Application of Shortest Path Algorithm to GPS Navigation in Grand Theft Auto V Game

Muhammad Zulfiansyah Bayu Pratama – 13521028[1]
Undergraduate Program in Informatics Engineering
School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13521028@std.stei.itb.ac.id

***Abstract*—Grand Theft Auto V (GTA V) is an Open World game developed by Rockstar Games. In this game, if the player rides a vehicle such as a motorbike, car, etc., there is a GPS navigation feature that functions to determine the shortest path from the player's position to the desired mission or destination. The map in GTA V uses the concept of graph to represent locations. GPS navigation in GTA V uses shortest path algorithm to find the best path to go through. In this paper the author will try to discuss how the graph concept is applied.**

***Keywords*—GTA V, GPS navigation, graph, shortest path**

I. Introduction

Grand Theft Auto (GTA) V is an Open World game made by Rockstar Games in 2013. There are three main characters in this game, namely Franklin, Michael, and Trevor. Players can carry out missions while following the story contained in this game or just explore the vast world in GTA V by driving. GTA V can be played on multiple devices like Playstation, XBOX, PC, etc.

Fig. 1.1 Grand Theft Auto V
Source: https://www.rockstargames.com/gta-v

In GTA V, there is a unique feature, namely GPS navigation. This feature has been around since GTA IV was released in 2008. When the player is driving, GPS navigation can determine the shortest path that the player can take to reach the desired destination. To activate the GPS navigation feature, players must determine the destination route by opening the map and marking the location they want to go. When carrying out missions and driving, navigation can show the direction of the path to the mission destination.

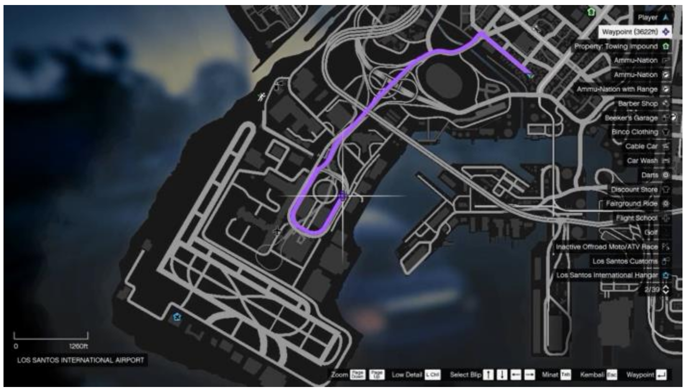


Fig. 1.2 GTA V map (after marking the destination)
Source: https://www.youtube.com/watch?v=UuRWiu83ybQ

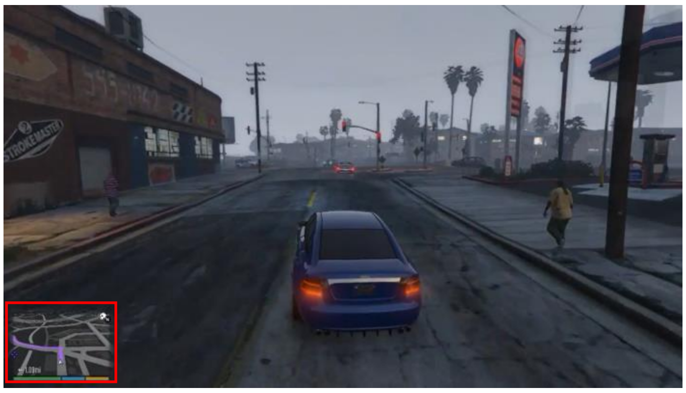


Fig. 1.3` GTA V gameplay (with GPS navigation)
Source: https://www.youtube.com/watch?v=UuRWiu83ybQ

The recommended travel route for GPS navigation is obtained from the results of the shortest path algorithm from the map graph in GTA. In this paper, the author will use Dijkstra's algorithm application to determine the shortest path from one place to another as an approach to how navigation works in GTA V.

II. Theoretical Basis

A. Graph Theory

In discrete mathematics, and more specifically in graph theory, a graph is a structure amounting to a set of objects in which some pairs of the objects are in some sense "related". The objects correspond to mathematical abstractions called vertices (also called nodes or points) and each of the related pairs of vertices is called an edge (also called a link or line).[1] Formally, 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 paired vertices, whose elements are called edges.[2]

B. *Graph Types*

Based on whether there are loops or multiple edges in a graph, then the graph is divided into two types :

1. *Simple Graph*

A simple graph is an unweighted, undirected graph containing no graph loops or multiple edges.[3]

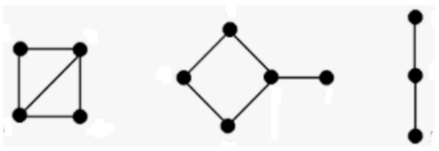


Fig. 2.1 Simple Graph
Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

2. *Unsimple Graph*

A graph that contains multiple edges or loops is called an unsimple graph (Munir, 2010, p. 357).

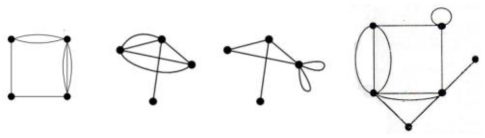


Fig. 2.2 Unsimple Graph
Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

Unsimple graphs are further divided into:

a. Multi-Graph

A multigraph is a graph that has many edges (Munir, 2010, p. 358).

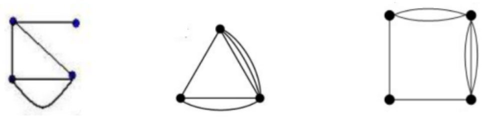


Fig. 2.3 Multi-Graph
Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

b. Pseudo-Graph

Pseudo-graph is a graph that contains loop edges. (Munir, 2010, p. 358).

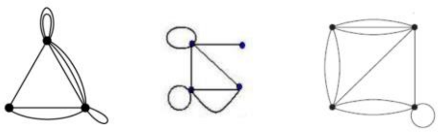


Fig. 2.4 Pseudo-Graph

Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

Meanwhile, based on the orientation of the edges, graphs are generally divided into two types :

1. *Undirected Graph*

A graph whose edges have no orientation is called an undirected graph. In this graph, the order of pairs of vertices connected by edges is not considered. So, $(u, v) = (v, u)$ are equal sides (Munir, 2010, p. 358).

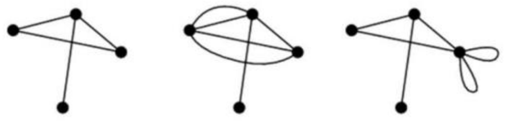


Fig. 2.5 Undirected Graph
Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

2. *Directed Graph*

A graph in which each edge is assigned a direction is called a directed graph. In a directed graph (u, v) and (v, u) represent two unequal arcs. For an arc (u, v), u is called an initial vertex and v is called a terminal vertex (Munir, 2010, p. 358).

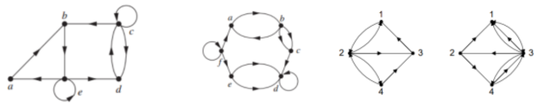


Fig. 2.6 Directed Graph
Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

C. *Basic Graph Terminology*

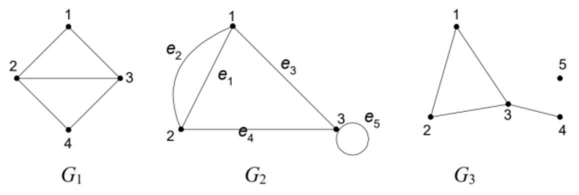


Fig. 2.7 Graph G_1, G_2, and G_3
Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

1. *Adjacent*

Two vertices in an undirected graph G are said to be adjacent if they are directly connected by an edge. In other words, u is next to v if (u, v) is an edge of graph G (Munir, 2010, p. 365).

In Figure 2.7 G_1, node 1 is adjacent to nodes 2 and 3. But node 1 is not adjacent to node 4. Likewise, node 4 is an adjacency of nodes 2 and 3 but not with node 1.

2. *Incidency*

For any edge $e = (u, v)$, edge e is said to be incidence to vertices u and v (Munir, 2010, p. 365).

3. *Isolated Vertex*

   Isolated nodes are nodes that have no edges side by side with him (Munir, 2010, p. 365). In Figure 2.7, G_3 has an isolated vertex, namely vertex 5.

4. *Degree*

   The degree of a vertex is the number of edges adjacent to that vertex. The degree notation for an undirected graph is $d(v)$, where v is a vertex. Meanwhile, in a directed graph, the degree of vertices v is expressed by $d_{in}(v)$ as the number of edges entering v and $d_{out}(v)$ as the number of edges leaving v.

5. *Path*

   A path of length n from the initial vertex v_0 to the destination vertex v_n in graph G is a sequence of alternating vertices and edges of the form $v_0, e_1, v_1, e_2, v_2, \ldots, v_{n-1}, e_n, v_n$ such that $e_1 = (v_0, v_1), e_2 = (v_1, v_2), \ldots, e_n = (v_{n-1}, v_n)$ are the edges of graph G (Munir, 2010, p. 369).

6. *Cycle/Circuit*

   A path that starts and ends at the same vertex is called a circuit or cycle (Munir, 2010, p. 370). In Figure 2.7 Graph G1, 1, 2, 3, 4, and 1 is a circuit. The length of a circuit is the number of edges in that circuit. If each edge is traversed differently on a circuit/cycle, then it's a simple circuit.

7. *Connected*

   Two vertices v_1 and v_2 are said to be connected if there is a path from v_1 to v_2. G is called a connected graph if for every pair of vertices v_i and v_j in set V there is a path from v_i for v_j.

8. *Weighted Graph*

   A weighted graph is a graph in which a number (the weight) is assigned to each edge.[4]

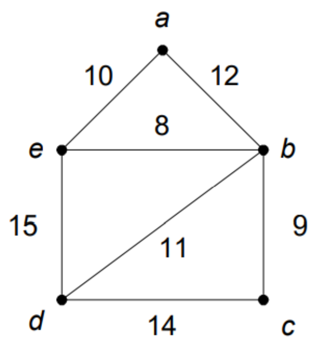


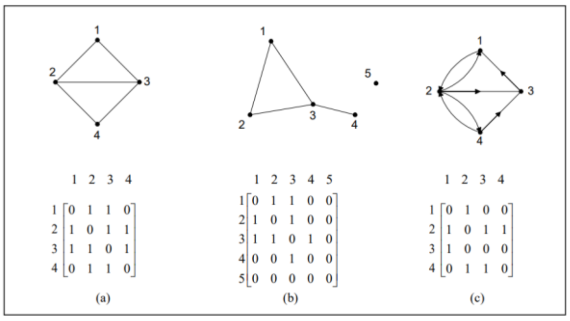
Fig. 2.8 Weighted Graph
Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf

D. Graph Representation in Programming

In programming, a structure is needed so that data regarding graphs can be created stored, and processed efficiently. The graph can be represented in three forms:

1. *Adjacency Matrix*

   The adjacency matrix is a square $n \times n$ matrix A such that its element $A_{i,j}$ is one when there is an edge from vertex u_i to vertex u_j, and zero when there is no edge.[5]

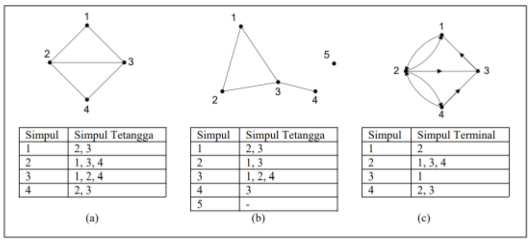


$$\begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 1 & 0 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 1 & 1 & 0 & 1 \\ 4 & 0 & 1 & 1 & 0 \end{array} \quad \text{(a)} \qquad \begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 & 0 \\ 3 & 1 & 1 & 0 & 1 & 0 \\ 4 & 0 & 0 & 1 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 \end{array} \quad \text{(b)} \qquad \begin{array}{c|cccc} & 1 & 2 & 3 & 4 \\ \hline 1 & 0 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 1 \\ 3 & 1 & 0 & 0 & 0 \\ 4 & 0 & 1 & 1 & 0 \end{array} \quad \text{(c)}$$

Fig. 2.9 Adjacency Matrix Representation
Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian2.pdf

2. *Adjacency List*

   In the adjacency list, connections between vertices are defined using a linked list. Each end has a list of vertices connected to that vertex.



Simpul	Simpul Tetangga
1	2, 3
2	1, 3, 4
3	1, 2, 4
4	2, 3

(a)

Simpul	Simpul Tetangga
1	2, 3
2	1, 3
3	1, 2, 4
4	3
5	-

(b)

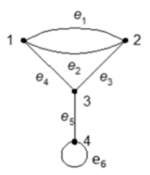
Simpul	Simpul Terminal
1	2
2	1, 3, 4
3	1
4	2, 3

(c)

Fig. 2.10 Adjacency List Representation
Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian2.pdf

3. *Incidency Matrix*

   In the incidence matrix, the rows and columns of the matrix contain 1's and 0s for an unweighted graph. For every $A = [a_{ij}]$, if vertices i and j are neighbors then a_{ij} will have a value of 1 if the i node is side by side with edge j, and a_{ij} is 0 if vertex i is not sided by side j.



$$\begin{array}{c|cccccc} & e1 & e2 & e3 & e4 & e5 & e6 \\ \hline 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 2 & 1 & 1 & 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 1 & 1 & 1 & 0 \\ 4 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}$$

Fig. 2.11 Incidency Matrix Representation
Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian2.pdf

E. Shortest Path

The shortest path is a problem for looking for a series of edges that need to be traversed starting from vertex to other vertices so that the total weight of the sides traversed is as small as possible.[6]

The definition of weight and shortest path depends on the problem at hand. As an example the edge weight on the graph representing the city and the road represents the travel time, meaning that the shortest path represents the least total travel time to move from one city to another.

F. Dijkstra Algorithm

Dijkstra's algorithm is a greedy algorithm that is used to solve the shortest path problem for a directed graph with non-negative line weights. The input of this algorithm is a weighted directed graph G and an origin s in a set of lines V. For example if the vertices of a graph represent the cities and the line weights represent the distances between the cities, Dijkstra's algorithm can be used to find the shortest distance between two cities. The cost of a line can be thought of as the distance between two nodes, i.e. the sum of the distances of all the lines in that path. For a pair of points s and t in V, this algorithm calculates the shortest distance from s to t.

The complexity of Dijkstra's algorithm (with heap optimization) runs in $O((V + E)\ log\ V)$ where V and E are the numbers of edges and vertices.

III. IMPLEMENTATION

A. Graph Formation

Fig. 3.1 GTA V map
Source: https://www.youtube.com/watch?v=UuRWiu83ybQ

The map in GTA V maps places based on the coordinates of the latitude and longitude of the place and these coordinates become the identity of a location, which in the graph we assume is a node. Then a collection of location nodes is formed. Then the roads that connect between places we consider as edges.

In this paper, the author will only take a few places to be a sample experiment. The location taken was around Franklin's house.

Fig. 3.2 GTA V map (near Franklin's house)
Source: https://www.youtube.com/watch?v=jbd3QTTwb98

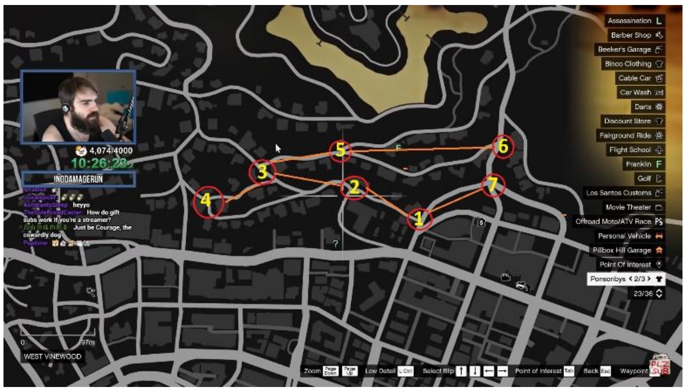


Fig. 3.2 GTA V map (near Franklin's house)
Source: https://www.youtube.com/watch?v=jbd3QTTwb98

B. Data Initialization

Suppose $adj[x, y]$ is the distance traveled from x to y or from y to x, then the list of $adj[x, y]$ is as follows:

- $adj[1,2] = 200;$
- $adj[2,3] = 250;$
- $adj\ [3,4] = 12;$
- $adj[4,5] = 300;$
- $adj[5,6] = 47;$
- $adj[6,7] = 190;$
- $adj[1,7] = 400;$

Suppose s is an initial node, then for each stage a data type is required to record information for all nodes:

•dist[x]: the shortest distance last found to reach the nodex from nodes.

•visited[x]: returns true if a nodex has been visited and returns false if nodex not yet visited.

Initially, all node x have the value dist[x]= ∞, and visited[x]= false. Especially for nodes, we know that the shortest distance to reach s from itself is 0. Therefore, dist[s]= 0.

C. Route Determination

The steps taken after the initialization process are as follows:

1.Select a node that has not been visited and has the smallest dist. Let's call this node u.

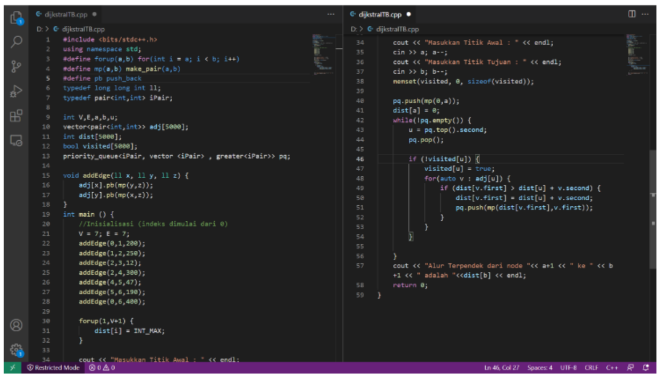
2. If there are no more nodes that have not been visited, or dist[u] has a value of ∞, it means that there are no more nodes

that can be visited. Dijkstra's algorithm ends.

3. Since u visited, then set visited[u] = true. Now ensure that the shortest path from s to u is dist[u].

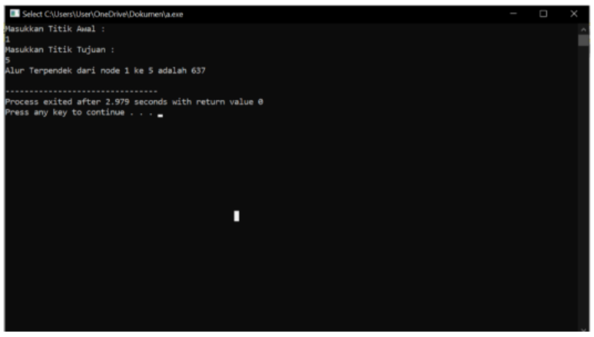
4. For each node that is a neighbor of u, do a process called "relax". The relax process for the u and v nodes is the process of updating the shortest distance to reach v, if it turns out that reaching v via u requires a smaller distance.

IV. EXPERIMENT



```cpp
#include <bits/stdc++.h>
using namespace std;
#define forup(a,b) for(int i = a; i < b; i++)
#define mp(a,b) make_pair(a,b)
#define pb push_back
typedef long long int ll;
typedef pair<int,int> iPair;

int V,E,a,b,u;
vector<pair<int,int>> adj[5000];
int dist[5000];
bool visited[5000];
priority_queue<iPair, vector <iPair> , greater<iPair>> pq;

void addEdge(ll x, ll y, ll z) {
    adj[x].pb(mp(y,z));
    adj[y].pb(mp(x,z));
}
int main () {
    //Inisialisasi (indeks dimulai dari 0)
    V = 7; E = 7;
    addEdge(0,1,200);
    addEdge(1,2,250);
    addEdge(2,3,12);
    addEdge(2,4,300);
    addEdge(4,5,47);
    addEdge(5,6,190);
    addEdge(0,6,400);

    forup(1,V+1) {
        dist[i] = INT_MAX;
    }

    cout << "Masukkan Titik Awal : " << endl;
    cin >> a; a--;
    cout << "Masukkan Titik Tujuan : " << endl;
    cin >> b; b--;
    memset(visited, 0, sizeof(visited));

    pq.push(mp(0,a));
    dist[a] = 0;
    while(!pq.empty()) {
        u = pq.top().second;
        pq.pop();

        if (!visited[u]) {
            visited[u] = true;
            for(auto v : adj[u]) {
                if (dist[v.first] > dist[u] + v.second) {
                    dist[v.first] = dist[u] + v.second;
                    pq.push(mp(dist[v.first],v.first));
                }
            }
        }

    }
    cout << "Alur Terpendek dari node "<< a+1 << " ke " << b+1 << " adalah "<<dist[b] << endl;
    return 0;
}
```

Fig. 5.1 Shortest Path Implemetation (C++)



```
Masukkan Titik Awal :
1
Masukkan Titik Tujuan :
5
Alur Terpendek dari node 1 ke 5 adalah 637

--------------------------------
Process exited after 2.979 seconds with return value 0
Press any key to continue . . .
```

Fig. 5.2 Shortest Path Implemetation's Result (C++)

V. CONCLUSION

Based on the analysis and testing of this paper, it can be concluded that Dijkstra's algorithm is the best solution for solving the problem of finding the shortest route in By finding all the fastest possibilities from a node to a nodex, Dijkstra's algorithm can run with more efficient time complexity and memory compared to other algorithms. Reasons for using Dijkstra's algorithm What can be learned from this paper are various shortest path solutions and the implementation of Dijkstra's algorithm in programming..

REFERENCES

[1] Trudeau. Richard J, *Introduction to Graph Theory* (Corrected, enlarged republication. ed.). New York: Dover Pub, 1993, p. 19.

[2] Munir, R, *Matematika Diskrit*, 4th ed. Bandung: Informatika Bandung, 2010, p. 356.

[3] Bronshtein, I. N. and Semendyayev, *K. A. Handbook of Mathematics*, 4th ed. New York: Springer-Verlag, 2004, p. 346.

[4] Fletcher. Peter, Hoyle. Hughes, Patty. C. Wayne, *Foundations of Discrete Mathematics* (International student ed.). Boston: PWS-KENT Pub. Co, 1991, p. 463.

[5] Biggs. Norman, *Algebraic Graph Theory*, Cambridge Mathematical Library (2nd ed.), Cambridge University Press, Definition 2.1, 1993, p. 7.

[6] Aji. Alham Fikri, Gozali. William, *Pemrograman Kompetitif Dasar*, Jakarata: CV. Nulisbuku Jendela Dunia, 2019, p. 128.

[7] Aji. Alham Fikri, Gozali. William, *Pemrograman Kompetitif Dasar*, Jakarata: CV. Nulisbuku Jendela Dunia, 2019, p. 129.

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, 10 Desember 2020

Ttd

Muhammad Zulfiansyah Bayu Pratama/13521028