
DEPARTMENT OF MATHEMATICS, FACULTY OF SCIENCE,
UNIVERSITY OF ILORIN, ILORIN, NIGERIA.

© THE MATHEMATICAL ASSOCIATION OF NIGERIA

THE MATHEMATICAL ASSOCIATION OF NIGERIA (M.A.N)

COUNCIL MEMBERS 2007/2008

Alh. A. O. Okuboyejo	-	National President
Prof. I. A. Ochepe	-	National Vice President
Dr. S. A. Okunuga	-	National Secretary
Prof. J. S. Sadiku	-	Editor-in-Chief, ABACUS (The Journal of Mathematical Association of Nigeria)
Dr.(Mrs.) M. F. Salman	-	National Financial Secretary
Mrs. P. N. Ezenwelu	-	National Treasurer
Mr. Christopher Yelposu	-	National Assistant Secretary
Mal. Mohammed Tukur	-	National Publicity Secretary
Mr. Ishaya Amutu	-	National Business Manager
Dr. (Mrs.) A. A. Eniayeju	-	Immediate Past National President and Ex-officio Member.
Mal. S. M. Sirajo	-	Immediate Past National Secretary and Ex-officio Member
Dr. M. O. Ibrahim	-	Ex-officio Member
Mr. E. S. Ebisine	-	Ex-officio Member

CHAIRPERSONS OF ALL STATE BRANCHES

ABACUS:

The Journal of the Mathematical Association of Nigeria, Volume 35, Number 2, August 2008.

Abacus, the Journal of the Mathematical Association of Nigeria, publishes very carefully selected papers covering all areas of Mathematics, Mathematics Education, Statistics Computer Science and Engineering.

The Journal publishes five types of contributions namely: survey papers, contributed papers, technical notes, technical comments and book reviews, the latter strictly by invitation of the editors through the Editor-in-Chief of the Journal.

Two issues (Mathematics series and Mathematics Education series) of the Journal are published in August or September of every year. Author(s) is/are to pay ₦1,200.00 for processing a paper. Three copies of the manuscript should be forwarded for review.

The papers should be typeset using 7" by 9" dimension, margin set at: left 1", right 0.5, top 0.5 and bottom 0.5". Send three copies of your article to the Editor-in-Chief:

EDITOR-IN-CHIEF

PROFESSOR J. S. SADIKU

DEPARTMENT OF MATHEMATICS, FACULTY OF SCIENCE,
UNIVERSITY OF ILORIN, ILORIN, NIGERIA.

The reviewers' comments are then sent to the authors(s). The revised article should be forwarded to the Editor-in-Chief in a CD or flash drive with two hard copies included. Microsoft word is preferred. The use of Microsoft Drawing for equation and grouping of equations is not accepted. Any other software used in typing your article aside from Microsoft word should be accompanied with the installation CD so as to make editing easy.

EDITORIAL BOARD

PROFESSOR J. S. SADIKU

(Editor-in-Chief)

Department of Mathematics, Faculty of Science,
University of Ilorin, Ilorin, Nigeria.

Professor P. Onumanyi
National Mathematical Centre,
Abuja, Nigeria.

Professor S. O. Iyehen
National Mathematical Centre,
Abuja, Nigeria.

Professor J. Amazigo
Department of Mathematics
University of Nigeria, Nsukka
Nigeria.

Professor M. S. Audu,
Department of Mathematics
University of Jos, Jos, Nigeria.

Professor S. O. Ale
National Mathematical Centre,
Abuja, Nigeria.
Nigeria

Professor S. A. Ilori
Department of Mathematics
University of Ibadan, Ibadan,

Professor R. O. Ayeni
Department of Pure & Applied
Mathematics, Ladoke Akintola
University of Technology, Ogbomoso,
Nigeria

Professor P. N. Lassa
Faculty of Education
University of Jos, Jos, Nigeria

Professor M. O. Fajemidagba
Department of Science Education
University of Ilorin, Ilorin, Nigeria.

Professor K. V. Aderogba
Department of Mathematics
University of Lagos, Lagos,
Nigeria.

ASSOCIATE EDITORS

Professor L. O. Adetula
National Mathematical Centre, Abuja
Nigeria.

Dr. I. U. Jahun
Federal College of Education
Bichi, Kano, Nigeria

Professor E. T. Jolayemi
Department of Statistics
University of Ilorin, Ilorin, Nigeria

Professor E. A. Akinrelere
Department of Mathematics
Obafemi Awolowo University
Ile-Ife, Nigeria.

Professor Herbert Wills
Department of Mathematics
Education, Florida State University,
Tallahassee, Florida, U.S.A.

Dr. A. A. Tijjani
Department of Mathematics
Ahmadu Bello University, Zaria,
Nigeria.

Dr. M.O. Oyesanya
Department of Mathematics
University of Nigeria, Nsukka,
Nigeria

Professor J. A. Adepoju
Department of Mathematics
University of Lagos, Lagos,
Nigeria.

Professor O.S. Adegboye
Department of Statistics
University of Ilorin, Ilorin, Nigeria

Professor M. A. Ibiejugba
Department of Mathematics
University of Ilorin, Ilorin, Nigeria

Professor G. M. Gladwell
University of Waterloo, Waterloo,
Canada

Professor A. U. Afuwape
Department of Mathematics
Obafemi Awolowo University
Ile-Ife, Nigeria.

Professor G. O. S. Ekhaguere
Department of Mathematics
University of Ibadan, Ibadan, Nigeria

TABLE OF CONTENTS

	Page
1. Reuben, U. and Iyai, D. - Asymptotic Null Controllability of Linear Neutral Volterra Integrodifferential Systems With Distributed Delays In Control	1
2. Okunuga, S. A. and Sofoluwe, A. B. - A Fifth Order Padé Integration Formula for Stiff Problems	6
3. Ette, A.M. - Perturbation Scheme on the Stability of Cylindrical Shells Stressed by External Pressure	14
4. Ifiok, A. E. - An Alternative Method for Calculating the Distance between Two Points on the Surface of the Earth.	27
5. Odule, T. J. - Understanding and Managing Computer Viruses	38
6. Oluyo, T.O and Ayeni, R.O - Mathematical Analysis of the Global Dynamics for Infectious Diseases	47
7. Ajala, O.A and Ayeni, R.O - A Note on the Existence and Uniqueness of a Bounded MHD Boundary Layer Flow Over a Moving Vertical Cylinder	50
8. Oladele, T. O., Bamigbola, O. M. and Gbadeyan, J. A. - Concept of Strings and Trees in Bioinformatics.	56
9. Amponsah, S. K., Darkwah, K. F., Oduro, F. T. and Ofori-Kusi, D. - Fibonacci Search Method On a TI-89 Graphing Calculator	68
10. Oladele, T. O. - A Survey of Programming Languages Relevant to Bioinformatics	85
11. Gbadeyan, J. A. Sadiku, J. S. and Babatunde, A. O. - Analysis of Pascal, Basic and C Programming Languages	92
12. Lasode, O. A. - Heatline Visualization of Unsteady Natural Convection in Square Enclosures with Top Conducting Wall	96
13. Ajibade, A. O., Yakubu, M. I. and Alkali, A. - Reversibility on the Set of Singular Matrices	133
14. Makanjuola, S.O. - A Class of Type w - Semigroup	140
15. Oyesanya, M. O. - Some Remarks on the Stability of the Imperfect Structure	143

16. Ngiangia, A., Ogbondah, C., Womuru, E. N., Wonu, N. & Ejekudo, N.
- A Study on the Effect of Transverse Magnetic Field on the Trajectory
of a Free Jet 154
17. Oyelami, B. O., Ale, S. O. & Sesay, M. S. - On Existence and Stability with
Respect to Invariant Sets for Impulsive Differential Equations with Variable
Times. 160
18. Kaurangini, M.L. & Jha, B. K. - Combined Effects of Free and Forced
Convection Flow in a Vertical Annulus Filled with Porous Material Having
Variable Porosity 179
19. Nwachukwu, G. C. & Otunta, F. O. - An Extension of Variable Order
One-Step Methods for Initial Value Problems. 193-199

Sufficient conditions for the asymptotic null controllability of linear neutral Volterra integrodifferential systems with distributed delays in the control variables are developed. The results are obtained by using the nondegeneracy of the controllability matrix. An example is provided to illustrate the obtained result.

1. INTRODUCTION

Neutral functional differential equations are characterized by a delay in the derivative. Equations of this type have applications in the study of electrical networks containing lossless transmission lines. It is well known that the mixed initial-boundary hyperbolic partial differential equation which arises in the study of lossless transmission lines can be replaced by an associated neutral functional differential equation [2]. Volterra integrodifferential equations occur often in applied mathematics [4]. For instance, in [8] a simplified model for compartmental systems with pipes is represented by nonlinear neutral Volterra integrodifferential equation.

The problem of controllability and null controllability of linear neutral systems has been studied by several authors [3,6,10,14]. Chukwu [11] and Angeli [7] discussed the functional controllability of nonlinear neutral systems and Underwood and Chukwu [9] studied the null controllability for such systems. Onwunatu [5] discussed the problem of null controllability for nonlinear systems of neutral functional differential equations with limited controls. The purpose of this paper is to establish a set of sufficient conditions for the asymptotic null controllability of linear neutral Volterra integrodifferential systems with distributed delays in control. This is motivated by the fact that models for systems with delay in the control occur in population studies and in some complex economic systems. Specifically, models for systems with distributed delays in the control occur in the study of agricultural economics and population dynamics [1, 12].

2. PRELIMINARIES

Consider the linear neutral Volterra integrodifferential systems with distributed delays of the form

$$\frac{d}{dt} \left[x(t) - \int_0^t C(t,s)x(s)ds - g(t) \right] = A(t)x(t) + \int_0^t G(t,s)x(s)ds + \int_0^t d_\theta B(t,\theta)u(t+\theta)$$

CONCEPT OF STRINGS AND TREES IN BIOINFORMATICS

¹OLADELE T. O. ²BAMIGBOLA O. M. and ³GBADEYAN J. A.

¹Department of Computer Science, University of Ilorin, Ilorin, Nigeria.

^{2,3}Department of Mathematics, University of Ilorin, Ilorin, Nigeria.

ABSTRACT

Strings have been found to be the most general medium for the representation of information. Considering the large amount of data stored in data dictionaries, databases or the massive data in genomic databases, text remains the main form of exchanging information. The representation of information from real-world problem may involve the use of many interlinked data structures.

In this paper, the suffix tree data structure was used to solve the Pattern Matching problem. We present the suffix tree as an efficient data structure that provides efficient access to all substrings of a string. It is able to rapidly align sequences containing millions of nucleotides and give significant biological information.

1. INTRODUCTION

The universe of strings is probably the most general medium for the representation of information.

Although data are stored in various ways, text remains the main form of exchanging information. This is particularly evident in literature or linguistics where data are composed of huge corpora and dictionaries. This applies as well to computer science where a large amount of data is stored in linear files. The same is also true of molecular biology where biological molecules can often be approximated as sequences of nucleotides or amino acids.

In this paper the suffix tree data structure was implemented using the McCreight Algorithm in order solve the Pattern Matching problem.

2. PRELIMINARIES

The meaning and definition of some terms and concepts that will provide a thorough understanding of the work in this paper are presented in this section.

Definition 2.1: A String is an ordered list of characters written contiguously from left to right. For any string S , $S[i..j]$ is the (contiguous) sub-string of s that starts at position i and ends at position j of string S .

Definition 2.2: An Alphabet is a finite non-empty set of symbol(s) which are assumed to be indivisible.

For example, the alphabet for English language consists of 26 upper case Letters $A, B, \dots, Z$ and 26 lower case Letters $a, b, \dots, z$.

We usually use the symbol Σ to denote an alphabet system.

Definition 2.3: A String over an alphabet Σ is a finite sequence of symbols of Σ .

Definition 2.4: Given a String S , $S[1 \dots i]$ is the prefix of the string S that ends at position i , and $[i \dots |S|]$ is the suffix of string S that begins at position i .

Example; California is a string, lifo is a substring, cal is a prefix, and ornia is a suffix.

Definition 2.5: For any string S , $S(i)$ denotes the i th character of S .

Definition 2.6: Let $x = a_1, a_2, \dots, a_n$ and $y = b_1, b_2, \dots, b_m$ be two strings. The concatenation of x and y is denoted by xy .

y , denoted by xy , is the string $a_1, a_2, \dots, a_n, b_1, b_2, \dots, b_m$ e.g Let $S_1 = \text{GOOD}$ and $S_2 = \text{DAY}$, The concatenation of S_1 and S_2 which is denoted by $S_1 S_2 = \text{GOODDAY}$.

Definition 2.7: A dictionary is a container of elements from a totally ordered universe that supports the following basic operations:-

- * **FIND:** search for an element.
- * **INSERT:** insert an element.
- * **REMOVE:** delete an element.

A major application of dictionaries is database systems.

In the most general setting, the element in a dictionary are pairs (x, y) where x is the key giving the ordering of the elements and y is the auxiliary information.

For example, in a database storing student records, the key could be the student's last name or matriculation number, and the auxiliary information the student's transcript.

TREE STRUCTURES

A tree consists of a finite set of elements, called **Nodes**, and a finite set of directed lines, called **branches**, that connect the Nodes. When the branch is directed to the node, it is an indegree branch, when the branch is directed away from the node, it is an outdegree branch. See figure 1.1 below:

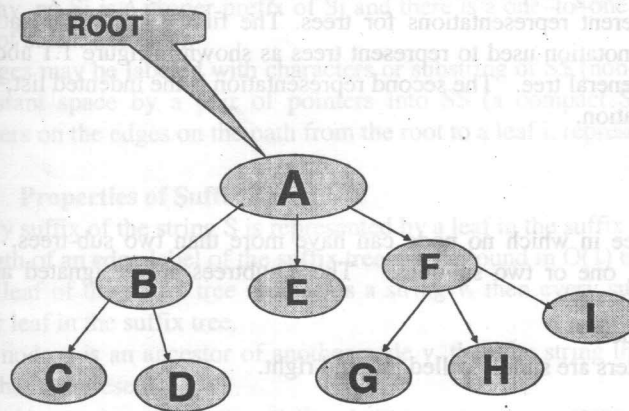


Figure 1.1: A tree

TREE TERMINOLOGIES

The first node of a tree is called the **root**. A **leaf** is any node with an outdegree of zero. i.e It has no outgoing branches e.g node C, D, G, H and I as shown above. Nodes that are not a root or a leaf are known as **Internal nodes** because they are found in the middle portion of a tree,

A node is a **parent** if it has successor nodes; i.e it has an outdegree greater than zero. Conversely, a node with a predecessor is a **child**. A child node has an in degree of one. Two or more nodes with the same parent are called **siblings**.

An ancestor is any node in the path from the root to the node. A **descendant** is any node in the path below the parent node.

CONCEPT OF STRINGS AND TREES IN BIOINFORMATICS

A path is a sequence of nodes in which each node is adjacent to the next one.

Every node in the tree can be reached by following a unique path starting from the root.

Example:

Parent: A, B, F

Children: B, E, F, C, D, G, H, I,

Siblings: {B, E, F}, {C, D}, {G, H, I}

Leaves: C, D, E, G, H, I

Internal nodes: B, F

A tree may be divided into sub-trees. A sub-tree is any connected structure below the root. The first node in a sub-tree is known as the root of the sub-tree. Furthermore, sub-trees can be subdivided into subtrees.

The concept of subtrees leads us to a recursive definition of a tree.

A tree is a set of nodes that is

1. either empty or
2. has a designated node called the root from which hierarchically descent zero or more subtrees which are also trees.

TREE REPRESENTATION

While a tree is generally implemented in the computer using pointers, outside the computer there are three different representations for trees. The first is the organization chart format, which is basically the notation used to represent trees as shown in figure 1.1 above. The term used for this notation is general tree. The second representation is the indented list, while the third is the graphical representation.

TYPE OF TREES

BINARY TREES

A binary tree is a tree in which no node can have more than two sub-trees. In other words, a node can have zero, one or two subtrees. These subtrees are designated as the left subtree and right subtree.

A null tree is a tree with no nodes.

Traditionally, the subtree pointers are simply called left and right.

BINARY TREE TRAVERSALS

The binary tree traversal requires that each node of the tree be processed once and only once in a predetermined sequence. These are two general approaches to tree traversals;

Depth first and breadth first.

In depth-first traversal all of the descendants of a child are processed before the next child.

In breadth-first traversal each level is completely processed before the next level is started.

3. DATA STRUCTURES

A data structure is a systematic way of organizing and accessing data. It is an aggregation of atomic and composite data types into a set with defined relationship.

3.1 Suffix Trees

Suffix trees are inter-connected structures (data structure), that provide efficient access to all substrings of a string. Suffix trees can be used to solve the exact matching problem in linear time (achieving the same worst-case bound that the Knuth-Morris-Pratt and the Boyer-Moore

algorithms achieved).

Definition 2.8: A suffix tree T for an m -character string S is a rooted directed tree with exactly m leaves numbered 1 to m . Each internal node, other than the root, has at least two children and each edge is labeled with a nonempty substring of S . No two edges out of a node can have edge-labels beginning with the same character. The key feature of the suffix tree is that for any leaf i , the concatenation of the edge-labels on the path from the root to leaf i exactly spell out the suffix of S that starts at position i . That is, it spells out $S[i..m]$.

As stated above, the definition of a suffix tree for S does not guarantee that a suffix tree for any string S actually exists. The problem is that if one suffix of S matches a prefix of another suffix of S then no suffix tree obeying the above definition is possible, since the path for the first suffix would not end in a leaf.

For example, if the last character of $xabxac$ is removed, creating string $xabxa$, then suffix xa is a prefix of suffix $xabxa$, so the path spelling out xa would not end in a leaf.

To avoid this problem we assume that the last character of S appears no-where else in S . Then no suffix of the resulting string can be a prefix of any other suffix. To achieve this, we add a character to the end of S that is not in the alphabet that string S is taken from. We use $\$$ for the "termination" character which is called the sentinel.

Informally, suffix trees of a string $S = s_1, \dots, s_n$, is a tree with $O(n)$ nodes and edges and n leaves. Each leaf in the suffix tree is associated with an index ($1 \leq i \leq n$). This is made possible by the concatenation of S with a special character $\$$, usually called the sentinel, that is not occurring in S .

This way, no S_i is a proper prefix of S_j and there is a one-to-one correspondence between $S\$$ and leaves of suffix tree (ST).

The edges may be labeled with characters or substring of $S\$$ (non-compact ST) or are represented in constant space by a pair of pointers into $S\$$ (a compact ST), but a concatenation of the characters on the edges on the path from the root to a leaf i , represent the i th suffix of string S .

3.1.1 Properties of Suffix Trees

1. Every suffix of the string S is represented by a leaf in the suffix tree.
2. Length of an edge label of the suffix tree can be found in $O(1)$ time.
3. If a leaf of the suffix tree represents a string v , then every suffix of v is also represented by another leaf in the suffix tree.
4. If a node u is an ancestor of another node v , then the string that u represents is a prefix of the string that v represents.
5. If nodes $v_1, v_2, \dots, v_k$ represent the strings $v_1, v_2, \dots, v_k$, respectively, then the lowest common ancestor of $v_1, \dots, v_k$ represents the longest common prefix of $v_1, \dots, v_k$.
6. With the Schieber-Vishkin method, the lowest common ancestor (lca) of two arbitrary nodes v_1 and v_2 can be computed in constant time.

There are two main reasons, why suffix tree with all its advantages have not been widely used in practice.

Implementing suffix trees is generally believed to be extremely involving. This is particularly due to the first two original papers that introduced suffix trees. They are generally difficult to understand, but recent works like Gusfield and Kurtz on suffix trees have simplified its implementation.

The suffix tree is also believed to be greedy for space. It required large space to be stored, that usually leads to excessive disk thrashing during its construction.

To tackle this problem, previous works have proposed alternative index structures which store less information than the suffix trees and are therefore more space efficient. These alternative structures include the suffix array, level compressed trie, suffix binary search tree and suffix cactus.

Apart from the fact that these structures are not as versatile as the suffix trees, their direct implementation do not run in linear worst case time as their indirect implementations, which normally begins with the construction of the suffix trees. Recently, another direction has been taken by Kurtz to tackle this problem. In his work, he showed that suffix trees store various kinds of redundant information and exploited these redundancies to obtain a more space-efficient representation in practice (average case) and also in worst case.

Specifically, he reduced the space requirement of suffix trees from $28.n$ bytes to $20.n$ bytes in worst case and to $10.1n$ bytes in practice. He further claimed that all formal implementation techniques of suffix trees can be implemented upon his representation, without any significant changes in these techniques. As an example, he modified McCreight's algorithm to implement suffix trees, while making use of his representation.

In this paper, we made use of Kurtz's representation and his modified McCreight's algorithm for constructing suffix tree.

3.1.2 Applications of Suffix Trees

Here some applications of suffix trees are discussed. Most of these applications allow surprisingly efficient, linear-time solution to complex string problems. These applications illustrate the power and utility of suffix trees.

(i) Exact String Matching:

There are three important variants of this problem depending on which string p or T is known first and held fixed. We have an instance of the use of suffix trees in the exact matching problem when the pattern and the text are both known to the algorithm at the same time. In that case, the use of a suffix tree achieves the same worst-case bound, $O(n+m)$ as the Knuth-Morris-Pratt or Boyer-Moore algorithms.

The exact matching problem often occurs in the situation when the text T is known first and kept fixed for some time. After the text has been preprocessed, a long sequence of patterns is input, and for each pattern P in the sequence, the search for all occurrences of P in T must be done as quickly as possible.

Let n denote the length of P and k denote the number of occurrences of P in T . Using a suffix tree for T , all occurrences can be found in $O(n+k)$ time, totally independent of the size of T .

(ii) Suffix Trees And The Exact Set Matching Problem:

This is the problem of finding all occurrences from a set of strings P in a text T , where the set is input all at once. There we developed a linear-time solution due to Aho and Corasick. Set P is of total length n and text T is of length m . The Aho-Corasick method finds all occurrences in T of any pattern from P in $O(n+m+k)$ time, where k is the number of occurrences. This same time bound is easily achieved using a suffix tree T of T .

When T is first known and fixed and the pattern P varies, all occurrences of any specific P (of length n) in T can be found in $O(n+k_p)$ time, where k_p is the number of occurrences of P . This exact matching problem is actually a simpler case because the set P is input at the same time the text is known. To solve it, we build suffix tree T for T in $O(m)$ time and then use this tree to successively search for all occurrences of each pattern in P . The total time needed in this approach is $O(n+m+k)$.

The Substring Problem For A Database Of Patterns:

In the most interesting version of this problem, a set of strings, or a database, is first given and fixed. Later, a sequence of strings will be presented and for each presented strings, the algorithm must find all the string in the database containing S as a substring. In the context of databases for genomic DNA data, the problem of finding substrings is a real one that cannot be solved by exact set matching. The DNA database contains a collection of previously sequenced DNA strings.

When a new DNA string is sequenced, it could be contained in an already sequenced string, and an efficient method is needed to check that it is of value. Suffix trees yield a very attractive solution to the database problem.

GENERALIZED SUFFIX TREES

Generalized suffix trees (hence forth GST), is the resulting suffix tree constructed for multiple strings, $S_1, S_2, \dots, S_n$ instead of a single string. The GST allows efficient access to common substrings and suffixes of strings $S_1, S_2, \dots, S_n$.

A simple way to build GST is to concatenate strings $S_1, S_2, \dots, S_N$ together, separated from each other by some special symbols not contained in any strings. The resulting suffix tree for the concatenated string is the required GST for strings $S_1, S_2, \dots, S_N$. In this tree, every suffix of every string is represented by a leaf. Note that, sometimes two or more strings of $S_1, S_2, \dots, S_N$ may have the same suffix. In this case, the GST has two or more leaves corresponding to the same suffix, each corresponding to a different string.

A GST can be built in $O(N.n)$ time and space, where N is the number of strings and n is the average length of each string.

A GST can be built fairly quickly. First build a suffix tree for S_1 (assuming an added terminal character \$). Then starting at the root of the tree, match S_2 against a path in the tree until a mismatch occurs. At that point add the remaining characters of the suffix of S_2 to the ST. When S_2 is fully processed, it will encode all the suffixes of S_1 and all the suffixes of S_2 .

A leaf's label consists of the string number (first number) and the starting position of the suffix in that string (second number).

Applications of Generalized Suffix Trees

Longest common substring (LCS) problem of two strings:

Given two strings S_1 and S_2 , the longest common substring is a substring that appears both in S_1 and S_2 and has the largest possible length. A simple and efficient way to find the longest common substring between two strings is to build a GST for S_1 and S_2 . Each leaf of the tree represents either a suffix from one of the two strings or a suffix that occurs in both strings. Mark each internal node v with a 1 (and 1 or 2) if there is a leaf in the subtree of u representing a suffix from S_1 (and / or S_2). The path-label of any internal node marked both 1 and 2 is a substring common to both S_1 and S_2 , and the longest such string in the LCS. The algorithm has to find the deepest node (that has the highest number of characters on the path to it) that is marked both 1 and 2 (and therefore, has two leaves). This problem can be solved in linear time, since the construction of the GST can be done in linear time (proportional to the total length of S_1 and S_2) and the node marking and calculations of string-depth can be done by known linear-time tree traversal methods.

All-Pairs Suffix-Prefix Matching Problem (APPSMP):

A suffix-prefix match of the sequences S_1 and S_2 refers to any suffix of S_1 that matches

a prefix of S_2 . Given a set of sequences $\{S_1, \dots, S_n\}$ the APPSMP is the problem of finding, for each ordered pair S_i, S_j in the set, the longest suffix-prefix match of (S_i, S_j) . This problem can be solved in linear time using the GST as the main data structure. To solve the APPSMP a GST is constructed for all the sequences in the set.

The algorithm also builds a list $L(v)$ for each internal node v . The $L(v)$ list contains the index i if and only if v is incident with a terminal edge (an edge or a GST that is only labeled with the string termination symbol, $\$$) whose leaf is labeled by a suffix of sequence S_i . That is, $L(v)$ holds the index i if and only if the path label to v is a complete suffix of sequence S_i .

(c) All Maximal Repeats Problem (AMRP) In Single Sequence:

A sequence can contain repetitive subsequences. These repeats can occur either adjacent to each other (Tandem repeats) or apart any where in the sequence. Some of the Tandem repeats consist of very short sub sequences, called Micro satellites, and have become the preferred markers in many genetic mapping efforts.

Maximal repeats are two identical subsequences in a sequence S , such that the character to the immediate left (right) of the first sub sequence is different than the character to the immediate left (right) of the second sub sequences.

That is, extending either of the two subsequences in either direction destroys the equality of the two subsequences. Our main interest in repetitive structures in sequences is due to the repetitive structures seen in biological sequences.

For example, most of the human Y chromosome consists of repeated subsequences. In addition, many diseases are caused by structural repeats. Therefore, solving the AMRP is useful in finding DNA repeats.

A suffix tree can be used to find all maximal repeats in a linear time. Once a suffix tree is constructed, any internal node implies a maximal repeat at two locations. The length of the repeat is the length of the subsequence from the root to that node.

(d) Circular DNA Sequences problems:

A circular sequence "S" of length n is defined as a sequence in which character n is considered to precede character 1. The characters in a sequence are numbered sequentially from 1 to n , starting as an arbitrary character in "S".

Given two circular sequences of the same length, a ST can be used to compare these two sequences, to determine if they are equal, in linear time. This is done by cutting the first circular sequence S_1 at an arbitrary point to obtain a linear sequence L_1 . Doubling L_1 , creates the sequence L_1L_1 . At this point, construct the suffix tree T_1 for L_1L_1 . Next, traverse T_1 with the rule that, at every node, the traversal follows the edge whose first character is lexically smaller over all the first characters on edges out of the node. The traversal continues until the traversal path P_1 has a sequence-depth n . Repeat for sequence S_2 to get the path P_2 . Then compare P_1 and P_2 to see if they are equal.

(e) The K-Mismatch (error) Problem:

This problem deals with finding sequences with errors. Specifically, it refers to matching sequences with an allowable (k) number of errors. For example, given a set of strings, find all occurrences of the pattern $P = \text{"Biology"}$ in this set of strings while allowing for only one error. P can occur differently as P' in any of the strings due to one of these possible errors: insertion, deletion or substitution. To solve this problem, we need to generate every possible string P' that can be derived from P by changing up to 1 character of P . Construct a GST containing all the given strings. Then traverse the tree to find all the starting locations of the possible occurrences.

This search can be done in linear time and is proportional to the length of P' .

APPLICATION OF STRINGS AND TREES

Hidden Markov Models

Hidden markov model is a computational structure for describing the subtle patterns that define families of homologous sequences. Hmms are powerful tools for detecting distant relatives, and for prediction of protein folding patterns. They are the only methods based entirely on sequences that is, without explicitly using structural information.

It is competitive with PSI – BLAST for identifying distance homologues. They also perform well at fold recognition, as assessed in CASP programs.

Within an Hmm is a multiple sequence alignment. However, Hmms are usually presented as procedures for generating sequences. A conventional multiple sequence alignment table could also be used to generate sequences by amino acids at successive position, each amino acid chosen from a position – specific probability distribution derived from the profile. But Hmms are more general than profiles.

They include the possibility of introducing gaps into the generated sequence, with position – dependent gap penalties.

Application of profiles requires that the multiple sequence alignment be specified upfront; the pattern statistics are then derived from the alignment. Hmms carry out the alignment and the assignment of probabilities together.

The internal structure of an Hmm shows the mechanism for generating sequences as shown below:

Begin at start, and follow some chain of arrows until arriving at end. Each arrow takes you to a state of the system. At each state (1) you take some action-emit a residue, perhaps and (2) choose an arrow to take you to the next state. The action and the choice of successor state are governed by sets of probabilities.

The dynamics of the system is such that only the current state influences the choice of its successor – the system has no memory of its history.

Software for applying Hmms to biological sequence analysis can achieve

Training – Given a set of unaligned homologous sequence, it can align them and adjust the transition and residue-output probabilities to define an Hmm capturing the patterns inherent in the sequences submitted.

Detection of distant homologues. Given an Hmm and a test sequences, it calculates the probability that the Hmm could generate the test sequence.

If an Hmm trained on a known family of sequences would generate the test sequence with relatively high probability, the test sequence is likely to belong to the family.

Align additional sequences, the probability of any sequence of states can be computed from the individual state – to – state transition probabilities. Finding the most likely sequence of states that the Hmm would use to generate one or more test sequences reveals their optimal alignment family.

SUFFIX TREE DATA STRUCTURES FOR ALIGNMENT OF WHOLE GENOMES:

The suffix trees has been found to be an efficient data structure for alignment of sequences containing millions of nucleotides.

The focus of most prior research has been on comparing single proteins or genomic DNA

sequences containing a single gene. The existing algorithms work extremely well on this task, but in most cases are ineffective in aligning entire genomes.

In addition, previous algorithms were designed primarily to discover insertions, deletions and point mutations; but not to look for the kinds of large changes that can be discovered in whole-genome comparisons, such as differences in tandem repeats and large scale reversal.

This new system is used for pairwise alignment and comparison of very large scale DNA sequences. The system is specifically designed to perform high resolution comparison. It outputs a base-to-base alignment of the input sequences, highlighting the exact differences in the genomes. It will locate all single nucleotide polymorphisms (SNPS), large inserts, significant repeats, tandem repeats and reversals, in addition to identifying the exact matches between the genomes.

4.2.1 The challenge of whole Genome Alignment.

The standard algorithms for sequence alignment rely on either dynamic programming or hashing techniques. In contrast, the suffix-tree approach directly finds maximal matches that are unique. This system uses a combination of three ideas: suffix trees, the longest increasing sequence (LIS) and Smith-Waterman alignment. The suffix tree is recommended as an efficient data structure for implementing pattern matching, whole genome alignment algorithm etc.

5. IMPLEMENTATION McCREIGHT ALGORITHM USING SUFFIX TREE DATA STRUCTURE

Here, the Pattern Matching problem is considered. The practical importance of pattern matching problems cannot be overemphasized. The problem arises in widely varying applications. In molecular biology there are several hundred specialized databases holding raw DNA, RNA and amino acid strings, or processed pattern (called motifs) derived from the raw string data.

5.1 PATTERN MATCHING PROBLEM

Given a string P called the pattern and a longer string T called the text, the pattern matching problem is to find all occurrences, if any, of pattern P in text T.

For example, if P = aba and T = bbabaxababay then P occurs in T starting at locations 3, 7 and 9. Note that two occurrences of P may overlap, as illustrated by the occurrences of P at locations 3 and 9.

5.2 PROGRAM SPECIFICATION

The program designed and implemented in this paper was written in C and it has a main program called Tree. The main program is an interactive menu-driven program which has four routines that can be accessed by selecting either of the numbers 1, 2, 3 and 4.

The menu is displayed as follows;

Please Select Routine

=====

- [1] Suffix Array
- [2] Suffix Tree
- [3] Exact String Matching
- [4] Exit Program

Choice (1 - 4):

If Choice is 1, the function suffix_array() is executed.

- Choice is 2, the function suffix-tree() is executed.
- Choice is 3, the function pattern_matching() is executed.
- Choice of 4, exits the menu and entire program.

THE SUFFIX ARRAY ROUTINE

The suffix array routine was implemented in this paper by the function suffix_array(). The function suffix_array() inputs a string which is entered from the keyboard.

The string length is determined by the library function strlen() and the main string is displayed on the screen. All the suffixes of the entered string and their respective starting positions in the main string are generated.

The suffix array is then generated by carrying out a lexicographic sorting of all the suffixes. The suffix array displays a lexicographic sorted list of suffixes with respect to their positions in the string.

Searching for patterns in the suffix array returns the position of occurrence of the pattern in the main string. The suffix_array() function was implemented using the string assassin.

THE SUFFIX TREE ROUTINE

The suffix tree routine was implemented in this paper by the function suffix_tree(). The suffix_tree() function was used to build the suffix tree for string abab.

The suffix_tree() function reads in the string abab. The McCreight algorithm builds the suffix tree in linear time $O(n)$.

5.2.3 THE EXACT STRING MATCHING ROUTINE

The exact string matching routine was implemented by the function pattern_matching(). The function pattern_matching() reads in a string of DNA sequence or protein sequence from a file. This is referred to as the main string. The length of this string is determined by the library function strlen().

A shorter string called the search string (pattern) is then entered. The main string is scanned for the first occurrence of the search string. Hence, the respective positions of occurrence of the search string in the main string are printed. The number of occurrence of the search string in the main string is also printed.

6. ANALYSIS OF RESULTS

SUFFIX ARRAY

The suffix array was implemented for the string $T = \text{assassin}$. This is a typical example given by Manber and Myers [10].

Pos is a lexicographically sorted array of start positions of all the suffixes in the string. Pos[k] is the start position of the kth smallest suffix. Pos[2] in the suffix array = 6. Hence, $T_6 = \text{in}$.

For large DNA and protein sequences the suffix array is found to be more space efficient than the suffix trees because it only stores array of positions without necessarily displaying all the suffixes in the sequence.

SUFFIX TREE

The suffix tree of $S = \text{abab}$ was implemented. The suffix tree represented by the string abab has two internal nodes called branching nodes. Leaf 1 spells out the entire string abab from the root.

CONCEPT OF STRINGS AND TREES IN BIOINFORMATICS

Leaf 2 spells out the suffix bab from the root since no two edges from the same node must begin with the same edge-label.

Leaf 3 spells ab, which is a branching after the second element in the suffix that ends in leaf 1.

Leaf 4 spells b, which is a branching after the first element in the suffix that ends in leaf 2.

Leaf 5 spells \$, the termination character from the root.

EXACT STRING MATCHING

The exact string matching problem was implemented by using the DNA sequence and the protein sequences.

The practical importance of the exact string matching problem should be obvious to anyone who uses the computer. Some of the common applications are in word processors, in textual information retrieval programs such as Medline, Lexis, in internet browser which sift through massive amounts of text; internet news readers that can search the articles for topics of interest. Also used for retrieving information in electronic journals that are already being "published" on-line; in telephone directory assistance; in on-line encyclopedias; in on-line dictionaries and thesauri, especially those with cross-referencing features; and in numerous specialized databases.

In molecular biology there are several hundred specialized databases holding raw DNA RNA, and amino acid strings, or processed patterns called motifs.

In this paper the DNA sequence that was used, was searched with the pattern AAT and 163 occurrences of this pattern was found in the sequence. On a second run, the pattern AGTC was search for and only one occurrence was found with their respective positions of occurrence in the DNA sequence displayed respectively.

The protein sequence given in use was searched with the pattern SSH and 36 occurrences were found with their respective positions of occurrence displayed respectively. On a second run the pattern DDV was used and 11 occurrences were found.

Note that, the patterns used here do not represent any particular biological concept, they were only chosen to test the efficiency of the implemented procedure. Hence, there is no restriction on the length of a pattern or search string. The pattern chosen will be determined by the type of information needed by the molecular biologist.

The position of occurrence of a pattern and the number of times it is repeated in a sequence are practically significant in drug design for preventive medicine and improved prescription.

7. CONCLUSION

In several applications text need to be structured before being searched. Even if no further information is known about their syntactic structure, it is possible and indeed extremely efficient to build a data structure that supports searches. Among several existing data structures equivalent to indices, the suffix tree, along with its constructions were presented and implemented in this paper using the McCreight algorithm. No other data structure exist that find diverse applications in text processing as the suffix trees.

In conclusion, algorithms based on suffix trees are more efficient for solving the exact string matching problems than the methods that involve fundamental preprocessing given by Boyer-Moore and Knuth-Morris-Pratt algorithms. The McCreight algorithm is found to be more efficient for building suffix trees for pattern matching. The suffix tree allows all the suffixes of a tree to be traversed during text processing.

REFERENCES

1. Adebiyi, E. F. (2002), Pattern Discovery in Biology and Strings Sorting: Theory and Experimentation. Ph.D. Dissertation. University of Tubingen, Germany.
2. Allen, B.T.(1997): The Computer Science and Engineering Handbook, CRC Press, Florida.
3. Delcher, A.L. , Kasif, S., Fleischmann, R.D., Petersen J., White, O., Salzberg, S.L. (1999): Alignment of Whole generous. Nucleic Acids Research Oxford University Press, Oxford.
4. Dipe, T.O. (2004): Suffix trees and pattern matching with Respect to Proteins and Deoxyribonucleic Acid (DNA) Sequence Analysis. MSc. Dissertation. University of Ilorin. Ilorin.
5. Gottfried, B. (1998), Programming with C. Tata McGraw-Hill. New Delhi.
6. Gilberg, R. F. and Forouzan, B. A. (1998), Data Structures: A pseudocode Approach with C. PWS publishing, Boston.
7. Gusfield, D. (1997): Algorithms on Strings, Trees and Sequences. Cambridge University Press, New York.
8. Kurtz, S. (1998), Reducing the Space Requirement of Suffix Trees. A Technical paper. University of Bielefeld, Germany.
9. Lesk, A. M. (2002): Introduction to Bioinformatics Oxford University Press, Oxford.
10. Manber, U. and Myers, G. (1993), Suffix Arrays: a new method for on-line string searches. SIAM J Comput. 22(5): 935 – 948.
11. Richard, F.G. and Behrouz, A.F. (1998), Data Structures. A Pseudocode Approach with C. Pws Publishing Company, Boston, M.A.
12. Schildt, H. (1997), Teach Yourself C. Third Edition. Osborne McGraw-Hill, California.
13. Standen, R. (1990), Methods for calculating the probabilities of finding patterns in sequences. Computer Application in Biosciences, 5, 8 – 96.
14. Standen, R. (1990), Searching for patterns in protein and nucleic acid sequences. Methods Enzymol. 183: 193 – 211.
15. The Wellcome Trust (2001), Deciphering the Genome. Wellcome News, London. Supplement 4, 10 – 11.