Analyzing a Tree-Theoretic Framework for Multiverse Structures Modeling

Covering Structure of Tree and Its Relevance to Many-World Interpretation

Ravinka Fathia Adinegara - 13525103
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: ravinkaadinegara@gmail.com, 13525103@std.stei.itb.ac.id

***Abstract*— The probability of the existence of the multiverse i.e. Many-World Interpretation has been investigated many times by the scientific community, arises as the contraposition from the Copenhagen theory in the famous Schrödinger's Cat measurement problem; proposing both universes as real and exist rather than collapsing, thus creating infinitely-many worlds. These many worlds can be visualized using tree due to its similar structure and properties, approaching the quantum measurement problem using structural and computational perspective. The simulation is implemented in C language and modelling the possibility of the observer's consciousness being in each world. The simulation, built with several functions and procedures including the randomizer LCG (Linear Congruential Generator) function, achieves a maximum complexity of $O(n^2)$. This paper delves into the use of discrete math in quantum mechanics, specifically the tree theory within the graph theory in visualizing and analyzing the Many-World Interpretation of quantum mechanics.**

Keywords—tree theory; many-world interpretation; multiverse; universe; LCG; multiverse simulation; complexity analysis; multiverse structures model

I. Introduction

For many decades, there have been still controversies within the quantum physics study field. One of the most-discussed issues within this study around the physicist is the quantum measurement paradox i.e. the paradox of Schrödinger's Cat; where there cannot be absolute outcome of an event when it comes to interpretation of quantum measurements at the microscopic level. On the other hand, human experiences occurrence in an environment where the outcome of an event can be measured exactly—all rely on Newtonian Mechanics or Classical Mechanics. Therefore, there can be several inconsistencies during the measurement of a quantum event as they differ with the classical ones [1].

The controversy has aroused some popular interpretations to either contradict—enhancing the said paradox—or support the quantum measurement paradox. One of the most intriguing proposals is the Many-World Interpretation which was proposed by Hugh Everett in 1957. This proposal argued that there is multiple—likely infinite—worlds in the universe along with the world we are experiencing. For example, when a quantum experiment takes place, all outcomes are realized in a newly created world; in other words, branching. This can happen tentatively, not only in quantum experiments, but also, for example, in explosion of stars during the supernova event [2].

Many experts consider Many-World Interpretation as a difficult concept to grasp; one needs to have strong understanding of philosophy and natural science to comprehend the theory. Indeed, most literature about Many-World Interpretation is discussing the topic from those points of view. However, this interpretation still lacks view from other fields, like structural and computational. Therefore, there must be ways to visualize branching and splitting over the multiverse.

In this paper, we find tree within the graph theory as one of the most suitable ways to simulate the branching process, which involves decision and quantum events. Many-World Interpretation contains branching among the universe; researchers described them as independent worlds; one does not affect the other (child branch). This characteristic shows the same behavior as tree in graph and tree theory. This paper discusses and analyzes what can be done among the process and the model limitation.

II. Theoretical Framework

A. Graph

To elaborate on tree, one must understand graph first, since tree is theoretically a graph with some specific and special requirements. In general, graphs can be used to describe relationships between objects. For one to be called a graph, it must contain collections of vertexes and edges. Therefore, a graph definition is

$$G = V, E \tag{1}$$

In Equation (1), G is acknowledged as graph, V as a set of vertices, and E as a set of edges. In graph theory, there cannot exist a graph where V is a null or empty set, hence V as a set must have at least one child for one to be called a graph. However, E is otherwise; there can be a graph where set of edges

is an empty set. A graph that has zero edges (its E is empty) is called an empty graph [3].

The following are important terminology on graph theory that will be useful before knowing the tree theory.

1) Adjacency: If two vertices are directly connected, they are *adjacent* to each other. For example, vertex A and vertex B are adjacent to each other only if they can reach one another through one edge.

2) Incidency: Say, there is an edge that connects two vertices (A, B). The name of the edge is (A, B) and it is incident to vertex A and vertex B.

3) Degree: Degrees of a vertex is the number of edges that is incident to that vertex. On directed graph—graph with direction on its edges, there are be two types of degree: in-degree and out-degree.

4) Path: If there is a graph G which has two or more adjacent vertices, if there is a way to traverse from vertex A to vertex B through one or more edges, a path which contains those edges to traverse will be established. Length of that path means the number of edges that is used to traverse from vertex A to vertex B.

5) Cycle or Circuit: A path can be referred to as cycle or circuit if the start and end of the path is the same vertex.

6) Connected: If there is a path from vertex A to vertex B, that two vertices are connected. If there is a path from every vertex to every other vertex on a graph G, G is considered a connected graph. Otherwise, G is a disconnected graph [3].

B. Tree

As mentioned earlier, tree is a special graph; to be a tree, a graph should not have a circuit (also called a cycle), which is described as when two vertices can be traversed from one to the other through more than one path across the edges. In addition, that graph must be connected, thus there is only and exactly one path to traverse from one vertex to another. Lastly, a tree must have no direction; a tree is not a directed graph.

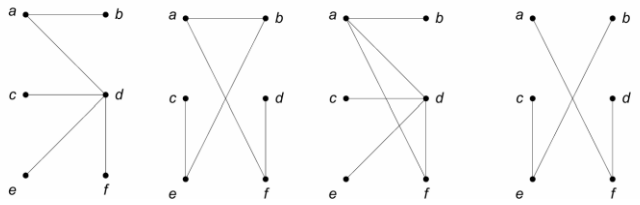


Figure 1 Tree and Not Tree
Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf

On Fig. 1, graph (a) and graph (b) are trees as they meet the criteria for trees: should not have circuits, connected, and have no direction. On the other hand, graph (c) is not a tree due to having a circuit created by vertices a, d, and f. Moreover, graph (d) is also not a tree because it has vertices that do not connect with each other; graph d is two disconnected trees built from vertices (a, f, d) and (b, c, e).

A set of connected trees is referred to as forest. In other words, because trees must not have any circuits, a disconnected graph that does not have circuit is also a forest.

There are several properties that a tree must have. The following are information about tree properties. If there is a graph G, which is as simple and the number of its vertices is n, all of these sentences are applied [4].

1) G is a tree.
2) Every two vertices in G are connected by a single path
3) G has m = n – 1 edge(s), and G is connected.
4) G has m = n – 1 edge(s) and G does not have cycle.
5) G do not have cycle and if there added one edge on G, there can only be one cycle.
6) G is a connected graph and all of its edges are bridges.

If there is one vertex in tree that is considered the “root” of the tree and the tree edges are considered directed edges to the children, that tree is called a rooted tree. The direction of the tree, because it is always assumed to be towards the children from the root, can be discarded in the visualization. There can be more than one vertex to be viewed as root in different visualization depending on how we would like to view it. The following are several terminologies on rooted tree [5].

1) Children and Parent: If vertex A and vertex B are part of a rooted tree T, vertex A is put on top of vertex B, and vertex A and vertex B are adjacent to one another, vertex A is the parent of vertex B and vertex B is the child of vertex A.

2) Path: Identical definition with graph’s; set of edges that can be traversed from one vertex to another vertex (or itself).

3) Sibling: If vertex B and vertex C have the same parent A, vertex B and vertex C are siblings. This only applies to direct parent; two vertices are not siblings if they have different parents even if they have the same ancestor.

4) Subtree: A subtree is part of a tree; for example, if there exists a rooted tree T with a root A that has vertices B, C, D as children, vertex B can be taken and one can create a new tree T’ from B down to its last descendant with B as the root. In this scenario, T’ is a subtree of T.

5) Degree: A bit different from graph’s definition of degree; a degree in tree theory is the number of children a vertex has. Accordingly, for an edge in rooted tree to be considered as a vertex’s degree, it has to be an out-degree—its children, not its parent. A tree’s degree is the maximum degree of all vertices.

6) Leaf: A vertex on tree that does not have any children.

7) Internal nodes: A vertex on a tree that does have children; the vertex that connecting the root to the leaves and the root itself.

8) Level: A root on a rooted tree has a level 0. Other vertices has a level according to number of edges on the path from the root to that said vertex (length).

9) Height/depth: The height/depth of a tree is the maximum level of the tree.

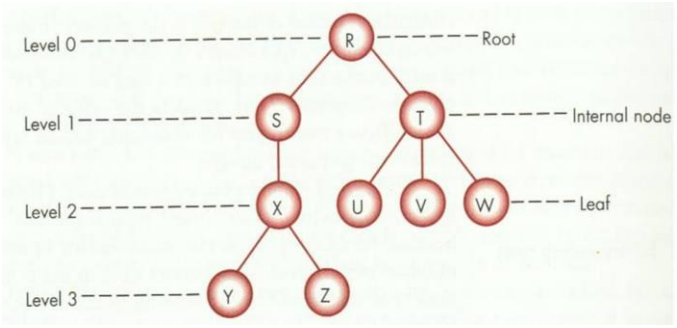


Figure 2 Tree and Its Terminologies
Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/24-Pohon-Bag2-2026.pdf

There are some types of rooted trees. A rooted tree that considers its root's children and descendant to be important is an ordered tree. Therefore, a structurally same ordered tree can be different because their children order are different.

Other than ordered tree, one of the most important and widely used rooted tree is the m-nary tree. If there exist a rooted-tree, which every single node—or vertex; in trees, vertices are often references as nodes—has only *m* or less degrees, that tree can be referred as m-nary tree. Furthermore, an m-nary tree which every single node has exactly *m* degrees except for the leaves is called a full m-nary tree. Full m-nary tree with k internal nodes has nodes of

$$n = mk + 1 \tag{2}$$

Out of all the poly-nary trees, the binary tree is the most crucial tree due to its wide range of applications. Following the m-nary tree rules, there can only be the maximum number of two children for every node. Since there are only two or less children, the children can be differentiated by referencing them as the left child and the right child. Due to this naming, a binary tree is an ordered tree—with the more general order considered the left child as the older sibling.

There are three ways to traverse the binary tree: pre-order, in-order, and post-order. To do the pre-order traversal, one must visit the root first, followed by the left child, then the right child. To do the in-order traversal, they must first visit the left child, root, and lastly the right child. For the post-order traversal, visit the left child, the right child, and end the traversal in root [5].

C. *Algorithm Complexity*

There are various elements and explanations about algorithm complexity, including time-complexity, space-complexity, T(n), notations, etc. However, the most widely used concept in algorithm complexity is the big-O notation, which is also called the asymptotic time complexity. It is a standard notation to show how much work a program does in large numbers. For larger numbers, we usually ignore the slower-growing element of a program and focus on the faster-growing ones. For example, for $T(n) = n^2 + n$, because n^2 growth is faster than the n, its big-O notation is n^2.

In the big-O theory, O(1) is considered the slowest growth (it does not grow; its work is constant and does not depend on input or n), O(log(n)) is the second slowest, followed by O(n), O(nlog(n)), $O(n^2)$, $O(n^3)$, other $O(n^a)$ where a is a constant. The faster-growing notation—usually considered highly inefficient—are $O(2^n)$ which usually appear in a brute-force solution of a problem and lastly, O(n!) [9].

D. *Many-World Interpretation*

Many-World Interpretation was introduced by Hugh Everett in 1957. Everett found paradoxes in previous theories; stating that the existence of more than one observer would lead to inconsistencies. It stated that there are two ways that a state function could alter:

1) Discontinuous change that was brought by the observation of a quantity that have eigenstates $\phi_1, \phi_2, \ldots$. Then, the state ψ will be changed to the state ϕ_j; the change has probability $|\psi, \phi_j|^2$.

2) Continuous i.e. deterministic change of state of a non-observed system according to the wave equation. In this equation, U is a linear operator.

$$\frac{\partial \psi}{\partial t} = U\psi \tag{3}$$

Say, there is a system S that is measured by an observer A. Together, (A + S) created another system for another observer B. If we ignore B's possible contribution to quantum mechanical description towards A + S, there must be an alternative description for a system that has an observer. On the contrary, if we consider B's quantum description to A + S, which will assign a state function ψ^{A+S}, if B does not interact with A + S, A + S would change its state according to Process 2; even if A is, too, considered an observer of system S. This is due to the existence of B as an observer of system A + S; there cannot be a way Process 1 can be happening if B was not in the position of interacting with A + S system.

To suit—and avoid—the paradox, five different alternatives were offered. The one that favored his proposal of Many-World Interpretation the most was alternative 5, which propose a complete abandonment of Process 1; pure wave mechanics is assumed to be applicable for all physics system, which include observers i.e. measuring apparata. The process of observation is assumed to be able to be described completely by the state function of a composite system containing the observer and the object-system. At all conditions, this composite system obeys the wave function [6].

As a result of this proposal/alternative, it is elaborated further that none of the states of a system is more "real" than the other. Each state of a system exists with the amplitude in superposition, but the observer's state has altered—it becomes entangled with the system. In conclusion, all the "branching" events are considered the universe's event, whereas we as the observers experience only one lives (branch).

E. *Quantum Events*

The following are several quantum events that, on quantum physics, can occur.

1) Spin-Up and Spin-Down: There are two measurable states on the spin of an electron, thus being $|0\rangle$ (spin up) and $|1\rangle$ (spin down); based on Stern-Gerland Experiments. State of the electron would be a superposition of up and down [7].

$$|\text{electron}\rangle = \alpha|0\rangle + \beta|1\rangle \quad (4)$$

2) Transmitted and Reflected: If a photon encounter the 50/50 beam-splitter, the whole photon is both transmitted and reflected; happened in different places (superposition) [7].

3) Photon-Detected and Photon-Missed: Within the same beam-splitter experiment, a detector can either detect or miss the photon—the whole photon is both transmitted and reflected—happened in different places (in this case, different world); therefore, on one universe like the one we experiences, its either the detector which detects the transmitted photon or (exclusive) the detector which detects the reflected photon receiving a photon [7].

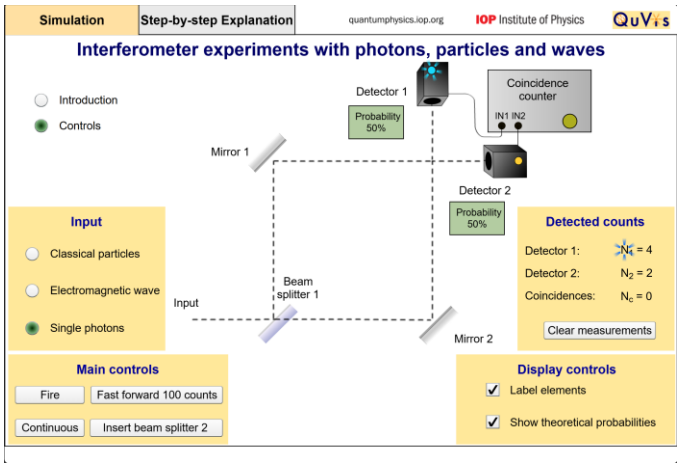


Figure 3 Simulation Attached in Literature [7]: Beam-Splitter Experiment
Source: https://www.st-andrews.ac.uk/physics/quvis/simulations_html5/sims/photons-particles-waves/photons-particles-waves.html

4) Stable and Decay: There is not a way we can predict when will the radioactive nucleus decays of a single atom would happen during the, say, Schrödinger's Cat thought experiment. On literature [7], it is stated that after the measurement (opening the box), the superposition collapse—this is the interpretation other than Many-World Interpretation. In Many-World Interpretation, the world does not collapse according to one's action; both world exist within the universe. We as an observer get to experience one single world; either the radioactive nucleus is stable or decay during our measurement.

F. LCG Algorithm

The Linear Congruential Generator or abbreviated by LCG Algorithm is considered the most famous algorithm that roles as random number generators in computer simulations. The definition of LCG Algorithm is

$$X_n = aX_{n-1}+c \ (mod\ M), \quad (5)$$

$$u_n = \frac{X_n}{M} \quad (6)$$

a, c, and M are integers and X_n with n = 1, 2, … is a sequence of integer with the range of $0 \leq X_n \leq M$ [8]. Thus, seed X_0 is determined first before doing the Eq. 5. If one wants to improve the randomness of the produced number, they can use methods to change the seed; for example, using time to determine the seed. Other ways are incrementing the seed with another random number and repeatedly doing the equation different times to produce sense of more-randomness.

III. Proposed Method

In this simulation, we visualize the multiverse structure as a binary tree. To visualize the multiverse structure as a tree, we must initialize what each property of tree represents what property of the universe. In this case, nodes from the tree represent both the past and current worlds that exist in branching process. The special nodes on the tree: leaves represent current lives in that world branch, as the internal nodes represent the past. The edges represent the branching process; visualizing how a (hypothetical) decision and a quantum event can result in two branches: both equally exist in different places. Lastly, the tree, given the tree is a weighted graph, the weight can represent a (random) possibility of the observer (player of the simulator) on being on that life.

A. Initializing the Universe

As rooted trees have roots, the universe has a starting point. Thus, we make the universe have a beginning point; there represented by the root. The beginning point of the universe (we called it the "origin") characterizes with 100% possibility of the observer to be in, has no child (given no branching process has occurred in initialization), and has no sibling. From this root/starting point, the player can thus make a decision to see the hypothetical branching if they ended up doing it or not, or do a (randomly picked) quantum experiment that produces a quantum event.

B. Making a Decision and Quantum Event

Hypothetically, when an observer is in a dilemma in making a decision, there are two choices that the observer can make: to do or not to do. This could be interpreted as two different uncertain outcomes (before the observer finalizes their decision) which can be related to the branching process of the universe. However, author would like to note that this branching process (of decision) is not supported by the Many-World Interpretation theory; it is treated as hypothetical in the paper and in the latter simulation.

The randomized percentage of the outcome, to do and not to do, will make up to its parent's percentage. For example, if the observer decides, branched from the origin of the universe, the possibility of the observer is on the world where they do the activity and the possibility of the observer is on the world where they do not do the activity will make up to 100%.

On the other hand, quantum events are supported by the Many-World Interpretation. There are pairs of quantum events that can occur in real world, including the ones that occur during

a quantum experiment. The ones that will be used in this simulation are Spin-Up and Spin Down, Transmitted and Reflected, Photon-Detected and Photon-Missed, lastly, Stable and Decay. Similar to the hypothetical making-a-decision branching, a pair of quantum events will branch from one or more parents and both of its percentage are added to make up for their parent's percentage.

IV. Program Implementation

A. Limitation

This program has several limitations, one of which is limited branching due to limited resources like memory. In real universe, the branching of the worlds can be infinitely-many, whereas this simulation does not cover. Moreover, the label (or an activity) for when making a decision is also limited to 95 characters. The floating-point rounding behavior can also decrease the accuracy of percentage. The percentage is printed down to 10^{-6}; as mentioned, representing the possibility of an observer to be in a specific life from branching.

B. Library and Constants

To ensure that the program runs accordingly, there are several libraries and constants that need to be included in the program. Using C language, we used stdio.h, stdbool.h, string.h, and stdlib.h libraries. Other than that, we defined some constants like MAX_CHILDREN, MAX_NODE, MAX_LABEL, MAX_DEPTH, and MAX_PARENTS. In addition, we differentiated each type of nodes with different character: percent (%) for root or the origin of the universe ROOT, ampersand (&) for DECISION, and hashtag (#) for quantum events QUANTUM.

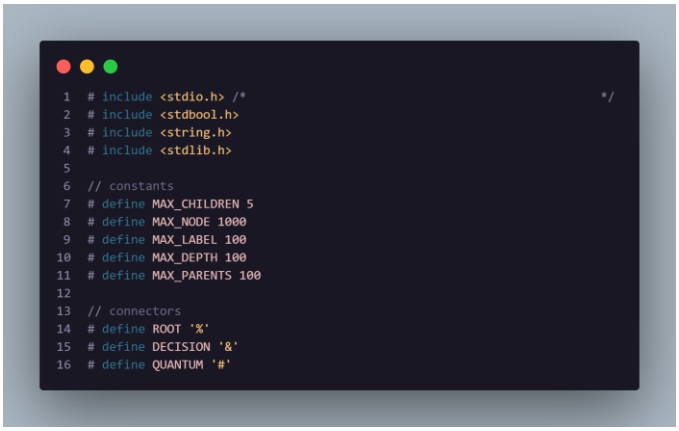


Figure 4 Libraries and Constants Needed
(Source: Author)

C. Abstract Data Structure

There are two abstract data structures that are used in order to make this program. One of them is Node which has connector, label, node_id, children_id, childcount, parent_id, indent, percentage, and is_last. The other is arrNode which has array of nodes, leafIds, nEff, and leafNeff.

Figure 5 Abstract Data Type
(Source: Author)

D. LCG Function

To randomize the number of parents a child can have, parents IDs, and percentage of each branching, we use the LCG (Linear Congruential Generator) which generates a random number when called. To ensure the seed is different each call, we use a counter incremented every call and then it will be added to the seed. The a, c, and M are standard-used variables for that specific function.

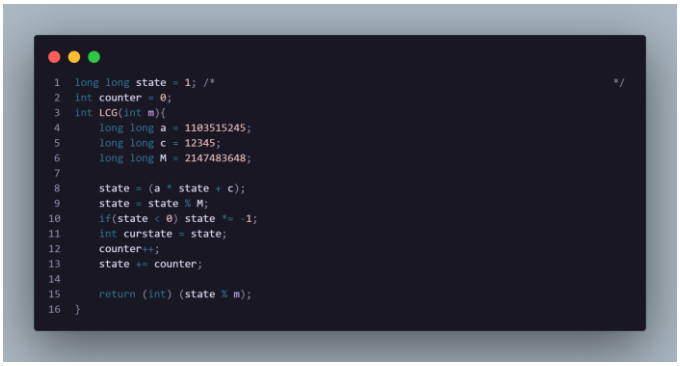
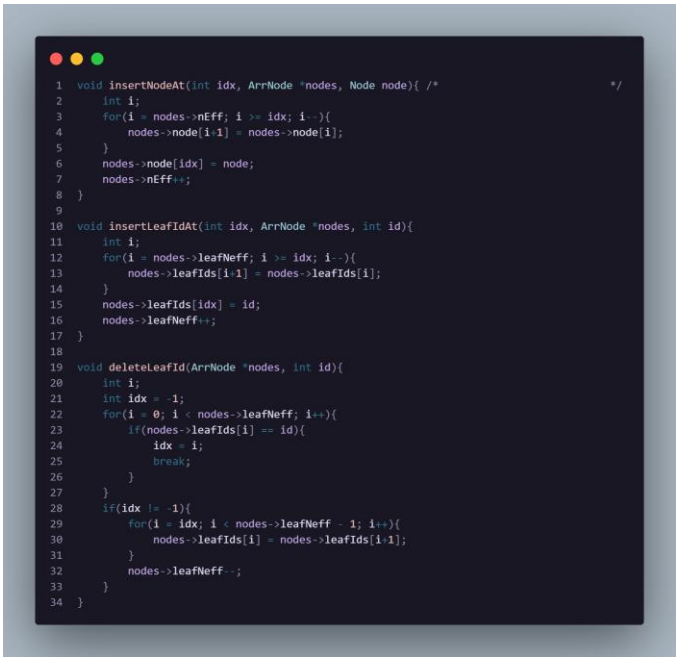


Figure 6 LCG Function
(Source: Author)

E. Helper Procedures

To manage the variables that will have the abstract data type, we will use insertNodeAt, insertLeafAt, and deleteLeafAt. insertNodeAt is a procedure that is used to insert a new node to the array of nodes on a variable with type arrNode. insertLeafAt has same logic as insertNodeAt, but is used to insert a new node ID to mark it as a leaf. Lastly, deleteLeafAt is used to delete a node ID on the leafIds array. This procedure is used when one leaf branches into two different worlds; that leaf is now an internal node and no longer a leaf.



```
void insertNodeAt(int idx, ArrNode *nodes, Node node){ /*                    */
    int i;
    for(i = nodes->nEff; i >= idx; i--){
        nodes->node[i+1] = nodes->node[i];
    }
    nodes->node[idx] = node;
    nodes->nEff++;
}

void insertLeafIdAt(int idx, ArrNode *nodes, int id){
    int i;
    for(i = nodes->leafNeff; i >= idx; i--){
        nodes->leafIds[i+1] = nodes->leafIds[i];
    }
    nodes->leafIds[idx] = id;
    nodes->leafNeff++;
}

void deleteLeafId(ArrNode *nodes, int id){
    int i;
    int idx = -1;
    for(i = 0; i < nodes->leafNeff; i++){
        if(nodes->leafIds[i] == id){
            idx = i;
            break;
        }
    }
    if(idx != -1){
        for(i = idx; i < nodes->leafNeff - 1; i++){
            nodes->leafIds[i] = nodes->leafIds[i+1];
        }
        nodes->leafNeff--;
    }
}
```

Figure 7 Miscellaneous Procedures
(Source: Author)

F. *Branching*

There are two logically similar procedures related to the branching of the universe, those are create_scenario_decision and create_scenario_quantum. Both procedures will pick random total of parents, pick random parents, and branch into two worlds from those parents with percentage. The difference between both events is that create_scenario_decision will produce two worlds with the events <activity>(y) and <activity>(n) with <activity> being a user input, whereas create_scenario_quantum will produce two worlds with randomly picked a pair of quantum events, along with the percentage.

In both procedures, the children are inserted right after the parents, hence creating an array which is a preorder sentence. This will make the printing process less difficult since it can loop the array of nodes from index zero up to the last index.

Figure 8 Branching Procedure
(Source: Author)

G. *Print Procedures*

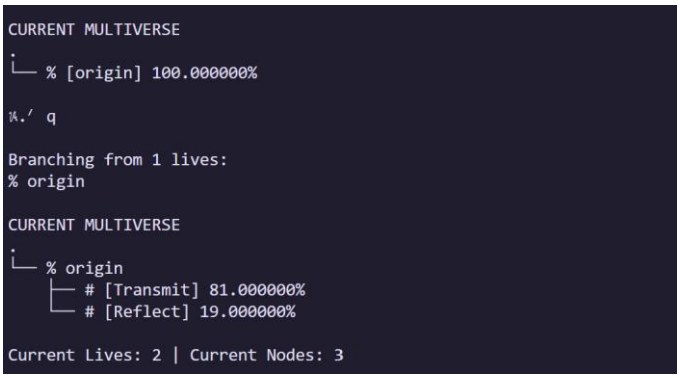
Lastly, the procedures in figure below are used to print current multiverse that have been created. Procedure print_tree will print the tree with a linear loop—as mentioned, the array of node already is a preorder sentence—and procedure print_stat will print the total of nodes that have been created and the total of leaves: the current life that exist.

```
void print_tree(ArrNode nodes){/*                                            */
    bool active[MAX_DEPTH] = {false};
    int i;
    printf("\nCURRENT MULTIVERSE\n");
    printf(".\n");
    for(i = 0; i < nodes.nEff; i++){
        int depth = nodes.node[i].indent;
        active[depth] = !nodes.node[i].is_last;

        int j;
        for(j = 0; j < depth; j++){
            if(active[j]) printf("│   ");
            else printf("    ");
        }

        if(nodes.node[i].is_last) printf("└── ");
        else printf("├── ");
        printf("%c ", nodes.node[i].connector);
        if(is_leaf(nodes.node[i])) printf("[");
        printf("%s", nodes.node[i].label);
        if(is_leaf(nodes.node[i])) {
            printf("]");
            printf(" %.6f%c", nodes.node[i].percentage, '%');
        }
        printf("\n");
    }
    printf("\n");
}

void print_stat(ArrNode nodes){
    printf("Current Lives: %d | Current Nodes: %d\n\n", nodes.leafNeff, nodes.nEff);
}
```

Figure 9 Print Procedures
(Source: Author)

H. *Simulation Example*

Firstly, after running the simulation, the player can see the origin universe or the root of the tree. After that, the player can run the first command, and the first two branches will appear out of the root.



```
CURRENT MULTIVERSE
.
└── % [origin] 100.000000%

%.' q

Branching from 1 lives:
% origin

CURRENT MULTIVERSE
.
└── % origin
    ├── # [Transmit] 81.000000%
    └── # [Reflect] 19.000000%

Current Lives: 2 | Current Nodes: 3
```

Figure 10 Beginning Testing Program
(Source: Author)

The available commands are m <activity> to make a decision, q to do a random quantum experiment (and thus producing a quantum event), and supporting commands like w to show the current multiverse, s to show the current stat, a to show the simulation's essence, h to help, x to quit.

After some more commands, the multiverse will increase in size and have more branches. This is due to branching which increases the total number of leaves available to be chosen as a parent. The tree growth (not the program) is close to $O(2^n)$ due to its binary structure.

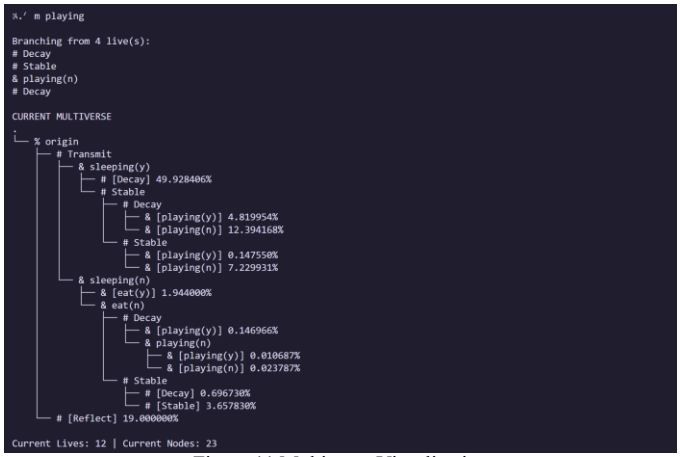


Figure 11 Multiverse Visualization
(Source: Author)

Therefore, the simulation works well to visualize the growth of a multiverse. We can see that the percentages are 49,93 + 4,82 + 12,39 + 0,15 + 7,23 + 1,94 + 0,15 + 0,1 + 0,02 + 0,69 + 3,65 + 19 which resulted in around ~100%.

I. Complexity Analysis

After knowing that the simulation works well, we can estimate the complexity of each function and procedure to know which function may noticeably or unnoticeably slows down the program if we were to increase the constants like MAX_NODES. The following are the estimated complexity, or the big-O, of each function and procedure.

1) LCG: The LCG implementation does not have any loops, thus resulting in O(1) complexity.

2) insertNodeAt: It is a function that shifts the array of nodes to put a new node in desired index. Therefore, its big-O is O(n).

3) insertLeafIdAt: Same logic as insertNodeAt; its big-O is also O(n).

4) deleteLeafId: It has two separated loops, one to ensure that the targeted ID exists in the array, which has the complexity of O(n), and one to shift the rest of the array to the left and overwrite the targeted ID, also has O(n) as its complexity. Therefore. O(n) + O(n) is the deleteLeafId complexity, which results in O(n).

5) make_universe: This procedure creates the first node without any loops, it has the complexity of O(1).

6) is_leaf: This is a function to check whether a node is a leaf. It only checks whether its child count is zero or not, therefore it has the complexity of O(1).

7) getNodeIdxbyId: We can use this function to search for the index of a node based on each ID. Like regular searching problem, its complexity is O(n).

8) make_decision: A procedure that extend the create_scenario_decision and create_scenario_quantum procedures. It does not have any loops, and its big-O is O(1).

9) create_scenario_decision: There is a loop based on LCG's output and inside the loop, it is calling the insertNodeAt and insertLeafAt. After the loop, it is calling the deleteLeafId procedure. Its complexity is closer to $O(n^2)$ since O(n) * (O(n) + O(n)) + O(n) is $O(n^2)$.

10) create_scenario_quantum: Very similar to create_scenario_decision; its complexity is $O(n^2)$.

11) print_tree: As mentioned, its main feature is a linear O(n) loop due to the array of nodes being a preorder sentence. However, there is a loop inside it that prints the indentation of a node, making it closer to $O(n^2)$.

12) print_stat: It only prints the effective n of array of nodes and array of leaf ID, making the complexity O(1).

After estimating each function and procedure complexity, we know that there are five functions and procedures that have O(1) complexity, four that have O(n) complexity, and three that have $O(n^2)$ complexity; not including the main function since it is infinitely-looped and only stopped if the user choose to stop the program. Therefore, the program can be considered sufficiently effective as it limits the maximum nodes to 1000; it is not handling a very large number.

V. Conclusion

This paper covers how a tree in graph theory has lots of similarities in structure with universe in Many-World Interpretation, therefore can be a tool to visualize the growth of the universe and the process of its branching, approaching the quantum measurement paradox better as quantum world can feel unnatural due to our macroscopic physical existence. Tree visualization helps build the structure of the universe in Many-World Interpretation; it intuitively guides us as an observer of a system to understand that on a quantum events, we are part of the system (due to the more-than-one observer paradox) and therefore branch along with the quantum system and create many different worlds.

Rooted trees have leaves, internal nodes, and edges along with degrees of the nodes; all these properties can represent properties of a hypothetical multiverse. The simulation made came along with percentages for each leaf (represent lives) not to represent the possibility of each world existing (because it will contradict the Many-World Interpretation itself), but rather the possibility of the observer's current consciousness (player) living in that specific world. Humans understand a concept better if they can relate themselves to the concept.

The complexity of the program is sufficiently effective as the maximum big-O of the functions and procedures is $O(n^2)$. However, the effectiveness of this program still has a long way to go to be improved and/or to be refactored. The feature can also be expanded or be investigated if needed. In conclusion, the tree theory relating to graph theory in discrete math has a wide use among lots of fields, and one of them is on the quantum mechanics interpretation, specifically Many-World Interpretation

Source Code

https://github.com/ajalku/simple-universe-tree-visualization

Video Link at Youtube

[Part 1] https://youtu.be/kQ8UwSNsEV8

[Part 2] https://youtu.be/KsbGSMpA3iU

ACKNOWLEDGMENT

Firstly, the author would like to thank The God, as she is not Their strongest soldier; she must learn about and approach Them more. Secondly, Dr. Rinaldi Munir is the one that she would like to thank because he provided his students with broad and deep resources in discrete math to write their papers and day-to-day learning. Lastly, the author can never thank her parents enough as they never stop trying to provide her with a lovely life and the people around her that let her coexist.

REFERENCES

[1] A.J. Leggett, "The quantum measurement problem," *Science*, vol. 307, no. 5711, pp. 871–876, Feb. 2005, doi: 10.1126/science.1109541. Accessed 17 June 2026.

[2] D. Wallace, "The Everett interpretation of quantum mechanics," in E. N. Zalta (ed.), *The Stanford Encyclopedia of Philosophy (Fall 2021 Edition)*. [Online]. Available: https://plato.stanford.edu/entries/qm-manyworlds/. Accessed 17 June 2026.

[3] R. Munir, "Graf (Bag. 1)," https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf, Accessed 17 June 2026.

[4] R. Munir, "Pohon (Bag. 1)," https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/23-Pohon-Bag1-2026.pdf, Accessed 17 June 2026.

[5] R. Munir, "Pohon (Bag. 2)," https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/24-Pohon-Bag2-2026.pdf, Accessed 17 June 2026.

[6] H. Everett III, J. A. Wheeler, B. S. DeWitt, L. N. Cooper, D. van Vechten, and N. Graham, The many-world interpretation for quantum mechanics, New Jersey, Princeton University Press, Accessed 18 June 2026.

[7] C. Hughes, J. Isaacson, A. Perry, R. F. Sun, and J. Turner, Quantum computing for the quantum curious, Cham, Switzerland, Springer Nature Switzerland AG, 2021, Accessed 18 June 2026.

[8] S. Tezuka, "Linear congruential generator," in The Springer International Series in Engineering and Computer Science, vol 315, 1995, (pp. 57-82), Accessed 18 June 2026.

[9] R. Munir, "Kompleksitas algoritma (bagian 2)," https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/24-Pohon-Bag2-2026.pdf, Accessed 17 June 2026.

STATEMENT

I hereby declare that this paper that I wrote is my own writing: not an adaptation or translation of someone else's paper and not plagiarized.

Bandung, 19 Juni 2026

Ravinka Fathia Adinegara (13525103)