Application of Graph Theory to Diagrammatically Solve the 'Gold in a Box' Logic Puzzle

James Chandra 13519078[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13519078@std.stei.itb.ac.id

***Abstract*—The application of various sub-topics in discrete mathematics can be obscure and often unthinkably odd. Graph theory also shares this characteristic where its applications might lie in fields such as crime analysis, molecular chemistry, and even sociology. The use of graph theory also proves to be prevalent in other sub-topics of discrete mathematics other than said graph theory itself, such as logic, among other sub-topics. This paper will explore ways of which to implement fundamental graph theory principles to the field of logic, in the form of solving logic puzzles which usually involves the process of manual mental labor. The diagrammatic method of solving said puzzles will also be implemented as an algorithm in the programming language Python with an uncomplicated complexity and relatively moderate degree of modularity. This method will hopefully prove to be more visually comprehensible in tackling the problem and can provide a more structured solution with a clear and concise flow of steps.**

***Keywords*—Algorithm, Diagrammatic, Graph theory, Logic.**

I. Introduction

Logic puzzles can be defined as problems that require some form of deduction or logical reasoning to solve. The first logic puzzle ever created can be found in "The Game of Logic", a book published in 1886 by Charles Lutwidge Dodgson, better known by his pseudonym Lewis Carroll, who was a children's fiction writer (notable works include "Alice's Adventures in Wonderland" and "Through the Looking-Glass") as well as a mathematician specializing in the field of mathematical logic. Dodgson's book contained games that required readers to confirm a conclusion from a handful of given premises.

Raymond M. Smullyan went on to expand the branch of logic puzzles and written over 10 publications on the topic over his career. Smullyan also popularized the Knights and Knave logic puzzle—one of the most recognizable logic problems and is heavily used in the field of logic and computer science. The puzzle above is based on a story containing a similar problem; the riddle of the two doors and two guards, where one door contained treasure while the other would bring death, and only one of the two guards spoke the truth.

The 'Gold in a Box' puzzle (other version such as the 'Thief Accusation' puzzle) also comes to mind, as the solution for it can be found with a similar course of thought which would be to reverse evaluate different scenarios, this puzzle's distinctive characteristic would be the fact that all the boxes already have signs stating things about itself/other boxes, instead of first having been asked to and only have one chance to ask so.

The process of reverse evaluation can be done using truth tables, as to give a more visually clear representation of the scenarios and statements on the boxes. Though that is the case, there still could be merit (especially for pedagogical purposes) in exploring other ways to do this such as by applying graph theory to further expand upon this visual representation to give a more intuitive perspective on the problem and its solution.

The following sections will cover a handful of topics from the fundamentals of graph theory itself, the use of graph theory, as well as other related terminologies that hopefully will give an image as to how we will achieve said things. The paper will also cover points such as creating algorithms based on the method and analyzing the applicability of the algorithm.

II. Fundamental Theorem

This section will mainly cover the definitions as well as theoretical knowledge needed as a pre-requisite to fully grasp what will be discussed in the following sections. Among other topics, an exposition of graph theory as a sub-discipline of discrete mathematics will heavily be focused upon, along with other minor subjects such as an overview of logic puzzles themselves.

A. Graph

In layman's terms, a graph could be seen as a network that helps show the interconnected-ness or relationships between various components. To better describe and help visualize the analogy, consider the following propositions. "Anne & Barry are friends, Barry & Charlie are friends, and Charlie & Anne are friends". The components referred to in the layman's definition in this instance would be the list of people which would be Anne, Barry, and Charlie while the relationship/interconnectedness between the components would be their friendship as stated in the original proposition.

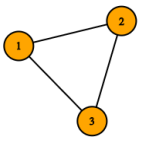

Fig. 2.1 Graph representation of referenced prompt

Parallels could be drawn from the prompt to a closer, much more accurate definition of a graph, the components Anne, Barry and Charlie are represented by the nodes/vertices of the graph labeled 1,2,3 respectively (note that in this case, the representation and labeling of the components are ordered arbitrarily). The bi-directional friendship (more on bidirectionality of graphs will be explained in later passages) of the components/people stated are defined by the edges of the graph, an example of this would be that of the node labelled '1', a relationship is shown between said node with the node labelled '2' and '3', which represents the friendship of components 'Anne' & 'Barry' as well as 'Anne' & 'Charlie'.

Hence to formalize the definition, according to West (1996), a graph $G = (V, E)$ is a set of vertices V and edges E where each edge (u, v) is a connection between vertices (note $u, v \in V$). If the graph in figure 2.1 were to be written in mathematical notation, the vertex set V would yield {1,2,3} while the edge set E would be denoted as {(1,2), (2,3), (1,3)}.

B. Graph Variant

Munir (2010) drew distinctions between different types of graphs, categorized based on the presence/lack thereof any looping edges or multiple (double-incidental) edges in a graph. These categories are as follow, simple graphs, and unsimple graphs that can be further categorized into multigraphs and pseudo-graphs.

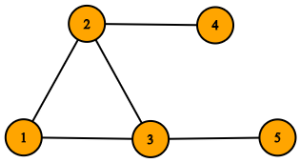

Fig. 2.2 Simple graph

Shown in the above figure is a simple graph, which can be defined as graphs that do not have any looping or multiple edges present.

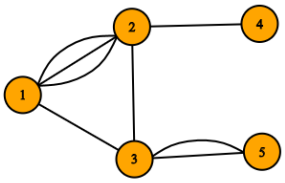

Fig. 2.3 Multigraph

As seen above, the multigraph can be identified by the number of coinciding/incidental edges over two of the same vertices, a graph can be categorized as a multigraph on the condition that it has one or more incidental edges.

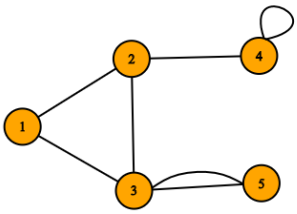

Fig. 2.4 Pseudo-graph

Pseudo-graphs, belonging to the same category of graphs as multigraphs—as unsimple graphs, can be identified by the presence of a looping edge in a graph or an edge that spans across two of the same vertices.

The following method of categorization will later prove to be more of practical use in this particular paper. Graphs can also be distinct from each other based upon its directionality, and can be categorized into undirected graphs, as well as directed graphs or digraphs.

Figures 2.2, 2.3, and 2.4 all depict an undirected graph, a graph that does not take consideration for the directionality of an edge, therefore in other words, an element of the edge set E that is of (u, v) is indistinguishable from (v, u) with a note that for a graph with a vertex set V, $u, v \in V$.

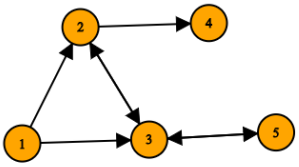

Fig. 2.5 Directed graph

A directed graph, such as shown above, demonstrates the distinction said graph makes upon the connection between two vertices. As opposed to the previous instance for undirected graphs, the visual representation of digraph draws a distinction between an element of an edge set (u, v) and of (v, u) for the fact that digraphs are drawn with arrows indicating the source vertex and the destination vertex.

As said in the prior paragraphs, this graph will prove to be useful as it will ease the process of monitoring specific vertices (more on this will be discussed in following paragraphs along with the supplementary terminologies required to cover the subject).

C. Terminology

A few basic terminologies of graph theory include adjacency, incidence, isolated vertex, null graph, and degree (Munir, 2010). Note that not all terminologies are going to be covered in this paper, as more deeper terminologies such as circuits/cycles and paths will not have much merit in discussing.

Adjacency describes the characteristic of two vertices that are connected with at least one edge (u, v), another term for this characteristic would be 'neighboring' vertices.

Incidence on the other hand, as touched upon previously as well describes the relationship between an edge and corresponding vertices/nodes that it is connected to, hence for

an element of an edge set $e = (v_j, v_k)$, e is said to be incidental with vertices v_j and v_k.

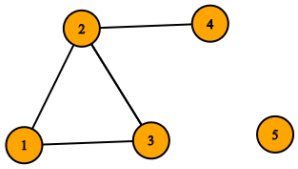

Fig. 2.6 Isolated vertex

In the above graph, the vertex labelled '5' is coined an isolated vertex because of the fact that it does not have any incidental edges connected to it. Hence it can be said that there are no isolated vertices for the graphs in previous figures.

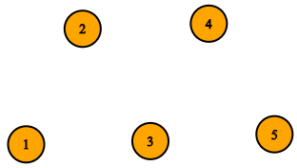

Fig. 2.7 Null graph

A null graph can be defined as a graph without the presence of edges or can also be stated as a graph with a null set as its edge set. From the previous definition of isolated vertices as well, we could also rephrase the definition of null graphs as only having isolated vertices for all the graph's vertices.

The final terminology discussed will be the concept of degrees, degrees can be defined as the amount of edges incidental with a specified vertex, this operation can be denoted as $d(v)$. Consider the simple graph in figure 2.2, an operation of $d(1)$ would yield the degree of 2, shown by edges (1,2) and (1,3) while an operation of $d(3)$ would yield the degree value of 3.

Note that the concept of degrees gets further broken down for directed graphs, this breakdown consists of inbound degree or in-degree(the amount of edges incidental with the specified vertex that has the specified vertex as the destination vertex) and outbound degree or out-degree (the amount of edges incidental with the specified vertex that has the specified vertex as the source vertex).

Consider the digraph in figure 2.5, an operation of $d_{in}(1)$ would yield the value 0 as there are no inbound edges going into the node labelled '1', while the operation $d_{out}(1)$ would yield the degree value of 2 which refers to the edge (1,2) and (1,3). Take note that the concept of directed degrees (in-degree and out-degrees) will be part of the main methodology into how a 'Gold in a Box' logic puzzle can be solved utilizing graph theory.

D. Graph Representation

As the paper will also eventually cover applicative ways of implementing the solution of the 'Gold in a Box' logic puzzle algorithmically, a way to restructure these graphs into process-able data structures must be thought of. There are several options to do this such as using adjacency matrices, incidency matrices, as well as adjacency lists.

Briefly, an adjacency matrix maps vertices to one another through a matrix, where the fundamental rule to form the matrix are as follow "exists an edge (u,v), then it shall be indicated with a 1" while if an edge between two vertices does not exist, it will be denoted as 0. Slightly varying methods of forming adjacency matrices exist, mainly differing in the type of graph that is trying to be represented. For example, an undirected graph can be represented as a symmetric matrix as edges have bidirectionality, while the opposite is true for digraphs [as edge (u,v) does not imply the existence of edge (v,u)]. Adjacency matrices can also be utilized to represent unsimple graphs by specifying the amount of coinciding/double edges present.

Among other methods, incidency matrices can also be utilized to represent a graph, this is done by mapping nodes to edges on the graph, an incidental edge would be denoted as 1, and would be denoted as 0 otherwise. Similar to the adjacency matrix, this graph representation can also be implemented as lists, in which the index of a list would represent vertices in a graph while the elements of each index would be put inside an array of all adjacent nodes with the original specified node.

E. Logic

Logic is a mathematical field that mainly discusses propositions (premise and conclusion derivation/evaluation) and interrelationship. Propositions can be defined as a declarative sentence that has a truth value (Genesereth & Kao, 2016).

In logic, propositions can be assigned to variables, then have truth values attributed to them, this process of determining truth value is evaluation, where an interpretation of a language or a set of variables is said to satisfy a proposition if and only if the proposition is true under that interpretation. Consider the following example, "Anne is Caucasian" and "Barry is Asian", the proposition "Anne and Barry are Caucasian" would evaluate to be false while the proposition "Anne or Barry is Caucasian" would evaluate to be true.

The same could also be done, but in reverse, hence it can be said that an interpretation can be derived from a set of propositions, this process of reverse evaluation can be done by using truth tables and eliminating rows where the interpretation do not satisfy the sentence, leaving behind the possible interpretations of the sentence.

III. METHODOLOGY

A. Knights and Knaves

The solution to the puzzle is first going to be shown by the method of reverse evaluation, as said in the previous section, and for that purpose, an example of the popular aforementioned knights and knaves puzzle will be used.

The prompt of the knights and knaves problem are as follow, consider the fact that knights always tell the truth, while knaves always lie. In an instance, there are two people labelled A and B (note their identity of whether or not they are knights or knave are initially obscured), in which person A says that both A and B are knaves.

Intuitively the solve for this puzzle would be to go through the scenario based on person A's statement. A scenario where A is an actual knight would be implausible because an actual knight would reveal their own identities to be knights, hence it

can be concluded that person A is a knave, and the lie of the statement lies within the fact that person B is a knight, not a knave.

Table 3.1 Truth table for knights and knaves

Line	A	B	We are both knaves
1	Knight	Knight	F
2	Knight	Knave	F
3	Knave	Knight	F
4	Knave	Knave	T

The table above shows a more structured way to think about the reverse evaluation process, the truth table tabulates all possible combination of roles A and B might have as well as the truth value of the statement person A gave which is why the statement "We are both knaves" only evaluate to be true in line 4. The process of elimination can be done by first crossing out line 1 and 2 for the reason of A being a knight and A telling a lie, then line 4 can also be eliminated because if so, A would be a knight and would have told the truth. Hence the only possible interpretation is for A to be a knave and for B to be a knight.

B. Gold in a Box

Now that a thorough understanding of reverse evaluation have been established, consider the following prompt for the 'Gold in a Box' logic puzzle. There are three boxes, one containining gold while the other two empty, each of the boxes have hints about where the gold lies, but only one of these hints are true. The hints on box #1 states that the gold is not in box #1, box #2 also states that the gold is not in box #2 while box #3 states that the gold is in box #2, the next prompt would be to figure out which box the gold is in.

Similarly, reverse evaluation can be done to solve for the solution for this puzzle, a traversal of all the scenarios can be done. For example, if the gold is indeed in box #1, then that implies that the hint on box #2 that states the gold is not in box #2 is true, while the hint on box #3 that states the gold is in box #2 is false and since it is known that the hint on box #1 is also untrue, that means there are 2 lies and 1 truth, which correctly corresponds to the original statement of the problem, hence this is the correct solution.

Another instance would be to assume the other scenario that box #2 contains the gold, this would imply the hint on box #2 to be a lie but would imply the hint on box #1 and box #3 to be true. Hence since there are 2 truths and 1 lie, this cannot be the case, as explicitly said in the original problem statement.

Table 3.2 Truth table for gold in a box

Line	Box 1	Box 2	Box 3	Not in box 1	Not in box 2	In box 2
1	Gold	-	-	F	T	F
2	-	Gold	-	T	F	T
3	-	-	Gold	T	T	F

It is observed that the same results also hold true, where it could be seen that the only line where there are two lies and one truth only is true for the first line, where the gold is in box 1, hence it can be concluded that the solution of this puzzle is that the gold is located in the first box as it is the only logically consistent solution with the problem statement.

However, the problem gets alot more complicated if it were scaled up to have ten more or one hundred more boxes added, intuitively solving the problem would not be as easy and utilising truth tables would no longer be feasible considering the increase of boxes.

C. Utilizing Graphs to Solve the Puzzle

The boxes in the original problem will be represented as points, or in graph theory's terminology, nodes/vertices, it is known that the hint written on box #1 states that the gold is not in that specified box, this statement can be represented as having outgoing/outbound edges going into every other vertices other than box #1. The same also applies with the hint statement of box #2, it will also be represented with one outbound edge going to the vertex that represent box #1 and box #3. In the case of box #3, since it already specified the box in which the gold is in (box #2), an outgoing edge can be drawn going into that one specific box (#2).

Using this graph, the cases that were considered can once again be run through, if a box had a degree value of two or more inbound edges, then—because edges ingoing edges represent boxes that state the gold is in the specified vertex—this implies that (for a specified problem where the number of truths n are explicitly stated) if there are n+1 or more numbers of an inbound degree, that must mean the vertex in discussion cannot be the one containing the gold, as that would mean the all the inbound edges tell the truth, and since there are n+1 inbound edges, that must mean this isn't the correct solution.

Hence a short algorithm of sorts can be constructed, the abstraction is as follows,

1. Take a node of any label.
2. Count the numbers of inbound edges which have arrows pointing at that specified node (this gives the number of boxes telling the truth if and only if the specified node represents the box containing the gold).
3. Move on to the next node and repeat until the solution is found, or there are no nodes left.

This method will prove to be more efficient than simply going through each and every scenario and confirming the truth values of the other corresponding boxes, solving the puzzle diagrammatically by using graphs will give the algorithm a theoretical framework to work upon, and the use of that algorithm in and of itself will allow for great automation as the results of the algorithm (list of all the inbound degrees) can be put inside a list that is easily accessible and process-able, this will also ease the process when it comes to different prompt conditions, as fixed constants can be replaced with variables, allowing for a higher degree of modularity of the algorithm. One instance of this would be if the prompt suggested that not only one box had a true hint, but two, or three, and so on and so forth. Hence for any set of boxes and hints, a simple matching of the number of truths that are there in the hints and the number of inbound degree in the list could be done to find the correct solution.

IV. Results and Discussion

A. Conceptual Implementation

Consider the prior prompt of the 'Gold in a Box' logic puzzle, the following the methods described before, a graph representing the problem shall be constructed.

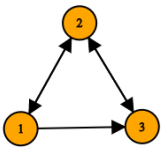

Fig. 4.1 Graph representation of the gold in a box puzzle

Following the methods stated in previous sections, a list of all the inbound degree is going to be made from isolating each node/vertex and then counting the in-degree. Note that the label '1', '2', and '3' represent the boxes #1, #2, and #3 respectively.

Hence, $d_{in}(1)$ would yield the degree value of 1 because of the edge (2,1), while $d_{in}(2)$ would yield the value of 2 because of edges (1,2) and (3,2), $d_{in}(3)$ would yield the value of 2 as well because of edges (1,3) and (2,3). Therefore, the correct solution would be that the gold is located in box #1 because it is the only logically consistent node with only one truth (an inbound degree of 1).

As touched upon previously, this diagrammatical method of solving the logic puzzle utilizing graph theory also allows us to replace the number of statements that are known to be the truth. For instance, if the original prompt of "it is known that there is only one truth from three of the hints on the boxes" were changed to "only two truths from three of the hints" then we would reference once again the list of d_{in} that has been made which would be $\{d_{in}(1), d_{in}(2), d_{in}(3)\}$ or {1, 2, 2}. Hence it can be concluded that for this particular case, the gold is either in box #2 or #3 because of a lack of information, but this hopefully will give an idea of the versatility of an algorithm of such.

B. Algorithmic Implementation

The algorithmic implementation of the method will be done in the programming language Python (version 3.8.2). As a means of explaining the source code of the program in a thorough manner, the program will be divided into its smaller abstractions so that it will be easier to get a grasp of the overall flow of the program.

```
truth = int(input("Insert known number of truths: "))
boxes = int(input("Insert known number of boxes: "))
```

Fig. 4.2 Initial input

The first part of the program is to request from the user an input of the fundamental information needed for the processes that will be done later on in the program. The values being referred to are the known number of truthful statements in the hint written on the boxes of the puzzle, as well as the known number of boxes in the puzzle itself, these values will be stored in the variable truth and boxes respectively.

```
adjacency = [[0 for i in range(boxes)] for j in range(boxes)]
print("")
for i in range(boxes):
  for j in range(boxes):
    adjacency[i][j] = int(input("Insert 1 if the hint on box #%d says
                                 that\nthe gold is in box #%d, otherwise
                                 insert 0:\n" % (i+1,j+1)))
```

Fig. 4.3 Adjacency matrix initialization

The next snippet of the code includes the declaration along with the initialization of an adjacency matrix (a square matrix of size that corresponds with the variable boxes), this method of graph representation has been discussed before in previous sections and is implemented here for an instance where a directed graph is the graph that is trying to be represented. The matrix-filling/input process also has been phrased in a way that allows the experience to be more intuitive to the users, having used descriptive language such as "if the hint on box N says that the gold is in box M" instead of directly prompting the user to determine if a directed edge exists between two specified vertices. This hopefully won't only be beneficial in terms of increasing ease of access, but also making the input process more efficient without having to first tabulate the puzzle into an adjacency table beforehand, even from an educated/academic standpoint.

```
print("\nThe resulting adjacency matrix:")
for i in range(boxes):
  for j in range(boxes):
    print(str(adjacency[i][j]),end=" ")
  print("")
```

Fig. 4.4 Printing the adjacency matrix

The current snippet of the code is overall very straightforward as its only purpose is to serve as a visualization tool for the users, as to print the matrix that has been made from the user's prior inputs.

```
degree = [0 for i in range(boxes)]
print("\nThe resulting list of each node's inbound degree:\n(",end="")
for i in range(boxes):
  for j in range(boxes):
    degree[i] += adjacency[j][i]
  print(degree[i],end="")
  if (i<boxes-1):
    print(",",end="")
print(")")
```

Fig. 4.5 Processing the adjacency matrix into a list of inbound degree

The next step of the program is to take the adjacency matrix and convert it into a list of all the inbound degree of each and every node in the graph. This is done by firstly declaring and initializing an array of a size that corresponds with the variable boxes.

A traversal through all of matrix's elements will be done and in short, a summation of each of the matrix's column's cumulative elements sum will be calculated and stored into a corresponding column/index of the inbound degree array. The final calculated array will then be printed by the program, as a

means of visualization to better understand the calculating process.

```
print("\nThe gold must be in either of the following boxes:")
for i in range(boxes):
  if (degree[i]==truth):
    print("Box #%d" % (i+1))
```

Fig. 4.6 Traversal search of the solution

After obtaining a complete array of the tally of inbound degrees for each of the boxes (in other words, a tally of all boxes that says the gold is in that specific box), a traversal search algorithm can be done, as to find a match between the number of truths that are known, and the value of the inbound degree. As previously stated, any inbound degree value that is not exactly equal to the number of truths known in the puzzle prompt cannot be the correct case/solution as it would mean there is a lie in the collection of statements.

Take note as well that the chosen algorithm is an all-element-spanning traversal search instead of using while-loop or a boolean-flag protection for the reason that if the reverse evaluation process that has been done yielded two or more possible boxes to contain the gold, then that would mean, without further information the actual box containing the gold cannot be determined, hence the traversal search, and if the two or more result were to be the case, the program would print out the corresponding amount of solutions that are deemed possible.

```
Insert known number of truths: 1
Insert known number of boxes: 3

Insert 1 if the hint on box #1 says that
the gold is in box #1, otherwise insert 0:
0
Insert 1 if the hint on box #1 says that
the gold is in box #2, otherwise insert 0:
1
Insert 1 if the hint on box #1 says that
the gold is in box #3, otherwise insert 0:
1
Insert 1 if the hint on box #2 says that
the gold is in box #1, otherwise insert 0:
1
Insert 1 if the hint on box #2 says that
the gold is in box #2, otherwise insert 0:
0
Insert 1 if the hint on box #2 says that
the gold is in box #3, otherwise insert 0:
1
Insert 1 if the hint on box #3 says that
the gold is in box #1, otherwise insert 0:
0
Insert 1 if the hint on box #3 says that
the gold is in box #2, otherwise insert 0:
1
Insert 1 if the hint on box #3 says that
the gold is in box #3, otherwise insert 0:
0

The resulting adjacency matrix:
0 1 1
1 0 1
0 1 0

The resulting list of each node's inbound degree:
(1,2,2)

The gold must be in either of the following boxes:
Box #1
```

Fig. 4.7 Output of the program

Hence, it can be seen that the output of the program that yielded the solution that box #1 contains the gold is indifferent from previous analysis and methods using intuitive reverse evaluation as well as by using truth tables.

V. Conclusion

To conclude the findings of this paper, the use of graph theory to diagrammatically solve logic puzzles can effectively do so in a manner that is clear, concise, and arguably more intuitive than simply going through all scenarios and trying to rule out logically inconsistent conditions only with thought-experiments.

It can also be concluded that the development and use of such an algorithm to have a more structured way of solving a problem can be simplified through the steps of proper computational thinking by the decomposition of larger problems into smaller issues, pattern recognizing as a means of ruling out patterns so that a solution can be made as general as possible with a high degree of modularity, abstraction to eliminate unimportant/less relevant subjects of the topic and finally to develop the algorithm itself.

The application of such an algorithm could be used in pedagogy as an introduction to logic, as the diagrammatical approach lends a good visual representation of the problem, especially with the nature of digraphs' clear directionality. In the field of computational/mathematical logic, the algorithm itself could be adapted to fit various other iterations of similarly constructed puzzles in a straightforward way. As for real world application, the author could see the potential of this algorithm to be used in a specific crime analysis case where amongst a number of suspects, the amount of truth-tellers can already be scoped out, therefore the algorithm can help finalize the deduction process, this is of course quite the farfetched proposition and scenario, but perhaps the author would like to end the academic literature in a more open-ended note as to incite inspiration and the inventiveness of the readers in regards to this subject.

VI. Acknowledgment

First and foremost, the author would like to express gratitude towards God for the opportunity to write this particular paper. The author would also like to thank all lecturers, professors, and aides of Institut Teknologi Bandung that have been involved in the pedagogical process of instilling the discipline of discrete mathematics to the author. The author would also like to specifically thank Dra. Harlili M.Sc. who is assigned to be the author's lecturer in the Discrete Mathematics course of code IF2120, as well Dr. Ir. Rinaldi Munir, MT. whose learning resources have been heavily used and contributed a very large amount towards the process of formulating and writing this paper. Among others, the author would also like to thank a handful of individuals from his group of friends who have helped him in proofreading parts of his paper. The author realizes that there are still many unrefined and imperfect parts of the paper and would like to apologize for any uncalled mistakes present in the literature.

REFERENCES

[1] D. B. West, Introduction to graph theory (Vol. 2). Upper Saddle River, New Jersey: Prentice Hall, 1996.
[2] G. Van Rossum, F. L. Drake, Introduction to Python 3: Python Documentation Manual Part 1. California: CreateSpace, 2009.
[3] J. L. Gross, J. Yellen, Graph theory and its applications. Florida: CRC Press, 2005.
[4] K. H. Rosen, K. Krithivasan, Discrete mathematics and its applications: with combinatorics and graph theory. Pennsylvania: Tata McGraw-Hill Education, 2012
[5] M. Genesereth, E. J. Kao, "Introduction to Logic," in Synthesis Lectures on Computer Science. Vermont: Morgan & Claypool Publishers, 2016.
[6] R. Munir, Matematika Diskrit (Revisi Ketujuh). Bandung: Informatika Bandung, 2020.
[7] S. Kapil, Clean Python. New York: Apress, 2019.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Jakarta, 5 Desember 2020

James Chandra - 13519078