Safety-prioritized Route Optimization of Post-Apocalyptic Moscow Metro's Tunnels Navigation in the Metro Series using Dijkstra's Algorithm

M. Aqsha Bagus R.I.B. - 13525060
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: aqshabgs13@gmail.com , 13525060@std.stei.itb.ac.id

***Abstract*—The Metro universe gives a glimpse of the danger-filled lives of the citizens of the Moscow Metro, filled with dangerous mutants, radiations, factions, and other hazardous conditions as consequences of a nuclear war. This paper provides a computational implementation of a pathfinding method used to search for the most optimal path with calculated safety weight considering distance, environment, and faction control. Using available map, the stations and tunnels are respectively modeled as vertices and edges. The pathfinding algorithm implemented in this paper is Dijkstra's Algorithm with an output of the path from a single chosen point to every other vertex mapped. The results show that the algorithm chosen is capable of choosing the path with the lowest safety weight risk to a specific point, with asymmetric results due to the bidirectional structure of the graph.**

Keywords—weighted graphs, Dijkstra, path optimization, video game, Metro game series, Moscow Metro, path safety

I. Introduction (*Heading 1*)

A fundamental problem in the field of informatics is determining the shortest path from one point to another using specific algorithms. Since its first well-known conception in the 1950s by computer scientist Edsger Dijkstra (which Dijkstra's Algorithm was named after), pathfinding algorithms has been an essential tool for efficiency determination across different sectors. For example, effective navigation systems using pathfinding algorithms are applied in GPS, video games, robotics, and logistics [1].

In real conditions, however, the shortest distance is not always the most feasible one. External factors such as real field conditions often have major effects on the actual implementation attempt. Because of this, mapping the shortest path without considering external factors could potentially cause more problems to occur, such as inhibiting efficiency by false identification or dangerous navigation that could harm lives.

The concept of field situation plotting and subsequent path optimization can be explored in many ways, but this paper will focus on the analysis of the navigation in the metro tunnels from the Metro franchise. The Metro series is a post-apocalyptic survival-action media in the form of games and novels set in the Moscow Metro system after a nuclear war has heavily irradiated the earth's surface including Russia, and as such, its citizens have repurposed the metro stations as their home [2]. Navigation is often dangerous due to mutants, radiation, roadblocks, and opponent factions.

Due to threats within the world of Metro, it is not feasible to plan navigation statically via the shortest path. Instead, traversing the metro stations and its tunnels requires proper preparation with field knowledge to map safe spots and danger zones. Using available data provided by the authors of the Metro series, it is possible to model the shown stations/bases and tunnels as vertices and edges respectively using the weighted graph representation of the graph theory. The danger weight (values) can be calculated from the various environmental factors in the metro tunnels. Subsequently, the most optimized route, defined as the safest route, can be retrieved using a pathfinding algorithm such as Dijkstra's Algorithm. Due to the linearity of the game, it is not possible to traverse to every single point from another point, but this paper will provide insights to the topic of save navigation in the metro tunnels.

This paper is divided into five parts: introduction, theoretical framework, methodology, results and discussion, and conclusion. The theoretical framework includes relevant frameworks to solve the path optimization problem such as graph theory and Dijkstra's Algorithm. The methodology explains the graph model representation of the metro system and Dijkstra's Algorithm calculation and implementation. Results and discussion consist of code testing (input/output) and an analysis on the results.

II. Theoretical Framework

A. Graph Theory

1) Graph (Vertices and Edges)

From a data structure perspective, a graph can be defined as a tuple G containing a set of vertices and edges [3]. A vertex v is an element within a graph which may or may not be connected to another vertex. Vertices (plural form of vertex) that are connected to each other or itself are connected by edges. Each edge is represented as a tuple consisting of two corresponding vertices. For example, edge $e = (v_m, v_n)$ is an edge connecting vertices v_m and v_n. Graphs can be mathematically represented as $G = (V, E)$ where $V = \{v_1, v_2, \dots, v_m\}$ and $E =$

$\{e_1, e_2, \ldots, e_n\}$. A few examples of graphs are illustrated in Fig. 1 as shown.

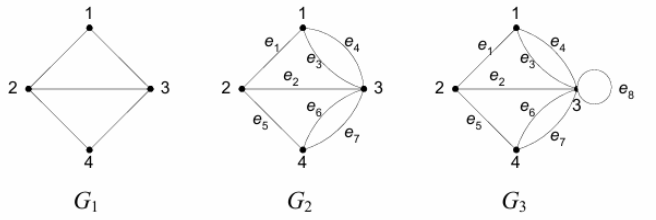

Fig. 1. Example showing three graphs
Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf, accessed on June 16th 2026

2) Weighted Graph

Weighted graphs are graphs with values $w(u, v)$, also known as weights or costs, for each edge $(u, v) \in E$. The weights of a weighted graph are highly dependent on the context of the problem. For example, the weights may measure the distance between one point to another, and in a different case, it may represent time needed between two activities. The figure below shows an image of a weighted graph.

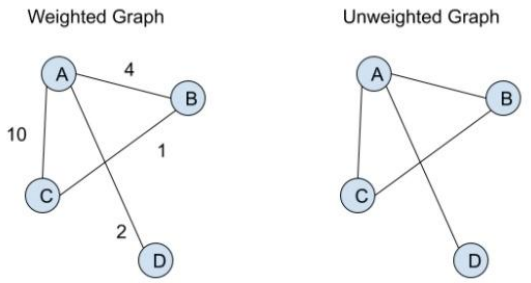

Fig. 2. Comparison between a weighted graph and an unweighted graph
Source: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf, accessed on June 16th 2026

Cost minimization in weighted graphs is essential in providing efficiency between a network of points. Cost-efficient linking can be achieved with pathfinding algorithms, such as Dijkstra's and A*. This paper will explore the topic using Dijkstra's Algorithm.

3) Graph Representations

A graph $G = (V, E)$ can be represented differently. Two common examples are using adjacency lists and adjacency matrices [11]. The choice of use between these two methods should depend on the structure of the graph.

The adjacency-list representation of graph G refers to an array of $|V|$ elements. Each element represents the adjacent connectivity for each vertex u where $u \in V$. The aforementioned adjacent connectivity are arrays containing all vertices v where $(u, v) \in E$. For weighted graphs, the weight $w(u, v)$ can be stored in the connectivity array of u and v. Because of its structure, the adjacency-list representation only needs to store into the memory as much as $\Theta(|V| + |E|)$. A disadvantage to consider when using this representation is the slower time to search the existence of an edge between two vertices.

The adjacency-matrix representation offsets the problems caused by the inefficiency in edge-searching. However, because of the inherent structure of matrices, a graph using this representation uses a memory of $\Theta(|V|^2)$. Let adj be an adjacency matrix, $0 \leq i, j \leq |V|$. An element $adj[i][j]$'s represents the linking between the vertex referred at row i and the vertex referred at column j. An unweighted graph has element values of 1s and 0s, where 1 means both vertices are linked and 0 means both are not linked. For weighted graphs, the values are simply the weight between the vertices.

B. Dijkstra's Algorithm

Dijkstra's Algorithm is a pathfinding algorithm designed to solve single-source shortest-path problems on weighted graphs with non-negative values [4]. Such solutions can be achieved on both directed or undirected graphs, which in the latter's case allows bidirectional navigation. The algorithm assumes that there is a path from source vertex a to every other vertex [5].

The main idea of Dijkstra's Algorithm is searching the shortest path from vertex a to vertex z while also minimizing the distance of in-between vertices. As such, we can not only find the minimum distance from vertex a to z, but also from the source to each vertex. Dijkstra's approach uses a greedy algorithm that inserts vertex w into a set of vertices S in which the distance from the source vertex a to vertex w is minimum while only navigating through vertices already in the set. The step-by-step operation of Dijkstra's Algorithm from vertex a is as follows:

1. Label source vertex a with 0 and the other vertices with ∞. The notation for labeling can be written as $L_k(v)$ where k is the current iteration and v is a vertex. In this case, we can construct the first two labels as $L_0(a) = 0,\ L_0(v) = \infty$ where $v \in V,\ V \in G(V, E)$, and $v \neq a$.
2. Construct an empty set $S_0 = \emptyset$. The set grows for each iteration k, where S_k is derived from S_{k-1} by inserting a new vertex u with the minimum label. The first inserted vertex will be a.
3. Update the label of each vertex v not in S_k so that the label of the vertex v at the k-th iteration, $L_{k(v)}$ is of the minimum cost between a and v as long as the path contains only vertices in S_k. The shortest path between a and v is either the shortest path from S_{k-1} (not including u) or the shortest path from a to u at the $(k-1)$-th iteration added by edge (u, v).
4. Iterate the procedure by successively adding non-member vertices into the S_k until the last vertex is added. Since the last node has no connected non-member vertices, we can safely insert it into the set.

For each iteration, labeling can be formulated as:

$$L_k(a, v) = min\{L_{k-1}(a, v), L_{k-1}(a, u) + w(u, v)\} \quad (1)$$

Provided below is the pseudocode for Dijkstra's Algorithm from references [4] and [6]:

procedure Dijkstra(input G: weighted graph with non-negative weight, input a: source vertex, output L: array [1..N] of integer)

{Initial State: $G(V, E)$ contains $|V| = N$ vertices where $a = v1, v2, \ldots, vn = z$. G also has lengths $w(v_i, v_j)$ where $w(v_i, v_j) = \infty$ if (v_i, v_j) is not an edge in G}
{Final State: array $L[1..N]$ where L_i is the shortest distance between a and v_i.}
Local Variables
i: integer
u, v: vertex
S: set of vertex
Algorithm
for $i := 1$ **to** N **do**
$L(v_i) := \infty$
endfor
$L(a) := 0$
$S := \infty$
for $k := 1$ **to** N **do**
$u :=$ a vertex not in S with $L(u)$ minimal
$S := S \cup \{u\}$
for all vertices v not in S **do**
if $L(u) + w(u, v) < L(v)$ **then**
$L(v) := L(u) + w(u, v)$
endif
endfor
endfor

III. METHOD

A. Map of the Metro Stations

Provided below is a modified map of the metro networks in the Metro universe with stations outside of the circle line removed [7].

Fig. 3. Metro systems map from Metro 2035
Source: https://metrovideogame.fandom.com/wiki/Locations?file=Map_Metro_2035_EN.png, accessed on June 16th 2026, modified by author

For simplicity, this paper will use a subgraph consisting of the stations of VDNKh, connecting into the stations inside the circular tunnels (known as Koltsevaya Line). This route is chosen due to the branching complexity compared to routes outside of the Koltsevaya Line which are relatively linear. The stations of VDNKh (we will address it as Exhibition in the code), Alekseyevskaya, and Rizhskaya are chosen due to its importance in the Metro series as the starting position of the main protagonist. Additionally, some modifications are made to the included stations:

1. Arbatskaya, Borovitskaya, Biblioteka im. Lenina, and Aleksandrovskiy Sad are combined into the Polis Station area.
2. Stations with two sections are treated as one vertex, the exception being Park Kultury due to the sections being closed off of each other.
3. Some stations are renamed following the conventions in metroguides.info and onlinecompass.net

The site metroguides.info is used to validate the connectivity of the stations by comparing the stations in Metro's map with the real-life map [8].

B. Weight Calculation

Dynamic weight calculations need to factor into account actual field conditions aside from the distance between two vertices. A mathematical model to calculate weight between two stations is formulated below:

$$w(u, v) = d(u, v) + H(u, v) + F(v) \quad (2)$$

The distance between two stations is defined as $d(u, v)$, while $H(u, v)$ represents the penalty score given by hazardous conditions in the area. Distance from each connected metro system is procured from straight-line measurements from the site onlinecompass.net using a conversion of 1 km to 10 points [9]. $H(u, v)$ is the hazard constant between the two stations, while $F(v)$ is the penalty score gained depending on the faction occupying the targeted station. The table below shows the type of hazards as well as information regarding the hazards.

TABLE I. HAZARD INFORMATION AND CONSTANTS

Type of Hazard	Constant
Radiation	50
Biohazard (Mutants or other dangerous non-human creatures)	75
Mental (Ghosts and hallucinations)	35
Tunnel collapse	50
Various hazards	100

This paper will assume the identity of the current path navigator as an independent traveler with no diplomatic affiliation, using standard metro equipment such as weaponry & ammunition, gas mask &` filters, and medical equipment. This assumption is made to simulate variation and prevent biases such as infinitely high values due to lack of equipment, or faction friendliness/hostility. The table below shows the faction factor constants

TABLE II. FACTION INFORMATION, CONSTANTS, AND DESCRIPTION

Faction	Constants	Description
VDNKh Commonwealth	0	Free-ride between occupied stations for travelers with a passport
Hanza	25	While its outposts are a trader's haven, Hanza is often suspicious of outsiders. If allowed entry, weapons are confiscated.
Red Line	75	Violent expansion, state of conflict with the Fourth Reich
Fourth Reich	75	Violent expansion, state of conflict with the Red Line, totalitarian eugenics dictatorship
Polis	0	High security but friendly
Bandits	65	Criminal activities with active hostility
Gangs	35	Criminal activities and may pose a threat
Arbat Confederation	0	Moderate, diversity respect. Severely weakened after the Hanza-Red Line peace treaty. Over all a peaceful station, though highly militarized.
1905 Confederation	0	Not much information regarding this faction except their population consists of free workers against tyrannical governments
Mutants	75	Most mutants found in the metro are hostile

C. *Station Graph Modeling*

The Python programming language will be used to create the graph form of the metro system and calculate optimized routes using Dijkstra's Algorithm. The adjacency list representation is used to model the graph data structure. The representation is justified by the topological structure of the metro stations. The metro network used in this paper inherently form a sparse structure due to the vertices having at least one connection and at most four connections (also known as a "sparse" graph).

A nested dictionary is used, consisting of a station name key with a corresponding value of a dictionary of station names and the weight.

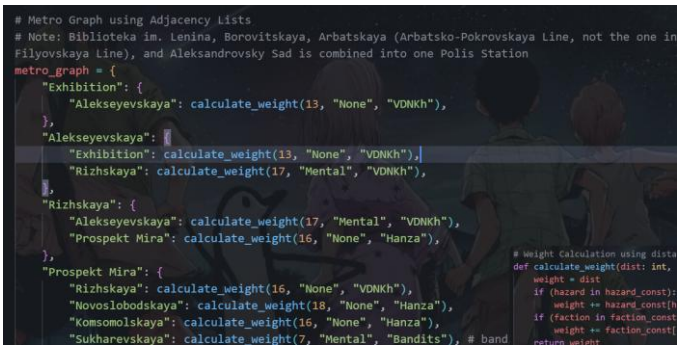

Fig. 4. Code snippet of the metro graph construction using adjacency list
Source: Author's documentation

D. *Program*

The program consists of three core functions/procedures. The first is a calculate_weight function which accepts an input of distance, hazard constant, and faction constant. The function sums up the arguments and returns the sum value.

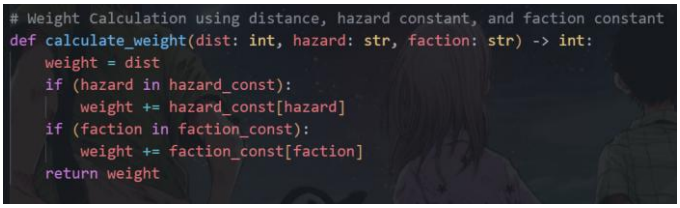

Fig. 5. The calculate_weight function
Source: Author's documentation

The second is a Dijkstra function which accepts a dictionary (in this case, an adjacency list), and a string which represents the starting point. The Dijkstra procedure runs Dijkstra's Algorithm and returns a nested dictionary "Info" consisting of a station name key and a value of a dictionary of "value" (total weight from start point until the specified station) and "prev" (previous station before the current station)

Figure 6. The Dijkstra function
Source: Author's documentation

The third is the Trace_Dijkstra procedure which accepts an "Info" dictionary and a start point string. The procedure traces the route and weight from a start point to every other vertex, and outputs the result.

```
def Trace_Dijkstra(Info: dict, a: str):
    for key in Info.keys():
        if (key == a):
            continue
        trace_list = []
        trace_list.append(key)

        cur_prev = Info[key]["prev"]
        while (cur_prev != None):
            trace_list.insert(0, cur_prev)
            cur_prev = Info[cur_prev]["prev"]
                                                    (variable) trace_list: list
        print(f"From {a} to {key} -> {' - '.join(node for node in trace_list)}")
```

Figure 7. The Trace_Dijkstra procedure
Source: Author's documentation

Additionally, the graph is visualized using the library NetworkX and Matplotlib is used. The visualization method uses a DiGraph model with weights and labels for both nodes (vertices) and edges (two bidirectional values) using references from NetworkX's official documentation [10]. This step is done to confirm the validity of the construction of the graph in the initial step.

IV. Results and Discussion

A. Results

The image below shows the visualization of the graph.

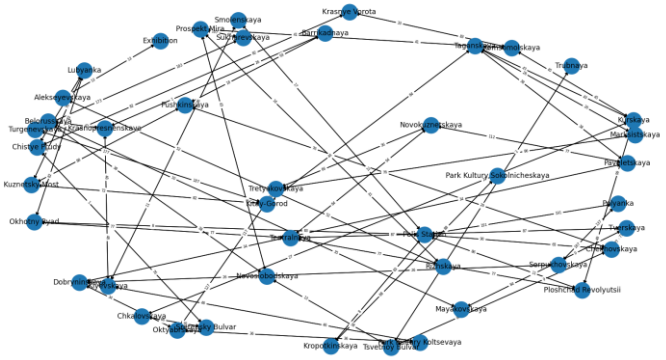

Fig. 8. Graph Visualization
Source: Author's documentation

By comparing the metro graph visualized in the program to the map used, the validity of the construction of the graph is confirmed. It is also possible to validate paths using figure 8.

The main program outputs a path and total weight from a starting vertex to every other vertex as seen in figure 9 with the starting point of "Exhibition".

Fig. 9. A snippet of a tracing output from "Exhibition"
Source: Author's documentation

A sample route from "Exhibition" to "Polis Station" shows the path taken: Exhibition, Alekseyevskaya, Rizhskaya, Prospekt Mira, Komsomolskaya, Kurskaya, Ploshchad Revolyutsii, and Polis Station. A total of 8 vertices, including the start point, are passed through, accumulating a weight of 302 units.

Another example from the start point of "Polis Station" is as shown below.

Fig. 10. A snippet of a tracing output from "Polis Station"
Source: Author's documentation

A sample route from "Polis Station" to "Exhibition" in figure 9 shows that the route is asymmetric compared to the route shown in figure 8. The path taken consists of Polis Station, Chekhovskaya, Tsvetnoy Bulvar, Novoslobodskaya, Prospekt Mira, Rizhskaya, Alekseyevskaya, and Exhibition.

B. Discussion

The sample route from "Exhibition" to "Polis Station" provides sufficient information regarding path-taking with safety-based weights. Initially, four vertices, including the start vertex, are compulsorily chosen due to the linear design of the first few stations. The next four stations have relatively low penalty scores with the exception of Ploshchad Revolyutsii's Red Line occupation, giving a grand total from the start point of 302 units. Comparatively, an alternate path with the same amounts of stations as the optimized path, branching in Prospekt Mira through Novoslobodskaya, Tsvetnoy Bulvar, Chekhovskaya, and Polis Station gives a higher weight of 311 units because of the threats of the Fourth Reich and the tunnel hazard. Interestingly, the optimized path has 117 distance points, more than the alternate path's 101 points. This insight signifies the importance of the safety factor in the path-making process.

Due to the faction constant factor and the difference in factions of some neighboring stations, the sample route from "Polis Station" to "Exhibition" has significantly lower weight than the opposite way around, scoring 261 units compared to the opposite's 302 units. This insight signifies the asymmetric navigation caused by the difference in edge weights in directed weighted graphs.

Deducing the asymptotic time complexity of the Dijkstra's Algorithm implementation in the Python program is possible by analyzing the core structure of the function. The outer loop (line 263) of the function loops N times, where N is the number of vertices $|V|$ in graph G. The inner loops (line 267 and 276) also

run N times each, both iterating through a dictionary of N keys due to either being directly the graph G itself or derived from graph G, which is the "Info" dictionary. The two inner loops can be considered a $|V|$ complexity because of how asymptotic time is calculated. It can be concluded from the analysis that the asymptotic time complexity of the program used is $O(|V|^2)$ [11].

For graphs with small amounts of vertices, for example the current metro graph with 43 vertices, the time complexity is negligible, but as the graph becomes more complex (more vertices and edges), an alternative and more efficient algorithm may be needed. A common paradigm used when implementing Dijkstra's Algorithm is using the priority queue data structure [12]. One such method is the min-priority queue with binary min-heap, which yields a total running time of $O\big((|V| + |E|)\log|V|\big)$. Another method is using the Fibonacci heap, giving a lower time complexity of $O(|V|\log|V| + |E|)$. Alternative solutions to the current program's architecture will not be discussed in this paper.

V. Conclusion

The implementation of Dijkstra's Algorithm in designing an optimal route based on safety parameters in the underground tunnels of the Metro universe has proven to be effective. The pathfinding method does not purely rely on the physical distance between each station, but also the dynamic and asymmetric factors of faction control and road hazard. While not currently applicable in the released games, the insight gained is significant nonetheless for navigation systems. A pathfinding algorithm which includes a robust safety-first paradigm is essential for preventing loss or harm from environmental hazard.

VI. Suggestions

For future work within the same topic, there are a few things that can be considered:

1. Making the program more dynamic using a selectable starting faction affiliation which would transform the weight of the graphs
2. Using the A* algorithm for a more heuristic approach of pathfinding.
3. Intensify research of the Metro world as well as information of specific areas.
4. Increasing the scope of the metro stations to stations outside of the Koltsevaya Line.

Video Link at Youtube

The program demonstration can be viewed through the link: https://youtu.be/HBLSrgBHlbE

Code Repository

The source code of the program can be accessed through the link: https://github.com/shamandeer/Dijkstra-Metro-Universe

Acknowledgment

First and foremost, I would like to give my praises and gratitude towards God for the grace given so that I could successfully finish the paper titled "Safety-prioritized Route Optimization of Post-Apocalyptic Moscow Metro's Tunnels Navigation in the Metro Series using Dijkstra's Algorithm" on time. I would also like to give my utmost gratitude to my lecturer, Prof. Dr. Ir. Rinaldi, M.T., for the guidance, knowledge, and references given for Discrete Mathematics within the 2nd semester. Finally, I would like to give my thanks to my colleagues for the support and guidance in completing this paper as well as my family for supporting me during the time of writing this paper.

References

[1] Z.A. Algfoor, M.S. Sunar, and H. Kolivand, "A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games", International Journal of Computer Games Technology, vol. 2015, pp. 1, 2015, http://dx.doi.org/10.1155/2015/736138.

[2] "Metro Series", Metro Video Games Wiki, [Online]. Available: https://metrovideogame.fandom.com/wiki/Metro_Series. [Accessed 19th June 2026]

[3] Rinaldi, "Graf (Bag 1)", [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf, pp. 7 & 55. [Accessed 15th June 2026]

[4] K.H. Rosen, Discrete Mathematics and Its Applications, 7th ed., New York: McGraw-Hill, 2012, pp. 710-712

[5] W. McQuain, "Weighted Graphs", [Online]. Available: https://courses.cs.vt.edu/~cs3114/Fall10/Notes/T22.WeightedGraphs.pdf , pp. 1-3. [Accessed 15th June 2026]

[6] Rinaldi, "Algoritma Greedy (Bagian 2)", [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/05-Algoritma-Greedy-(2026)-Bag2.pdf, pp. 20. [Accessed 15th June 2026]

[7] "Locations", Metro Video Games Wiki, [Online]. Available: https://metrovideogame.fandom.com/wiki/Locations, [Accessed 15th June 2026)

[8] MetroGuides, "Moscow Metro Station Guide and Route Map", [Online]. Available: https://metroguides.info/city/moscow#scheme/0/0. [Accessed: 17th June 2026]

[9] OnlineCompass, "Distance Calculator and Map Measurement Tool", [Online]. Available: https://www.onlinecompass.com. [Accessed: 17th June 2026]

[10] NetworkX, "Weighted Graph", [Online]. Available: https://networkx.org/documentation/stable/auto_examples/drawing/plot_weighted_graph.html. [Accessed: 18th June 2026]

[11] Rinaldi, "Kompleksitas Algoritma (Bag 2)", [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/26-Kompleksitas-Algoritma-Bagian2-2026.pdf, pp. 28, [Accessed 19th June 2026]

[12] T.H. Cormen et al., Introduction to Algorithms, 3rd ed., Cambridge: The MIT Press, 2009, pp. 662

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.

Bandung, 19 Juni 2026

M. Aqsha Bagus R.I.B. - 13525060