Recursive Algorithm for Statistical Analysis of Bivariate Data Using Abstract Data Type Binary Tree

O.E. Oguike
Department of Computer Science
University of Nigeria
Nsukka, Nigeria
osondu.oguike@unn.edu.ng

ABSTRACT
Bivariate data can be defined as a collection of n pairs of data. Bivariate numeric data can be very useful in statistical analysis because it can be used to compute some important statistics, like correlation coefficient, regression analysis parameters, covariance etc. Though the recursive algorithms that will use binary tree to store and manipulate bivariate data will have double recursion, it has become a matter of academic concern to consider how the binary tree can be used to store and manipulate bivariate data. This paper develops novel algorithms for the operations of a binary tree that has two data items at the node, and it uses these novel algorithms to develop novel recursive algorithms that will compute some important bivariate statistics from the bivariate data stored in a binary tree.

Keywords- Bivariate data, correlation, regression analysis, covariance, statistical analysis, binary tree, recursive algorithm, abstract data type.


1. INTRODUCTION

Binary tree can be used to store, access and manipulate a collection of bivariate data. Though the traditional binary tree [3] requires that the nodes store a single data item, the binary tree that will be used to store bivariate data will have two data items at the node of each subtree. This implies that the operations of this binary tree will be extended from the traditional binary tree that each node stores single data item, to a binary tree that each node stores a pair of data item. Furthermore, the extension can continue in order to store multivariate collection of data. Data stored in a binary tree are stored in such a way that when they are accessed, the data will be in ascending or descending order. Figure 1 shows a collection of bivariate data in a tree.

Though tree is a traditional abstract data type with its access operations, it can be regarded as a nature-inspired model of computation. It depicts a natural tree, with its branches forming the left and right subtrees, and its fruits are the data that are stored at the nodes of the tree. The famous cocoa tree found in the western part of Nigeria is an example of such natural tree.

Figure 1: Bivariate Data in a Binary Tree Structure
The importance of bivariate statistical analysis cannot be overemphasized; this is because the various statistics that we can obtain from bivariate data are very useful in statistical analysis. Example, the bivariate regression analysis statistics help us to develop prediction model that can be used to predict a dependent variable, given an independent variable, the bivariate correlation coefficient can be used to establish the type of relationship that exist between two variables, while the statistic, covariance can be used to establish the extent of variation of bivariate data.

2. SURVEY OF RELATED LITERATURE

On the definition of abstract data type, the authors in [5] defined it as an abstract model of data objects with a formal encapsulation of logical architecture and valid operations of the data object. According to them, ADT encapsulates a data object and presents the user with an interface through which data can be accessed. In [6], the authors opined that a type might be viewed as a set of clothes (or a suit of armor) that protects an underlying untyped representation from arbitrary or unintended use. It provides a protective covering that hides the underlying representation and constrains the way objects may interact with other objects.

According to them, in an untyped system, untyped objects are naked in that the underlying representation is exposed for all to see. Even on parallel machines, writing on the importance of abstraction, the author in [7] said that abstraction is essential to enable clean separation of application code from the assembly-level mechanisms coordinating parallel execution, thus encapsulating the issues of communication, synchronization, and distribution. Still on the importance of data abstractions, [8] maintained that it improves the modularity of programs by encapsulating implementation details, and by providing a clear delineation between design and implementation. It has been noted in [10] that the abstract data type facility (ADT) in a programming language allows the user to create a collection of representing types, and functions defined on these types, hiding the representation of the types, and the implementation of the functions, and allowing only the use of the functions defined.

Though it has been argued that the recursive approach to transversing a binary tree is not very efficient [9], but one should also consider other factors for the choice of recursive approach, which include, easy to implement, understand, debug and very structured when implemented. The statistical formulae that compute the various bivariate statistics have been expressed in different, but equivalent forms in [1],[2],[4].

3. DESIGN OF ALGORITHM FOR THE OPERATIONS OF ABSTRACT DATA TYPE TREE

The algorithms that will be used to store and access bivariate data in the abstract data type tree need to be defined. The structure of this tree will be defined in a java class called stattree as follows:

\[
\text{Stattree leftree} \\
\text{Double datax} \\
\text{Double datay} \\
\text{Stattree rightree}
\]

The algorithms for the operations of the abstract data type, tree can be defined as follows:

\[
\text{stattree emptytree}() \\
1. \text{Determine emptytree} \\
1.2 \text{emptytree} = \text{null}; \\
2. \text{Display emptytree}
\]

The algorithm emptytree takes nothing as parameter, but it returns a tree that does not have bivariate data, and does not have the left and right subtrees.

\[
\text{boolean istreeempty(stattree t)} \\
1. \text{Request t} \\
2. \text{Determine istreeempty} \\
2.1 \text{IF t = null THEN} \\
2.1.1 \text{istreeempty} = \text{true}; \\
2.1.2 \text{ELSE istreeempty} = \text{false} \\
3. \text{Display istreeempty}
\]

The algorithm, istreeempty takes a tree as parameter, and it returns true if the tree is empty, otherwise it returns false.

\[
\text{stattree makenode(double x, double y, stattree l, stattree r)} \\
1. \text{Request data} \\
1.1 \text{Request x} \\
1.2 \text{Request y} \\
1.3 \text{Request l} \\
1.4 \text{Request r} \\
2. \text{Determine makenode} \\
2.1 \text{Declare and allocate, temp of the type stattree} \\
2.2 \text{temp.leftree} = l; \\
2.3 \text{temp.datax} = x; \\
2.4 \text{temp.datay} = y; \\
2.5 \text{temp.rightree} = r; \\
2.6 \text{makenode} = \text{temp}
3. \text{Display makenode}
\]
The algorithm, makenode makes a new node of the defined tree structure. It takes four parameters, two real numbers that make up the bivariate data and two trees that make up the left and the right subtrees. The algorithm puts the parameters in their positions in the new node and returns the new node.

\[
\text{double treedatax(stattree } t) \]
\[
1. \ \text{Request } t \\
2. \ \text{Determine treedatax} \\
2.1 \ \text{IF istreeempty}(t) \ \text{THEN} \\
2.1.1 \ \text{treedatax} = \text{"Empty tree does not have any data"}; \\
\text{ELSE} \\
2.1.2 \ \text{treedatax} = t.datax \\
3. \ \text{Display treedatax}
\]

The algorithm, treedatax takes a tree as parameter and it returns the first part of the bivariate data at the node of the tree.

\[
\text{double treedatay(stattree } t) \]
\[
1. \ \text{Request } t \\
2. \ \text{Determine treedatay} \\
2.1 \ \text{IF istreeempty}(t) \ \text{THEN} \\
2.1.1 \ \text{treedatay} = \text{"Empty tree does not have any data"}; \\
\text{ELSE} \\
2.1.2 \ \text{treedatay} = t.datay \\
3. \ \text{Display treedatay}
\]

The algorithm, treedatay takes a tree as parameter and it returns the second part of the bivariate data at the node of the tree.

\[
\text{stattree leftsubtree(stattree } t) \]
\[
1. \ \text{Request } t \\
2. \ \text{Determine leftsubtree} \\
2.1 \ \text{IF istreeempty}(t) \ \text{THEN} \\
2.1.1 \ \text{leftsubtree} = t \\
\text{ELSE} \\
2.1.2 \ \text{leftsubtree} = t.leftree \\
3. \ \text{Display leftsubtree}
\]

The algorithm leftsubtree takes a tree as parameter and it returns the left subtree of that tree if the tree is not empty, otherwise it returns the tree.

\[
\text{stattree rightsubtree(stattree } t) \]
\[
1. \ \text{Request } t \\
2. \ \text{Determine rightsubtree} \\
2.1 \ \text{IF istreeempty}(t) \ \text{THEN} \\
2.1.1 \ \text{rightsubtree} = t \\
\text{ELSE} \\
2.1.2 \ \text{rightsubtree} = t.rightree \\
3. \ \text{Display rightsubtree}
\]

The algorithm rightsubtree takes a tree as parameter and it returns the right subtree of that tree if the tree is not empty, otherwise it returns the tree.

\[
\text{stattree insertree(double } x, \ \text{double } y, \ \text{stattree } t) \]
\[
1. \ \text{Request data} \\
1.1 \ \text{Request } x \\
1.2 \ \text{Request } y \\
1.3 \ \text{Request } t \\
2. \ \text{Determine insertree} \\
2.1 \ \text{IF istreeempty}(t) \ \text{THEN} \\
2.1.1 \ \text{insertree} = \text{makenode}(x, y, \text{null, null}); \\
\text{ELSE} \\
2.1.2 \ \text{IF } (x < \text{treedatax}(t)) \ \text{THEN} \\
2.1.2.1 \ \text{insertree} = \text{makenode}(\text{treedatax}(t), \text{treedatay}(t), \text{insertree}(x, y, \text{leftsubtree}(t)), \text{rightsubtree}(t)); \\
\text{ELSE} \\
2.1.2.2 \ \text{insertree} = \text{makenode}(\text{treedatax}(t), \text{treedatay}(t), \text{leftsubtree}(t), \text{insertree}(x, y, \text{rightsubtree}(t))); \\
3 \ \text{Display insertree}
\]

The algorithm insertree takes three parameters, which are the two real numbers that form the pair of data and a tree, the algorithm insertree makes new node and put the node at the correct position in the binary tree.

4. DESIGN OF RECURSIVE ALGORITHM FOR BIVARIATE STATISTICAL ANALYSIS

The algorithms that will calculate the following bivariate statistics, covariance, correlation and bivariate regression analysis parameters will be designed. The following algorithms will use the algorithms for the operations of binary tree to design the algorithms, which will be used to compute the three bivariate statistics mentioned above.
Double sumpro(stattree t)
1. Request t
2. Determine sumpro
   2.1 IF istreeempty(t) THEN
   2.1.1 sumpro = 0
   ELSE
   2.1.2 sumpro = treedatax(t)*treedatay(t) + sumpro(leftsubtree(t)) + sumpro(rightsubtree(t))
3. Display sumpro

The algorithm sumpro takes a tree as parameter and it returns the sum of the product of the bivariate data in the tree.

Double sumx(stattree t)
1. Request t
2. Determine sumx
   2.1 IF istreeempty(t) THEN
   2.1.1 sumx = 0
   ELSE
   2.1.2 sumx = treedatax(t) + sumx(leftsubtree(t)) + sumx(rightsubtree(t))
3. Display sumx

The algorithm sumx takes a tree as parameter and it returns the sum of datax attribute of the bivariate data in the tree.

Double sumy(stattree t)
1. Request t
2. Determine sumy
   2.1 IF istreeempty(t) THEN
   2.1.1 sumy = 0
   ELSE
   2.1.2 sumy = treedatay(t) + sumy(leftsubtree(t)) + sumy(rightsubtree(t))
3. Display sumy

The algorithm sumy takes a tree as parameter and it returns the sum of datay attribute of the bivariate data in the tree.

Double sumsqx(stattree t)
1. Request t
2. Determine sumsqx
   2.1 IF istreeempty(t) THEN
   2.1.1 sumsqx = 0
   ELSE
   2.1.2 sumsqx = treedatax(t)*treedatax(t) + sumsqx(leftsubtree(t)) + sumsqx(rightsubtree(t))
3. Display sumsqx

The algorithm sumsqx takes a tree as parameter and it returns the sum of square of the datax attribute of the bivariate data in the tree.

Double sumsqy(stattree t)
1. Request t
2. Determine sumsqy
   2.1 IF istreeempty(t) THEN
   2.1.1 sumsqy = 0
   ELSE
   2.1.2 sumsqy = treedatay(t)*treedatay(t) + sumsqy(leftsubtree(t)) + sumsqy(rightsubtree(t))
3. Display sumsqy

The algorithm sumsqy takes a tree as parameter and it returns the sum of square of the datay attribute of the bivariate data in the tree.

int count(stattree t)
1. Request t
2. Determine count
   2.1 IF istreeempty(t) THEN
   2.1.1 count = 0
   ELSE
   2.1.2 count = 1 + count(leftsubtree(t)) + count(rightsubtree(t))
3. Display count

The algorithm count takes a tree as parameter and it returns the number of pairs of data in the tree.
5. DESIGN OF ALGORITHM FOR THE BIVARIATE REGRESSION ANALYSIS PARAMETERS

One of the parameters of bivariate regression analysis is the beta parameter. For any given collection of bivariate data, \(X_i, Y_i\), the beta parameter is defined, using this statistical formula:

\[
\beta = \frac{\sum_{i=1}^{n} X_i Y_i - \left( \sum_{i=1}^{n} X_i \right) \left( \sum_{i=1}^{n} Y_i \right)}{\left( \sum_{i=1}^{n} X_i^2 \right) - \left( \sum_{i=1}^{n} X_i \right)^2}
\]

Using the statistical algorithms defined in the previous section, the algorithm that will compute the bivariate regression parameter, beta can be defined as follows:

**Double beta(stattree t)**
1. Request t
2. Determine beta
   2.1 \(\text{top} = \text{count}(t) \times \text{sumpro}(t) - \text{sumx}(t) \times \text{sumy}(t)\)
   2.2 \(\text{bottom} = \text{count}(t) \times \text{sumsqx}(t) - \text{sumx}(t) \times \text{sumx}(t)\)
   2.3 \(\beta = \text{top} / \text{bottom}\)
3. Display beta

The second bivariate regression parameter is called alpha, and it is defined statistically as:

\[
\alpha = \frac{\sum_{i=1}^{n} Y_i - \beta \cdot \sum_{i=1}^{n} X_i}{n}
\]

Using the recursive statistical algorithms defined in the previous section, the algorithm that will compute the bivariate correlation coefficient can be defined as follows:

**Double alpha(stattree t)**
1. Request t
2. Determine alpha
   2.1 \(\text{Alpha} = \frac{\text{sumy}(t)/\text{count}(t) - \beta \cdot \text{sumx}(t)/\text{count}(t)}{\text{sumpro}(t)/\text{count}(t)}\)
3. Display alpha

6. DESIGN OF ALGORITHM FOR THE BIVARIATE CORRELATION COEFFICIENT

The correlation coefficient is used to establish the type of relationship, if any, between the two variables. The statistical formulae that can be used to compute the correlation coefficient is given below as:

\[
\rho = \frac{\sum_{i=1}^{n} X_i Y_i - \left( \sum_{i=1}^{n} X_i \right) \left( \sum_{i=1}^{n} Y_i \right)}{\sqrt{\sum_{i=1}^{n} X_i^2 - \left( \sum_{i=1}^{n} X_i \right)^2 \frac{n}{\sum_{i=1}^{n} Y_i^2 - \left( \sum_{i=1}^{n} Y_i \right)^2}}}
\]

Using the recursive statistical algorithms defined in the previous section, the algorithm that will compute the bivariate correlation coefficient can be defined as follows:

**Double correlate(stattree t)**
1. Request t
2. Determine correlate
   2.1 \(\text{top} = \text{sumpro}(t) - \frac{\text{sumx}(t) \times \text{sumy}(t)}{\text{count}(t)}\)
   2.2 \(\text{first} = \frac{\text{sumsqx}(t) - \frac{\text{sumx}(t) \times \text{sumx}(t)}{\text{count}(t)}}{\text{count}(t)}\)
   2.3 \(\text{second} = \frac{\text{sumsqy}(t) - \frac{\text{sumy}(t) \times \text{sumy}(t)}{\text{count}(t)}}{\text{count}(t)}\)
   2.4 \(\text{down} = \sqrt{\text{first} \times \text{second}}\)
   2.5 \(\text{correlate} = \frac{\text{top}}{\text{down}}\)
3. Display correlate
7. DESIGN OF ALGORITHM FOR THE BIVARIATE STATISTICAL ANALYSIS, COVARIANCE

This is another important bivariate statistic, it is defined as :

\[
\sum_{i=1}^{n} X_i Y_i - \frac{\left( \sum_{i=1}^{n} X_i \right) \left( \sum_{i=1}^{n} Y_i \right)}{n} (n - 1)
\]

Using the recursive statistical algorithms defined in the previous section, the algorithm that will compute the bivariate statistic covariance can be defined as follows:

\[\text{Double covariance(stattree } t)\]
\[1 \text{ Request } t\]
\[2 \text{ Determine covariance}\]
\[2.1 \text{ IF count}(t) \leq 1 \text{ THEN}\]
\[2.1.1 \text{ covariance } = \text{"Covariance is not defined."}\]
\[2.1.2 \text{ ELSE}\]
\[2.1.2 \text{ covariance } = (\text{sumpro}(t) - \text{sumx}(t) * \text{sumy}(t) / \text{count}(t) / (\text{count}(t) - 1))\]
\[3. \text{ Display covariance}\]

8. IMPLEMENTATION OF THE ALGORITHMS

A structured approach to programming was used to implement the recursive algorithms, using Java programming language. All the algorithms for the operations of the abstract data type, tree were implemented as java methods in a java class called stattree, thereby encapsulating the operations of the abstract data type, tree. Furthermore, all the recursive statistical algorithms were implemented as java methods in another java class called statistical. Finally, all the algorithms for the various bivariate statistics were implemented as java methods in another class called bivariate. A test program was written in another java class called test, which provided the options that the user would use to read in bivariate data into the abstract data type tree and compute and display the relevant bivariate statistics. The four java classes interacted with each other using the concept of composition and inheritance. Figure 2 below shows the various classes and the way they interacted with each other.

![Class Diagram Showing Interaction Between the Classes.](image-url)
9. RESULTS OF THE IMPLEMENTED ALGORITHMS

Sample test data was used to test the implemented algorithms. Table 1 below shows a sample test data that was used to test the implemented algorithm.

Table 1. Sample Test data

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>34</td>
</tr>
<tr>
<td>15</td>
<td>38</td>
</tr>
<tr>
<td>23</td>
<td>45</td>
</tr>
<tr>
<td>25</td>
<td>46</td>
</tr>
<tr>
<td>30</td>
<td>50</td>
</tr>
<tr>
<td>32</td>
<td>51</td>
</tr>
<tr>
<td>35</td>
<td>55</td>
</tr>
<tr>
<td>40</td>
<td>59</td>
</tr>
<tr>
<td>42</td>
<td>61</td>
</tr>
<tr>
<td>45</td>
<td>65</td>
</tr>
<tr>
<td>47</td>
<td>67</td>
</tr>
<tr>
<td>50</td>
<td>70</td>
</tr>
<tr>
<td>51</td>
<td>71</td>
</tr>
<tr>
<td>55</td>
<td>75</td>
</tr>
<tr>
<td>56</td>
<td>77</td>
</tr>
<tr>
<td>58</td>
<td>79</td>
</tr>
<tr>
<td>60</td>
<td>81</td>
</tr>
<tr>
<td>61</td>
<td>82</td>
</tr>
<tr>
<td>65</td>
<td>87</td>
</tr>
<tr>
<td>67</td>
<td>88</td>
</tr>
<tr>
<td>70</td>
<td>91</td>
</tr>
<tr>
<td>75</td>
<td>96</td>
</tr>
<tr>
<td>77</td>
<td>98</td>
</tr>
<tr>
<td>80</td>
<td>100</td>
</tr>
</tbody>
</table>

The test program was used to read the collection of bivariate data and store them in a tree, using the implemented algorithms for the operations of a tree. The test program also tested all the implemented algorithms and the following results were obtained: bivariate regression parameter, beta is 1.0168, bivariate regression parameter, alpha is 19.617.

![Bivariate Regression Analysis](image)

When the two parameters were combined, the line of best fit was obtained as, \( Y = 1.0168X + 19.617 \). Other results obtained from the implemented algorithms were bivariate correlation coefficient, which was obtained as 0.997926, and covariance, which was obtained as 480.36. The results were validated using an alternative approach (Excel spreadsheet), and figure 3 above confirms the result for the regression analysis.

9. CONCLUSION

This paper has been able to use a binary tree to store and access a collection of bivariate data. Novel recursive algorithms have been developed using the operations of the binary tree. The paper has been able to use the operations of the abstract data type, tree to develop recursive algorithms that can be used for statistical analysis. It has also been able to use the recursive algorithms to compute some bivariate statistics that are of interest. All the algorithms were implemented and tested and the resulted were validated using an alternative approach (Excel spreadsheet).
REFERENCES