Implementasi Algoritma A* (Star) untuk Menentukan Rute Jarak Terpendek

Melisa Tryastie, Yukandri, Lia Nelda, Ressa Priskila, Viktor Handrianus Pranatawijaya

Program Studi Teknik Infomatika, Fakultas Teknik,
Universitas Palangka Raya,
Jalan Yos Sudarso, Palangka, Kec. Jekan Raya, Kota Palangka Raya, Indonesia
tryastie@mhs.eng.upr.ac.id
yukandri01@mhs.eng.upr.ac.id
lia003@mhs.eng.upr.ac.id
ressa@it.upr.ac.id
viktorhp@it.upr.ac.id

Abstract

In this research, the implementation of the A star algorithm is used to find efficient routes, reduce the travel time, and optimize the use of existing road infrastructure at waypoints. The A star algorithm uses the concept of Open List and Close List, which helps to reduce the number of rechecks on the points travelled, thus speeding up the search process. The A star algorithm stops when there are no more points on the open list or when the end point has been determined. This research method uses primary data, which consists of five street locations in Palangka Raya city, as nodes and combines the distance between the street points. The implemented program uses the A Star algorithm to calculate the shortest route and displays the path along with the distance. The purpose of this research is to achieve the shortest route and calculate the distance travelled for the waypoints.

Keywords: *shortest route, A* algorithm, waypoint.*

1. Pendahuluan

Dalam pengembangan sistem navigasi, penentuan rute terpendek antara dua lokasi merupakan salah satu aspek penting. Algoritma pencarian jalur terpendek, atau yang dikenal sebagai *shortest path*, digunakan untuk menemukan rute tercepat dalam sebuah *graph* [1]. Ada dua pendekatan umum dalam pencarian rute terpendek, yaitu pencarian buta (*blind search*) dan pencarian heuristik (*heuristic search*). Pencarian buta lebih sederhana namun membutuhkan waktu lebih lama, sedangkan pencarian heuristik memberikan hasil yang lebih optimal dalam waktu yang lebih singkat. Salah satu teknik pencarian heuristik yang efektif adalah algoritma A* [1].

Algoritma A* merupakan algoritma pencarian jarak terpendek yang optimal dan berfitur lengkap. Dengan menggunakan konsep *Open List* dan *Close List*, algoritma ini dapat mengurangi jumlah pengecekan ulang pada titik-titik yang dilalui, sehingga mempercepat proses pencarian. Algoritma A* berhenti saat tidak ada lagi titik pada open list atau saat titik akhir telah ditentukan [2].

Implementasi algoritma A* untuk menentukan rute jarak terpendek diharapkan dapat membantu meningkatkan efisiensi, mengurangi waktu tempuh, serta mengoptimalkan penggunaan infrastruktur jalan yang ada. Dengan demikian, implementasi ini diharapkan dapat memberikan kontribusi positif. Dalam penelitian ini, kami mengimplementasikan algoritma A* untuk menentukan rute terpendek. Data yang digunakan dalam penelitian ini adalah data titik jalan, jarak antar jalan, dan estimasi waktu tempuh. Algoritma ini diimplementasikan dalam bahasa Python untuk memudahkan integrasi dengan sistem navigasi yang ada.

2. Metode Penelitian

2.1. Data Penelitian

Data yang dipergunakan dalam penelitian ini adalah data primer dan melibatkan informasi terkait lokasi titik-titik jalan di wilayah Kota Palangka Raya. Setiap titik jalan diwakili sebagai simpul dalam graf, dengan lokasi sebagai simpulnya, dan jarak antara titik-titik jalan sebagai sisi yang menghubungkannya. Jarak antara titik-titik jalan yang dimasukkan ke dalam sistem dan jumlah simpul bervariasi tergantung pada tujuan titik jalan yang ditetapkan. Evaluasi rute didasarkan pada perhitungan jarak, tanpa mempertimbangkan faktor kemacetan lalu lintas atau kondisi geografis rute. Penentuan posisi menggunakan Google My Maps sebagai panduan, dengan menghitung jarak antar titik berdasarkan koordinat lintang dan bujur lokasi yang dipilih [3].

2.2. Graph

Menggunakan *graph* pada peta untuk meyediakan jalur yang terhubung ke *node* untuk keseluruhan rute pada objek penelitian, dan mengimplementasikan algoritma A* untuk menentukan rute terpendek dengan cepat dan efisien [4].

2.3. Tahapan Algoritma A Star

Algoritma A* merupakan algoritma pencarian yang menghitung rute paling efisien dengan mengevaluasi *node* terkecil. Ini dirancang untuk menemukan jalur tercepat antara dua titik menggunakan teori *graph* atau grafik dan kumpulan *node* yang mewakili titik awal dan titik akhir [5]. Untuk mencari jarak terpendek pada suatu peta, peta tersebut harus direpresentasikan dalam bentuk diagram. Setiap titik pada graph mewakili persimpangan atau lokasi tertentu pada peta, sedangkan tepi graph mewakili jalur antara persimpangan tersebut. Tepi dari graph tersebut memiliki bobot yang mencerminkan panjang jarak antara titik-titik tersebut. Algoritma A* digunakan untuk mengevaluasi setiap node dengan menggabungkan g(n), yang merupakan biaya untuk mencapai node tersebut, dan h(n), yang merupakan perkiraan biaya yang diperlukan untuk mencapai tujuan dari node tersebut. [6]. Dalam notasi matematika dapat direpresentasikan sebagai berikut:

Rumus:

$$f(n) = h(n) + g(n) \quad (1)$$

Keterangan:

a. f(n) adalah biaya estimasi terendah untuk mencapai *node* n.
b. h(n) adalah biaya dari *node* awal ke *node* n.
c. g(n) adalah perkiraan biaya dari *node* n ke *node* akhir.

Dalam penerapannya algoritma A* mempunyai beberapa istilah dasar antara lain titik awal (*starting point*), simpul (*nodes*), S, *open list*, *closed list*, harga (*cost*) [1]. Dapat dijelaskan sebagai berikut:

a. “Titik awal (*starting point*)” adalah titik awal dalam menentukan algoritma A*.
b. “Simpul (*nodes*)” Adalah sebuath titik diagram yang mewakili untuk mencari rute terpendek.
c. "S" adalah node yang digunakan sebagai titik awal dalam pencarian rute terpendek.
d. "*Open List*" adalah kumpulan node yang belum dieksplorasi sepenuhnya dan masih dapat diakses dari titik awal atau node yang sedang dieksplorasi.
e. “*Close List*” adalah tempat menyimpan data sebelum *node* S, yang juga merupakan bagian dari jalur terpendek yang berhasil diperoleh.
f. “Harga (*Cost*)” adalah nilai yang diperoleh dari penjumlahan nilai setiap *node* pada jalur terpendek dari titik awal ke S, dan penjumlahan nilai estimasi dari *node* hingga *node* tujuan.

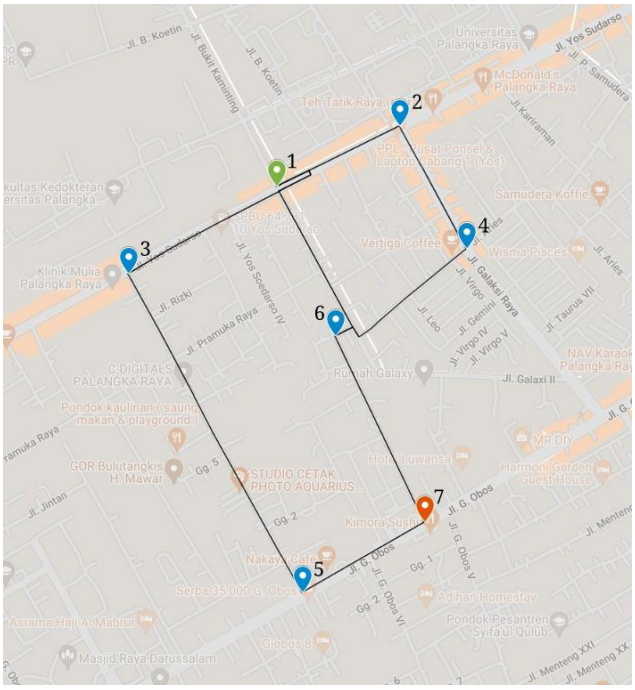
Inti dari algoritma A* adalah untuk menemukan rute terpendek dari titik awal ke node tujuan dengan meminimalkan biaya atau *cost*.

Langkah – langkah untuk mencari rute terpendek dengan algoritma A* [1] adalah sebagai berikut:

a. Langkah pertama adalah memulai dari node S sebagai titik awal..
b. Selanjutnya, tambahkan semua *node* yang terhubung dengan *node* S ke dalam *Open List*.
c. Pilih *node* dengan nilai terkecil dari *Open List*, yaitu *node-node* yang terhubung dengan *node* S.
d. Pindahkan *node* S ke *node* dengan nilai *Cost* terkecil. Simpan *node* sebelum S sebagai *node* induk dari S dan tambahkan ke dalam *Close List*. Jika terdapat *node* lain yang terhubung dengan S (yang sudah dipindahkan) namun belum ada di *Open List*, masukkan *node* tersebut ke dalam *Open List*.
e. Bandingkan nilai g(n) dengan nilai g(n) sebelumnya. Jika nilai g(n) yang baru lebih kecil, kembalikan S ke posisi sebelumnya atau titik awal. *Node* yang sudah dimasukkan ke dalam *Close List*. Ulangi langkah ini sampai solusi ditemukan atau tidak ada lagi *node* yang tersisa di *Open List*.

3. Hasil dan Pembahasan

Dalam penelitian ini, pencarian rute terpendek dilakukan dengan mengidentifikasi lokasi pada peta menggunakan informasi koordinat geografis, yaitu latitude dan longitude. Implementasi algoritma A* untuk mencari rute terpendek antara titik jalan atau tujuan dilakukan menggunakan bahasa pemrograman Python. Gambar 1 dan Tabel 1 di bawah ini menunjukkan perjalanan dari titik awal (ditandai dengan warna hijau) ke titik akhir (ditandai dengan warna merah) yang digunakan dalam penelitian.



Gambar 1. Goggle My Maps Titik awal dan Titik akhir

Tabel 1. Koordinat latitude dan Longitude dari titik-titik jalan.

No	Titik Jalan	Latitude	Longitude
1	Jl. Yos Sudarso	-2.21916	113.89514

p-ISSN: 2986-3929
e-ISSN: 3032-1948

No	Titik Jalan	Latitude	Longitude
2	Jl. Yos Sudarso	-2.21769	113.8981
3	Jl. Yos Sudarso	-2.22125	113.89156
4	Jl. Galaksi Raya	-2.22064	113.8997
5	Jl. G. Obos	-2.22894	113.89576
6	Jl. G. Garu	-2.22275	113.89655
7	Jl. G. Obos	-2.22723	113.8987

Berdasakan koordinat titik jalan lintang (Latitude) dan titik jalan (Longitude) yang diperoleh dari Goggle My Maps, Jarak antara satu titik dengan titik lainnya dihitung menggunakan rumus berikut:

Rumus [3] :

$$d_{ij} = \left(69 \cdot \sqrt{\left(lon_i - lon_j\right)^2 + \left(lat_i - lat_j\right)^2}\right) \cdot 1{,}60934 \quad (2)$$

Keterangan:

a. "d_{ij}" adalah jarak dalam satuan kilometer antara dua titik koordinat i dan j.
b. "lon_i" adalah nilai longitude titik i.
c. "lon_j" adalah nilai longitude titik j.
d. "lat_i" adalah nilai latitude titik i.
e. "lat_j" adalah nilai latitude titik j.

Rumus di atas menggunakan rumus Haversine, di mana konstanta 69 digunakan untuk mengubah perbedaan lintang menjadi jarak dalam mil. Konstanta ini didasarkan pada asumsi bahwa jarak sejajar satu derajat lintang sekitar 69 mil, atau sekitar 111,045 kilometer, pada garis lintang rata-rata Bumi. Setelah menghitung jarak dalam mil dengan konstanta 69, kita dapat mengonversinya menjadi kilometer dengan menggunakan faktor 1,60934. Faktor ini merupakan hasil dari konversi 1 mil menjadi kilometer (1 mil = 1,60934 kilometer), sehingga kita dapat mengalikan jarak dalam mil dengan faktor ini untuk mengubah satuan menjadi kilometer [6].

Program Python mencari jarak semua titik ke titik akhir:

```
import math
class Node:
    def __init__(self, x, y):
        self.x = x
        self.y = y

def haversineDist(n1, n2): # fungsi untuk menghitung jarak antara dua node
    # Mengubah koordinat menjadi radian
    lat1 = math.radians(n1.x)
    lon1 = math.radians(n1.y)
    lat2 = math.radians(n2.x)
    lon2 = math.radians(n2.y)

    # Menghitung jarak menggunakan rumus Haversine
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = math.sin(dlat / 2)**2 + math.cos(lat1) * math.cos(lat2) * math.sin(dlon / 2)**2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1-a))
    R = 6373.0 # Perkiraan radius bumi dalam km
    distance = R * c
```

p-ISSN: 2986-3929
e-ISSN: 3032-1948

```
    # Membulatkan jarak menjadi 2 tempat desimal
    distance = round(distance, 2)
    return distance

n1 = Node()
n2 = Node()
distance = haversineDist(n1, n2)
print(f"Jarak antara ({n1.x}, {n1.y}) dan ({n2.x}, {n2.y}) adalah {distance} km.")
```

Berdasarkan perhitungan sebelumnya tentang jarak antar titik, jumlah jarak akan dinyatakan dalam satuan kilometer dan dibulatkan menjadi dua angka di belakang koma, sehingga hasilnya adalah sebagai berikut:

Tabel 2. Jarak Masing-masing Titik Tujuan

x	1	2	3	4	5	6	7
1	0.00	0.37	0.46	0.53	1.09	0.43	0.98
2	0.37	0.00	0.83	0.37	1.28	0.59	1.06
3	0.46	0.83	0.00	0.91	0.97	0.58	1.04
4	0.53	0.37	0.91	0.00	1.02	0.42	0.74
5	1.09	1.28	0.97	1.02	0.00	0.69	0.38
6	0.43	0.59	0.58	0.42	0.69	0.00	0.55
7	0.98	1.06	1.04	0.74	0.38	0.55	0.00

Setelah mengetahui jarak antara setiap titik ke titik lainnya, langkah selanjutnya adalah menghitung pencarian rute terpendek dengan mengimplementasikan algoritma A* pada program berikut:

Program pencarian rute terpendek:

```
def aStarAlgo(start_node, stop_node):
    open_set = set(start_node)
    closed_set = set()
    g = {}               # Menyimpan jarak dari node awal
    parents = {}         # parents berisi peta kedekatan dari semua node
    # Jarak node awal dari dirinya sendiri adalah nol
    g[start_node] = 0
    # start_node adalah node root yaitu tidak memiliki node induk
    # jadi start_node diatur sebagai node induknya sendiri
    parents[start_node] = start_node
    while len(open_set) > 0:
        n = None
        # node dengan nilai f() terendah ditemukan
        for v in open_set:
            if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                n = v
        if n == stop_node or Graph_nodes[n] == None:
            pass
        else:
            for (m, weight) in get_neighbors(n):
                # node 'm' yang tidak ada dalam set pertama dan terakhir ditambahkan ke
                set pertama
                # n diatur sebagai induknya
                if m not in open_set and m not in closed_set:
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight
                # untuk setiap node m, bandingkan jaraknya dari start yaitu g(m) dengan
                # dari start melalui node n
                else:
```

p-ISSN: 2986-3929
e-ISSN: 3032-1948

```
                    if g[m] > g[n] + weight:
                        # update g(m)
                        g[m] = g[n] + weight
                        # ubah induk m menjadi n
                        parents[m] = n
                        # jika m ada dalam set tertutup, hapus dan tambahkan ke set
                        terbuka
                        if m in closed_set:
                            closed_set.remove(m)
                            open_set.add(m)
        if n == None:
            print('Path does not exist!')
            return None

        # jika node saat ini adalah stop_node
        # maka kita mulai merekonstruksi jalur dari node tersebut ke start_node
        if n == stop_node:
            path = []
            total_distance = 0
            while parents[n]!= n:
                path.append(n)
                total_distance += g[n] - g[parents[n]]
                n = parents[n]
            path.append(start_node)
            path.reverse()
            print('Rute Terpendek: {}'.format(path))
            print('Total Jarak Tempuh: {:.2f}'.format(total_distance))
            return path
        # hapus n dari open_set, dan tambahkan ke closed_set
        # karena semua tetangganya telah diperiksa
        open_set.remove(n)
        closed_set.add(n)
    print('Path does not exist!')
    return None

# mendefinisikan fungsi untuk mengembalikan tetangga dan jaraknya
# dari node yang dilewati
def get_neighbors(v):
    if v in Graph_nodes:
        return Graph_nodes[v]
    else:
        return None

def heuristic(n):
    H_dist = {
        'S': 0.98,
        'A': 1.06,
        'B': 1.04,
        'C': 0.74,
        'D': 0.38,
        'E': 0.55,
        'Z': 0
    }
    return H_dist[n]

Graph_nodes = {
    'S': [('A', 0.37), ('B', 0.46), ('E', 0.43)],
    'A': [('C', 0.37)],
    'B': [('D', 0.97)],
    'C': [('E', 0.42)],
    'D': [('Z', 0.38)],
    'E': [('Z', 0.55)],
}

aStarAlgo('S', 'Z')
```

Berdasarkan proses pencarian rute terpendek di atas, maka rute terpendek ditentukan berdasarkan titik awal (warna hijau) dan titik akhir (warna merah) yang diinginkan.

p-ISSN: 2986-3929
e-ISSN: 3032-1948

Hasil output program:

```
Rute Terpendek: ['S', 'E', 'Z']
Total Jarak Tempuh: 0.98
```

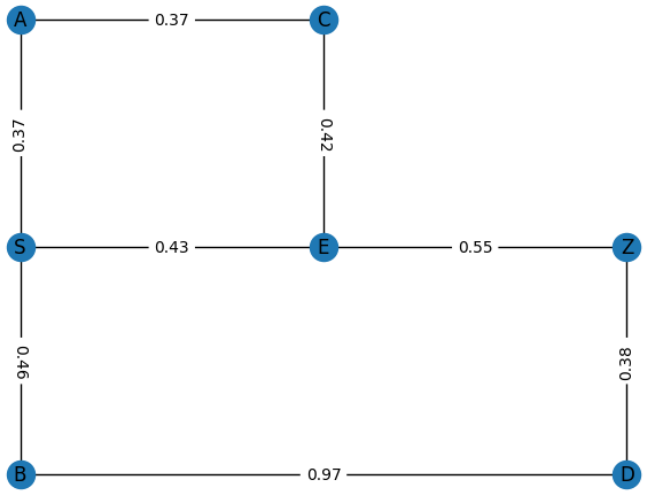
Hasil eksekusi program di atas ditentukan oleh proses pencarian rute terpendek menggunakan algoritma A*. Langkah selanjutnya adalah memberikan input agar dapat diimplementasikan dalam graf. Implementasi ini bertujuan untuk memudahkan visualisasi jalur rute pada peta dengan lebih detail.

Program implementasi graph dalam mencari rute terpendek:

```
import networkx as nx
import matplotlib.pyplot as plt

# Membuat objek graf
G = nx.Graph()

# Menambahkan node ke graf
G.add_node('S', pos=(0, 1))
G.add_node('A', pos=(0, 2))
G.add_node('B', pos=(0, 0))
G.add_node('C', pos=(2, 2))
G.add_node('D', pos=(4, 0))
G.add_node('E', pos=(2, 1))
G.add_node('Z', pos=(4, 1))

# Menambahkan edge (sambungan) antara node
G.add_edge('S', 'A', weight=0.37)
G.add_edge('S', 'B', weight=0.46)
G.add_edge('S', 'E', weight=0.43)
G.add_edge('A', 'C', weight=0.37)
G.add_edge('B', 'D', weight=0.97)
G.add_edge('C', 'E', weight=0.42)
G.add_edge('D', 'Z', weight=0.38)
G.add_edge('E', 'Z', weight=0.55)

# Mendapatkan posisi node
node_pos = nx.get_node_attributes(G, 'pos')

# Menampilkan graf dengan posisi node yang telah ditentukan
nx.draw(G, pos=node_pos, with_labels=True)

# Menampilkan jarak pada edge
labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos=node_pos, edge_labels=labels)

plt.show()
```

p-ISSN: 2986-3929
e-ISSN: 3032-1948

Hasil output program:



Gambar 2. Hasil Tampilan Graph

Hasil perhitungan lengkap rute terpendek dari setiap titik awal ke titik akhir dapat lihat dalam tabel di bawah berikut:

Tabel 3. Hasil perhitungan pengujian

No	Titik Awal	Titik Akhir	Rute	Jarak (Km)
1	S (Jl.Yos Sudarso)	Z (Jl.G.Obos)	S → A → C → E → Z	1.71
2	S (Jl.Yos Sudarso)	Z (Jl.G.Obos)	S → B → D → Z	1.81
3	S (Jl.Yos Sudarso)	Z (Jl.G.Obos)	S → E → Z	0.98

Berdasarkan hasil perhitungan pengujian, dapat disimpulkan bahwa program ini dibuat untuk menerima input berupa graph agar dapat menghitung rute terpendek dan menampilkan rute tersebut beserta jaraknya.

4. Kesimpulan

Dalam penelitian ini, sistem yang menggunakan algoritma A* untuk menentukan rute terpendek pada titik jalan telah berhasil dibangun dan diuji coba. Hasil dari sistem menunjukkan bahwa algoritma A* efektif digunakan untuk mencari rute terpendek dalam suatu peta. Setiap lokasi direpresentasikan sebagai simpul dalam graf, dengan jalur yang menghubungkan lokasi tersebut sebagai tepi graf. Bobot atau jarak setiap tepi digunakan untuk menghitung nilai heuristik. Proses pencarian rute dengan algoritma A* melibatkan pembentukan pohon pencarian dan penggunaan antrian prioritas. Dengan menggunakan algoritma ini, sistem dapat mengolah graf masukan, menghitung rute terpendek, dan menampilkan hasil rute terpendek beserta jaraknya. Sebagai contoh, jika titik awalnya adalah S (Jl. Yos Sudarso) dan titik akhirnya adalah Z (Jl. G. Obos), maka rute terpendek adalah 0,98 km melalui rute S → E → Z.

Daftar Pustaka

[1] Ida Bagus Gede Wahyu Antara Dalem, “Penerapan Algoritma A* (STAR) Menggunakan Graph Untuk Menghitung Jarak Terpendek,” *Jurnal RESISTOR (Rekayasa Sistem Komputer)*, vol. 1, no. 1, pp. 41–47, Apr. 2018, Accessed: Apr. 17, 2024. [Online]. Available: https://ejournal.instiki.ac.id/index.php/jurnalresistor/article/view/253

[2] Yusra Fernando, Muhammad Ativ Mutsaqov, and Dyah Ayu Megawaty, “Penerapan Algoritma A-STAR Pada Aplikasi Pencarian Lokasi Fotografi Di Bandar Lampung Berbasis Android,” *Jurnal Teknoinfo*, vol. 14, no. 1, pp. 27–34, Jan. 2020, doi: 10.33365/jti. v14i1.509.

[3] Rasita Natasya Br Sitepu and I Gusti Ngurah Anom Cahyadi Putra, “Penentuan Rute Terpendek Menggunakan Algoritma a Star (Studi Kasus: Distributor Barang),” *Jurnal Nasional Teknologi Informasi dan Aplikasnya*, vol. 1, no. 1, pp. 431–440, Nov. 2022, Accessed: Apr. 17, 2024. [Online].
Available: https://ojs.unud.ac.id/index.php/jnatia/article/view/92697

[4] Rizal Ahmad Fauzi and Rizal Rachman, “Implementasi Algoritma A* Menggunakan Graph Pada Aplikasi Route at Location Berbasis Web,” *eProsiding Sistem Informasi (POTENSI)*, vol. 2, no. 1, pp. 121–129, Jun. 2021, Accessed: Apr. 17, 2024. [Online].
Available: https://eprosiding.ars.ac.id/index.php/psi/article/view/369

[5] Susilawati, Robby Rizky, Sri Setiyowati, and Aghy Gilar Pratama, “Penerapan Metode A*Star Pada Pencarian Rute Tercepat Menuju Destinasi Wisata Cagar Budaya Menes Pandeglang,” *Geodika: Jurnal Kajian Ilmu dan Pendidikan Geografi*, vol. 4, no. 2, pp. 192–199, Dec. 2020, doi: 10.29408/geodika. v4i2.2754.

[6] I Putu Andi Wiratama Putra and I Gede Arta Wibawa, “Implementasi Algoritma A* (Star) dengan Graf untuk Menentukan Rute Terpendek Distributor Kopi,” *Jurnal Nasional Teknologi Informasi dan Aplikasnya*, vol. 1, no. 4, pp. 1053–1062, Aug. 2023, Accessed: Apr. 17, 2024. [Online].
Available: https://ojs.unud.ac.id/index.php/jnatia/article/view/102421

p-ISSN: 2986-3929
e-ISSN: 3032-1948

Halaman ini sengaja dibiarkan kosong