Utilization of Dijkstra's Algorithm in Finding Best Route for Medical Check-Up with Water Ambulance

Christopher Brian - 13522106[1]
Undergraduate Program in Informatics
School of Electrical Engineering and Informatics
Bandung Institute of Technology, Jalan Ganesha 10 Bandung 40132, Indonesia
[1]13522106@itb.ac.id

***Abstract*—Enhancing response time and minimizing fuel consumption are critical in the use of water ambulances. Fortunately, optimization efforts are both possible and well-established by the field of graph theory. Route optimization has long been one of the main focuses in this field of discrete mathematics. Weighted graphs are more frequently used as they provide more information compared to unweighted ones. By utilizing proven algorithms in solving the problem, this paper introduces a scenario-based approach to route planning in maritime environments. By integrating scenario-specific factors, such as optimal response time as opposed to optimal fuel consumption, this paper aims to provide an adaptive solution to this critical concern.**

***Keywords*—discrete mathematics, graph, route optimization, water ambulance.**

I. Introduction

Water ambulance is a vehicle used for emergency medical care in island communities. The use of water ambulances allows islander populations to receive quick and adequate health services. Water ambulances also largely contribute in introducing modern medicine and providing healthcare access to isolated and semi-isolated communities living beyond land with established medical infrastructures.

On the other hand, the fuel versus time dilemma have long been a debate in the use of transportations. Moving from one place to another using any conventional transportation will require more fuel consumption than average. On the other side, maximum fuel consumption optimization means using a lower than maximum speed. While several small breakthroughs have been made in order to shrink the gap between fast travel time and optimal fuel consumption, the dilemma will always exist since they are natural opposites. Maximum performance will always require maximum effort, which in turn require often maximum fuel consumptions.

However, there is also another side to the argument. Maximum time efficiency, at least roughly speaking, can not only be reached by maximum vehicle performance, but also shortest path possible taken. This will also correlate with lower fuel consumption, since optimizing the route taken also means optimizing fuel consumption. Finding the shortest path has long been a popular problem in discrete mathematics, especially the field of graph theory.

In order to study graph theory, it would be wise to learn its history first. Graph theory was in fact created to solve problems related to route planning and route optimization. A paper written by Leonhard Euler in 1736, popularly regarded as the first paper in the history of graph theory, was in fact an analysis on routes between places [2]. Modern graph theory represents places with nodes/vertices and routes with edges. A graph is defined by its vertices and edges.

In order to convey more information than just connection between nodes, weighted graphs are used as opposed to unweighted graphs. A weighted graph is a graph whose vertices have values relating to their property. A route-finding algorithm could then be used on this weighted graph in order to find the best possible route according to everyday scenarios or prioritized factors.

II. THEORY

A. Graph Theory

In discrete mathematics, graph theory is the study of graphs, mathematical structures made up of vertices and edges used to represent relations between objects. In formal definition, a graph is defined as a tuple G = (V, E), where V stands for the set of vertices/nodes, and E stands for the set of edges which connect the vertices/nodes. Graphs are used to model discrete objects and how those objects relate mathematically.

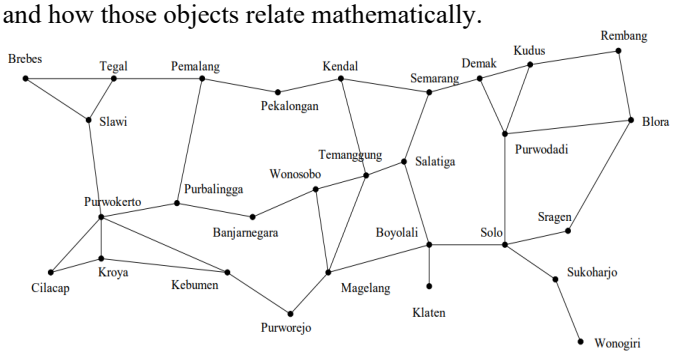


Fig. 1. Graph representing network of roads between several cities in Central Java. Adapted from [5]

A loop is and edge that connects a vertex to itself, while multiple edges are two or more edges that connect the same two vertices. Graphs that do not allow the presence of neither loops nor multiple edges are called simple graphs. On the other hand, graphs that allow the presence of loops and multiple edges are

called non-simple graphs. Non-simple graphs that contain multiple edges are called multigraphs, while those that contain loops are called pseudographs.

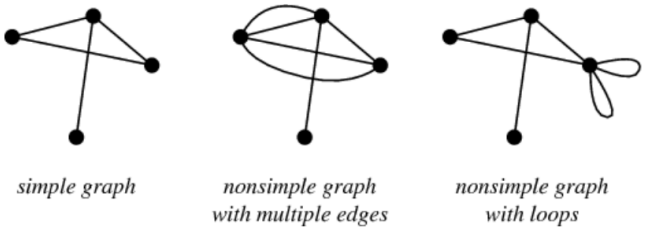


Fig. 2. Simple graph, non-simple graph with multiple edges (multigraph), and non-simple graph with loops (pseudograph). Adapted from [4]

Based on the orientation properties of the edges, graphs are differentiated into two types. Directed graphs are graphs in which edges have orientations. On the other hand, graphs where all the edges are bidirectional are called undirected graphs.

There are also unique kinds of graphs, one of which being the weighted graph. A weighted graph is a graph where each edge is assigned a number or value. These numbers are used to represent many properties, including but not limited to cost and distance.

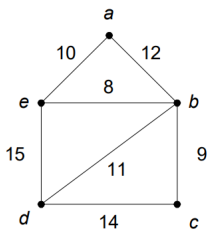


Fig. 3. Example of a weighted graph. Adapted from [5]

While representing graphs as normal drawings of edges and vertices, this method will get increasingly hard as the number of edges and/or vertices increase. This problem however can be easily managed by using more efficient graph representation methods. One such example, perhaps the most popular one, is the adjacency matrix. An adjacency matrix is a square matrix used to represent a finite graph. The elements of the matrix correspond to the presence of edges between vertices used as the row index and column index of the element. For undirected graphs, the number 1 is used to indicate the presence of an edge between two vertices, while the number 0 is used to indicate that those two vertices are not connected by any edge.

In the case of directed graphs, an adjacency matrix can still be used as a representation tool, albeit with some modifications. Instead of only two numbers indicating the presence or absence of an edge between any given two vertices, the elements in the adjacency matrix now represent the number of edges directed from the row index vertex into the column index vertex. To count the total of inward edges of a vertex, all one must do is to calculate the sum of all elements in the same column of the column index representing the vertex. On the other hand, to count the total of outward edges of a vertex, calculate the sum of elements in the same row of the row index representing the vertex.

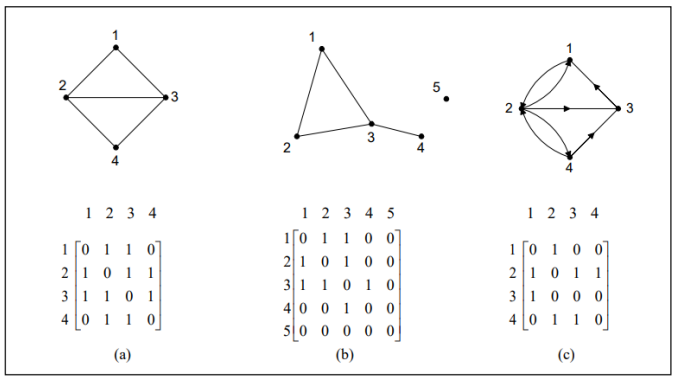


Fig. 3. Three different graphs each represented by its adjacency matrix. Adapted from [6]

B. Dijkstra's Algorithm

Dijkstra's algorithm is an algorithm used for finding shortest path from any node into any other node in a weighted graph. It is classified as a greedy algorithm, meaning it will always select the node with the smallest distance at each step. The algorithm will create two sets, one for visited nodes, and one for unvisited nodes. The user will then be required to pick a source and destination node at the start. For the full steps of the algorithm, see below.

1. Prepare a weighted graph where the weight represents distance between the two nodes it connects.
2. Pick a source node and use it as the beginning vertex, pick a destination node.
3. The algorithm will then conduct a search from one vertex to another until it reaches the destination node.
4. In the beginning vertex, set the initial value as 0, and set weights between nodes not connected by any edges as infinity.
5. Still in the beginning vertex, compare every weight from other nodes not yet visited, and accumulate the weight from the beginning vertex.
6. If a weight smaller than the previous weight is found, set the compared weight into the smaller one being found.
7. After comparing distances to neighboring nodes, mark all the visited nodes, in order to avoid comparing again evaluated nodes and only maintaining the last minimum distance.

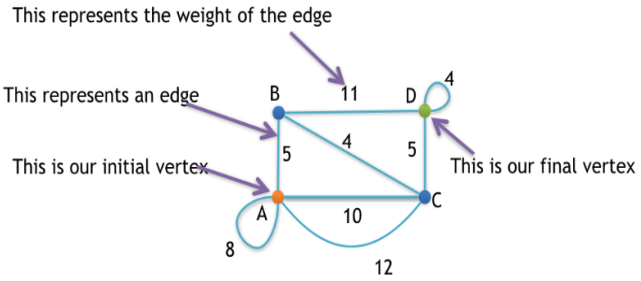


Fig. 4. Weighted graph example that will be used in Dijkstra's Algorithm. Adapted from [2]

After following the steps detailed above the image, we will then find the best (shortest) path from vertex A as the source node to vertex D as the destination node.

Marked	A	B	C	D
A	0	∞	∞	∞
B	0	Min(∞, 0+5) **5**	Min(∞, 0+10) 10	∞
C	0	5	Min(10, 5+4) **9**	Min(∞, 5+11) 16
D	0	5	9	Min(16, 9+5) **14**

Note! Column D has the marked value 14.
This means our shortest path has the weight 14.

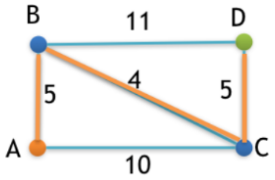


Fig. 4. Route finding table and the shortest path found by using Dijkstra's Algorithm. Adapted from [2]

By implementing Dijkstra's Algorithm, the shortest path from source node A to destination node D has a distance of 14, with the specified route taken being A, then B, then C, then D.

C. Water Ambulance

A water ambulance, also sometimes called a marine ambulance, is a dedicated emergency medical service vehicle that moves on rivers, lakes, or coastal areas to provide quick health assistance and transportation to individuals needing them. These ambulances are equipped with medical facilities and trained personnel to respond to emergencies, just like their land counterparts, albeit having the capability to access places where land ambulances are not capable of. Water ambulances is the key for overcoming challenges related to medical access critical for communities located near water bodies. Water ambulances can also provide aa rapid response to incidents such as water accidents, medical emergencies for water vehicle passengers, or situations where land ambulances are deemed impractical.

Water ambulances are capable of providing initial medical care and also stabilizing patients during transit from their location into a determined medical facility. The biggest advantage owned by water ambulances are their versatility in regions with extensive water networks, optimizing time needed for transportation while also increasing the survival outputs of medical emergencies. An additional advantage is the ability to operate in disaster response scenarios, where they can operate through flooded areas or reach locations deemed unable or difficult to be reached by land. The appropriate use of water ambulances will give medical practitioners an edge in dealing with special scenarios related to water bodies, expanding the reach of medical services by a significant margin, especially in archipelagos and spread-out island communities.

III. METHODOLOGY

A. Map Representation

In order to represent the islands and water routes connecting them, a weighted graph will be used. This graph will have a set of vertices representing islands and a set of edges representing water routes connecting those islands.

An ordinary weighted graph is used for its capability of representing different scenarios related to the mapping. The ordinary weighted graph is capable to contain the distance between any two islands as the weight of the edge connecting those two islands' nodes. It is capable of representing map of islands in the open sea with determined distances between every pair of two islands. It can also represent river-connected communities, or islands in the open sea with predetermined possible routes (not all pairs of two islands could be traversed).

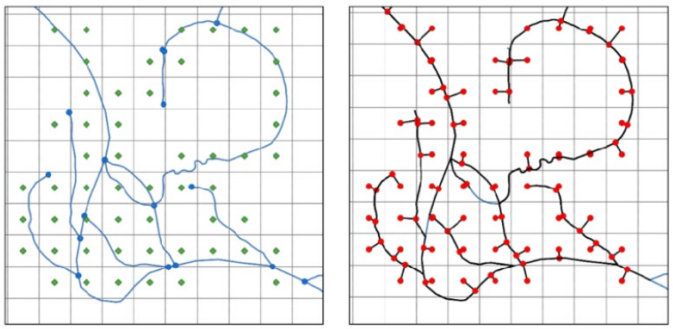


Fig. 5. Representing a map of centroids largely connected by a river network and the processed graph as a result. Adapted from [4]

The method used in this paper is most definitely not new. Researchers has long used the weighted graph representation as a way to analyze the network of routes in a given map and deal with their properties. In case of finding the shortest route possible, the weighted graph made as a representation of the map used will then be processed by Dijkstra's Algorithm, which will then find the most optimal route from the source node to the destination node.

Given the simplicity of the method used, it can actually be an advantage when we take a look into how it will be used. Efficient planning will surely benefit from the quick time required to run the algorithm and get the final needed result. In fact, there are several other algorithms useful for finding the shortest possible route, including but not limited to Bellman-Ford Algorithm and Floyd-Warshall Algorithm. Dijkstra's Algorithm is known as the algorithm with the lowest time complexity, giving it an advantage compared to the other algorithms

Breadth-first search (BFS)	$O(V+E)$
Dijkstra	$O(V^2)$ by linear array for priority queue $O((V+E)\lg V)$ by binary heap $O(V\lg V+E)$ by Fibonacci heap
Bellman-Ford-Moore	$O(VE)$
Directed acyclic graphs (DAG)	$Q(V+E)$
Floyd-Warshall	$O(V^3)$
Transitive closure	$O(V^3)$
Johnson	$Q(V^2\lg V+VE)$

Fig. 6. Comparing the time complexity of different algorithms used in finding shortest possible route. Adapted from [1]

B. Graph Representation and Algorithm

To represent the graph in a way that could be understood by computers, the adjacency matrix will be used. In this particular case, the code will assume that if multiple edges are present, then the edge with the lowest weight will be used in the graph. For now, our code does not include an extensive versatility or error checking mechanisms.

In terms of simplicity, the code is indeed quite simple but with some modifications. Due to the constraints of the problem, some changes have to be made to the original Dijkstra's Algorithm. While the original version finds the shortest possible route between two nodes on a graph, the modified version will find the shortest possible route to start from a node, visit every other node at least once, and then come back to the start node. The code generates all permutations of nodes, calculates the total route length for each permutation, then selects the minimum length as the solution. As the number of edges and vertices increase, the time complexity will however increase quite significantly too. In addition, while the original version stops when the shortest path to all reachable nodes is found, this modified version continues until it has explored all permutations of nodes, ensuring that each of them is visited at least once.

C. Medical Check-Up

Our code is focused on determining the shortest possible route of a routine medical check-up done on every single node represented in the map. The Dijkstra's Algorithm used in this code is the largely unmodified version, so visiting one node or place twice is allowed in finding the shortest possible route.

The only main target of the medical check-up is to start from one node, visit every other node at least once (more than once is allowed), and then return back to the same node the journey started from. In order to optimize fuel consumption and reduce total time needed, this code will surely be helpful and the result will be easy to understand.

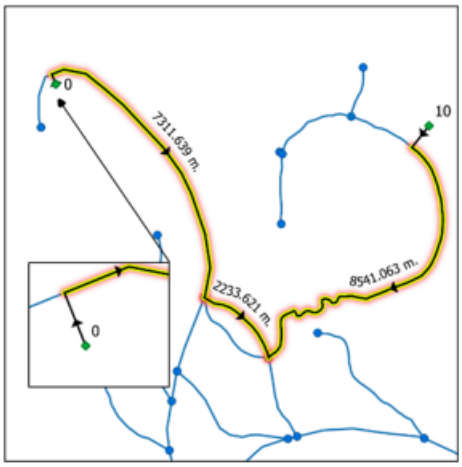


Fig. 7. An example of the shortest route possibly taken determined by the Dijkstra's Algorithm. Adapted from [1]

D. Complete Design

Here is the complete design for the program produced in this paper.

1. The program will contain previously defined graph containing all the vertices representing the places and edges representing the water routes connecting those places.
2. The graph will be represented as an adjacency matrix, with the row index and column index representing each node, and the elements valued as 0 for inexistant route connecting the two nodes or the distance between two nodes in integer.

IV. RESULT

A. Program

```python
import sys
import itertools

def dijkstra(graph, start):
    num_nodes = len(graph)
    all_nodes = set(range(num_nodes))

    min_length = sys.maxsize
    min_path = None

    for permuted_nodes in itertools.permutations(all_nodes - {start}):
        current_path = [start] + list(permuted_nodes) + [start]
        current_length = calculate_route_length(graph, current_path)

        if current_length < min_length:
            min_length = current_length
            min_path = current_path

    return min_path

def calculate_route_length(graph, route):
    length = 0
    for i in range(len(route) - 1):
        length += graph[route[i]][route[i + 1]]
    return length

def main():
    # Example graph contained inside
    graph = [
        [0, 34, 31, 64, 92, 84, 86, 50, 56, 83],
        [34, 0, 27, 63, 83, 32, 13, 72, 16, 43],
        [31, 27, 0, 59, 55, 16, 23, 53, 60, 88],
        [64, 63, 59, 0, 13, 82, 14, 73, 61, 14],
        [92, 83, 55, 13, 0, 27, 43, 16, 84, 13],
        [84, 32, 16, 82, 27, 0, 80, 56, 50, 57],
        [86, 13, 23, 14, 43, 80, 0, 39, 12, 55],
        [50, 72, 53, 73, 16, 56, 39, 0, 69, 53],
        [56, 16, 60, 61, 84, 50, 12, 69, 0, 12],
        [83, 43, 88, 14, 13, 57, 55, 53, 12, 0]
    ]

    start_node = int(input("Enter the start node (0-based index): "))

    if 0 <= start_node < len(graph):
        shortest_route = dijkstra(graph, start_node)
        route_length = calculate_route_length(graph, shortest_route)
        print(f"Shortest route starting from node {start_node}: {shortest_route}")
        print(f"Route length: {route_length}")
    else:
        print("Invalid start node.")

if __name__ == "__main__":
    main()
```

Fig. 8. Full code used for finding the shortest possible route for medical check-up using water ambulances.

B. Testing

```
    graph = [
        [0, 34, 31, 64, 92, 84, 86, 50, 56, 83],
        [34, 0, 27, 63, 83, 32, 13, 72, 16, 43],
        [31, 27, 0, 59, 55, 16, 23, 53, 60, 88],
        [64, 63, 59, 0, 13, 82, 14, 73, 61, 14],
        [92, 83, 55, 13, 0, 27, 43, 16, 84, 13],
        [84, 32, 16, 82, 27, 0, 80, 56, 50, 57],
        [86, 13, 23, 14, 43, 80, 0, 39, 12, 55],
        [50, 72, 53, 73, 16, 56, 39, 0, 69, 53],
        [56, 16, 60, 61, 84, 50, 12, 69, 0, 12],
        [83, 43, 88, 14, 13, 57, 55, 53, 12, 0]
    ]
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

PS C:\Users\Hp\Documents\MATDIS> python -u "c:\Users\Hp\Documents\MATDIS\tes.py"
Enter the start node (0-based index): 1
Shortest route starting from node 1: [1, 5, 2, 0, 7, 4, 3, 9, 8, 6, 1]
Route length: 209
PS C:\Users\Hp\Documents\MATDIS>
```

```
def main():
    # Example graph contained inside
    graph = [
        [0, 34, 31, 64, 92, 84, 86, 50, 56, 83],
        [34, 0, 27, 63, 83, 32, 13, 72, 16, 43],
        [31, 27, 0, 59, 55, 16, 23, 53, 60, 88],
        [64, 63, 59, 0, 13, 82, 14, 73, 61, 14],
        [92, 83, 55, 13, 0, 27, 43, 16, 84, 13],
        [84, 32, 16, 82, 27, 0, 80, 56, 50, 57],
        [86, 13, 23, 14, 43, 80, 0, 39, 12, 55],
        [50, 72, 53, 73, 16, 56, 39, 0, 69, 53],
        [56, 16, 60, 61, 84, 50, 12, 69, 0, 12],
        [83, 43, 88, 14, 13, 57, 55, 53, 12, 0]
    ]
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

PS C:\Users\Hp\Documents\MATDIS> python -u "c:\Users\Hp\Documents\MATDIS\tes.py"
Enter the start node (0-based index): 6
Shortest route starting from node 6: [6, 1, 5, 2, 0, 7, 4, 3, 9, 8, 6]
Route length: 209
PS C:\Users\Hp\Documents\MATDIS>
```

```
def main():
    # Example graph contained inside
    graph = [
    [0, 47, 65, 42, 55, 59, 78, 91, 88, 11],
    [47, 0, 47, 53, 66, 89, 30, 70, 46, 58],
    [65, 47, 0, 21, 52, 45, 25, 86, 81, 41],
    [42, 53, 21, 0, 95, 55, 24, 87, 24, 42],
    [55, 66, 52, 95, 0, 34, 69, 48, 63, 31],
    [59, 89, 45, 55, 34, 0, 82, 64, 15, 94],
    [78, 30, 25, 24, 69, 82, 0, 22, 39, 66],
    [91, 70, 86, 87, 48, 64, 22, 0, 37, 13],
    [88, 46, 81, 24, 63, 15, 39, 37, 0, 83],
    [11, 58, 41, 42, 31, 94, 66, 13, 83, 0]
    ]
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

PS C:\Users\Hp\Documents\MATDIS> python -u "c:\Users\Hp\Documents\MATDIS\tes.py"
Enter the start node (0-based index): 5
Shortest route starting from node 5: [5, 4, 7, 9, 0, 1, 6, 2, 3, 8, 5]
Route length: 268
PS C:\Users\Hp\Documents\MATDIS>
```

```
def main():
    # Example graph contained inside
    graph = [
    [0, 47, 65, 42, 55, 59, 78, 91, 88, 11],
    [47, 0, 47, 53, 66, 89, 30, 70, 46, 58],
    [65, 47, 0, 21, 52, 45, 25, 86, 81, 41],
    [42, 53, 21, 0, 95, 55, 24, 87, 24, 42],
    [55, 66, 52, 95, 0, 34, 69, 48, 63, 31],
    [59, 89, 45, 55, 34, 0, 82, 64, 15, 94],
    [78, 30, 25, 24, 69, 82, 0, 22, 39, 66],
    [91, 70, 86, 87, 48, 64, 22, 0, 37, 13],
    [88, 46, 81, 24, 63, 15, 39, 37, 0, 83],
    [11, 58, 41, 42, 31, 94, 66, 13, 83, 0]
    ]
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  COMMENTS

PS C:\Users\Hp\Documents\MATDIS> python -u "c:\Users\Hp\Documents\MATDIS\tes.py"
Enter the start node (0-based index): 7
Shortest route starting from node 7: [7, 4, 5, 8, 3, 2, 6, 1, 0, 9, 7]
Route length: 268
PS C:\Users\Hp\Documents\MATDIS>
```

V. Conclusion

In conclusion, Dijkstra's Algorithm is useful in determining best possible route for medical check-up using water ambulances in maritime communities.

VI. Appendix

Appendixes, if needed, appear before the acknowledgment.

VII. Acknowledgment

The preferred spelling of the word "acknowledgment" in American English is without an "e" after the "g." Use the singular heading even if you have many acknowledgments. Avoid expressions such as "One of us (S.B.A.) would like to thank ... ." Instead, write "F. A. Author thanks ... ." Sponsor and financial support acknowledgments are placed in the unnumbered footnote on the first page.

References

[1]N. L. Biggs, E Keith Lloyd, and Robin James Wilson, Graph theory 1736-1936. Oxford: Clarendon Press, Dr, 2006.

[2]R. Munir, "Graf (Bag.1) Bahan Kuliah IF2120 Matematika Diskrit Program Studi Teknik Informatika STEI-ITB." Accessed: Dec. 11, 2023. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

[3]R. Munir, "Graf (Bag.2)." Accessed: Dec. 11, 2023. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian2.pdf

[4]Alireza Boloori and M. Mahmoudi, "Networks," IGI Global eBooks, pp. 150–178, Jan. 2013, doi: https://doi.org/10.4018/978-1-4666-2661-4.ch012.

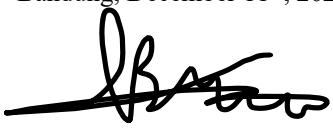
[5]"Dijkstra Algorithm - Finding Shortest Path - Graph - dyclassroom | Have fun learning :-)," dyclassroom.com. https://dyclassroom.com/graph/dijkstra-algorithm-finding-shortest-path (accessed Dec. 11, 2023).

[6]N. Cadieux, M. Kalacska, O. T. Coomes, M. Tanaka, and Y. Takasaki, "A Python Algorithm for Shortest-Path River Network Distance Calculations Considering River Flow Direction," Data, vol. 5, no. 1, p. 8, Mar. 2020, doi: https://doi.org/10.3390/data5010008.

STATEMENT OF ORIGINALITY

I hereby declare that the paper I have authored is an original work, and is not a copy or a translation from another person's paper, nor is it plagiarized.

Bandung, December 11th, 2023



Christopher Brian
13522106