NON LINEAR DATA STRUCTURE SIMULATION SYSTEM [+]

Nevart A. Yousif [*]

Abstract :

A data structure is an arrangement of data in a computer's memory or even disk storage. It is an example of several common data structures as arrays, linked lists, queues, stacks, graphs, and trees. Data structures provide a means to manage huge amounts of data efficiently, such as large databases and internet indexing services. In Non-linear data structure, every data item is attached to several other data items in a way that is specific for reflecting relationships and are not arranged in a sequential structure, like tree and graph data structures.

Computer simulation is the discipline of designing a model of an actual or theoretical physical system then executing the model on a digital computer and analyzing the execution output. The aim of this research is to design non linked data structure simulator to simulate the graph structure like: graph types and representation in memory, and also tree structure operations as traversing methods, tree conversions, building Binary Search Tree (BST) and some of tree applications), because many students find difficulty in understanding various parts of this subject. With the help of this simulator, the user will be able to study and run different operations of the non linear data structure, for each operation there are some examples to show how to do the processes using step by step before executing the process practically on different trees that he(she) can construct by himself. The simulator can tell the student whether the solution is correct or not and give a chance to correct the solution to achieve a better understanding or read more examples for learning. This model can serve as a tool for teaching and helping the student to get an ideas of how a real data analysis is done inside computer.

The simulator is designed with Microsoft Visual Basic 6.0 Programming language which is classified among the object-oriented programming languages to provide a good graphical user interface (GUI) to process the tree functions. The system has been presented on some of the arbitrators in the same field, as well as it is applied in practice on a random sample of students from the second year of the computer system department in Technical Institute of Kirkuk, which proved better understanding for graph and tree data structure and its operations.

Key words: Simulation, visualization, non linear linked lists, Graph Data structures, Tree data structure.

نظام محاكاة لهياكل البيانات اللاخطية

نفارت الياس يوسف

المستخلص :

ان هياكل البيانات اللاخطية هي تنظيم البيانات في ذاكرة الحاسوب وكذلك في ذاكرة الاقراص, وهو أحد الامثلة على هياكل البيانات الشائعة كالمصفوفات, القوائم الموصولة, المكدسات الطوابير المخططات والاشجار. توفر هياكل البيانات الوسائل لمعالجة كميات هائلة من البيانات بكفاءة كقواعد البيانات الضخمة وخدمات فهرسة الانترنيت. يرتبط كل عنصر بياني في هياكل البيانات اللاخطية بعدة عناصر بيانية اخرى بطريقة خاصة كي تعكس العلاقات بينها وهي غير منظمة في هيكل متسلسل كهياكل الاشجار والمخططات.

[+] **Received on 22/1/2013 , Accepted on 30/12/2014**

[*] **Assistant Lecturer / Technical Institute / Kirkuk / Nothern Technical University**

ان المحاكاة باستخدام الحاسوب هي عبارة عن تصميم موديل لنظام حقيقي او مفهوم لنظام مادي ثم تنفيذ الموديل على حاسوب رقمي, وتحليل نتائج التنفيذ. ان الهدف من هذا البحث هو تصميم نظام محاكاة لهياكل البيانات اللاخطية لمحاكاة هيكل المخطط مثل: انواع المخططات وتمثيلها في الذاكرة وكذلك العمليات التي تجري على هيكل الاشجار مثل (مسح الاشجار , التمثيل في الذاكرة, وتحويلات الشجرة, بناء شجرة البحث الثنائية, وبعض التطبيقات الاخرى للاشجار), لان العديد من الطلبة يواجهون صعوبة في فهم الاجزاء المختلفة لهذا الموضوع. وبمساعدة هذا النظام سيتمكن المستخدم من دراسة وتنفيذ عمليات مختلفة لهياكل البيانات اللاخطية, ولكل عملية أمثلة لإظهار كيفية معالجة العمليات خطوة خطوة قبل تنفيذها عمليا على أشجار مختلفة يقوم المستخدم ببناءها بنفسه. يقوم البرنامج باعلام المستخدم اذا كان حله صحيح ام خاطيء ويعطي فرصة لتصحيح الخطأ في الحل ان وجد ليعطي فهم افضل للموضوع او يعود لمراجعة امثلة اخرى لتعلم الموضوع. هذا النموذج يعمل كاداة تعليمية ومساعدة للطالب للحصول على افكار حول كيفية تحليل البيانات الحقيقية داخل الحاسوب.

أنجز هذا البحث باستخدام لغة البرمجة (Microsoft Visual Basic 6.0) والتي تصنف ضمن لغات البرمجة الشيئية ذات التصميم المرئي والواجهة الرسومية. تم عرض النظام على بعض المحكمين في حقل الاختصاص وكذلك تم تطبيقه عمليا على عينة عشوائية من طلبة المرحلة الثانية لقسم أنظمة الحاسوب في المعهد التقني/كركوك, وقد اثبت انه يوفر فهم افضل لهيكل بيانات المخطط والشجرة وعملياتها.

الكلمات المفتاحية: **Non-linear Data structures, Simulation, Tree structure, graph structure**

1. Introduction :

It is common to use illustrations when teaching abstract concepts in computer science to help make these concepts more concrete. The instructor will often draw a given data structure on the whiteboard and use it to step through the process of performing an algorithm on that data structure. Textbooks often include a series of diagrams of data structures to accompany the textual description and explanation of the given algorithm[1].

A more effective educational strategy is the use of computerized simulations for presenting more complex concepts and algorithms. Several studies have suggested that student learning is improved by using simulations as opposed to traditional paper and pencil methods ([2-4]). Since simulations are dynamic, it is possible to watch the algorithm in action as it manipulates a data structure, and the transitions between steps is much more pronounced. Since the entire process is automated, these transitions can be demonstrated without needing to manually erase or recopy the data structures. A computer simulation can be developed to contain domain of specific knowledge of the algorithm, thus making it easier to prevent errors, and to provide immediate feedback for students who use these tools to study an algorithm and work practice exercises. It also facilitates the grading of student homework, since much of the process can be automated and not require hours of instructor time to carefully read student solutions. Simulations can also allow operations to be logged to provide a means of reverting to previous steps of an algorithm for repetition and practice. Computer simulations also can be saved for later review and practice, either for instructors to present in their lectures, or for students to study the workings of the algorithms. [1]

Data structures, is one of the most important subject on Computer Engineering and Information Technology, which is required by all the software programmers. As a programmers, we would be handling the data in huge quantity. The data we are given required to be stored in the memory. The memory should be used efficiently. Graphs and

trees are flexible, powerful non-linear data structures that can be used to represent data items which has hierarchical relationship The major problem in teaching data structures and algorithms has been the difficulty of capturing the dynamic nature of the material. A proper tool for classroom demonstration would provide an ideal way to teach these kinds of concepts[5]. Simulation embodies the principle of "learning by doing" to learn about the system we must first build a model of some sort and then operate the model. The use of simulation is an activity that is as natural as a child who role plays. To understand reality and all of its complexity, we must build artificial objects and dynamically act out roles with them. Computer simulation is the electronic equivalent of this type of role playing and it serves to drive synthetic environments and virtual worlds [6].

As applied to the study of research design, simulations can serve as a tool to help the teacher, evaluator, and methodologist address the complex interaction of data construction and analysis, statistical theory, and the violation of key assumptions. In a simulation, the analyst first creates data according to a known model and then examines how well the model can be detected through data analysis. Teachers can show students that measurement, sampling, design, and analysis issues are dependent on the model that is assessed[7].

In this research, a new simulator for non Linear Linked lists that helps the students to execute the operations of graph and tree data structures to be easy to understand especially binary tree conversions and traversing methods. Two versions of procedures are implemented to teach the non linear algorithms: a teaching version for instructors to teach graph and tree algorithms during their lectures, and a tutoring version for students to practice examples and complete homework exercises.

2. Related Works :

First, let us make a survey about the research that has been done to help in learning tree data structures via visualization and simulation researches, and other computer simulation researches which are really an effective tool in computer science education:

1- Raghavendra C. S, 1981 proposed a programming exercise that to developed a visual cache memory simulator and then use it to analyze several memory reference trace files and analyze several memory reference trace files [8], and Danial N. et al, improved the quality of learning for students [9].

2- Korn, J.L. et al, 1988, presented a method to provide higher level and more informative visualizations in a debugger using a technique called traversal based visualization. The debugger traverses a data structure using a set of user supplied patterns to identify parts of the data structure to be drawn a similar way. A declarative language is used to specify the patterns and the actions to take when the patterns are encountered. Alternatively, the user can construct traversal specifications through a graphical user interface to the declarative language. Furthermore, the debugger supports modification of data [10].

3- Ryan S. B. et al, 1999, Presented two tools to support the teaching of data structures and algorithms visualizers, which provide interactive visualizations of user-written data structures, and Testers, which check the functionality of user-written data structures. They get an outline about a prototype implementation of visualizers and testers for data structures written in Java, and they gave a report about the usefulness of testers and visualizers in an introductory Data Structures and Algorithms (CS2) course [11].

4- Ville Karavirta, et al, 2004, introduced a new tool for illustrating algorithms in action in terms of direct manipulation of the library data structures, the process called visual algorithm simulation. The user does not need to code anything to build animations. Instead, he(she) can graphically invoke ready-made operations available in the library data structures to simulate the working of real algorithms[12].

5- Rusu, A. et. al, 2006, created a system that determines the type of a binary tree and then selects an algorithm to draw the tree depending upon the specified quality measures. The system recognizes many types of binary trees (complete, Fibonacci, random, unbalanced-to-the-left and unbalanced-to-the-right) and allows the user to choose from many quality measures. Experiments show that their adaptive visualization system can perform any of these systems using a single binary tree drawing algorithm and select multiple quality measures[13].

6- Kristy S. V., 2012, proposed a research involved developing and testing a sketching, visualization, and simulation tool for demonstrating graph algorithms commonly taught in under graduate computer science courses. This research has two broad objectives: developing a tool that supports computer aided manual simulations of algorithms to augment traditional white board presentations, allowing and a tool that supports computer-aided student practice of algorithms providing incremental feedback about their performance and about how to solve a problem when they get stuck[1].

3. Graph Data Structure :

A graph G consists of a set V , whose members are called the vertices of G, together with a set E of pairs of distinct vertices from V . These pairs are called the edges of G. If e =(v, w) is an edge with vertices v and w, then v and w are said to lie on e, and e is said to be incident with v and w. If the pairs are unordered, then G is called an undirected graph; if the pairs are ordered, then G is called a directed graph. The term directed graph is often shortened to digraph, and the unqualified term graph usually means undirected graph. The natural way to picture a graph is to represent vertices as points or circles and edges as line segments or arcs connecting the vertices. If the graph is directed, then the line segments or arcs have arrowheads indicating the direction. Figure (1) shows two examples of graphs. The second part of Figure (1) shows a directed graph, where the nodes of the network (A, B, ... , F) are the vertices and the edges from one to another have the directions shown by the arrows.

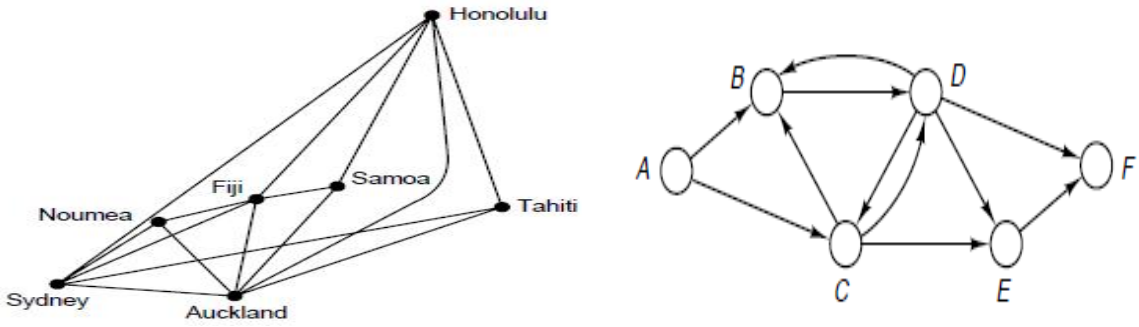


Figure (1): Examples of Graphs

If we are to write programs for solving problems concerning graphs, then we must first find ways to represent the mathematical structure of a graph as some kind of data structure. There are several methods in common use, which differ fundamentally in the choice of abstract data type used to represent graphs, and there are several variations depending on the implementation of the abstract data type

Graphs are defined in terms of sets, and it is natural to look first to sets to determine their representation as data. First, we have a set of vertices, and, second, we have the edges as a set of pairs of vertices. Rather than attempting to represent this set of pairs directly, we divide it into pieces by considering the set of edges attached to each vertex separately. In other words,

we can keep track of all the edges in the graph by keeping, for all vertices v in the graph, the set E_v of edges containing v, or, equivalently, the set A_v of all vertices adjacent to v.
In the foregoing implementation, the structure Set is essentially implemented as an array of bool entries. Each entry indicates whether or not the corresponding vertex is a member of the set. If we substitute this array for a set of neighbors, we find that the array neighbors in the definition of class Graph can be changed to an array of arrays, that is, to a two-dimensional array.
Another way to represent a set is as a list of its elements. For representing a graph, we shall then have both a list of vertices and, for each vertex, a list of adjacent vertices. We can consider implementations of graphs that use either contiguous lists or simply linked lists. For more advanced applications, however, it is often useful to employ more sophisticated implementations of lists as binary or multiway search trees or as heaps. Note that, by identifying vertices with their indices in the previous representations, we have ipso facto implemented the vertex set as a contiguous list, but now we should make a deliberate choice concerning the use of contiguous or linked lists [5].

4. Tree Data Structure :

The tree data structure can be generalized to represent directed graphs by removing the constraints that a node may have at most one parent, and that no cycles are allowed. Edges are still abstractly considered as pairs of nodes. Node is a structure which may contain a value, a condition, or represent a separate data structure. The topmost node in a tree is called the root node. It is the node at which operations on the tree commonly begin. Each node in a tree has zero or more child nodes, which are below it in the tree. An internal node (also known as an inner node or branch node) is any node of a tree that has child nodes. Similarly, an external node (also known as an outer node, leaf node, or terminal node) is any node that does not have child nodes. The height of a node is the length of the longest downward path to a leaf from that node. The height of the root is the height of the tree. The depth of a node is the length of the path to its root as shown in figure (2).
There are many different ways to represent trees; common representations represent the nodes as dynamically allocated records with pointers to their children, their parents, or both, or as items in an array, with relationships between them determined by their positions in the array (e.g., binary heap)[4].

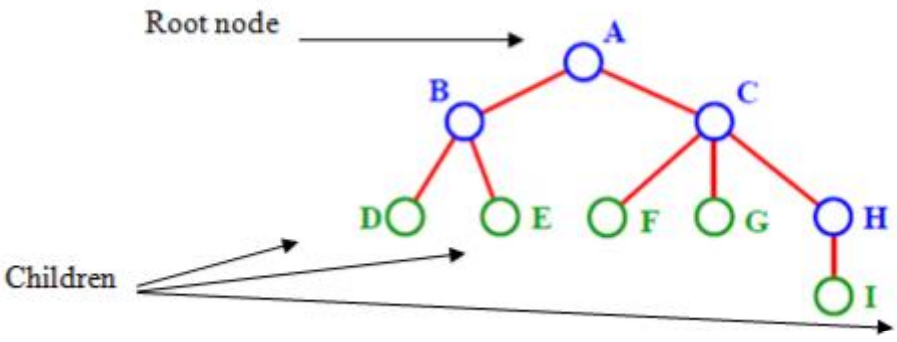


Figure (2): Tree data structure

In this paper, a Tree simulator is designed to simulate the tree operations and algorithm in the sense that it allows real interaction between the user and the underlying data structures. The user is able to watch many examples on each process and also build his own tree structures[4].

4.1 Binary Trees :

A binary tree is a tree data structure in which each node has at most two child nodes, usually distinguished as "left" and "right". Nodes with children are parent nodes, and child nodes may contain references to their parents. Outside the tree, there is often a reference to the "root" node (the ancestor of all nodes), if it exists. Any node in the data structure can be reached by starting at root node and repeatedly following references to either the left or right child. Binary trees are used to implement binary search trees and binary heaps. The essential feature of the comparison tree is that, when we move to the left subtree, we move to smaller keys, and, when we move to the right subtree, we move to larger keys.

One of the most important operations on a binary tree is traversal, moving through all the nodes of the binary tree, visiting each one in turn. If we name the tasks of visiting a node V, traversing the left subtree L, and traversing the right subtree R, then there are six ways to arrange them:

V L R, L V R, L R V, V R L, R V L, R L V.

These three names are chosen according to the step at which the given node is visited. With preorder traversal, the node is visited before the subtrees; with inorder traversal, it is visited between them; and with postorder traversal, the root is visited after both of the subtrees[5].

An expression tree is built up from the simple operands and operators of an (arithmetical or logical) expression by placing the simple operands as the leaves of a binary tree and the operators as the interior nodes. For each binary operator, the left sub-tree contains all the simple operands and operators in the left operand of the given operator, and the right sub-tree contains everything in the right operand.

For a unary operator, one of the two sub-trees will be empty. We traditionally write some unary operators to the left of their operands, such as '−' (unary negation) or the standard functions like log( ) and cos( ). Other unary operators are written on the right, such as the factorial function ( )! or the function that takes the square of a number, Sometimes either side is permissible, such as the derivative operator, which can be written as d=dx on the left, or as 0 on the right.

Binary trees can be constructed from programming language primitives in two ways:

a. Nodes and references: in a language with records and references, binary trees are typically constructed by having a tree node structure which contains some data and references to its left child and its right child.
b. Arrays: binary trees can also be stored in breadth-first order as an implicit data structure in arrays, and if the tree is a complete binary tree, this method wastes no space. This method benefits from more compact storage and better locality of reference, particularly during a preorder traversal[4].

5. Non-Linear Lists Simulator :

The simulator is designed with Visual Basic Programming language to provide a good graphical user interface (GUI) to process the Graphs and trees functions. The first designed interface is shown in figure (3) and it give the student the choice to select to learn either graph or tree data structure.

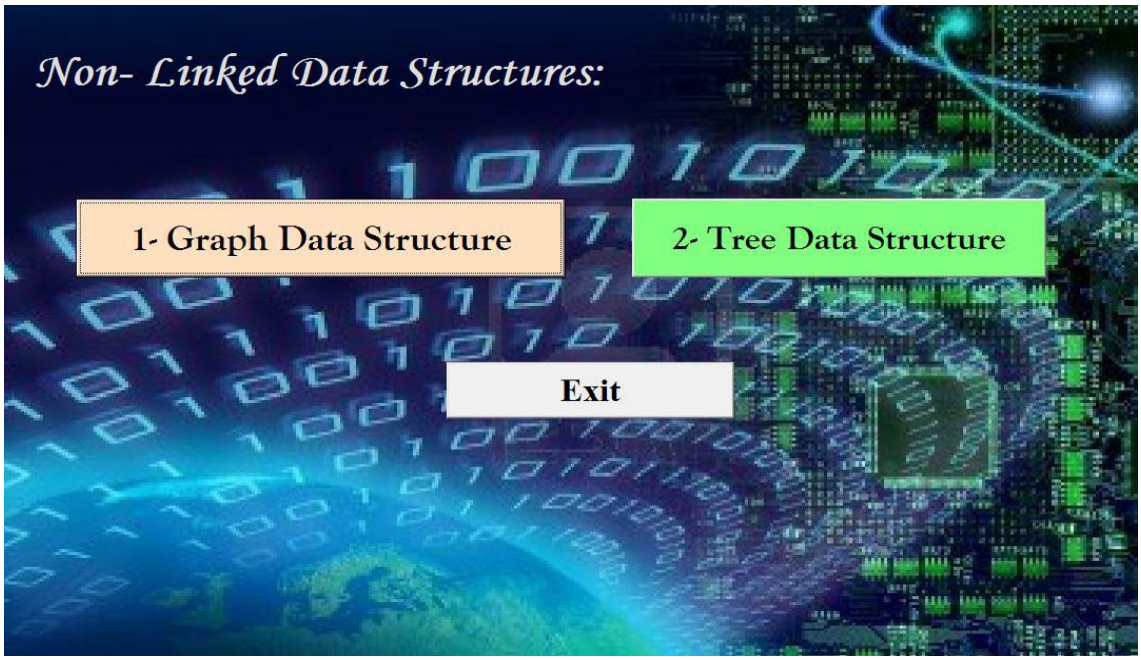


Figure (3) :The User interface window

5.1 Graph Simulator Design :

The main menu bar consists of menu options for the most important operations on the graphs that the student need to understand and it can be used as an efficient tool in teaching this subject (figure A1). These menu options are :

1. Graph Help
2. Types
3. Graph Representations
4. Back to Main Menu

The first menu option " Graph Help" provided with basic concepts , definitions and types of graphs, it is also provided with additional links to a three external word files as resources of additional and detailed information about graphs. These resources enables the teacher to add or change any information he/she need in the future or add information in any language to improve the students understanding by changing the content of these files (figure A2).

The second menu option " Types" contain "Examples" and "types" options. Examples as shown in figure(A3) is designed to improve the students information about graph types and it contain many different graphs, the student can choose the graph from menu at the top of the screen, and the graph number, then will be highlighted and its type will appear (connected or not , cyclic or not, directed or not, or if it is tree graph). The "Types" menu option shown in figure (A4) contain many graphs differs than the examples graphs, here the student himself must choose the type by selecting the correct answers depending on the chosen graph and check his/her solution using the "check solution" button, but this button will not give the correct answer if it false to make the student try again and go back to the examples or concepts if he/she wanted to learn more.

The third menu option "Graph Representation" shows three sub-options for graph representations as shown in figure A1:

- Adjacency Set
- Adjacency Matrix
- Adjacency List

At each of these representations the student can choose his own graph by pressing on any node of a prepared graph nodes to highlight them, so the student can choose different graphs and represent them using the selected method as shown in figures (A5- A7). After solving the representation also he can check the solution or return back to the examples.

5.2 Tree Simulator Design :

The main menu bar consists of options for the most important operations on the trees that the student required and it can be used as an efficient tool in teaching the subject as shown in figure(A8). These menu options are :

1. Tree Help
2. Tree Operations
3. Back To Main Menu

The first menu option " Concept" contain tree definitions, types and representation, also it is provided with additional links to three external word files as resources of additional and detailed information about Trees like the graph to support more detailed information about the tree structures improve students understanding (figure A9) and allow changing the content of these files at future.

The second menu option contain the following operations on the trees:

1. Representing in Memory
2. Binary Search Tree (BST)
3. Tree Traversal
4. Tree Conversions
5. Representing Arithmetic Expression as Binary tree

Each of these options contains submenus of three options: examples, analysis and code/algorithm.

The example option open forms that help to understand the tree functions like: traversing methods (in-order, pre-order, and post-order), BST, tree representation in memory, binary to normal tree and arithmetic expression tree examples.

Two types of example forms are designed, the first supports the user with many examples to allow him/her to select and run, like 'tree conversion' in figure (A15) and arithmetic expression tree examples in figure(14-a).

Another type of example forms is provided to give the student ability to construct the example by himself as traversing methods in figure(A10-a), representing the tree in memory in figure (A11-a) which supports the student with an empty input tree that he/she can fill with any needed data, then select the required method to traverse (in-order, pre-order or post –order) or to represent in memory, then press the "solution" button to see the solution. To see the solution step by step, the simulator is provided with a programmed button to see the execution steps of the required function with animation, as in figure(A10-a). The student can clear the solution to see the execution of the same example or start another example using the "New" button.

The same type of example forms are used with the BST in figure(A12-a) that provide the user the ability to write his own list and convert it to BST, and the tree conversion from normal to binary tree figure(A13-a), the proposed system provides the student with more examples, in each example, information about the function are displayed to help in understanding it.

The tester commands will display a form to check the understanding of the student by trying to solve one of many provided tasks like building arithmetic expression tree in figure(A14-b).

Another type of analysis forms are used in such a way that the student can build his task by himself and solve it like: building the tree in the tree traversal by pressing on a selected nodes

and changing their colors to red in figure(A10-b), filling an empty list with random numbers to convert it to BST as in figure(A12b), build a binary tree also by pressing on the selected nodes to be represent in memory after numbering these nodes as in figure(A11-b), and building a normal tree and convert it to binary in figure(A13-b).
In each analysis the solution must be entered by the student, and check it using "check solution" button, the student can try to change his solution if it is not correct, he/she can solve different tasks given from his teacher. These forms do not provide the solution when it is not correct, so the student must try many times until reaching the correct solution with help of the "clear" buttons, or return back to learn by the examples. Code/Algorithm menu option contain the codes or algorithms in Visual Basic for the required method.

6. Conclusions :

The ultimate goal of this simulation model is to improve the quality of learning for students in higher education by providing opportunities for academics to enhance their understanding of good practice in assessment. The simulator provided an interactive tool that described the Non- Linear Data Structure simulator exercises and applications because the student can't see these processes actually inside the computer. Using this simulator the student can apply any number of examples and tasks, besides the available help and algorithms are provided to enhance the learning operation. The simulator was tested by taking the views of specialists and teachers in the technical institute in Kirkuk-Iraq, and it achieved significantly increase in the student's ability to understand the operations on the data structures. The simulator can be used by teachers in teaching tree an graph data structures processes to improve the students general understanding.

7. Future Work :

One of the future extensions involves an additional visualization processes to tree programming by combining this simulator with other data structures simulators, like primitive and linear data structures as integers, characters, arrays, stack, queue and linear linked lists.

8. References :

1. Kristy S. V., "Sketchmate: A Computer-Aided Sketching and Simulation Tool for Teaching Graph Algorithms", University of Tennessee, Knoxville, Trace: Tennessee Research and Creative Exchange, Doctoral Dissertations Graduate School, Aug 2012. http://trace.tennessee.edu/utk_graddiss/1437.
2. Hundhausen, C., Douglas, S., and Stasko, J. "A Meta-Study of Algorithm Visualization effectiveness", Journal of Visual Languages and Computing, Vol 13, No 3, p:259-290, 2002.
3. Kumar, A., "Need to Consider Variations within Demographic Groups When Evaluating Educational Interventions". In ITiCSE '09: Proceedings of Innovations and Technology in Computer Science Education, pages 176-180. Paris, France, 2009.
4. Laakso, M. and Salakoski, T., "Automatic Assessment of Exercises for Algorithms and Data Structures - A Case Study with TRAKLA2", In Proceedings of Kolin Kolistelut / Koli Calling - Fourth Finnish/Baltic Sea Conference on Computer Science Education, pages 28-36, 2004.

5. Paul A. F., "The Art and Science of Digital World Construction", Computer & Information Science and Engineering Department, University of Florida, USA, copyright Nikos Drakos, Computer Based Learning Unit, University of Leeds 1995. http://www.cise.ufl.edu/~fishwick/introsim/paper.html
6. Robert L. K. and Alexander J. R, "Data Structures and Program Design in C++", Prentice Hall in Upper Saddle River, New Jersey 07458, ISBN 0-13-087697-6, 2000.
7. Trochim, W. and Jossey B., "Advances in Quasi-Experimental Design and Analysis". New Directions for Program Evaluation Series, Number 31, San Francisco, 1996. http://www.socialresearchmethods.net/simul/introsim.htm
8. Raghavendra C. S. and M. D. Ercegovac , "A Simulator for On-Line Arithmetic ",Proceedings of the 5th IEEE Symposium on Computer Arithmetic, University of Michigan, Michigan, May 1981.
9. Danial N. et al, "A Visual Cache Memory Simulator" , 35th ASEE/IEEE Frontiers in Education Conference, Indianapolis, ISBN 0-7803-9077, October, 2005.
10. Korn, J. L. and Appel, A.W., "Traversal-based Visualization of Data Structures", Information Visualization, 1998. Proceedings. IEEE Symposium on, pages 11-18, 1998.
11. Ryan S. B. et al., "Testers and Visualizers for Teaching Data Structures", ACM SIGCSE Bulletin, Volume 31, Issue 1, Pages 261-265, Mar 1999.
12. Ville K., et al., "A Tool for On-The-Fly Demonstration of Data Structures and Algorithms", Conference of Proceedings of the Third Program Visualization Workshop, pp. 26–33, The University of Warwick, UK.,2004.
13. Rusu, A., Clement, C. and Jianu, R., "Adaptive Binary Trees Visualization with Respect to User-Specified Quality Measures", Information Visualization, Tenth International Conference**,** Page(s):469–474, 2006.

Appendix (A)

List of Non-Linear Simulator Forms

Figure (A1): Use Interface for Graph Data Structure

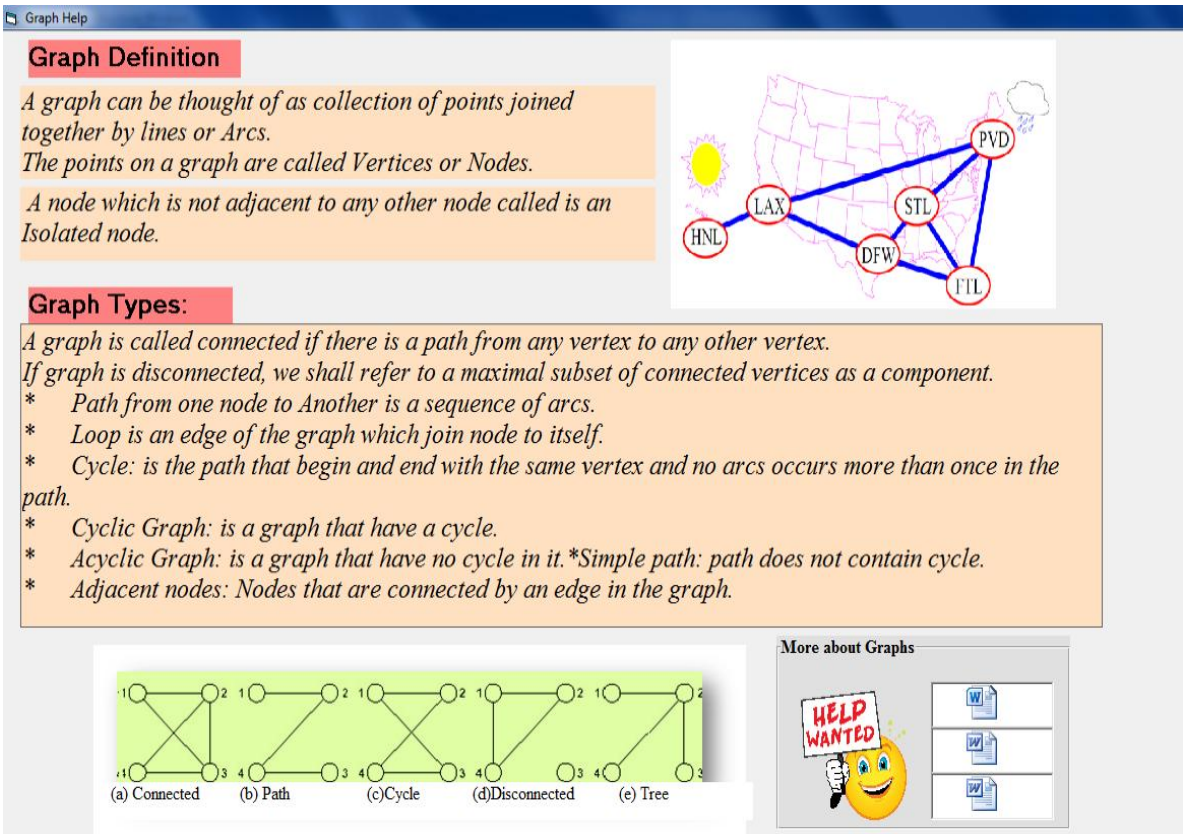


Figure (A2): Graph Help (definitions, types and more external references about Graphs)

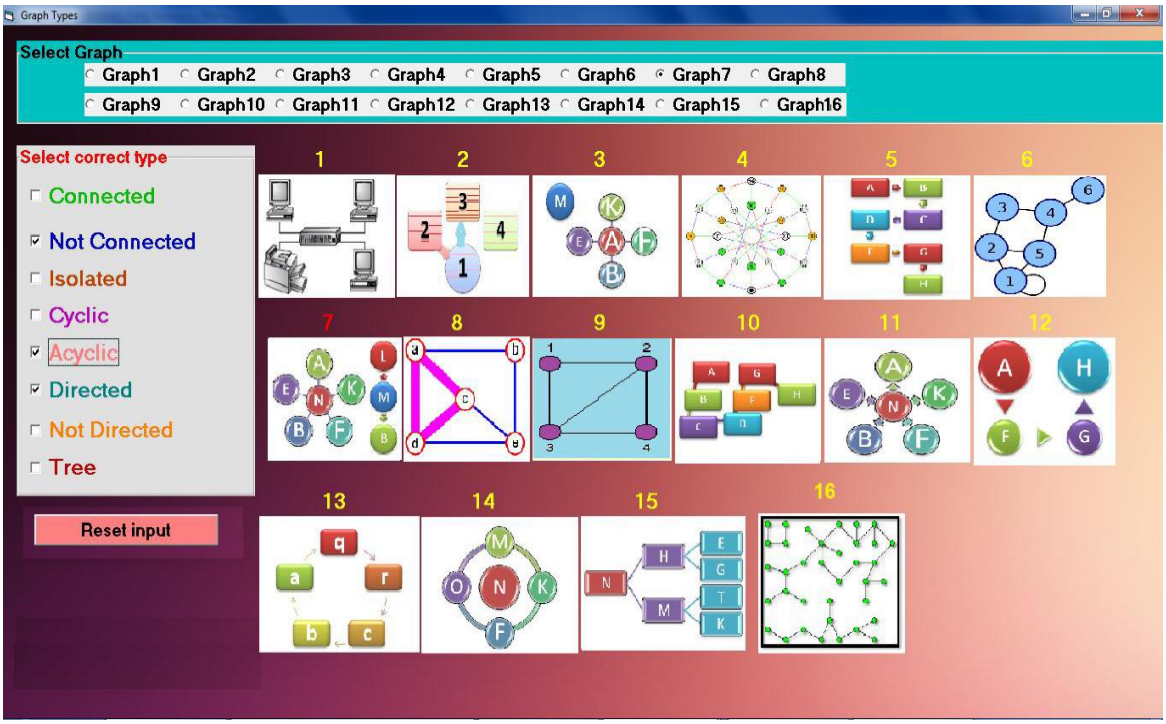


Figure (A3): Examples of graph types

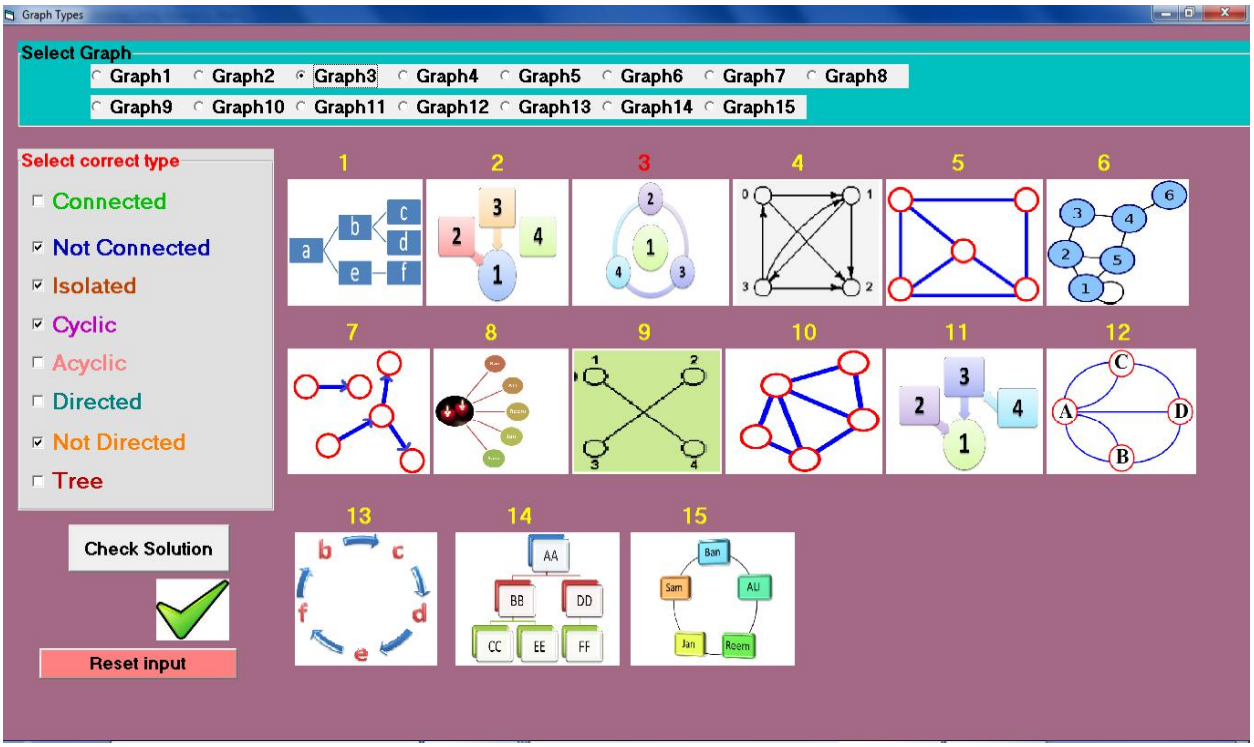


Figure (A4): Graph types shows a correct solution of Graph No3

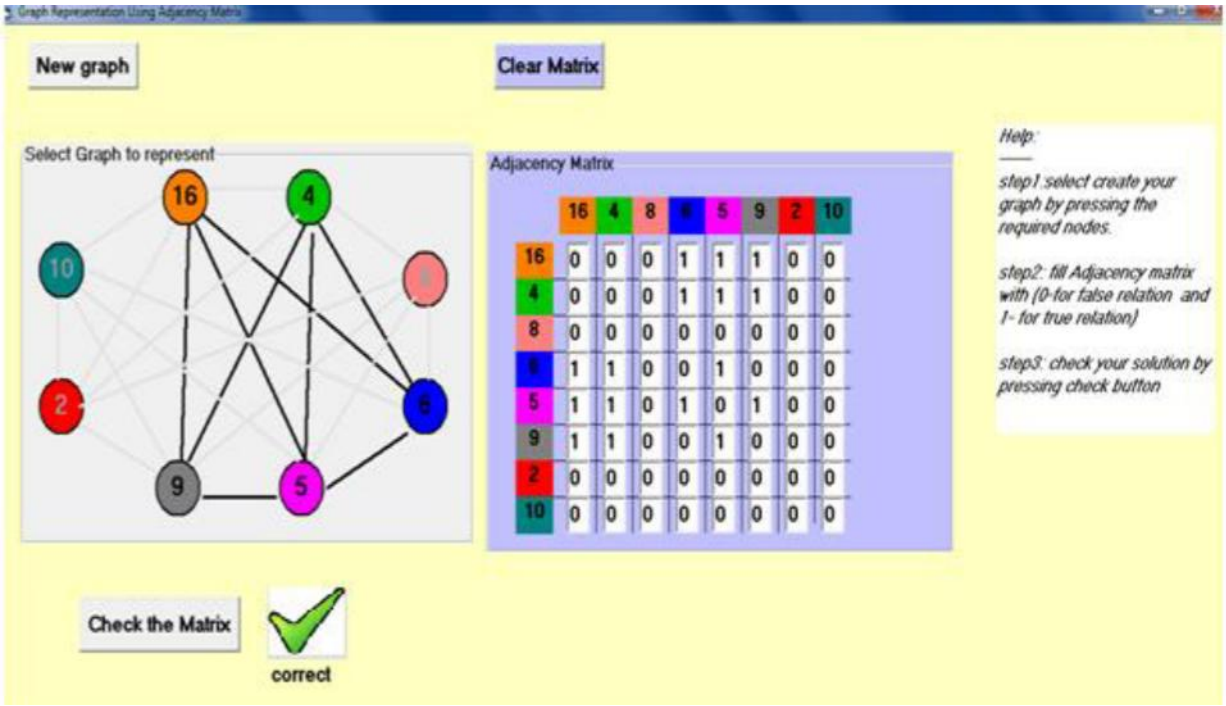


Figure (A5): Correct Solution for representing a selected graph using adjacency matrix

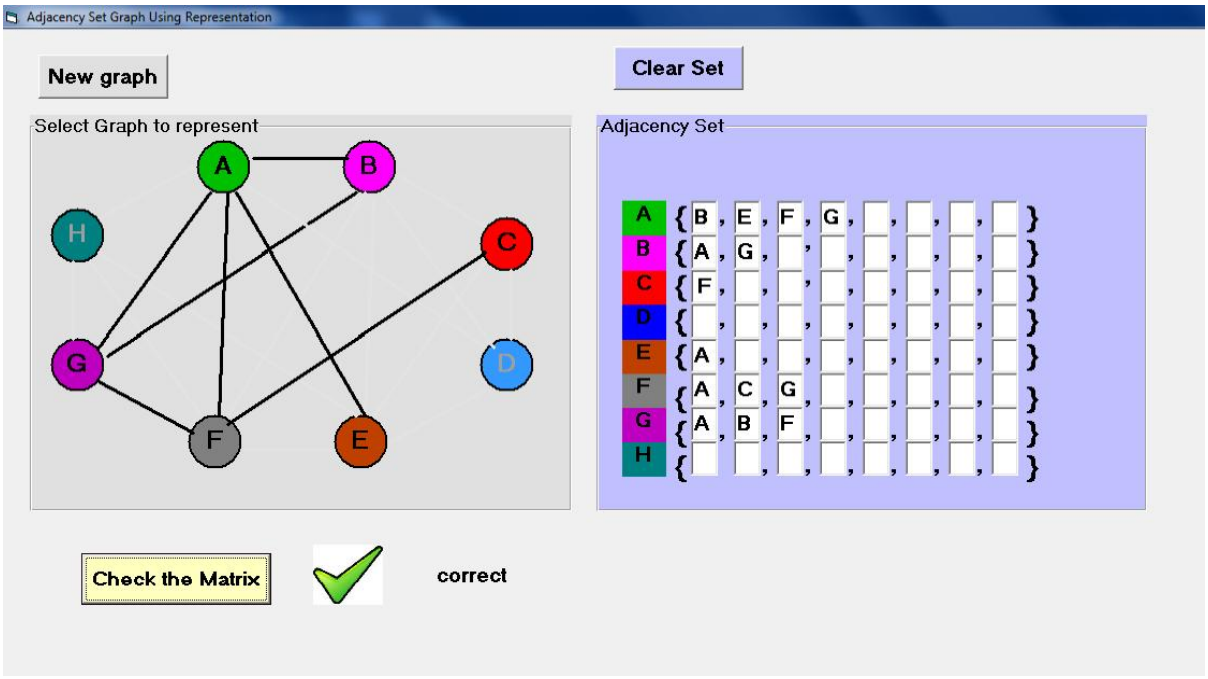


Figure (A6): Correct Solution for representing a selected graph using adjacency set

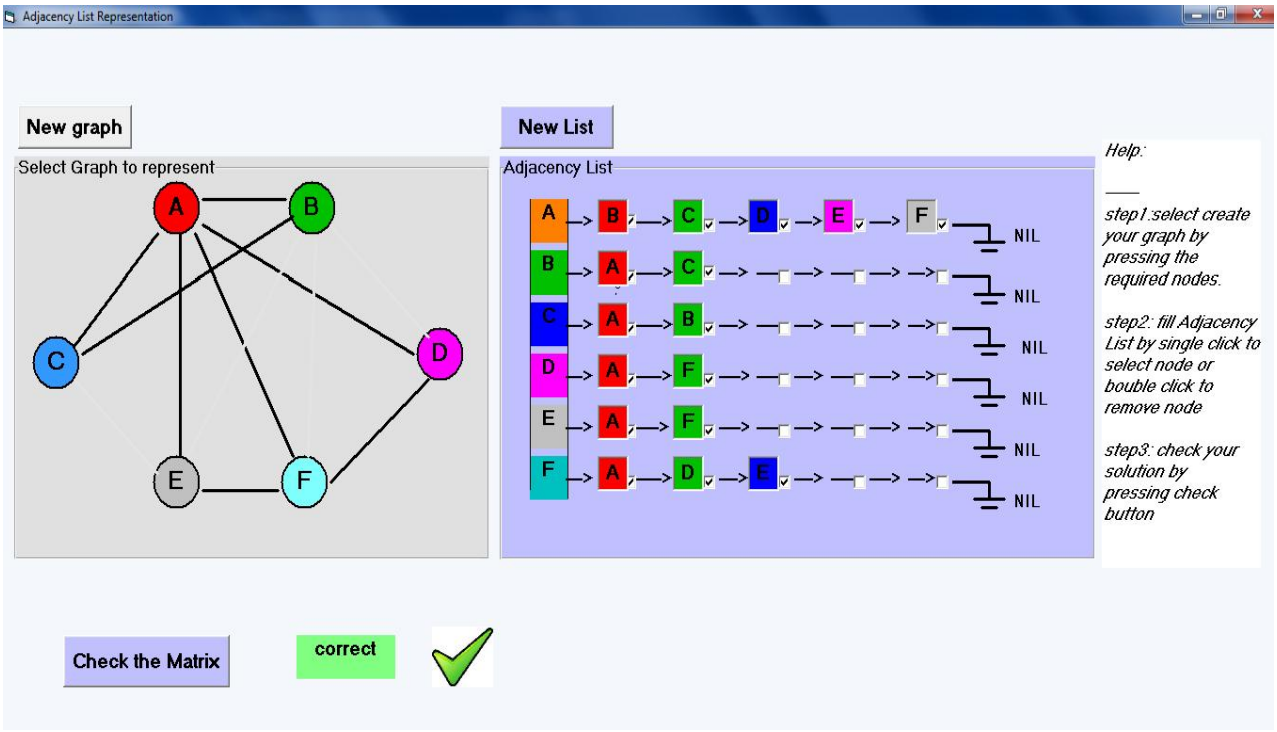


Figure (A7): Correct Solution for representing a selected graph using adjacency list

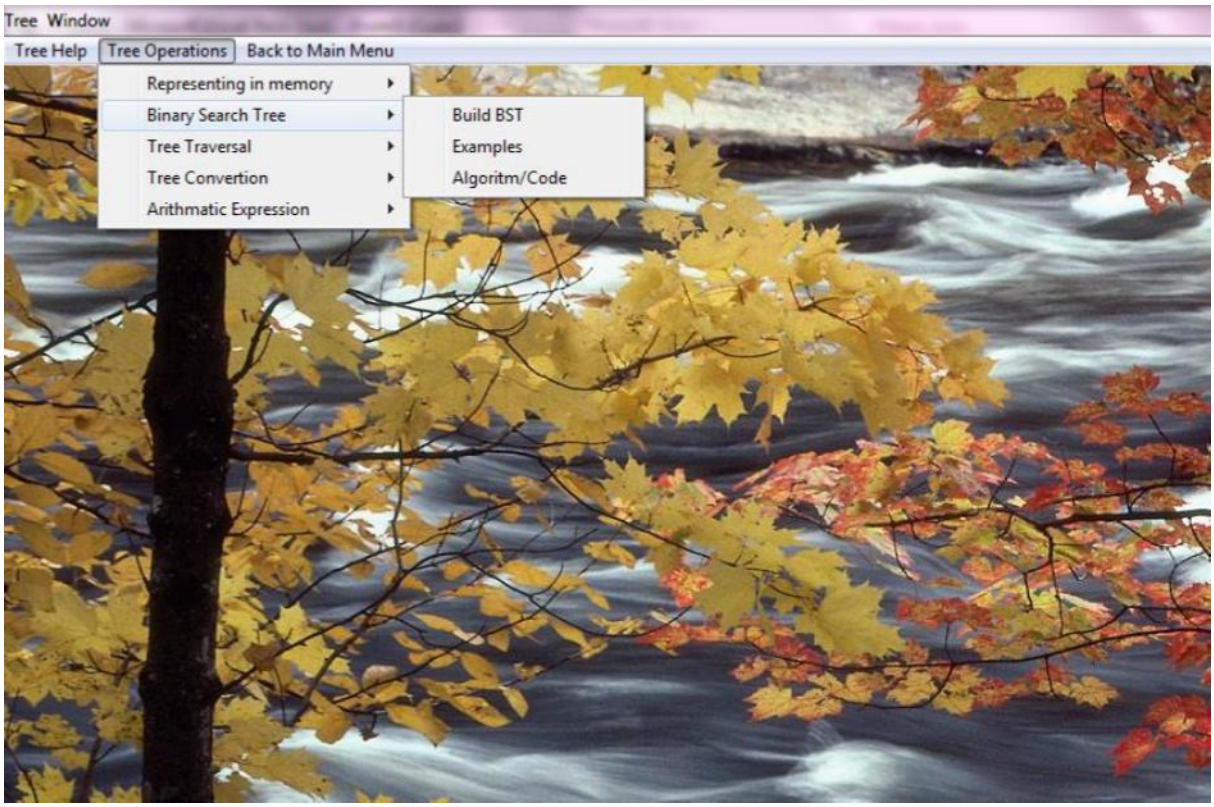


Figure (A8): User Interface for Tree Data Structure

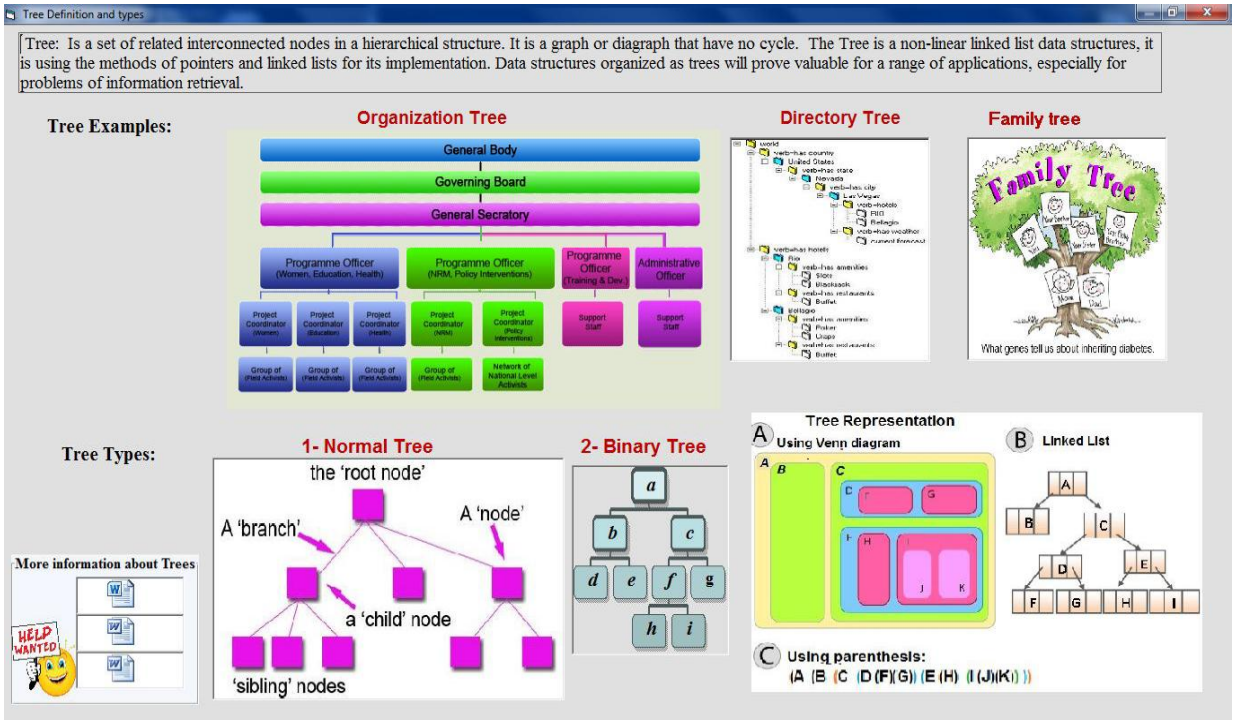
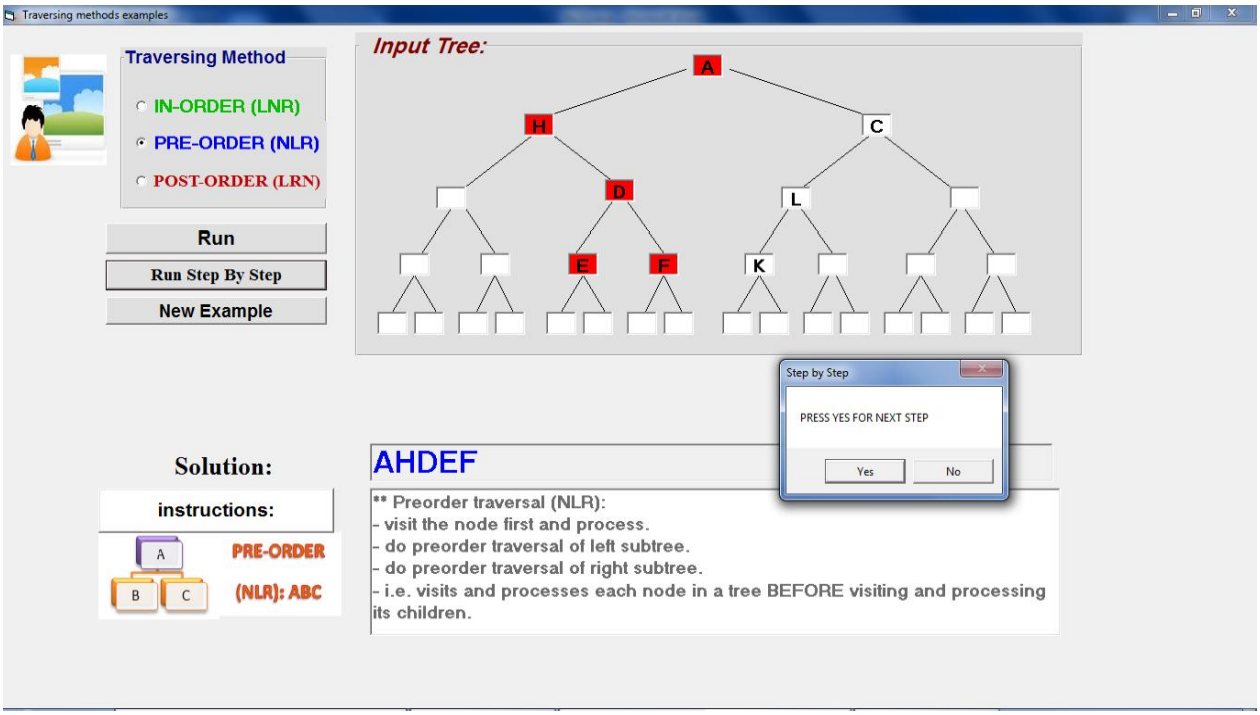
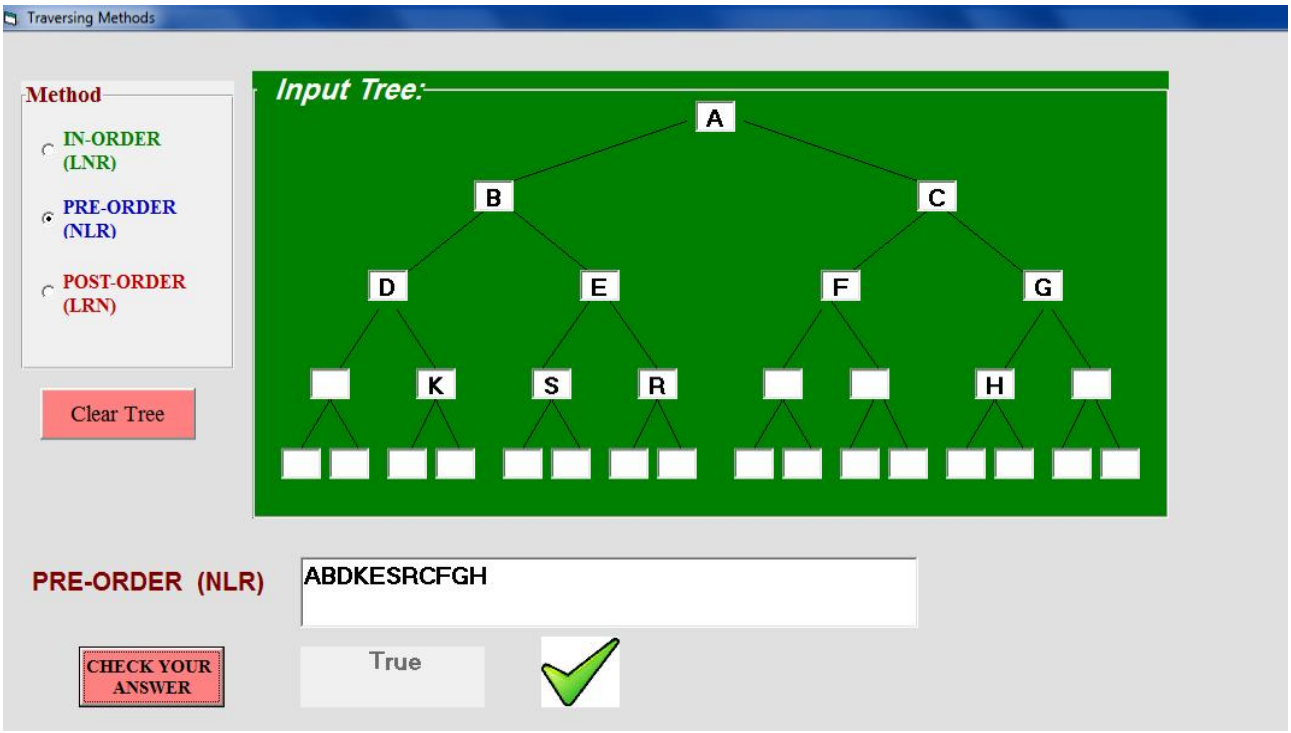


Figure (A9): Tree Help (definition, types, representation and more external references about Trees)

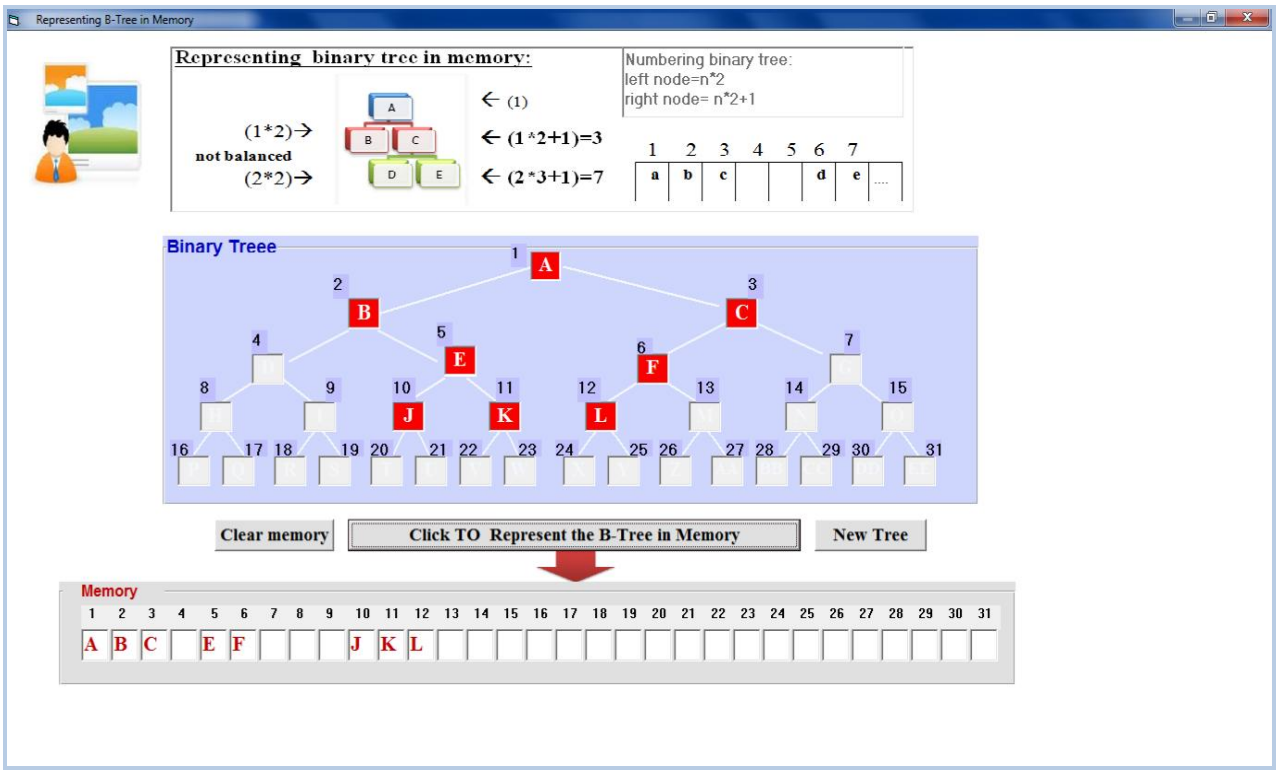


(a)

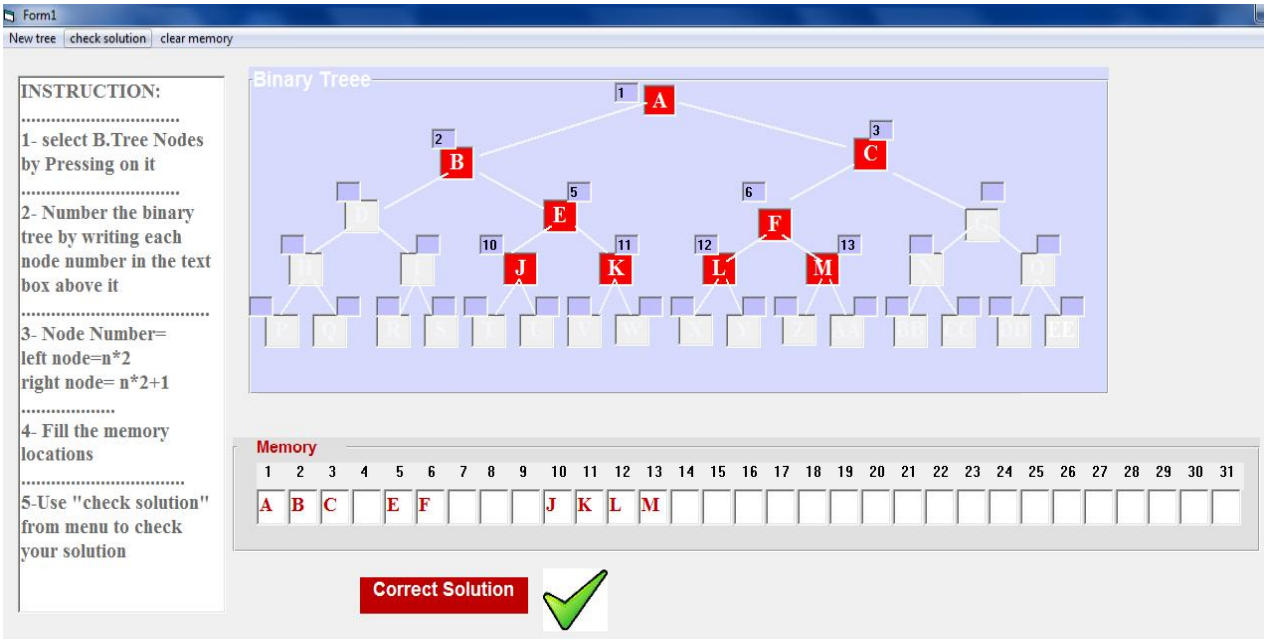


(b)

Figure (A10): a- Traversing methods examples using pre-order method with step by step execution.
b- Traversing method using post-order method with correct solution

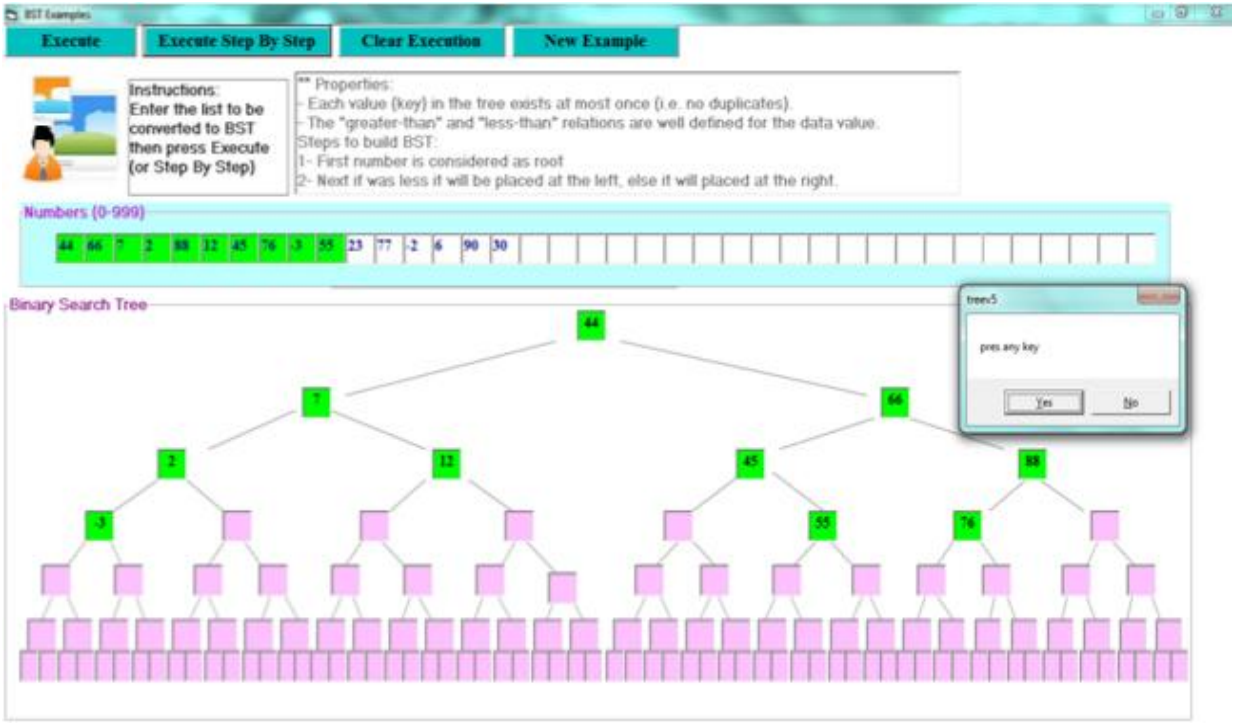


(a)

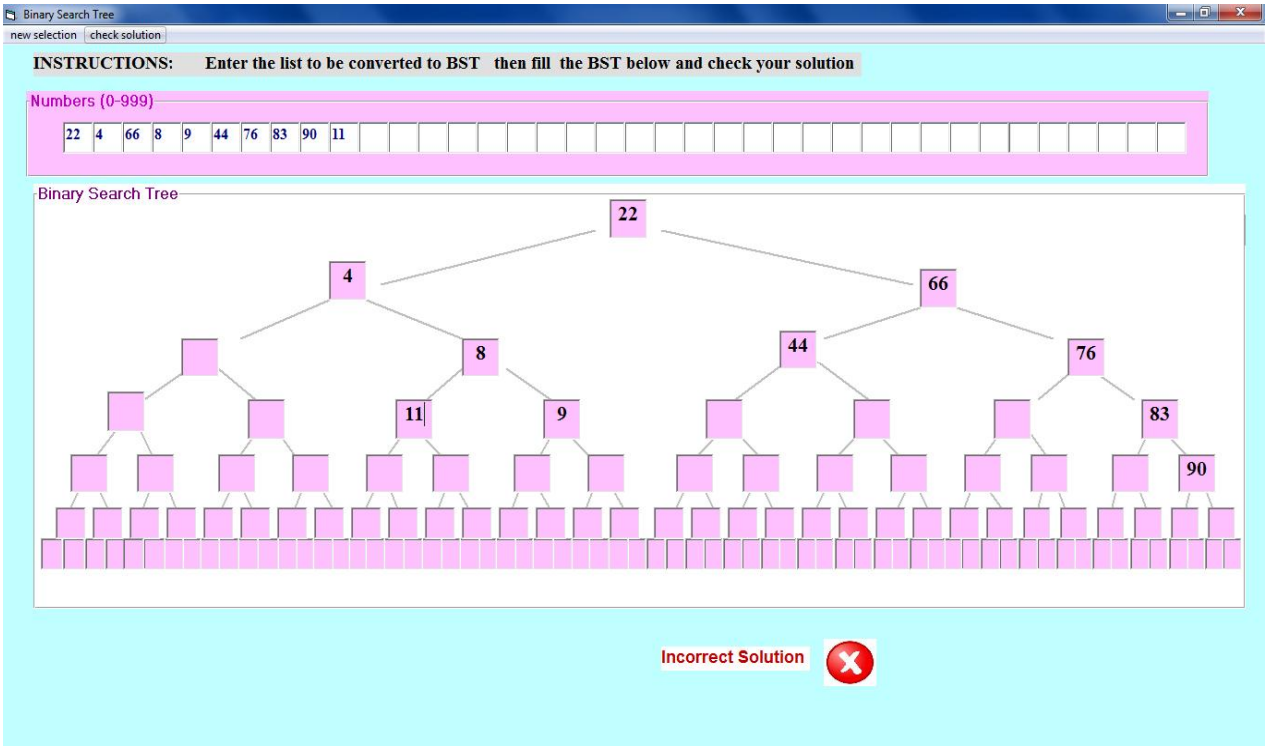


(b)

Figure (A11): a- Example for tree representation in memory,
b- tree representation in memory with correct solution

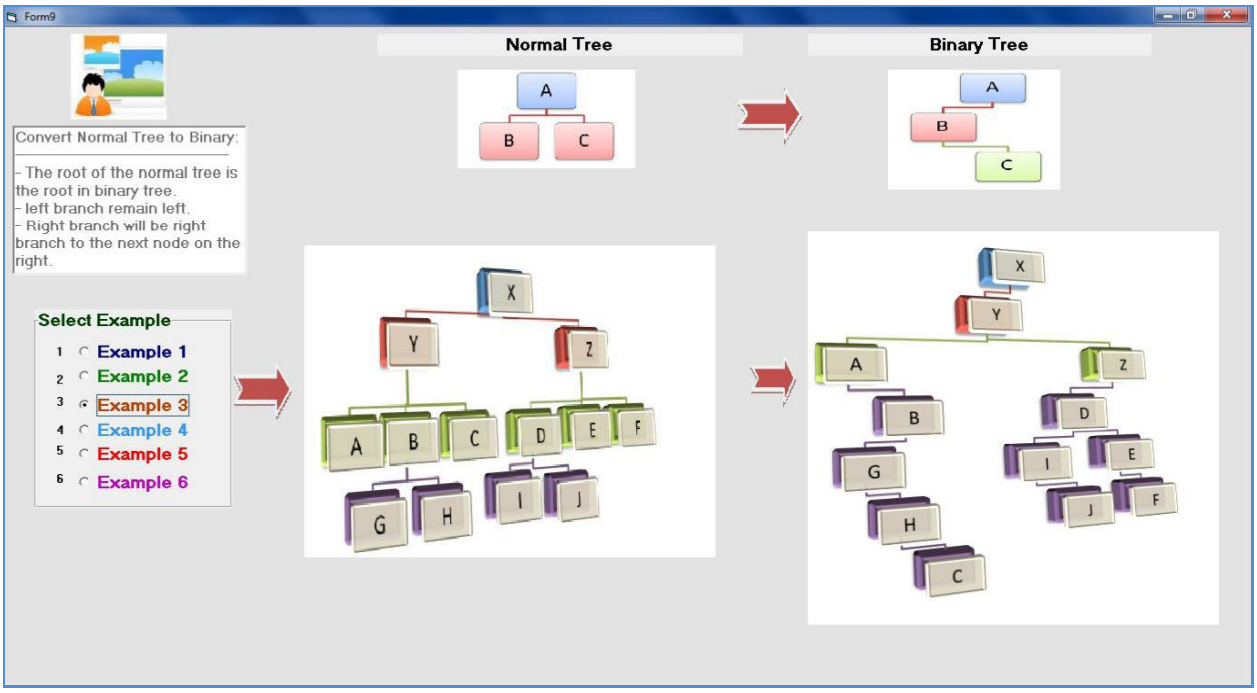


(a)

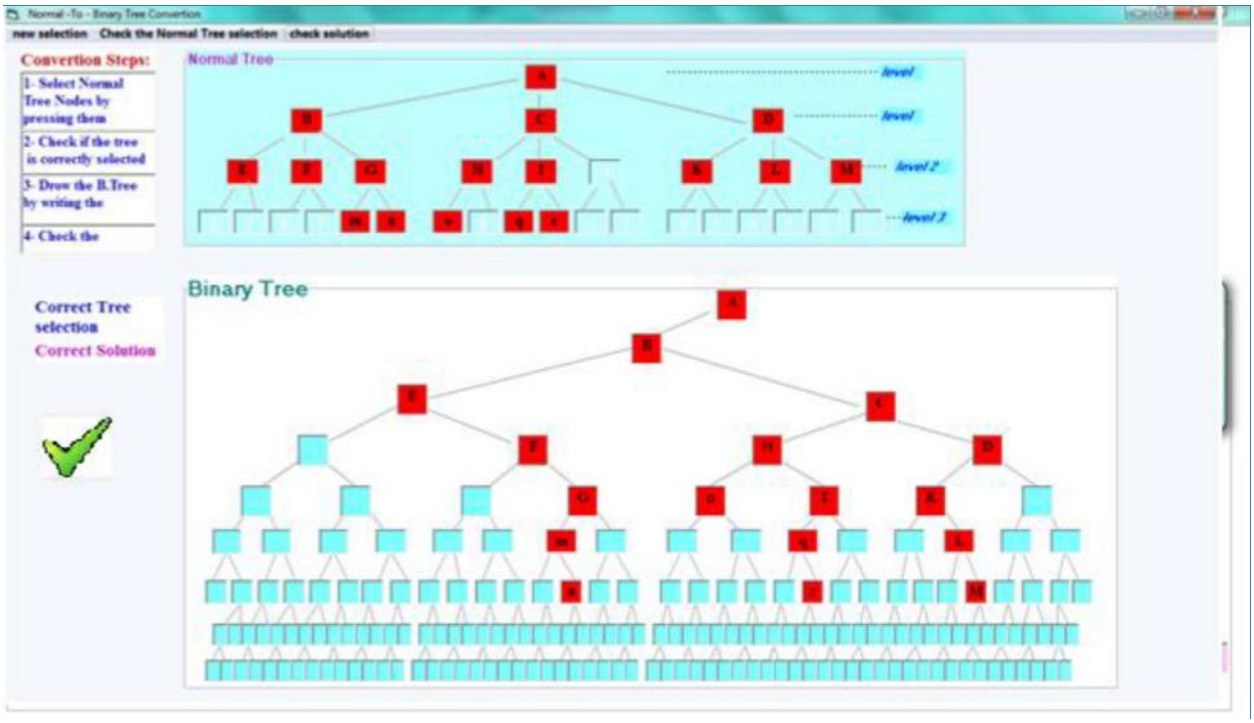


(b)

Figure (A12): a- Binary Search Tree example using step by step execution
b- Binary Search Tree with error solution

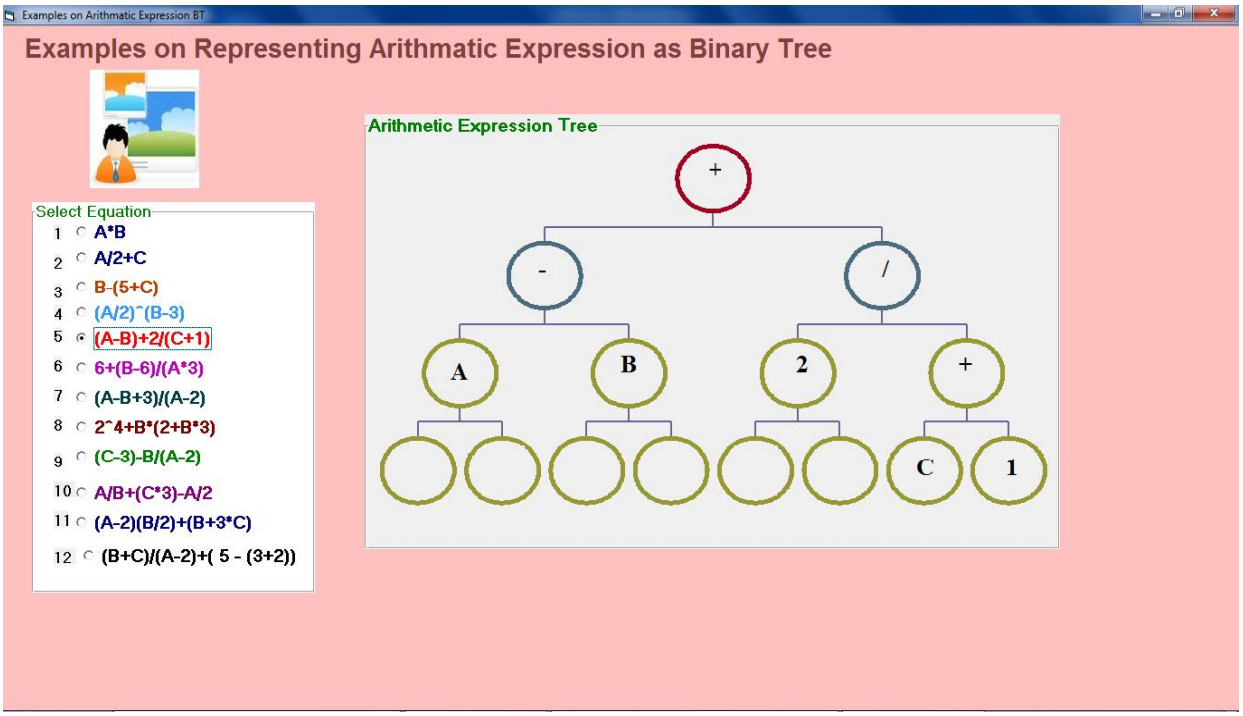


(a)

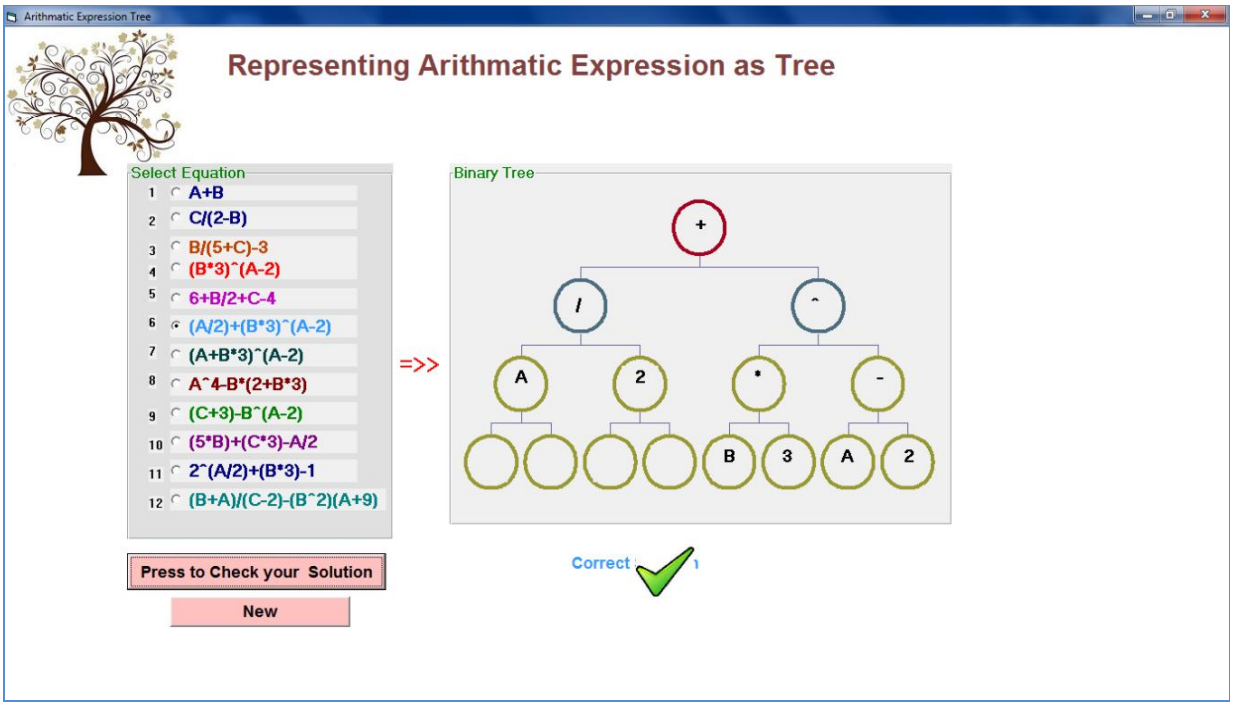


(b)

Figure (A13): a- example for Normal -to- Binary tree conversion
b- Normal -to- Binary tree conversion with correct solution



(a)



(b)

Figure (A14): a- Example for Arithmetic expression representation as binary tree
b- Arithmetic expression as binary tree with correct solution

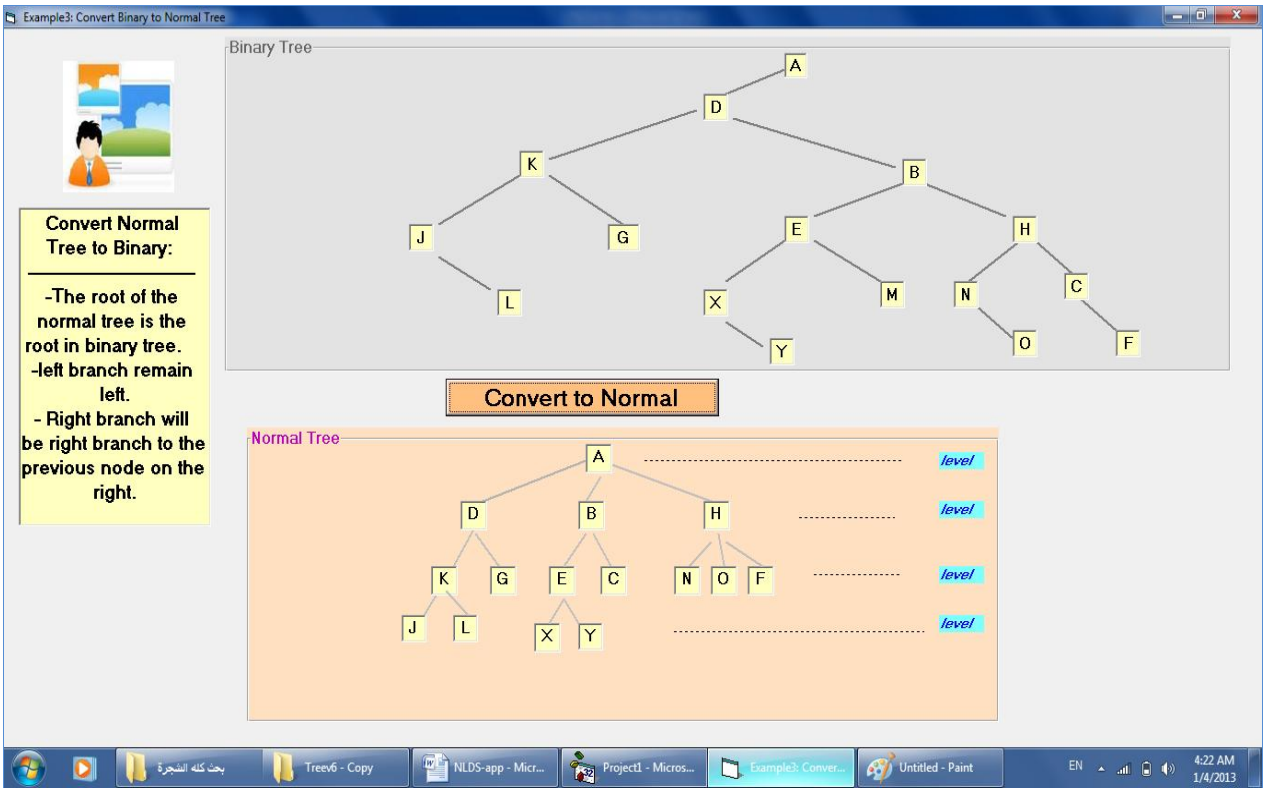


Figure (A15): example on conversion of Binary tree to Normal tree