Greedy, or not Greedy. That is the Question. Dream Girls Part 2. (21++)

Jones Napoleon Autumn - 13518086
Informatics Major
School of Electrical Engineering and Informatics
Bandung Institute of Technology, Jl. Ganesha 10 Bandung 40132, Indonesia
13518086@std.stei.itb.ac.id

***Abstract* — Any type of connection across individual is usually stagnant, including those with special romance. The urges to find new connections would generally exist in the mind of a living individual. An instance would also include a partner who would love to explore his or her potential partner(s). This paper shall cover the implementation of Graph theory and its traversing algorithm in discovering connection path from one person to another under seven connections – Six Degrees of Separation as the curtail, and hopefully, help any little greedy boy in fulfilling his desire to learn more about his possible connection combination for a good use.**

***Keywords* — Connection distance, Dream Girl(s), Breadth-first search, Six Degrees of Separation, Graph traversing.**

I. INTRODUCTION

Greedy algorithm has been notably one of the most popular algorithm in solving general problems in the field of Computer Science. Despite the paper's title, however, this paper will not discuss the Greedy algorithm and would instead, covers Graph traversing algorithm, specifically the renowned Breadth-first Search. Little disclaimer: this paper is the continuation of author's previous paper [1] titled "Getting Closer to Your Dream Girl through Six Degree of Separation with Graph Theory." which discusses graph theory in general and its capability in connecting people (and a guy to his dream girl, indeed).

In this paper, we shall take every features in the previous paper including its curtail of Six Degrees of Separation. Six Degrees of Separation has been a popular idea that indicates the connectedness of individuals. The idea that all people are six, or fewer, social connections away from each other, is often called as *6 Handshakes rule*. As a result, a chain of "a friend of a friend" statements can be made to connect any two people in a maximum of six steps.

The idea, again, was re-popularized by a Hollywood actor Kevin Bacon with the invention of the game "Six Degrees of Kevin Bacon". The concept is rather simple: link any celebrity to Kevin Bacon in under seven connections, and the jump of connection is being referred to as 'Bacon Number'. But rather than linking any celebrity to Kevin Bacon, we could actually link people from any side of the world to the other. We could even specifically link a guy from one end to a stranger girl on the other - previous paper's purpose.

In this paper, however, we shall allow the guy to be "greedy" in the sense of being able to have multiple dream girls at the same time (close to cheating). Onward, it would be advisable to assume having multiple dream girls as having multiple girlfriends at one moment. The idea was incited when the theory of Six Degrees is not necessarily applicable in the real world. With that, **the basis of this paper stands on the theory that people with less than seven connections have the capacity to befriend each other and those with more than six connections would never get to know one another.**

A classical approach in getting acquaintance to other people would be through connection from a mutual friend. This paper is specifically written with those way of acquaintance in mind. With the global population exceeding 7.7 billion people, there are numerous graphs with countless interconnection within individuals. With the Six Degree of Separation as the number of connection curtail, Graph theory is ready to help a guy to find his fullest potential in acquainting as many girls at once.

Note that the discussion in this paper is purely imaginational and educational. No cheating is done for this paper, and no cheating should be done because of this paper.

II. THEORETICAL BASIS

A. Graph

A graph is a pair $G = (V, E)$, where V is a set whose elements are called *vertices* (singular: *vertex*), and E is a set of two-sets of vertices, whose elements are called *edges*. The vertices x and y of an edge $\{x, y\}$ are called the *endpoints* of the edge and the vertices x and y being called *adjacent*, whereas the edge is said to be *incident* on x and y (represented as (x, y)) [2].

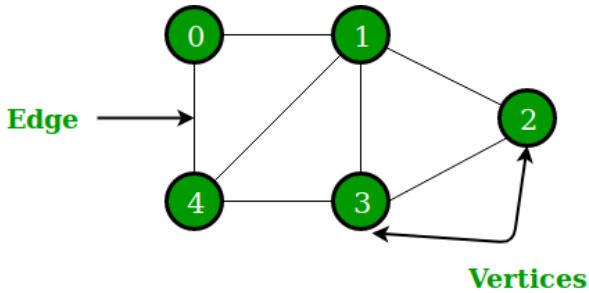

Figure 1. Example of graph. [6]

A vertex may not belong to any edge, called isolated vertex. If every vertex in a graph is an isolated vertex, the graph is called a *null graph*; a complete opposite of this graph is called a *complete graph* where each pair of vertices is joined by an edge, containing all possible edges in a single graph. Besides, one interesting concept of graph is *subgraph*. If there exist a couple of graph pair $G = (V, E)$ and $G1 = (V1, E1)$, where $V1 \subseteq V$ and $E1 \subseteq E$, then the graph $G1$ is said to be the subgraph of G.

In general, graph can be categorized by several of its properties. Based on the orientation of the edges, graph is divided into directed graph and undirected graph. Directed graph is a graph where all of its edges has direction (displayed with arrow); one of the endpoints acts as the source and the other as the destination. Undirected graph is one where all the edges are bidirectional, thus, should a vertex *u* connects to a vertex *v*, *v* must also connect to *u*. In computer science, directed graph is often utilized to represent conceptual graph, like finite state machines. Figure 1 also represents an unweighted and undirected graph.

Another property of a graph is weight. A weighted graph is one in which every edges has a value assigned, whereas an unweighted graph is one in which every edges has the same value (weight). The application of weighted graph becomes prominent in cases like *Minimum Spanning Tree* – an interesting topic out of this paper scope, while that of the unweighted is usually used in social network where connections only has 'True' or 'False' value.

Graph can be represented in several ways. Two of the most commonly used representations are Adjacency Matrix and Adjacency List.

Adjacency matrix defines the graph by using matrix where the number of rows and columns equal to the number of vertices and the possible value for every matrix element is only 1 (True) and 0 (False). An edge (u, v) exist in the graph if and only if the element $A_{u,v}$ has the value of 1. Otherwise, the value of element $A_{u,v}$ must be 0. In weighted graph, the value of element $A_{u,v}$ can be replaced as the weight value. A complete graph has an adjacency matrix full of 1 except its main diagonal and an undirected graph has an adjacency matrix where its main diagonal acts as a mirror.

Adjacency list defines the graph by a list with effective indices equals to the number of total vertices, with each index represents the corresponding vertex. Then, only if a vertex v is connected to vertex u will u be listed on v's adjacency list. If the graph is weighted, the adjacency list will be populated with tuple (u, w), where w is the weight.

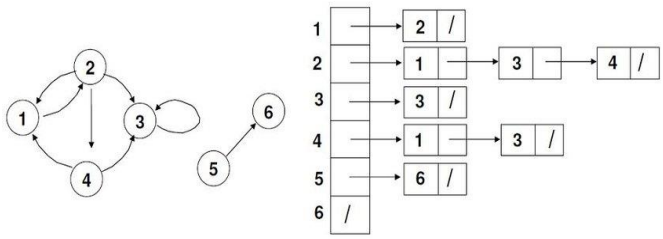

Figure 2. A directed graph represented by adjacency lists. [7]

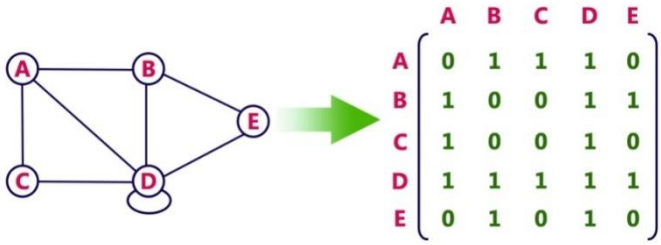

Figure 3. A graph represented by adjacency matrix. [8]

B. Graph-traversing Algorithm

Traversing a graph means visiting each of its node and perform the intended algorithm on that node. But unlike an array or list where it is apparent that the process of iteration starts from the first element to the last, a graph presents an unordered pattern where it is not obvious on which node(s) to visit first. Two algorithms that are often used to traverse a graph include Depth-first Search and Breadth-first Search.

In short, depth-first search (DFS) starts at the root (top node) of the graph and goes all the way down until meeting a connectionless node or a node whose all its edges have been traversed, then backtracks until it finds an unexplored path. Meanwhile, breadth-first search (BFS) starts at the graph's root and explores all of the neighbor nodes at the present depth, with its depth increasing as a process of iteration. A great representation of traversing process for DFS and BFS would be a stack and a queue respectively.

Since BFS increases its depth gradually, it is often used to find the shortest path. However, the drawback with BFS is it consumes memory and time if the solution is far from the root. Although this problem can be mitigated by using DFS as it requires a less time and space complexity, DFS is not often used as it might not confer the shortest path.

With the usage of BFS, there exist a couple of new terms: active node and expanded nodes. Active node is one doing the algorithm expansion, and the expanded nodes are those being expanded but not yet expand. For instance, while A is on the process of expansion, it becomes the active node, and B, C, and D become the expanded nodes. While C is being an active node, the expanded nodes would include E and F.

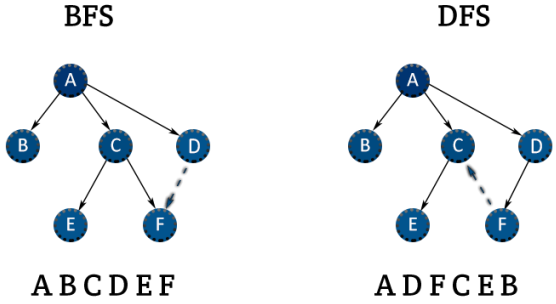

Figure 4. BFS vs DFS. [10]

Besides BFS and DFS, there are several other variations in graph traversing, such as Iterative Deepening Search that does DFS in a BFS fashion, and Depth-limited Search that curtail DFS depth; both of which are Uninformed search (leaps of edges equals represented value). There are also Informed Search (leaps of edges means nothing), like Uniform Cost Search where expanded nodes might be re-expanded again, and Branch and Bound that decides BFS neighboring nodes by a bound function.

C. Six Degree of Separation theory

Six Degrees of Separation is the idea that all people are six, or fewer, social connections away from each other, often called as *6 Handshakes rule*. As a result, a chain of "a friend of a friend" statements can be made to connect any two people in a maximum of six steps. It was originally coined by Frigyes Karinthy in 1929 and later popularized in an eponymous 1990

play written by John Guare. It is sometimes generalized to the average social distance being logarithmic in the size of the population [3].

The idea has been well supported by other early conceptions as well. Shrinking world; due to technological advances in all fields, including communication and travel, friendship networks has grown larger and has spanned a greater distance. Karinthy himself believed that the modern world was '*shrinking*' due to this ever-increasing connectedness of human beings. He posited that despite great physical distances between the globe's individuals, the growing density of human networks made the actual social distance far smaller [4].

With its popularity, several studies have specifically been conducted to measure the connectedness of global people empirically. Initially developed by Karl Bunyan, a Facebook platform application named "Six Degrees" was established to calculate the degrees of separation between people. After some redevelopment of the application, Facebook's data team released two papers in November 2011 which document that amongst all Facebook users at the time of research (721 million users with 69 billion friendship links), there is an average distance of 4.74. In 2016, the figure fell to 4.57, showing the interconnectedness among social people. Probabilistic algorithms were applied on statistical metadata to verify the accuracy of the measurements, and in return, 99.91% of Facebook users were found interconnected, forming a large connected component [5].

After the release of John Guare's play in 1990, the idea, again, was popularized by a Hollywood actor Kevin Bacon with the invention of the game "Six Degrees of Kevin Bacon". The concept is rather simple: link any celebrity to Kevin Bacon in under seven connections, and the jump of connection is being referred to as 'Bacon Number' [9].

III. APPLICATION OF GRAPH IN SIX DEGREES OF SEPARATION THEORY

Six degrees of separation represents at most six jumps or seven people (including the people to be linked) in connections leap. In graph representation, the people can be represented by vertices, while the connections can be represented by edges. The total of which for the former should not exceed 7, and that for the latter should not exceed 6. Therefore, the first requirement for a single *Six Degrees* connection is a subgraph *G1* (of a global graph *G*) with a maximum of 6 nodes and 7 vertices.

Since social connectedness (relations) could not be measured quantitatively (at least for this paper's purpose) - for instance connection in social media means a truly-connected (*True*) value, otherwise falsely-connected (*False*), the graph used in Six Degrees of Separation theory would be unweighted. In addition, a true connection would be referred to as *True* if and only if both parties know each other well, here it means that the right graph representation would be that of undirected.

With a vast connection potentially occur in the global graph (human population exceeding billions), the usage of Adjacency Matrix would be inefficient as it requires the square of those number (billions x billions). Thus, Adjacency List is of a better choice for an effective memory usage.

Since connection degree equals and directly proportionate to the graph's traversing depth, the suitable algorithm in this case would definitely be Breadth-first Search (BFS).

In summary, a better approach of Six Degrees of Separation with graph would harness an unweighted and undirected graph, with the people's relations being represented by Adjacency List, the subgraph not surpassing six degrees, and Breadth-first Search as the traversing algorithm.

IV. IMPLEMENTATION OF GRAPH IN SIX DEGREES OF SEPARATION THEORY

For a better dynamic Graphical User Interface (GUI) of the graph representation, author uses JavaScript language to demonstrate the implementation in Six Degrees of Separation.

First and foremost, create the object of Graph.

```
export class Graph {
    constructor(){ ···
    }
    getNumberOfVertices(){ ···
    }
    getAllVertices(){ ···
    }
    addVertex(v){ ···
    }
    addEdge(v1, v2){ ···
    }
    deleteVertex(v){ ···
    }
    deleteEdge(v1, v2){ ···
    }
```

```
    getVertexOrder(v){ ···
    }
    hasEdge(v){ ···
    }
    hasEdgeGotConnection(v1, v2){ ···
    }
    getEdge(v){ ···
    }
    getNumberOfEdges(v){ ···
    }
    getNumberOfConnections(){ ···
    }
```

Figure 5. Graph object with its constructor and some prominent methods.

The **constructor** set up an existing graph variable and set its node list to nothing. The **getNumberOfVertices** method returns the number of existing node in the graph, while **getAllVertices** returns its nodes. The following six methods (**addVertex**, **addEdge**, **deleteVertex**, **deleteEdge**) just do what it should, and the following two (**hasEdge**, **hasEdgeGotConnection**) respectively return a *Boolean* value of whether a vertex has edge(s) or connection and whether an edge exists between two nodes.

getVertexOrder returns the order of the vertex, **getEdge** returns all the edges (direct connection) a vertex (node) has, **getNumberOfEdges** returns its number of edges from a vertex, and **getNumberOfConnections** returns the total number of existing connections in the graph.

After finishing off the model, now look a little glance at the graphical interface

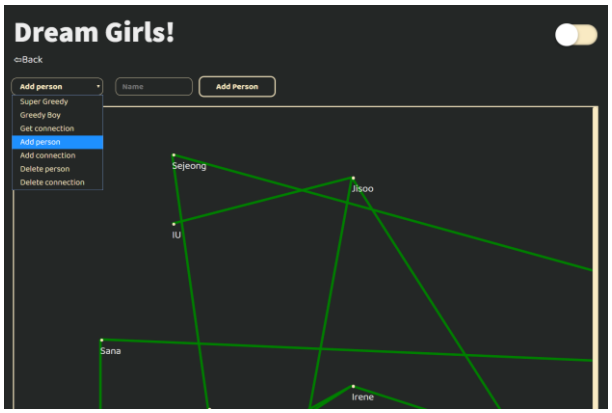

Figure 6. Dream Girls GUI with several features.

It is evident from the seven features, **Add Person**, **Add Connection**, **Delete Person**, and **Delete Connection** are used to serve the fundamental graph action. All four of which respectively trigger the **addVertex**, **addEdge**, **deleteVertex**, **deleteEdge** functions after some validity checking, such as checking whether the input name exists in the graph before adding or deleting a person, and checking if the input names are both obtainable from the graph as well as their current connection status before adding or removing their connection.

To simplify computation, author declares only fourteen *people* and set several connections across the graph in the database; despite knowing the case in real-life is not going to be this smooth. Also, note that any names used in this paper is only for academic purpose and do not represent any individual/entities in the real world.

For the sake of better User Interface, author implements little code in JavaScript to enable the visualization of the graph, full with each of its people's connections. The corresponding result is as follows in Figure 7. Note that this graph's instance will be used for the whole paper's discussions.

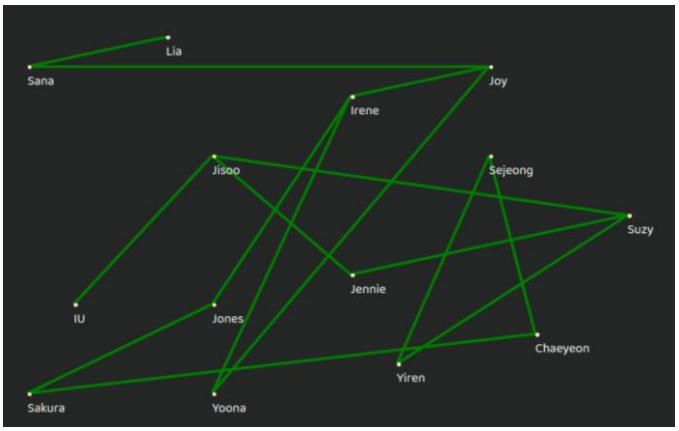

Figure 7. Graphical representation of the interconnection in the graph.

V. IMPLEMENTATION OF GRAPH TRAVERSING ALGORITHM

In my previous paper, we discuss about exploring and generating all the connection paths between two people, namely a small guy and his single dream girl, that stands under seven connections – abiding by the Six Degrees of Separation Theorem. Take for example the inputs of *Jones* and *Joy*, would generate the result *Jones -> Irene -> Joy*, *Jones -> Irene -> Yoona -> Joy*. Another example is those of *Jones* and *IU* would result in none, as they are separated by more than six degrees in this paper.

1. Get Connection

However, generating all possibilities of connection paths means nothing if there is no additional insight about it, moreover the algorithm cost would be heavy if the graph has traversed all the way six connections deep from the main root (*Jones*). Thus, to ease up computation, we would revamp the procedure of generating connection into only the smallest leap of connections. In the above *Jones* and *Joy* example, the answer would then be only *Jones -> Irene -> Joy*, as it takes the smallest number of connection jumps (2). In this case, the usage of Breadth-first search as the traversing algorithm would preside over that of Depth-first search.

Now, the algorithm of getting the shortest connection paths plays when *Get Connection* is clicked. There is no specific rule other than the generic BFS; only with additional maximum depth of 6 as its graph iteration curtail. Figure 8 and Figure 9 display the output of *Jones – Jisoo*, and *Jones – IU* respectively. Following that, Figure 10 shows the algorithm (graph's *computeConnection* method) for such results.

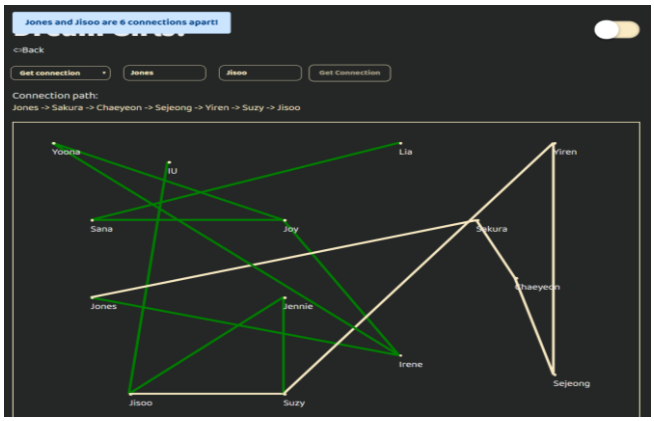

Figure 8. Graphical output of Getting Connection for *Jones* and *Jisoo*

Figure 9. Graphical output of Getting Connection for *Jones* and *IU*

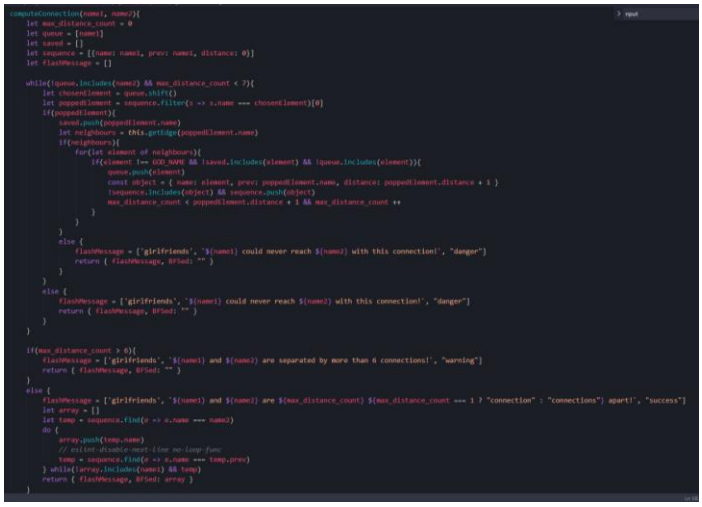
Figure 10. BFS Algorithm that returns the first node encounter.

The variable *max_distance_count* is used to keep up with the value of current traverse depth. Remember that the representation of BFS would resemble a queue, and the *queue* variable is used just for that purpose – it keeps track of all the expanded nodes in order, while the purpose of *saved* is to save all nodes that are or have been active. The *sequence* acts as a *memoization* that monitors each of the node as well as its corresponding data structure (*distance*: connection distance from the root node to *this* node, *prev*: the previous node that

generate/expand *this* node).

In essence, the algorithm traverses the graph until *max_distance_count* reaches 7 or the intended node was met. If the former scenario emerges, the function returns nothing and displays a flash with message "*{person1}* and *{person2}* are separated by more than 6 connections"; otherwise, it would rearrange the *memoization* to find the path from *person1* to *person2*, resulting in *Jones -> Sakura -> Chaeyeon -> Sejeong -> Yiren -> Suzy -> Jisoo* from Figure 8, and displays relevant flash message.

2. Greedy Boy

Nothing could possibly have gone wrong with generating a connection path, but no fun would come out of it as well. The fun part starts when *person1* is changed to *greedyboy*, *person2* to *girlfriend*, and the method to Greedy Boy – a method to find *greedyboy* his *alternative* or *extra* girlfriend(s) beside his current *girlfriend*. This method is possible because not everyone is interconnected within 6 jumps and as long as the *extra* is under seven connections with the *greedyboy* and over six connections with the boy's *girlfriend*, she is good to go (*extra*-girlfriend material).

Figure 11 perfectly displays what it looks like when *Jones* have *Suzy* as girlfriend, yet still be greedy and craving more. The output are four people, all with more than six connection leaps from *Suzy* and less than seven from *Jones*. Figure 12 depicts the case when all *Jones*'s under-seven possible connections are reachable through *Sakura*'s six connections distance as well. In other words, $J \subseteq S$, where J is all people within six connections from *Jones*, and S is those from *Sakura*. Figure 13 occurs when *Jones* (the *greedyboy*) and *IU* (the prospective *girlfriend*) are more than six connections apart, meaning *Jones* would never befriend *Sakura* all his life. Well, are you dreaming, boy?

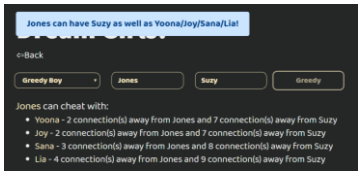

Figure 11. Text output of Greedy Boy method (*Jones - Suzy*)

Figure 12 (left). Text output of Greedy Boy method (*Jones - Sakura*)

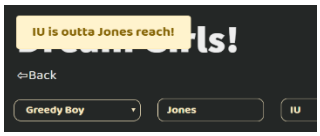

Figure 13 (right). Text output of Greedy Boy method (*Jones - IU*)

```
const handleGreedy = () => {
    if(graph.hasEdge(greedyName) && graph.hasEdge(mustHaveName)){
        const dangerousSequence = graph.getUnderSeven(mustHaveName)
        const dangerousPeopleForGreedy = dangerousSequence.map(s => s.name)
        const greedyPossibleObject = graph.getEveryoneExcept(dangerousPeopleForGreedy, greedyName)
        const { mustHaveAcquintance: greedyPossibleCheatingPartners, greedyExist } = greedyPossibleObject

        if(!greedyExist){
            setFlashInfo(['girlfriends', `${mustHaveName} is outta ${greedyName} reach!`, "warning"])
        }
        else {
            if(greedyPossibleCheatingPartners.length === 0){
                setFlashInfo(['girlfriends', `Impossible to cheat if ${greedyName} really wants ${mustHaveName}!`, "warning"])
            }
            else {
                let girlConnectionDistance = graph.getConnectionsDistance(mustHaveName, greedyPossibleCheatingPartners)
                let greedyConnectionDistance = graph.getConnectionsDistance(greedyName, greedyPossibleCheatingPartners)
                let deleted = []
                for(let i = 0; i < greedyConnectionDistance.length; i ++){
                    if(greedyConnectionDistance[i].distance > 6){
                        deleted.push(greedyConnectionDistance[i].name)
                    }
                }
                greedyConnectionDistance = greedyConnectionDistance.filter(girl => !deleted.includes(girl.name))
                girlConnectionDistance = girlConnectionDistance.filter(girl => !deleted.includes(girl.name))
```

Figure 14. High-abstraction program that settles all possible *extra* partners for *greedyboy*

The *mustHaveName* variable represents the (prospective) *girlfriend* that *greedyName* (*greedyboy*) has. Starting off, we first filter everyone that are still reachable from the *girlfriend* – thus, *getUnderSeven*-connection people from *mustHaveName*. Then, we continue to *getEveryoneExcept greedyName* and those resulted from the *mustHaveName's getUnderSeven*-connection, so the *potentialCheatingPartners* would remain unknown to his girlfriend, storing the potential extra partners to a local array named *greedyPossibleCheatingPartners*. If the array is not empty, then we would *getConnectionsDistance* - traversing each element and find its connection distance to both *greedyName* and *mustHaveName*. Finally, *greedyConnectionDistance* saves each of *greedyName*'s *potential* partners under seven connections jump, and *girlConnectionDistance* correspondingly saves each of those partner's connection distance value (must be above 6).

```
getConnectionsDistance(greedyName, greedyPossibleCheaters){
    let copyCheatingPartners = [...greedyPossibleCheaters]
    let queue = [greedyName]
    let saved = []
    let sequence = [{name: greedyName, prev: '', distance: 0}]

    while(copyCheatingPartners.length !== 0){
        let chosenElement = queue.shift()
        let poppedElement = sequence.filter(s => s.name === chosenElement)[0]
        if(poppedElement){
            saved.push(poppedElement.name)
            let neighbours = this.getEdge(poppedElement.name)
            if(neighbours){
                for(let element of neighbours){
                    if(element !== GOD_NAME && !saved.includes(element) && !queue.includes(element)){
                        queue.push(element)
                        const object = { name: element, prev: poppedElement.name, distance: poppedElement.distance + 1 }
                        !sequence.includes(object) && sequence.push(object)
                    }
                    if(copyCheatingPartners.includes(element)){
                        copyCheatingPartners.splice(copyCheatingPartners.indexOf(element), 1)
                    }
                }
            }
        }
        else {
            break
        }
    }
    const finalSequence = sequence.filter(s => greedyPossibleCheaters.includes(s.name))
    return finalSequence
}
```

Figure 15. Implementation of *getConnectionsDistance*

The implementation of *getConnectionsDistance* is similar to that of *computeConnection* at Figure 10, just that instead of iterating until connection distance hits seven, this algorithm iterates until everyone in the *greedyPossibleCheatingPartners* has had her connection distance computed.

3. Super Greedy

Above everything, though, this *Greedy Boy* method is only plausible when he (already) has a (prospect) girlfriend. But, what if that's not the case? *Super Greedy* is here to give a hand.

In short, what it does is take an input of a *greedyboy*'s name and generate all cheating combinations that he could have.

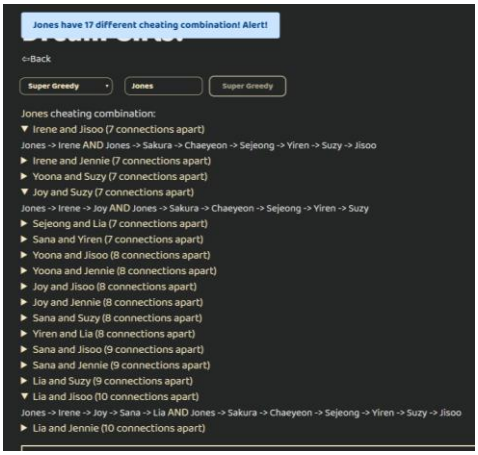

Figure 16. Text output of Super Greedy method (*Jones*)

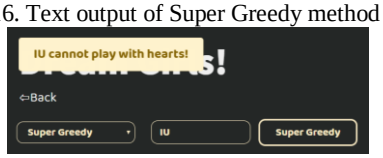

Figure 17. Text output of Super Greedy method (*IU*)

Figure 16 displays all cheating possibility, whereas Figure 17 showcase the final output of Super Greedy, and since IU has no possibility in cheating – this is apparent as she is considered as leaf in the node, meaning that she has only one direct connection, and thus no way for her to cheat.

The rule for this algorithm is also pretty straightforward; if two girls are under seven connections with the *greedyboy* with the two girls being over six connections, then two of them are one of the cheating combinations.

```
const handleSuperGreedy = () => {
    if(graph.hasEdge(greedyName)){
        const dangerousSequence = graph.getUnderSeven(greedyName)
        const operations = graph.operate(dangerousSequence)
```

Figure 18. Procedural short algorithm in handling Super Greedy

After *getUnderSeven*-connection from the *greedyboy* (*greedyName*), execute the *operate* method shown in Figure 19.

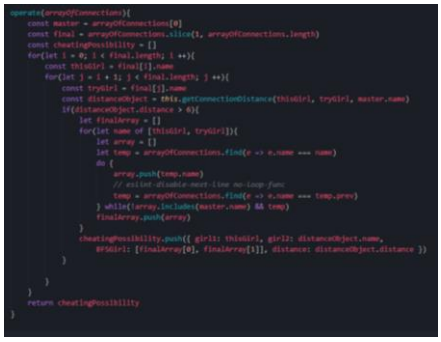

Figure 19. Algorithm behind *operate*

The *master* variable stores the *greedyName*, and *final* is the result of the *array*'s slicing without the *greedyName*. Then, we harness the Brute force technique for every couple combination in the *final* and appending their relevant data structure if they meet the condition of over six connection distance, resulting in all connection possibilities for the *greedyName*.

VI. CONCLUSION

With three new features of being greedy looking for partner(s) in a relationship (Get Connection, Greedy Boy, Super Greedy), author wishes people to learn through them from the good side as a prevention to cheat.

VII. APPENDIX

1. The playground of this paper can be accessed at https://jonesnapoleon.com/projects/dreamgirls.
2. The source code would not be publicly shared at any version control platform.
3. For a visual reference in Indonesian, feel free to access https://www.youtube.com/watch?v=9vQdc4L1GQc.
4. The instances and idea examples used are only for educational purposes and don't embody any individual/entities in the real world.

VIII. ACKNOWLEDGMENT

First and foremost, author would love to thank his family for supporting him in everyday choices that he takes. Author would also not forget to thank Bandung Institute of Technology as well as all of its lecturers, including those from the Informatics background for the opportunity to write this Computer Science paper.

IX. REFERENCES

[1] Jones Napoleon, "Getting Closer to Your Dream Girl through Six Degrees of Separation With Graph Theory", accessed 2nd May 2020, <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2019-2020/Makalah2019/13518086.pdf>

[2] Biggs, Norman (1993) Algebraic Graph Theory (2nd ed.). Cambridge University Press. ISBN 978-0-521-45897-9

[3] Guare, John (1990) Six Degrees of Separation. Vintage. ISBN 9780679734819

[4] Karinthy, Frigyes (1929) Everything is Different

[5] Ugander, J., Karrer, B., Backstrom, L., & Marlow, C. (2011). The anatomy of the Facebook social graph.

[6] Geeks for geeks, Undirected graph, accessed 4th December 2019, <https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/>

[7] Carla Osthoff, Adjacency List, accessed 4th December 2019, <https://www.researchgate.net/figure/A-directed-graph-represented-by-adjacency-lists_fig3_274143903>

[8] Oreilly, Adjacency Matrix, accessed 4th December 2019, <https://www.oreilly.com/library/view/php-7-data/9781786463890/6bddc759-aa6e-4f1a-a2b3-a5460b5de121.xhtml>

[9] "Actor's Hollywood career spawned 'Six Degrees of Kevin Bacon'". Telegraph. 6 June 2011. Retrieved 7 May 2012.

[10] Open4Tech, accessed 2nd May 2020. <https://open4tech.com/bfs-vs-dfs/>

DECLARATION

I hereby declare that this paper is of original work, neither an adaptation, nor a translation of any existing paper, and not an act of plagiarism.

Bandung, May 3th 2020

Jones Napoleon Autumn
13518086