International Journal of Natural and Applied Sciences, 4(3): 262-277, 2008
www.tapasinstitute.org/journals/ijotafs

PQ TREES, CONSECUTIVE ONES PROBLEM AND APPLICATIONS

O. O. Olugbenga, E. F. Adebiyi, S. Fatumo and A. Dawodu
Department of Computer and Information Science, College of Science and Technology, Covenant University Ota, Nigeria
Corresponding author: O. O. Olugbenga

ABSTRACT

A PQ tree is an advanced tree–based data structure, which represents a family of permutations on a set of elements. In this research article, we considered the significance of PQ trees and the Consecutive ones Problem to Computer Science and bioinformatics and their various applications. We also went further to demonstrate the operations of the characteristics of the Consecutive ones property by simulation, using high level programming languages. Attempt was also made at developing a PQ tree–Consecutive Ones analyzer, which could be instrumental not only as an educative tool to inquisitive students, but also serve as an important tool in developing clustering software in the field of bioinformatics and other application domains, with respect to solving real life problems.

Keywords: PQ trees, Consecutive ones problem, bioinformatics, computer science

INTRODUCTION

Tree is one of the most important and fundamental data structures used in computer science (Ahrabiani and Nowzari-dalinii, 2007), but the applications of a particular type of tree known as the PQ trees, especially in the field of computer science, have been underestimated over the years. Computational biologists and bioinformaticians have employed PQ trees to proffer solutions to several health problems. The solutions to many real life problems in computer science could be efficiently proffered using the PQ tree algorithms, but little or less attention has been paid to it.
The PQ tree data structure was first introduced by Booth and Lueker (1976) in their papers entitled "Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-Tree algorithms (Booth and Lueker, 1976)". In their work, Booth and Lueker (1976) described PQ trees as a data structure for representing the permutations of a set U in which various subsets of U are constrained to occur consecutively. They presented efficient algorithms for manipulating PQ trees. The most desirable property of these algorithms was that they required a number of steps, which were linear in the size of their inputs.
Thus, a PQ tree is an advanced tree-based data structure, which represents a family of permutations on a set of elements. It is a rooted, labeled tree, in which each element is represented by one of the leaf nodes, and each non-leaf node is labeled P or Q. A P node has at least two children, and a Q node has at least three children. The children of a P node may be reordered in anyway. The children of a Q node may be put in reverse order, but may not otherwise be reordered
A matrix M has a consecutive ones property for rows if there exist a permutation of the columns of M such that each row of M, the ones occur consecutively. Thus, there exist a relationship between PQ Trees and the Consecutive Ones Property. Therefore, the PQ Tree Algorithm can be employed as a Clustering tool in solving computationally related problem both in Computer Science, Bioinformatics and Other related areas.
The purpose of this research article is to elaborate more on PQ trees and the Consecutive Ones problem, consider their various applications, to initiate, stimulate research interest in this direction, and highlight the interconnection between them. Finally, it demonstrates a simple transient prototype design of a PQ tree application by applying the programming knowledge of computer science in simulating the operations of the PQ trees and the Consecutive ones property in solving real life problems.

MATERIALS AND METHODS

The PQ tree data structure: It is essential to have a basic understanding of a PQ tree and its corresponding operations. A PQ tree representing permutations of a set U is built step by step. In

each step, a new restriction, given as a subset S C U is introduced. The tree is also altered to represent this restriction (Gørril, 1998).
PQ-trees are rooted trees, whose internal nodes are of two types, called P and Q-nodes. The leaves of the tree correspond to elements of a set U. The PQ-tree is designed to constrain the permutations of elements of U, revealing the permissible permutations with respect to S, a family of subsets of U. Permissible permutations of the set U with respect to S_i ε S, are the permutations where no two elements of Si are separated by an element not belonging to S_i. The PQ-tree ensures this by imposing restrictions on how P- and Q-nodes can rearrange their children (Gørril, 1998).
Definition 1: The universal tree, figure 1, has all elements of U as leaves and children of the root, a single P-node.
Definition 2: Every element a ε U is a tree, consisting of one leaf, with itself as the root.
Definition 3: Every element of U appears exactly once, as a leaf, in the PQ-tree representing U.
Definition 4: P-nodes have at least two children
Definition 5: Q-nodes have at least tree children
Definition 6: Two trees are identical if and only if one can be obtained from
the other by zero or more equivalence transformations. There are two equivalence transformations:

- P-nodes may permute their children arbitrarily.
- Q-nodes may only be flipped over, reversing the order of their children, and always leaving the same two children endmost, and the rest interior.

Definition 7: When a PQ-tree T is restricted by S_i, the tree is said to be reduced with respect to S_i, and the new tree is denoted T (U, S_i) .
Definition 8: Definition (Frontier of a PQ−tree)
Let T be a PQ−tree on {0, . . . , n}. The frontier of T, denoted by fron (T), is the ordered list of signed integers visited during a depth-first traversal of T.
Definition 9: Definition (−permutation et └ −reversal) Let T a PQ−tree on {0, . . . , n}. Performing a O −permutation on a P−node of T consists to shuffle (in any order) its children as well as the signs of the leaves-children. Performing a └ −reversal on a Q−node of T consists to reverse the order of its children (together with a sign change on the leaves-children in the signed case).
Definition 10: Definition (Equivalence of two PQ−trees) Let T and T′ two PQ−trees on a same set. They are said to be equivalent, denoted by T≡ T′, if there exists a sequence of O−permutations and └ −reversals transforming T into T′ (Chateau, 2006).
Description of operations: A PQ tree representing permutations of a set U is built step by step. In each step, a new restriction, given as a subset S ε U is introduced. The tree is also altered to represent this restriction.

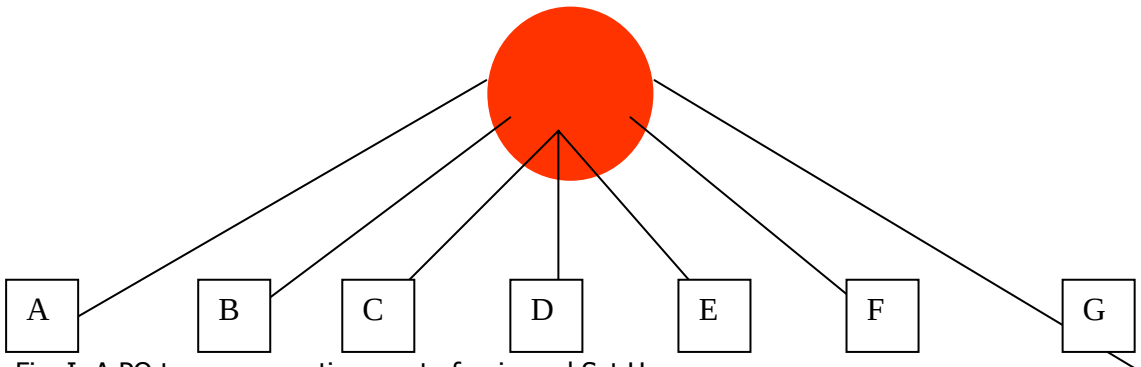

Fig. I: A PQ tree representing a set of universal Set U

However, when a Subset S1= {A, B, C} is introduced, the tree must ensure that the elements of the subset can permute.

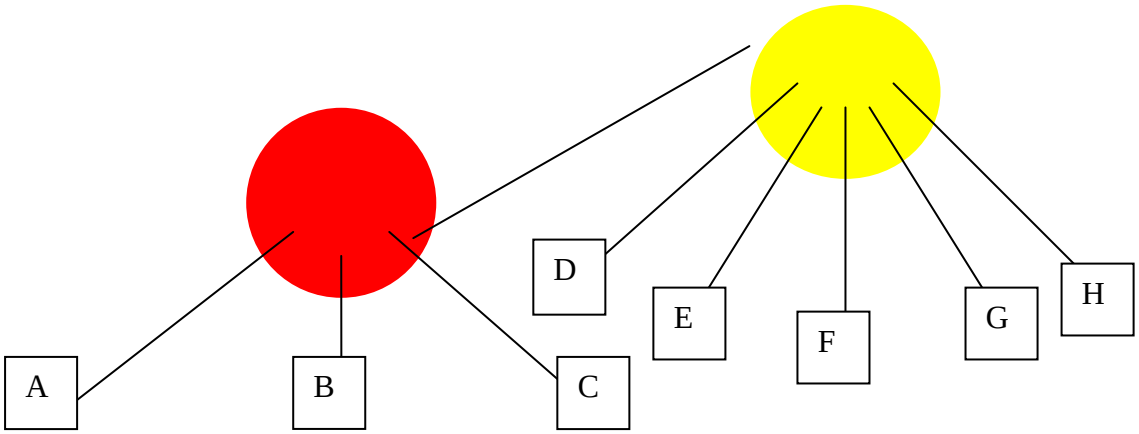

Fig. II: A PQ tree represented when the subset S1 was introduced

When additional subsets (restrictions) [s2= {E,G,H}, s3= {E,D}] are introduced, they may not be disjoint from or contained in the sets already reduced.

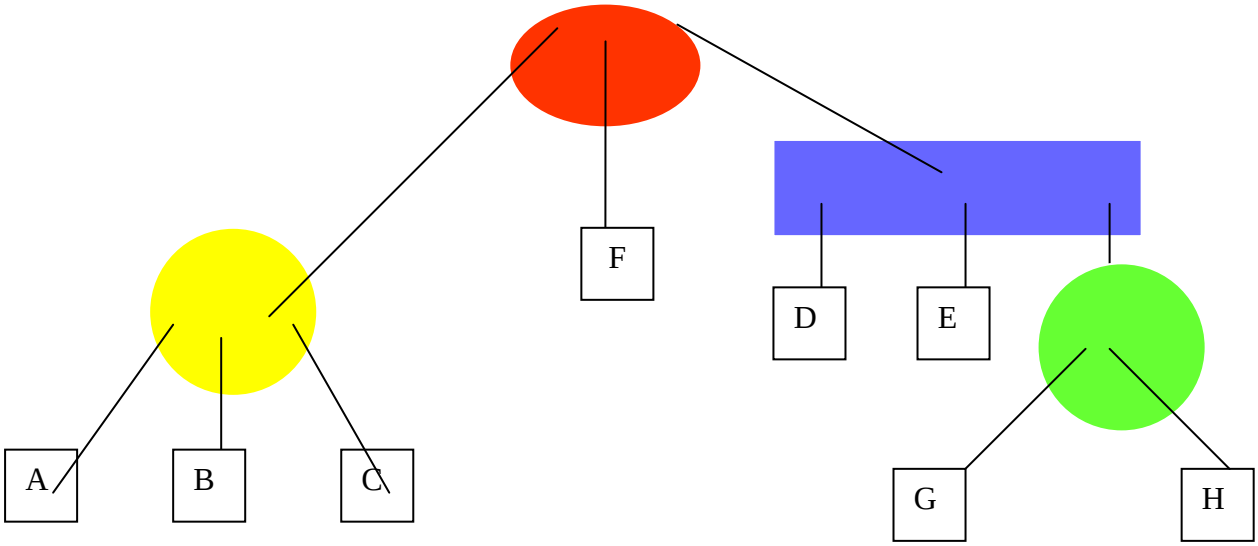

Fig. III: A PQ tree produced when a subset S2 and S3 were introduced

When we apply the subsets s4= {B, C, G} and edges pointing to them are made invisible, we obtain a pruned sub-tree.

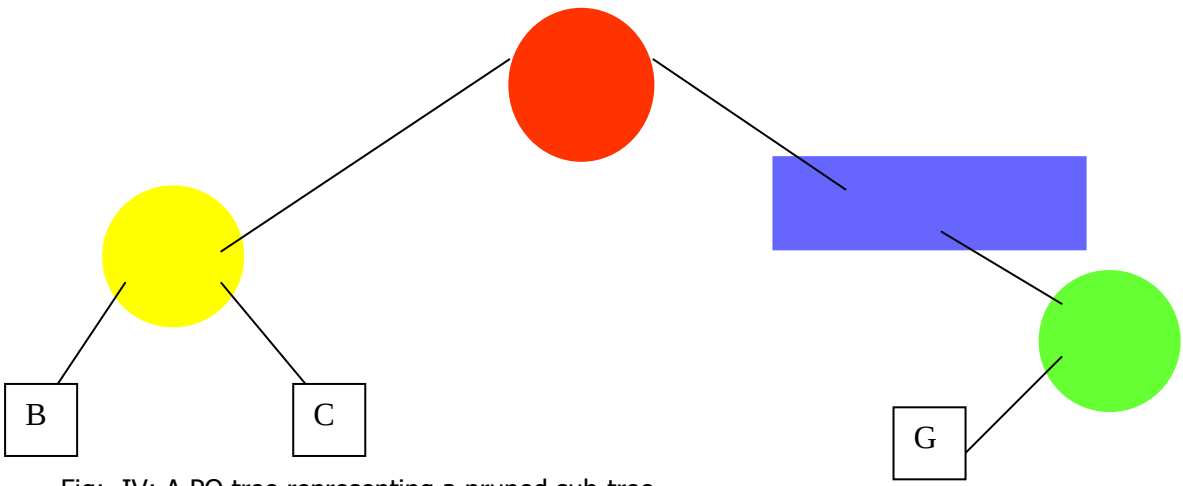

Fig: IV: A PQ tree representing a pruned sub-tree

The Consecutive One's Problem: A Matrix 0/1 M has the Consecutive Ones Property for rows if there is a permutation of the columns of M such that in each row of M, the ones occur

consecutively. Let π € S (n) is such a permutation of the columns. Then we can say that π establishes the consecutive ones property for rows of M.
For example any 0/1 Matrix M with 2 rows consists of at most four (4) different types of columns i.e.)

0 0 1 1
0 1 1 0

If we consider any column permutation π that sorts the columns along the order above; obviously, all such permutations establishes the Consecutive Ones Property of the Matrix M.
Another example,

The Matrix M= 1 1 0
1 0 1
0 1 1

Does not have the Consecutive Ones Property and this can be verified by testing all 6 possible column permutations
Symmetrically, the Consecutive Ones Property can be defined for columns.
Thus, this (4, 3) matrix M has the Consecutive Ones Property.

M = 1 1 1
1 0 0
0 1 0
0 0 1

Since, its transpose helps to establish this
Here, the Consecutive One's Problem comes into play when we try to (test) whether a given matrix has the Consecutive One's Property by checking all of its possible column permutations using an efficient algorithm
The Fundamental Theorem of PQ Trees: Theorem: Let S ε U be a subset of U, and let T be a PQ-tree with exactly the elements of U in its frontier. Let T (U, S) be T reduced with respect to S. The permutations of U permissible by T (U, S) are then exactly those permutations permissible by T in which the elements of S occur consecutively.
This shows that there exist a good relationship between PQ Trees and the Consecutive Ones Property.
Applying PQ tree algorithm in solving the consecutive ones problem

1. For $0 \leq i \leq k$
2. let Mi ={ $S_1, S_2, \ldots, S_i$ }
3. Construct the Universal tree
4. Now reduce the Universal tree gradually with respect to s_i to construct the PQ tree of M_i Perform this reduction steps k times after each other with respect to set s_i. (Reduction algorithm)
5. After the i-th step, the current PQ tree is the one related to Mi.
6. Whenever the null tree arises, (after the reduction step i), the algorithm can be terminated.

The PQ Tree Algorithm (Reduction Algorithm): REDUCE(T,T'): Given a collection T of subsets of N={ 1,2,3……,n} . The function REDUCE (T, T') builds a PQ tree T such that f ε C (T) iff f ε C (T') and every i εT appears as a consecutive substring of f.
The Procedure REDUCE (T, T') will return the null tree if no frontier f ε C (T') is such that every i ε T appears as a consecutive substring of f (Gørril Vollen ,1998).
The PQ tree reduction algorithm consists of two sub-algorithms, namely;
(i) Procedure Bubble
(ii) Procedure Reduce
The PQ tree (Reduction Algorithm)
Procedure Bubble [3]

```
Procedure Bubble (T, S)
begin
    for each leaf X ε2 S do enqueue X;
        while |queue| + BLOCK_COUNT + OFF_THE_TOP > 1 do
            if |queue| = 0 then return T( Ø ; Ø);  {No consecutive
                sequence is  possible}
```

```
          else
                    dequeue X and mark X BLOCKED;
               if X has valid parent pointer or is adjacent to unblocked sibling then
                         mark X UNBLOCKED;
    if X is marked UNBLOCKED then
          if X is ROOT(T; Si) then OFF_THE_TOP := 1;
   else
               Y: = PARENT;
               update blocked siblings of X;
if Y has not been enqueued earlier then enqueue Y ;
       fi;
  else update BLOCK_COUNT;
         fi;;
    od;
   if BLOCK_COUNT = 1 then make PSEUDONODE ROOT(T; Si);
      return T;
   end;
```

The PQ tree (Reduction Algorithm)

Procedure Reduce [3]

```
Procedure Reduce (T, S)
begin
                for each leaf X  ε S do enqueue X;
                     while |queue|>0 do
                                dequeue X;
          if X is ROOT(T, Si ) then
                    if some template for ROOT(T, Si) applies to X then
                              substitute the replacement for X in T;
                                   else return T( Ø ; Ø );
               else {X is not ROOT(T, Si)}
if some template for nodes not ROOT(T, Si) applies to X then
                    substitute the replacement for X in T;
           else return T(Ø ;Ø );
     fi;
if ROOT(T, Si ) is reached then return T;
          else if every pertinent sibling of X has been matched then
          enqueue the parent of X;
      od;
end;
```

PQ tree as a clustering tool: Clustering can be defined as the process of organizing objects into groups whose members are similar in some ways. Considering Figures I-IV, it will be observed that the PQ tree can gradually assume the characteristics of a clustering tool, based on some specified parameters. Thus, clustering has to do with arrangement.

Booth and Leuker (1976), introduced the PQ tree data structure as a tool in solving the general consecutive arrangement problem. General Consecutive Arrangement Problem states that: Given a finite set X and a collection T of subsets of X, does there exist a permutation π of X in which the members of each subset I € T appears as a consecutive substring of π?

Booth and Leuker (1976) introduced an algorithm (Reduction Algorithm to solve this problem which is linear in the length of the input (O (n^2))

A graphical illustration of Clustering

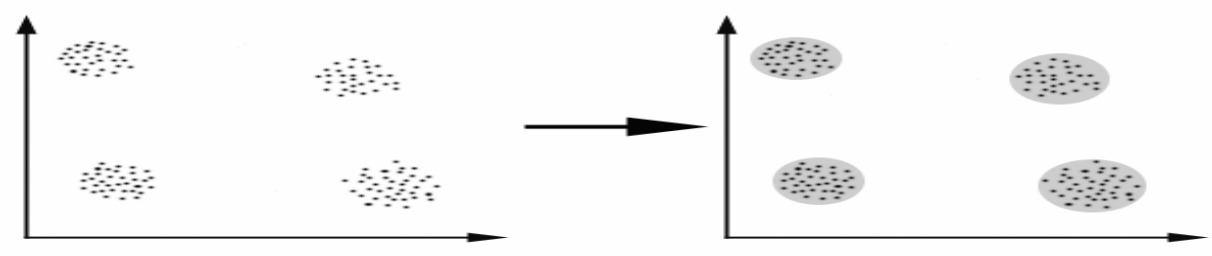

Fig. V: Graphical illustration of clustering

Importance of clustering

- Understanding the current data we are dealing with and finding groups.
- It helps to aid feature extraction for classification
- It also supports Summarization /Compression

There are several scientific works involving the use of PQ trees that have been carried out in the past by scientists in computer science and bioinformatics. They are as follows:

(i) Efficient sub-typing tests with PQ-encoding

In this research work, a new scheme for encoding multiple and single inheritance hierarchies was developed. This scheme is called PQ–encoding after the PQ tree data structure which was previously used in graph theory (Zibin and Gil, 2001).

(ii) PQ-trees and the Set of All Distinct Planar Embeddings of a Graph

In this research work, a forest of PQ tree was used to efficiently represent the set of all distinct planar embeddings of a bi-connected graph, a linear time algorithm was then given for computing this representation (Stallmann, 1986).

(iii) Erratum: The Travelling Salesman and the PQ-Tree

In their research work, they investigated specially structured sets of permutations that could be represented via PQ-trees, a well-known data structure from Theoretical Computer Science. They developed a dynamic programming algorithm for finding the shortest pyramidal Traveling Salesman Problem tour (Rainer *et al.,* 1999).

(iv) Planarity Algorithms via PQ-Trees

In their research work, they gave a linear-time implementation that simplified and unified the Shih-Hsu and Boyer-Myrvold methods. Their algorithm extended to generate embeddings uniformly at random, to count embeddings, to represent all embeddings, and to produce a Kuratowski subgraph of a non-planar graph. Their algorithm kept track of all possible embeddings by re-interpreting Booth and Lueker's PQ-tree data structure to represent circular instead of linear orders. This interpretation of PQ-trees gave the PC-trees of Shih and Hsu and leads to a simpler, more-symmetric form of PQ-tree reduction (Bernhard and Tarjan, 2008).

(v) Discovering functional gene expression patterns in the metabolic network of Escherichia coli with wavelets transforms

Part of this bioinformatics research work was carried out using PQ trees algorithm (Rainer *et al.,* 2006).

(vi) Gene Proximity Analysis Across Whole Genomes via PQ Trees

In this research paper, they presented a new tool for the representation and detection of gene clusters in multiple genomes, using PQ trees (Gad *et al.,* 2005). This described the inner structure and the relations between clusters succinctly, aided in filtering meaningful from apparently meaningless clusters and also gave a natural and meaningful way of visualizing complex clusters.

Implementation: As a means of contributing to knowledge, we developed a transient prototype for a Consecutive ones – PQ tree analyzer and also a program to test for the consecutive ones property in 0/1 matrices.

Java program codes

Java Codes for the PQ tree-Consecutive Ones Analyzer

```
/*
 * MenuTest.java
 *
 * Created on November 12, 2007, 5:06 PM
 *
 * To change this template, choose Tools|Template Manager
 * and open the template in the editor.
 */

/**
 *
 * @author Gbenga
 */
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;

public class MenuTest {
```

```
      public static void main(String[] args) {

       MenuFrame frame = new MenuFrame();
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.setVisible(true);

               }

}

// design of the frame with the menu bar

class MenuFrame extends JFrame
{

   public MenuFrame()
   {
        setTitle("Consecutive Ones's- PQTree Analyzer-Designed By
           Oluwagbemi Olugbenga");
        setSize(DEFAULT_WIDTH,DEFAULT_HEIGHT);

        JMenu matrixMenu = new JMenu("MATRIX");
        JMenuItemnewItem = matrixMenu.add(new TestAction("Number_Of_Rows"));
        newItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_R,
InputEvent.CTRL_MASK));
        //demonstrate accelerators

        JMenuItem openItem = matrixMenu.add(new TestAction("Number_Of_Columns"));
        openItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_C,
InputEvent.CTRL_MASK));
        matrixMenu.addSeparator();

        matrixAction = new TestAction("Matrix_Dimension");
        JMenuItem saveItem = matrixMenu.add(matrixAction);
        saveItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_D,
InputEvent.CTRL_MASK));

        matrixAsAction = new TestAction("Matrix_Type");
        JMenuItem saveAsItem = matrixMenu.add (matrixAsAction);
        matrixMenu.addSeparator ();

        matrixMenu.add(new
              AbstractAction("Exit")
        {
             public void actionPerformed(ActionEvent event)
               {
                  System.exit(0);
               }
        });

        //

        readonlyItem = new JCheckBoxMenuItem("Read-only");
        readonlyItem.addActionListener(new
              ActionListener()

                 {
                      public void actionPerformed(ActionEvent event)
                          {
                                boolean saveOk = !readonlyItem.isSelected();
```

```
                        matrixAction.setEnabled(saveOk);
                        matrixAsAction.setEnabled(saveOk);
                    }
                });

    ButtonGroup group = new ButtonGroup();

    JRadioButtonMenuItem insertItem = new JRadioButtonMenuItem("");
                insertItem.setSelected(true);
    JRadioButtonMenuItem overtypeItem = new JRadioButtonMenuItem("");

                group.add(insertItem);
                group.add(overtypeItem);

                //
                Action  PnodeAction = new TestAction("P-NODE");
                PnodeAction.putValue(Action.SMALL_ICON, new ImageIcon("Pnode.gif"));
                Action QnodeAction = new TestAction("Q-NODE");
                QnodeAction.putValue(Action.SMALL_ICON, new
                  ImageIcon("Qnode.gif"));
                Action FrontiersAction = new TestAction("Frontiers");
                FrontiersAction.putValue(Action.SMALL_ICON, new
                   ImageIcon("paste.gif"));

                JMenu nodeMenu = new JMenu("NODES");
                nodeMenu.add(PnodeAction);
                nodeMenu.add(QnodeAction);
                nodeMenu.add(FrontiersAction);

                 //
                JMenu optionMenu = new JMenu("Options");

                optionMenu.add(readonlyItem);
                optionMenu.addSeparator();
                optionMenu.add(insertItem);
                optionMenu.add(overtypeItem);

                nodeMenu.addSeparator();
                nodeMenu.add(optionMenu);

                //

JMenu TestMatrixMenu = new JMenu ("TEST FOR CONSECUTIVE ONES PROPERTY");
                TestMatrixMenu.setMnemonic('T');

                JMenu PQTreeMenu = new JMenu ("DRAW THE PQTree");
                PQTreeMenu.setMnemonic('D');

                JMenuItem indexItem = new JMenuItem("Smaller_Diagram");
                indexItem.setMnemonic('I');
                PQTreeMenu.add(indexItem);

                // you can also add the mnemonic key to an action

                Action aboutAction = new TestAction("Bigger_Diagram");
                aboutAction.putValue(Action.MNEMONIC_KEY,new Integer('A'));
                PQTreeMenu.add(aboutAction);

                //add all top-level menus to menu bar
```

```
                JMenuBar menuBar =  new JMenuBar();
                setJMenuBar(menuBar);

                menuBar.add(matrixMenu);
                menuBar.add(nodeMenu);
                menuBar.add(TestMatrixMenu);
                menuBar.add(PQTreeMenu);

                //demonstrate pop-ups

                popup = new JPopupMenu();
                popup.add(PnodeAction);
                popup.add(QnodeAction);
                popup.add(FrontiersAction);

                JPanel panel = new JPanel();

                panel.setComponentPopupMenu(popup);
                add(panel);

            // the following is a workaround for bug 4966109
             panel.addMouseListener(new MouseAdapter() {});
   }

   public static final int DEFAULT_WIDTH = 300;
   public static final int DEFAULT_HEIGHT = 200;

   private Action matrixAction;
   private Action matrixAsAction;
   private JCheckBoxMenuItem readonlyItem;
   private JPopupMenu popup;

}

//A sample action that prints the action name to System output

class TestAction extends AbstractAction
{
      public TestAction (String name) { super(name); }

         public void  actionPerformed(ActionEvent event)
         {
              System.out.println(getValue(Action.NAME) + " selected.");


         }
}
```

C++ program codes for the program to test for the consecutive ones property in a 0/1 matrices

```
//  The program written and compiled by Ayo  Dawodu
#include<iostream.h>
#include<stdlib.h>
#define MAX 99
int mata[MAX][MAX];
int temp[9];
void main()
{
                 int a,b,i,j,k=0;
```

```
                cout << "Enter the no of rows and colunm of the matrix \n";
                cout << "ROWS: ";
                cin >>  a;
                cout << "COLUMNS: ";
                cin >> b;
                cout << "\nMatrix A is a "<< a <<" by "<<b<<" matrix!\n";
                cout<< "You are to enter "<<a*b<<" Values!\n";


                cout << "Enter the values in 0s and 1s, PRESS ENTER AFTER EACH
VALUE:\n";

                //inputting the matrix
                for (i=0;i<a;i++)
                {
                         //cout <<"\t";
                        for (j=0;j<b;j++)
                        {
                                //cout <<"\t";
                                cin>> mata[i][j];
                        }
                }

//displaying the matrix
      cout <<"\nHere's the matrix:->\n";
                for (i=0;i<a;i++)
                        {
                        cout <<"\n\t";
                              for (j=0;j<b;j++)
                                 {
                                       cout<<mata[i][j];
                                       cout <<"\t";
                                 }
                             cout<<"\n";
                        }

                //Checking for consecutive ones property
                cout <<"\nChecking through for 'the consecutive ones property'...\n";
                int z=0;
                for(j=0;j<a;j++)
                   {
                         k=0;
                           for(i=0;i<b;i++)
                             {
                                   temp[k]=mata[i][j];
                                      if(k>0)
                                {
                                      if(temp[k]==temp[k-1])
                                      z+=1;
                                 }
                           k++;
                        }
                  if(z==2)
                  {
cout<<"\n    The matrix above possess the \"CONSECUTIVE ONE PROPERTY\"\n";
                        exit(0);
                        break;
                  }
                  else
```

```
                {
                        z=0;
                        goto t;
                }
            }
          //transposing to re-check
                t:cout<<"\nTRANSPOSING.....\n";
                  cout <<"\t";
                   for (i=0;i<b;i++)
                        {
                            cout <<"\n\t";
                                for (j=0;j<a;j++)
                                    {
                                        cout<<mata[j][i];
                                        cout <<"\t";
                                    }
                                cout<<"\n";
                        }
                      cout<<"\nRe-checking for \"CONSECUTIVE ONES PROPERTY\"...\n";
                z=0;
        for(j=0;j<a;j++)
                {
              for(i=0;i<b;i++)
                {
                      temp[k]=mata[i][j];
                        if(k>0)
                        {
                                if(temp[k]==temp[k-1])
                                z+=1;
                        }
                     k++;
                }
               }
               if (z==2)
                {
cout<<"\nThe Transpose of the matrix above possess the \"CONSECUTIVE ONE PROPERTY\"\n";
                    exit(0);
                }
               else
                {
 cout << "\n The matrix in both forms does not possess the \"CONSECUTIVE ONE PROPERTY\"\n";
                }

}
```

RESULTS AND DISCUSSION

Results of the Java codes

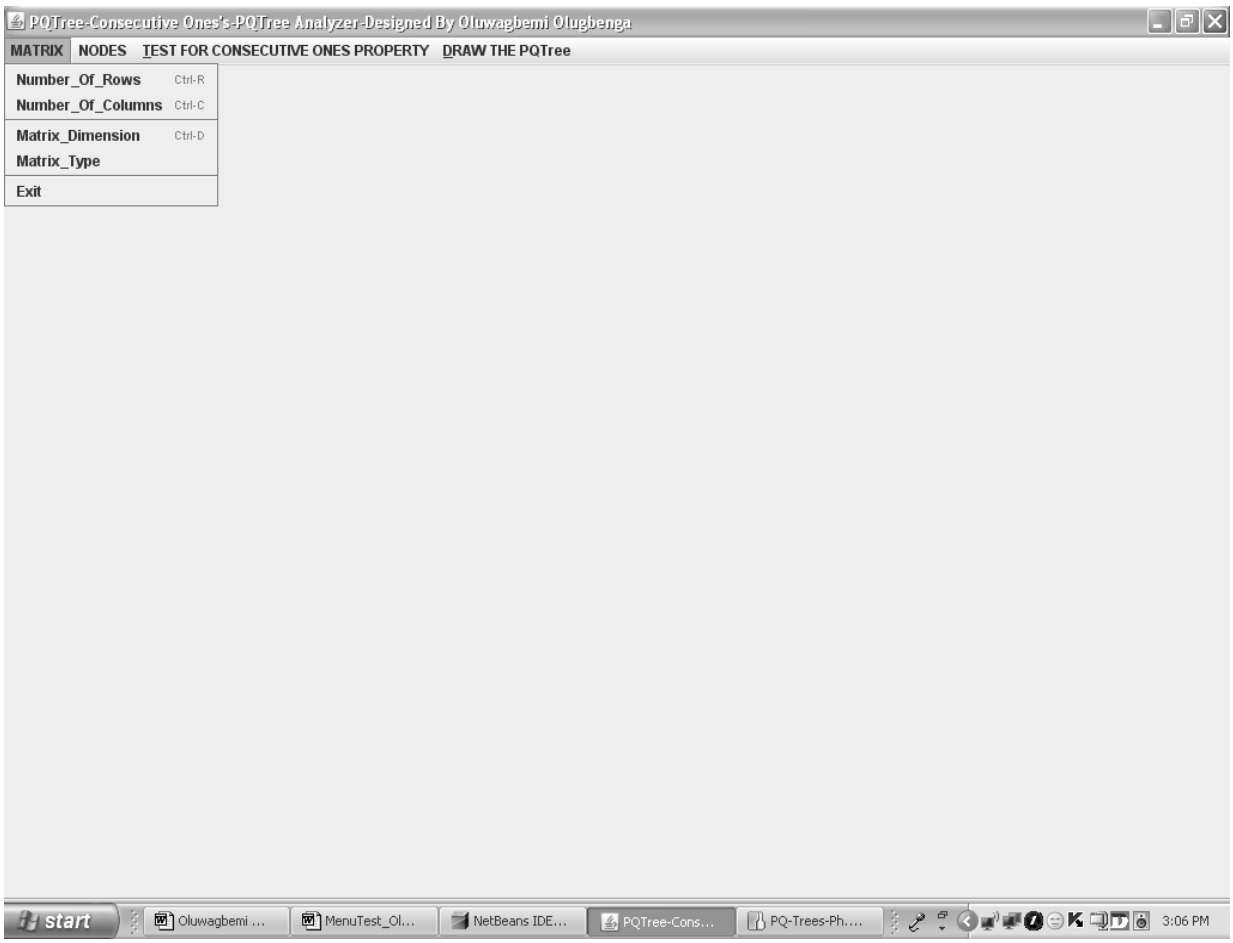

Fig. VI: Display of the input for the number of rows, columns, matrix dimensions and the matrix type

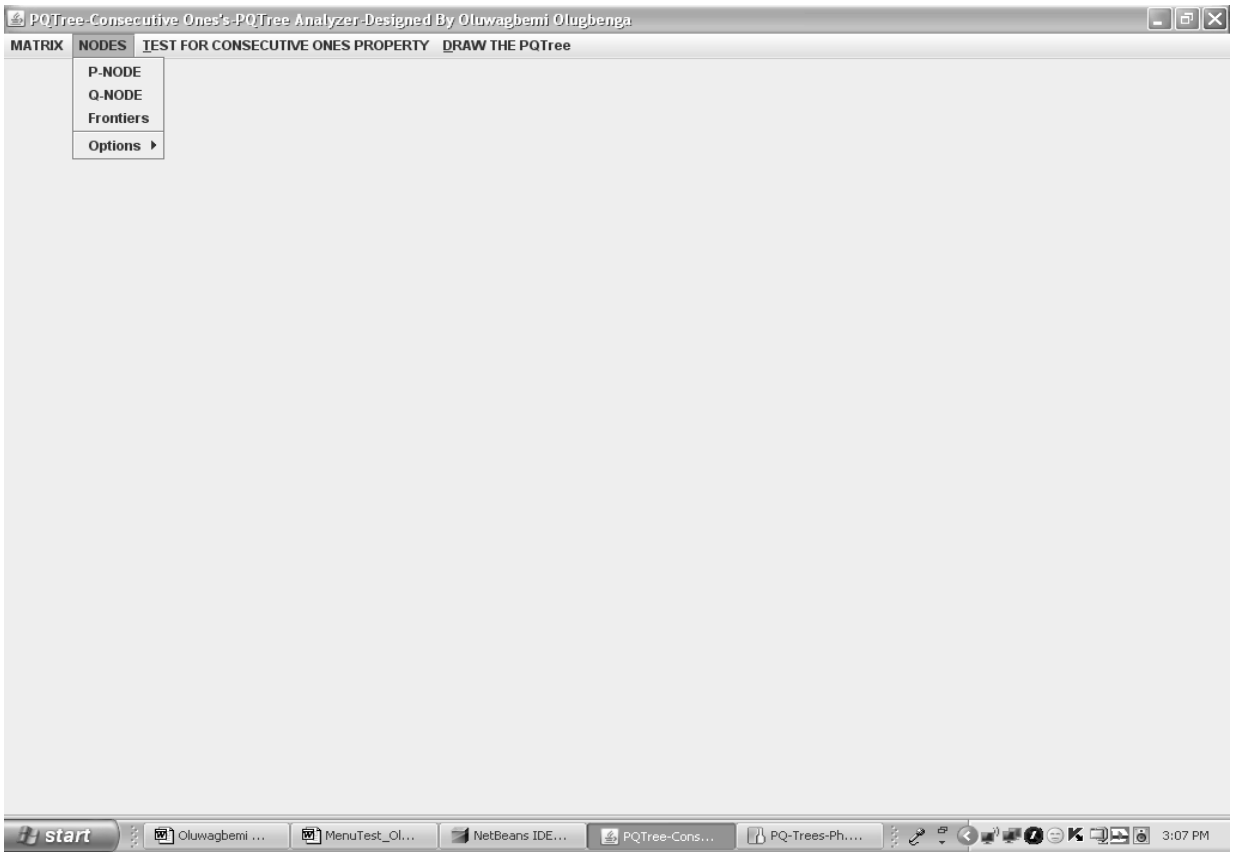

Fig. VII: Display of the design for P-node and Q-node

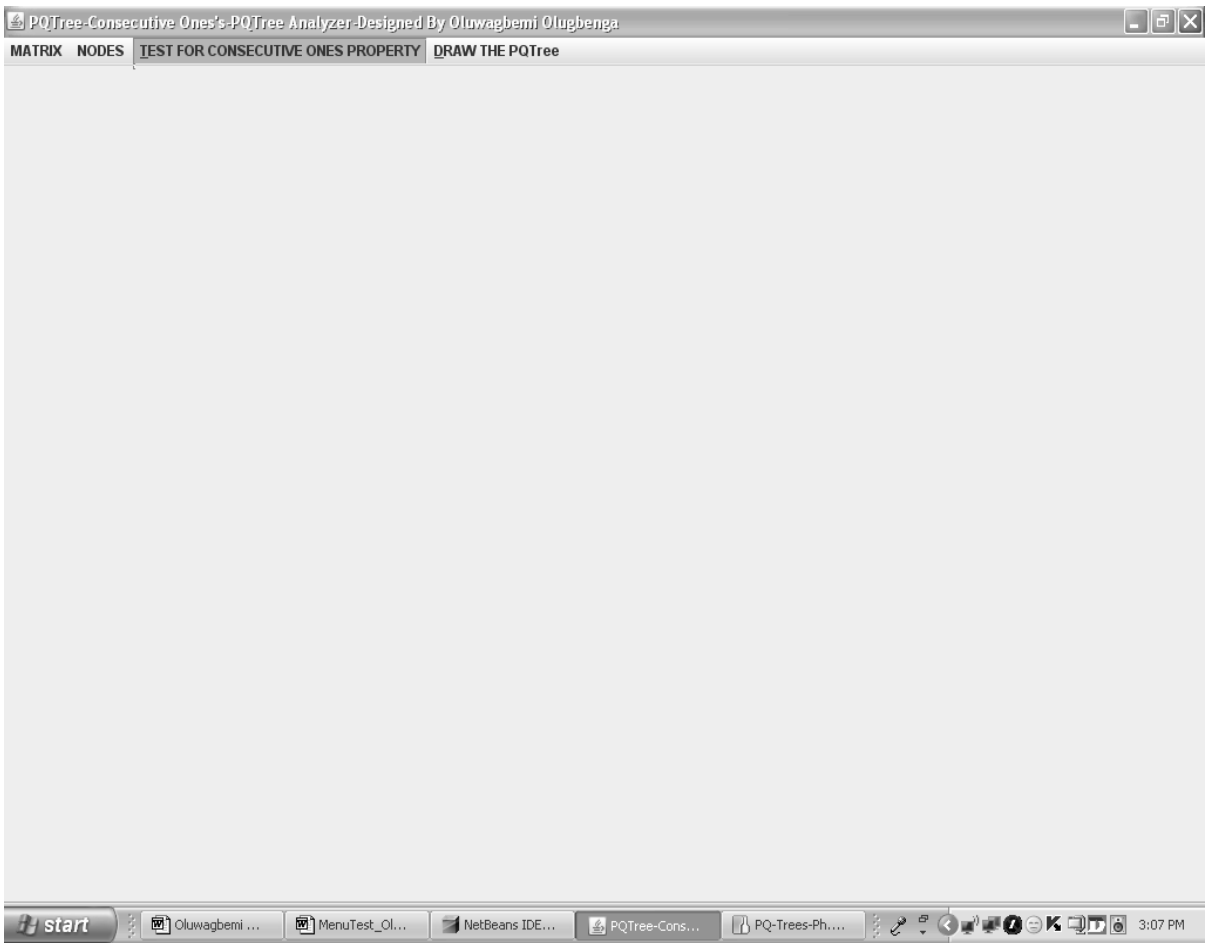

Figure VIII. Display of the module to test for the Consecutive ones property.

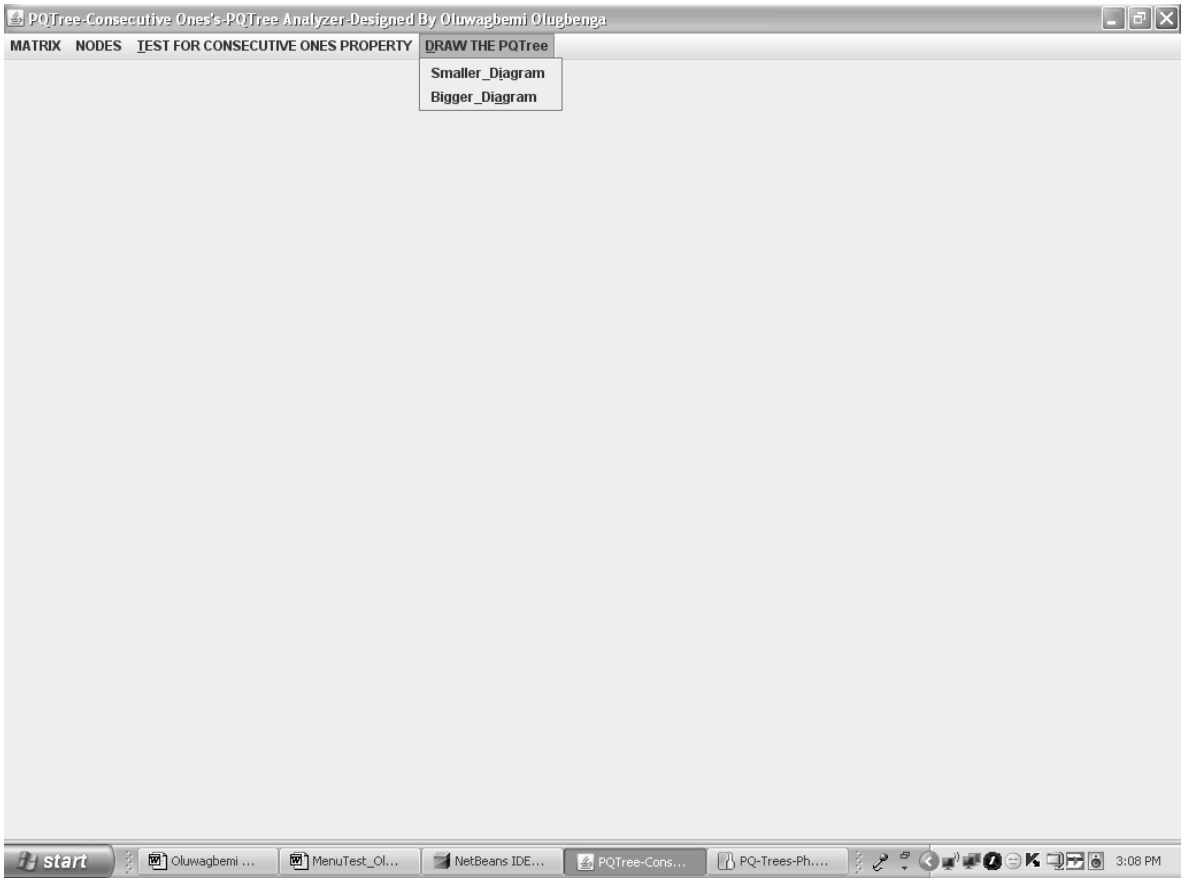

Fig. IX: Display of the module for displaying the PQ tree for a given consecutive ones matrix.

Results of the C++ codes

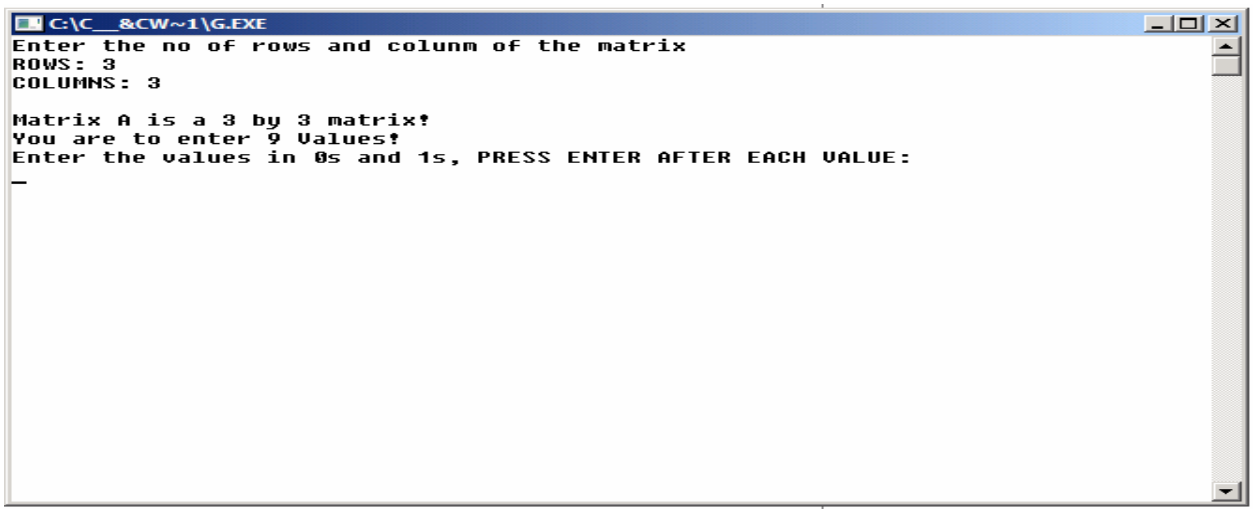

Fig. X: C++ program to display the matrix entry

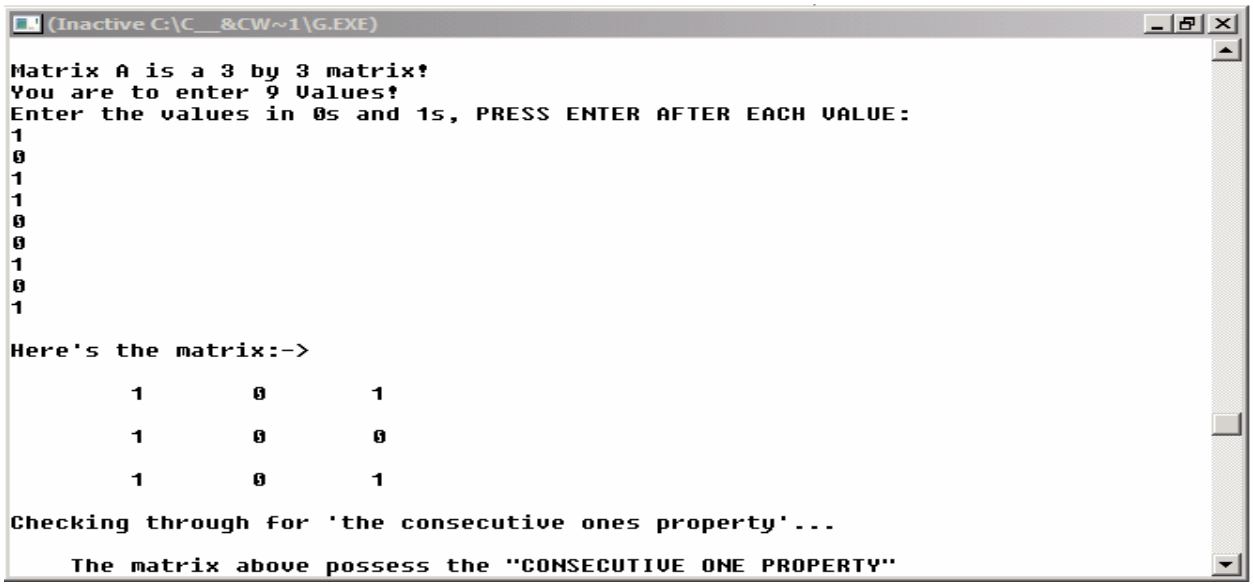

Fig. XI: C++ program to test for the consecutive ones property and display the matrix satisfying this property

Applicational areas of PQ trees

- In Computer Science, it can be used for the purpose of Data mining of complex and large data sets.
- It can be used in creating contig map from DNA fragments in Bioinformatics research.
- It can be used to recognize interval graphs and to determine whether a graph is planar.
- It can also be applied to test the consecutive ones property of (0, 1) matrices.
- It can be used in the representation and detection of Gene Clusters in Multiple Genomes.

Applications of PQ tree as a clustering tool

- Marketing: it can be used in finding groups of customers with similar behavior.
- In Biology, it can serve as a very important tool in the classification of plants and animals given their features.
- In secondary and tertiary institutions, it can be very useful in Libraries, especially for book ordering.
- In Insurance, it can be used in identifying groups of motor insurance policy holders.
- In Earthquake studies, PQ trees can be used in developing clustering softwares in order to observe and give earthquake alerts as regards earthquake epicenters to identify dangerous zones;

CONCLUSION

PQ Tree Algorithm serves as a very efficient algorithm which can be used in solving the Consecutive Ones Problem and it is also useful as a clustering technique in Bioinformatics, Computer Science and other areas.
Future work can be done in the following areas:
• Development of a fully functional PQ Tree Analyzer
•Development of a fully functional PQ Tree Clustering Software which can be applied to Marketing, Earthquake zone detection through clustering techniques, Banking, Biology in clustering plants and animals ,etc
• Development of fully-functional Software for Discovering the Functional Gene Expression Patterns in the Metabolic Network of *E. Coli.*

REFERENCE

Ahrabiani, H. and Nowzari-dalinii, A. (2007). *Parallel generation of t-Ary trees in A-order.* Oxford University Press, United Kingdom.

Booth, K. S. and Lueker, G. S. (1976). Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree Algorithms. *Journal of Computer and System Sciences,* 13(3): 335-379.

Chateau, A. (2006). Conserved intervals, Common intervals and PQ-trees, October 17, 2006

Gad, M., Landau, L. P. and Oren, W. (2005). Gene proximity analysis across whole genomes via PQ Trees. *Journal of Computational Biology,* 12(10): 1289-13.

Gørril, V. (1998). PQ-trees and maximal planarization: An approach to skewness and Scient. Thesis, University of Oslo, Department of Informatics.

Haeupler, B. and Tarjan, R. E. (2008). Planarity algorithms via PQ-Trees. Department of Computer Science, Princeton University, Princeton NJ.

König, R., Schramm, G., Oswald, M., Seitz, H., Sager, S., Zapatka, M., Reinelt, G. and Eils, R. (2006). Discovering functional gene expression patterns in the metabolic network of Escherichia coli with wavelets transforms. *BMC Bioinformatics,* 7: 119-133.

Rainer, E., Burkard, V., G. D. and Gerhard J. W. (1999). Erratum: The traveling salesman and the PQ-tree. *Mathematics of Operations Research,* 24(1):

Stallmann, M. (1986). PQ-trees and the set of all distinct planar embeddings of a graph. Department of Computer Science, North Carolina State University Raleigh, Sort h Carolina.

Zibin, Y. and Gil, J. (2001). Efficient Sub-typing tests with PQ encoding. In: Proceedings of the 16th ACM Conference on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 2001). Available at http://citeseer.ist.psu.edu/zibin01efficient.html