Desain Rute Optimal di Jalan Raya Jatinangor Menggunakan Algoritma Greedy pada Graf Berbobot

Mohammad Akmal Ramadan - 13522161
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13522161@std.stei.itb.ac.id

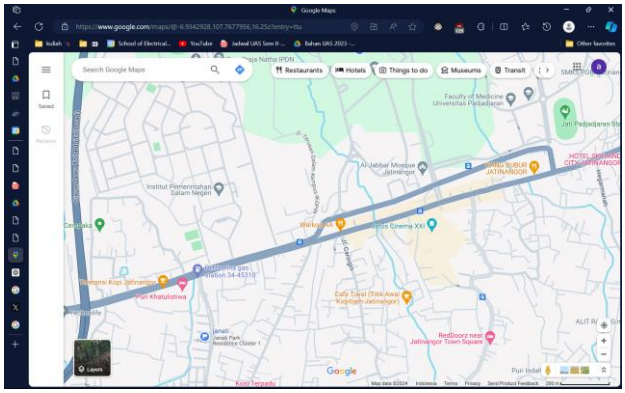
***Abstract*—Algoritma *greedy* adalah salah satu algoritma yang banyak digunakan dalam penyelesaian masalah pencarian rute optimal. Pada makalah ini, akan dicari rute optimal di Jalan Raya Jatinangor menggunakan algoritma Greedy. Data yang didapat untuk dijadikan bobot dalam graf adalah jarak antar tempat. Faktor-faktor lainnya yang diperhitungkan dalam pencarian rute optimal adalah titik belok dan putar balik, waktu tempuh, frekuensi melawan arah, dan kepadatan lalu lintas. Implementasi algoritma *greedy* dilakukan dalam bahasa Python. Hasil yang didapat adalah perlu adanya belokan dari arah IPDN menuju arah SPBU.**

Kata kunci—Algoritma greedy; rute optimal; Jalan Raya Jatinangor

I. PENDAHULUAN

Jatinangor adalah salah satu kawasan yang mengalami perkembangan pesat di Kabupaten Sumedang, terutama karena adanya beberapa institusi perguruan tinggi seperti Universitas Padjajaran dan Institut Teknologi Bandung yang memiliki peningkatan jumlah mahasiswa tiap tahunnya. Dengan meningkatnya populasi penduduk di Jatinangor, kepadatan jalan raya akan semakin meningkat juga.

Jalan raya menjadi faktor penting dalam suatu kawasan karena menjadi tempat atau fasilitas utama yang dipakai untuk melakukan mobilisasi khususnya di kawasan padat penduduk. Keefektivitasan desain dari sebuah jalan juga berkontribusi penting dalam kemajuan sebuah kawasan karena akan mempercepat waktu mobilisasi sehingga waktu produktif penduduk akan meningkat. Untuk menentukan keefektivitasan sebuah jalan dapat dilakukan sebuah analisis untuk membentuk sebuah data yang mencakup perlakuan dan rute yang paling sering dilalui oleh pengguna jalan.



Gambar 1. Jalan Raya Jatinangor (https://www.google.com/maps/@-6.9342928,107.7677956,16.25z?entry=ttu)

Jalan Raya Jatinangor merupakan salah satu jalan yang tingkat keefektivitasannya perlu ditingkatkan. Selama pengalaman peneliti berkuliah di Jatinangor selama dua tahun, desain Jalan Raya Jatinangor tidak efektif dibuktikan dengan banyaknya kemacetan yang disebabkan oleh letak belokan yang tidak tepat, pengendara motor yang melawan arah karena belokan sangat jauh, hingga kendaraan besar yang terpaksa untuk melakukan putar balik di jalan yang sempit.

Maka dari itu, pada makalah ini akan dilakukan sebuah analisis keefektivitasan Jalan Raya Jatinangor dan solusi desain yang diajukan oleh penulis melalui algoritma *Greedy*.

II. DASAR TEORI

A. *Algoritma*

Algoritma merupakan urutan langkah-langkah yang sistematis dalam memecahkan suatu masalah komputasional untuk mencapai suatu tujuan tertentu. Algoritma memainkan peran penting dalam bidang ilmu komputer dan matematika karena mereka menyediakan cara yang jelas dan logis untuk menyelesaikan masalah.

Dalam konteks ini, beberapa parameter utama yang menunjukkan seberapa bagus suatu algoritma adalah efisiensi dan kebutuhan sumber daya. Efisiensi algoritma diukur berdasarkan dua aspek utama:

1. **Kompleksitas Waktu (Time Complexity), T(n):** Kompleksitas waktu adalah jumlah tahapan komputasi (operasi) yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan nnn. Ini menunjukkan seberapa cepat suatu algoritma dapat memproses data masukan yang diberikan.

2. **Kompleksitas Ruang (Space Complexity), S(n):** Kompleksitas ruang adalah ruang memori yang dibutuhkan oleh algoritma sebagai fungsi dari ukuran nnn. Ini mengukur seberapa banyak memori yang diperlukan untuk menyimpan data yang digunakan selama eksekusi algoritma.

Untuk menganalisis efisiensi algoritma, digunakan tiga notasi asimptotik yang sering dipakai:

1. **O(f(n)) - Big O Notation:** Menyatakan batas atas kebutuhan waktu algoritma. Ini memberikan perkiraan terburuk dari waktu eksekusi algoritma sebagai fungsi dari ukuran masukan.

2. **Ω(g(n)) - Omega Notation:** Menyatakan batas bawah kebutuhan waktu algoritma. Ini memberikan perkiraan terbaik dari waktu eksekusi algoritma sebagai fungsi dari ukuran masukan.

3. **Θ(h(n)) - Theta Notation:** Menyatakan batas ketat kebutuhan waktu algoritma. Ini menunjukkan bahwa waktu eksekusi algoritma adalah proporsional terhadap fungsi h(n)h(n)h(n) dalam batas atas dan bawah.

Beberapa contoh bentuk strategi algoritma yang dipelajari di Mata Kuliah IF2211 Strategi Algoritma:

1. Algoritma *Brute-Force*
2. Algoritma *Greedy*
3. Algoritma *Divide and Conquer*
4. Algoritma *Decrease and Conquer*
5. Algoritma *Backtracking*
6. Algoritma *Branch and Bound*
7. *Dynamic Programming*

B. Algoritma Greedy

Algoritma greedy adalah salah satu teknik pemecahan masalah yang bekerja dengan membuat pilihan lokal terbaik pada setiap langkah dengan harapan bahwa solusi akhir yang dihasilkan adalah solusi optimal secara global. Algoritma ini sederhana dan intuitif, serta sering digunakan dalam berbagai aplikasi, seperti penjadwalan, pengurutan, dan optimasi jalur.

Prinsip dasar dari algoritma greedy adalah:

1. **Memilih Pilihan Terbaik Lokal:** Pada setiap langkah, algoritma greedy memilih opsi yang tampak paling baik saat itu, tanpa mempertimbangkan konsekuensi jangka panjang. Pilihan ini didasarkan pada kriteria tertentu yang mendefinisikan "terbaik" dalam konteks masalah yang sedang dipecahkan.

2. **Keputusan Irrevocable:** Setiap keputusan yang diambil tidak dapat diubah, dan algoritma terus bekerja hingga mencapai solusi akhir.

Contoh aplikasi dari algoritma greedy adalah dalam masalah **Knapsack**, di mana tujuannya adalah untuk mengisi tas dengan barang-barang yang memiliki nilai tertentu agar nilai totalnya maksimum tanpa melebihi kapasitas tas. Algoritma greedy akan memilih barang dengan rasio nilai-terhadap-berat tertinggi terlebih dahulu, dan terus menambahkan barang berikutnya berdasarkan kriteria ini hingga kapasitas tas terpenuhi.

Kelebihan Algoritma Greedy:

- Sederhana dan mudah diimplementasikan.
- Bekerja dengan cepat dan efisien dalam banyak kasus praktis.

Kekurangan Algoritma Greedy:

- Tidak selalu menghasilkan solusi optimal untuk semua masalah.
- Kinerja tergantung pada sifat masalah dan kriteria yang digunakan untuk membuat pilihan.

Secara keseluruhan, algoritma greedy sangat berguna dalam konteks tertentu di mana pilihan lokal yang terbaik dapat menghasilkan solusi yang mendekati optimal atau bahkan optimal. Namun, untuk masalah yang lebih kompleks, diperlukan analisis lebih lanjut untuk memastikan bahwa pendekatan greedy benar-benar memberikan solusi yang diinginkan.

C. *Graf Berbobot*

Graf berbobot (*weighted graph*) adalah salah satu jenis struktur data dalam teori graf di mana setiap sisi (edge) dari graf memiliki bobot atau nilai tertentu yang merepresentasikan biaya, jarak, atau metrik lainnya yang relevan. Graf berbobot dapat direpresentasikan sebagai G=(V,E,W)G = (V, E, W)G=(V,E,W), di mana:

- **V** adalah himpunan simpul (vertices) dalam graf.
- **E** adalah himpunan sisi (edges) yang menghubungkan simpul-simpul dalam graf.
- **W** adalah fungsi bobot yang menetapkan nilai atau bobot tertentu untuk setiap sisi dalam graf

Bobot pada graf berbobot dapat mencerminkan berbagai faktor tergantung pada konteks aplikasinya. Misalnya, dalam masalah jaringan transportasi, bobot bisa berupa jarak antara dua titik, waktu tempuh, atau biaya perjalanan.

Komponen dan Karakteristik Graf Berbobot:

1. **Simpul (Vertices):** Titik-titik dalam graf yang mewakili objek atau lokasi tertentu.
2. **Sisi (Edges):** Garis yang menghubungkan dua simpul dalam graf dan menunjukkan hubungan atau koneksi di antara mereka.
3. **Bobot (Weights):** Nilai yang terkait dengan setiap sisi dalam graf. Bobot ini digunakan untuk menentukan jalur optimal berdasarkan kriteria yang ditetapkan (misalnya, jarak terpendek atau waktu tercepat).

Contoh Aplikasi Graf Berbobot:

1. **Jaringan Transportasi:** Menentukan rute terpendek atau tercepat antara dua lokasi dalam peta jalan raya.
2. **Jaringan Komputer:** Mengoptimalkan jalur data dalam jaringan untuk meminimalkan latensi atau penggunaan bandwidth.
3. **Masalah Alokasi Sumber Daya:** Mengalokasikan sumber daya secara efisien berdasarkan biaya atau keuntungan.

Algoritma yang Digunakan pada Graf Berbobot:

1. **Algoritma Dijkstra:** Digunakan untuk menemukan jalur terpendek dari satu simpul ke semua simpul lainnya dalam graf berbobot non-negatif.
2. **Algoritma Bellman-Ford:** Digunakan untuk menemukan jalur terpendek dalam graf berbobot yang mungkin memiliki bobot negatif.
3. **Algoritma Prim dan Kruskal:** Digunakan untuk menemukan pohon rentang minimum (minimum spanning tree) dalam graf berbobot.

Dengan menggunakan graf berbobot, berbagai masalah optimasi dapat diselesaikan secara efisien dengan algoritma yang tepat. Pada konteks penelitian ini, graf berbobot akan digunakan untuk memodelkan jaringan jalan raya di Jatinangor dan algoritma greedy akan diterapkan untuk menemukan rute optimal berdasarkan bobot yang telah ditetapkan.

D. *Teori Jaringan Transportasi*

Teori jaringan transportasi adalah studi tentang bagaimana infrastruktur transportasi seperti jalan raya, rel kereta, dan rute udara diorganisasi dan dioperasikan untuk memfasilitasi pergerakan barang dan orang secara efisien. Teori ini mencakup berbagai aspek seperti perencanaan, desain, operasi, dan manajemen jaringan transportasi.

Komponen Jaringan Transportasi:

1. **Node (Simpul):** Titik-titik dalam jaringan yang mewakili lokasi penting seperti kota, persimpangan jalan, atau terminal transportasi.
2. **Link (Sisi):** Koneksi antara node yang mewakili jalan raya, jalur kereta, atau rute penerbangan.

Tujuan Teori Jaringan Transportasi:

1. **Optimasi Rute:** Menemukan rute terbaik berdasarkan berbagai kriteria seperti jarak, waktu, atau biaya.
2. **Analisis Kapasitas:** Menentukan kapasitas maksimum dari jaringan dan mengidentifikasi bottleneck atau hambatan dalam aliran lalu lintas.
3. **Manajemen Lalu Lintas:** Mengelola aliran lalu lintas untuk mengurangi kemacetan dan meningkatkan keselamatan.

Dengan memahami teori jaringan transportasi, peneliti dapat merancang sistem yang lebih efisien dan efektif untuk mengelola lalu lintas dan transportasi. Dalam konteks penelitian ini, teori jaringan transportasi akan digunakan untuk menganalisis dan mengoptimalkan rute perjalanan di Jatinangor.

III. ANALISIS DAN PEMBAHASAN

1. Analisis Keefektivitasan Jalan Raya Jatinangor

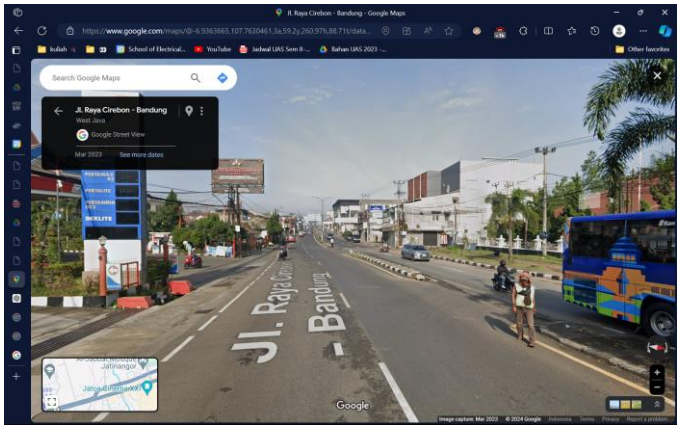
Untuk menganalisis keefektivitasan Jalan Raya Jatinangor, langkah pertama adalah mengumpulkan data perilaku pengguna jalan, seperti:

1. **Titik Sering Belok dan Putar Balik:** Mengidentifikasi titik-titik di mana pengendara sering kali belok atau putar balik.
2. **Frekuensi Melawan Arah:** Mencatat seberapa sering pengendara melawan arah di jalan tertentu.
3. **Kepadatan Lalu Lintas:** Mengukur tingkat kepadatan di berbagai segmen jalan pada waktu yang berbeda.
4. **Waktu Tempuh:** Menghitung waktu tempuh rata-rata dari berbagai titik awal ke tujuan di sepanjang Jalan Raya Jatinangor.

Data ini diperoleh peneliti dengan pengamatan secara langsung. Selanjutnya akan ditentukan titik-titik yang akan dijadikan simpul.

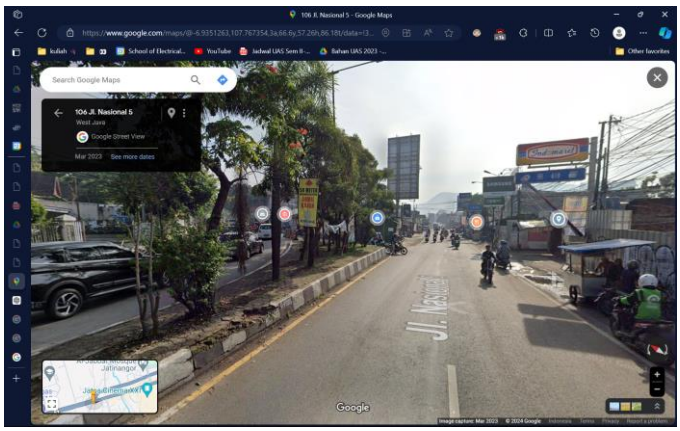
2. Penentuan Titik-titik Simpul

Titik-titik simpul yang digunakan dalam penelitian ini adalah ITB, UNPAD, Jatos, Jonas Photo, dan belokan-belokan yang rentan terjadi lawan arah dan dilewati oleh pengguna jalan.



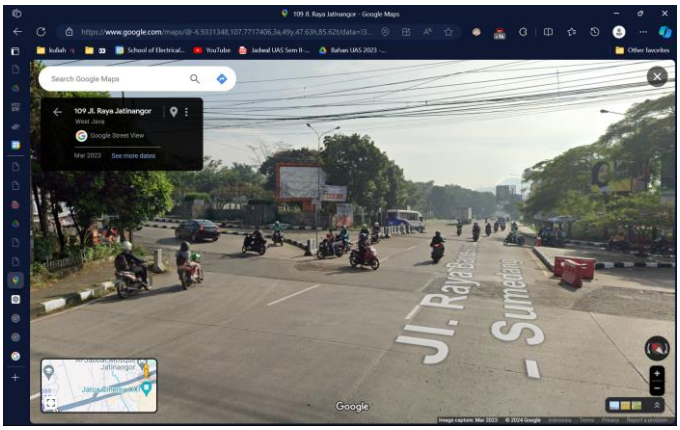
Gambar 2. Belokan SPBU (https://www.google.com/maps/@-6.9364818,107.7630232,3a,75y,314.73h,79.03t/data=!3m6!1e1!3m4!1sAPdP62YQ0yvSh5oCYnCSgQ!2e0!7i16384!8i8192?coh=205409&entry=ttu)

Pada gambar 2, merupakan belokan SPBU Pertamina, belokan ini sangat sering dilewati oleh para pengguna jalan, mulai dari kendaraan bermotor hingga kendaraan besar seperti bus dan truk. Belokan ini kerap dilanggar karena untuk belok langsung ke SPBU harus menggunakan belokan selanjutnya yang dipakai untuk putar balik dari lawan arah.



Gambar 3. Belokan IKOPIN (https://www.google.com/maps/@-6.9351263,107.767354,3a,75y,43.05h,80.51t/data=!3m6!1e1!3m4!1sjb1of9i0kcSaUs2GV-2wkA!2e0!7i16384!8i8192?coh=205409&entry=ttu)

Belokan IKOPIN menjadi salah satu belokan yang sering dilanggar oleh pengguna jalan, terutama pengguna jalan yang melawan arah agar bisa menggunakan belokan ini.



Gambar 4. Belokan Masjid Salman (https://www.google.com/maps/@-6.9331348,107.7717406,3a,75y,48.66h,85.2t/data=!3m6!1e1!3m4!1sVRft3wVUOyutOYvg_23ZSg!2e0!7i16384!8i8192?coh=205409&entry=ttu)

Belokan terakhir yang menjadi titik simpul adalah belokan Masjid Salman. Belokan ini menjadi belokan yang sering terjadi pelanggaran utamanya kendaraan bermotor yang melawan arah. Seperti pada Gambar 4, terlihat banyak motor yang melakukan lawan arah, hal ini sangat berbahaya karena jalan ini merupakan jalan yang seringkali kendaraan memiliki kecepatan yang tinggi.

Bahkan, belokan ini pernah terjadi kecelakaan yang melibatkan angkot dan motor yang ingin melawan arah, yang membuat salah satu mahasiswa meninggal di tempat.

Dari 3 belokan ini, akan dijadikan simpul dan pembobotan berat yang sesuai dengan kondisi kemacetan, waktu tempuh, frekuensi melawan arah dan frekuensi belokan dan putar balik.

3. Pembentukan Model Graf Berbobot

Graf berbobot merepresentasikan jaringan jalan di Jatinangor dengan setiap simpul mewakili persimpangan atau titik penting, dan setiap sisi mewakili jalan yang menghubungkan dua simpul dengan bobot tertentu (misalnya, jarak, waktu tempuh, atau frekuensi kendaraan melawan arah). Berikut adalah model graf berbobot berdasarkan analisis perilaku pengguna jalan di Jatinangor:

```
graph = {
    'ITB': {'Belokan Masjid Salman': 450, 'Belokan IKOPIN': 190},
    'Belokan Masjid Salman': {'ITB': 450, 'UNPAD': 850},
    'UNPAD': {'Belokan Masjid Salman': 850, 'Jatos': 2490},
    'Jatos': {'UNPAD': 2490, 'Belokan SPBU': 480},
    'IPDN': {'Belokan SPBU': 250, 'Belokan IKOPIN': 400},
    'Belokan SPBU': {'Jatos': 480, 'Belokan IKOPIN': 370},
    'Belokan IKOPIN': {'Belokan SPBU': 370, 'ITB': 190}
}
```

Gambar 5. Pembobotan Graph

Pada graph ini, simpul sebagai tempat dan belokan, yaitu ITB, belokan Masjid Salman, belokan IKOPIN, UNPAD, IPDN, Jatos, dan belokan SPBU. Dengan sisi-sisi adalah jarak antar simpul dalam meter.

4. Pembentukan Algoritma Greedy

```python
def greedy_shortest_path(graph, start, end):
    shortest_paths = {start: (None, 0)}
    current_node = start
    visited = set()

    while current_node != end:
        visited.add(current_node)
        destinations = graph[current_node]
        weight_to_current_node = shortest_paths[current_node][1]

        for next_node, weight in destinations.items():
            weight = weight_to_current_node + weight
            if next_node not in shortest_paths:
                shortest_paths[next_node] = (current_node, weight)
            else:
                current_shortest_weight = shortest_paths[next_node][1]
                if current_shortest_weight > weight:
                    shortest_paths[next_node] = (current_node, weight)

        next_destinations = {node: shortest_paths[node] for node in shortest_paths if node not i
        visited}
        if not next_destinations:
            return "Rute tidak ditemukan"
        current_node = min(next_destinations, key=lambda k: next_destinations[k][1])

    path = []
    while current_node is not None:
        path.append(current_node)
        next_node = shortest_paths[current_node][0]
        current_node = next_node
    path = path[::-1]
    return path
```

Gambar 6. Algoritma *Greedy*

Implementasi menggunakan Bahasa python, algoritma terlebih dahulu menginisialisasi sebuah dictionary bernama shortest_paths dengan titik awal sebagai kunci dan sebuah tuple yang berisi nilai None (untuk menunjukkan tidak ada titik sebelumnya) dan 0 (bobot/jarak ke titik saat ini) sebagai nilai. Selain itu, algoritma menetapkan current_node ke titik awal dan membuat sebuah set kosong visited untuk menyimpan titik-titik yang telah dikunjungi.

Algoritma kemudian memasuki loop utama yang terus berlanjut sampai current_node mencapai titik tujuan. Dalam setiap iterasi loop ini, algoritma pertama-tama menandai current_node sebagai telah dikunjungi dengan menambahkannya ke dalam set visited. Kemudian, algoritma mendapatkan semua tetangga dari current_node dari graf dan menyimpan bobot/jarak dari titik awal ke current_node dalam variabel weight_to_current_node.

Selanjutnya, algoritma menghitung bobot total untuk setiap tetangga dari current_node. Untuk setiap tetangga (next_node), bobot total dihitung dengan menjumlahkan weight_to_current_node dan bobot dari current_node ke next_node. Jika next_node belum ada di shortest_paths, algoritma menambahkannya dengan current_node sebagai titik sebelumnya dan bobot total yang dihitung. Namun, jika next_node sudah ada di shortest_paths, algoritma membandingkan bobot baru yang dihitung dengan bobot yang sudah ada. Jika bobot baru lebih kecil, algoritma memperbarui nilai di shortest_paths dengan bobot yang lebih kecil tersebut.

Setelah semua tetangga dari current_node diproses, algoritma memilih titik berikutnya yang belum dikunjungi dengan bobot/jarak terkecil. Ini dilakukan dengan membuat dictionary next_destinations yang berisi semua titik di shortest_paths yang belum dikunjungi dan kemudian memilih titik dengan bobot terkecil. Jika tidak ada titik yang tersisa untuk dikunjungi, algoritma mengembalikan pesan "Rute tidak ditemukan", menandakan bahwa tidak ada jalur yang tersedia dari titik awal ke titik tujuan.

Jika ada titik yang tersisa untuk dikunjungi, algoritma memperbarui current_node ke titik dengan bobot/jarak terkecil dan melanjutkan ke iterasi berikutnya dalam loop.

Setelah current_node mencapai titik tujuan, algoritma membangun rute terpendek dengan melacak kembali dari titik tujuan ke titik awal menggunakan shortest_paths. Algoritma menambahkan setiap titik ke dalam list path dan kemudian membalikkan urutan list tersebut untuk mendapatkan rute dari titik awal ke titik tujuan. Akhirnya, algoritma mengembalikan list path yang berisi rute terpendek yang ditemukan.

5. Pencarian Rute Terpendek

Setelah mengimplementasi algoritma *greedy*, selanjutnya akan dicari rute terpendek dari satu tempat ke tempat lainnya, dengan fungsi berikut:

```python
# Fungsi untuk iterasi jarak terdekat dari semua pasangan titik
def find_all_shortest_paths(graph):
    nodes = list(graph.keys())
    all_paths = {}
    for i in range(len(nodes)):
        for j in range(len(nodes)):
            if i != j:
                start = nodes[i]
                end = nodes[j]
                path = greedy_shortest_path(graph, start, end)
                all_paths[(start, end)] = path
    return all_paths

# Menjalankan fungsi untuk mendapatkan semua jarak terdekat dari setiap pasangan titik
all_paths = find_all_shortest_paths(graph)
for (start, end), path in all_paths.items():
    print(f"Rute terpendek dari {start} ke {end} adalah: {path}")
```

Gambar 7. Fungsi untuk mencari rute terpendek

Dengan menjalankan fungsi mencari rute terpendek dari algoritma *greedy* yang sudah dibuat, didapat hasil seperti ini:

```
Rute terpendek dari UNPAD ke IPDN adalah: Rute tidak ditemukan
Rute terpendek dari UNPAD ke Belokan SPBU adalah: ['UNPAD', 'Belokan Masjid Salman', 'ITB', 'Bel
okan IKOPIN', 'Belokan SPBU']
Rute terpendek dari UNPAD ke Belokan IKOPIN adalah: ['UNPAD', 'Belokan Masjid Salman', 'ITB', 'B
elokan IKOPIN']
Rute terpendek dari Jatos ke ITB adalah: ['Jatos', 'Belokan SPBU', 'Belokan IKOPIN', 'ITB']
Rute terpendek dari Jatos ke Belokan Masjid Salman adalah: ['Jatos', 'Belokan SPBU', 'Belokan IK
OPIN', 'ITB', 'Belokan Masjid Salman']
Rute terpendek dari Jatos ke UNPAD adalah: ['Jatos', 'Belokan SPBU', 'Belokan IKOPIN', 'ITB', 'B
elokan Masjid Salman', 'UNPAD']
Rute terpendek dari Jatos ke IPDN adalah: Rute tidak ditemukan
Rute terpendek dari Jatos ke Belokan SPBU adalah: ['Jatos', 'Belokan SPBU']
Rute terpendek dari Jatos ke Belokan IKOPIN adalah: ['Jatos', 'Belokan SPBU', 'Belokan IKOPIN']
Rute terpendek dari IPDN ke ITB adalah: ['IPDN', 'Belokan IKOPIN', 'ITB']
Rute terpendek dari IPDN ke Belokan Masjid Salman adalah: ['IPDN', 'Belokan IKOPIN', 'ITB', 'Bel
okan Masjid Salman']
Rute terpendek dari IPDN ke UNPAD adalah: ['IPDN', 'Belokan IKOPIN', 'ITB', 'Belokan Masjid Salm
an', 'UNPAD']
Rute terpendek dari IPDN ke Jatos adalah: ['IPDN', 'Belokan IKOPIN', 'Belokan SPBU', 'Jatos']
Rute terpendek dari IPDN ke Belokan SPBU adalah: ['IPDN', 'Belokan IKOPIN', 'Belokan SPBU']
Rute terpendek dari IPDN ke Belokan IKOPIN adalah: ['IPDN', 'Belokan IKOPIN']
Rute terpendek dari Belokan SPBU ke ITB adalah: ['Belokan SPBU', 'Belokan IKOPIN', 'ITB']
Rute terpendek dari Belokan SPBU ke Belokan Masjid Salman adalah: ['Belokan SPBU', 'Belokan IKOP
IN', 'ITB', 'Belokan Masjid Salman']
Rute terpendek dari Belokan SPBU ke UNPAD adalah: ['Belokan SPBU', 'Belokan IKOPIN', 'ITB', 'Bel
okan Masjid Salman', 'UNPAD']
Rute terpendek dari Belokan SPBU ke Jatos adalah: ['Belokan SPBU', 'Jatos']
Rute terpendek dari Belokan SPBU ke IPDN adalah: Rute tidak ditemukan
Rute terpendek dari Belokan SPBU ke Belokan IKOPIN adalah: ['Belokan SPBU', 'Belokan IKOPIN']
Rute terpendek dari Belokan IKOPIN ke ITB adalah: ['Belokan IKOPIN', 'ITB']
Rute terpendek dari Belokan IKOPIN ke Belokan Masjid Salman adalah: ['Belokan IKOPIN', 'ITB', 'B
elokan Masjid Salman']
Rute terpendek dari Belokan IKOPIN ke UNPAD adalah: ['Belokan IKOPIN', 'ITB', 'Belokan Masjid Sa
lman', 'UNPAD']
```

Gambar 8. Hasil rute terpendek

Dari hasil di Gambar 8. Dibutuhkan beberapa filterisasi rute yang tidak mungkin, seperti rute yang melawan arah, rute yang membutuhkan jembatan untuk dibuat, dan lain-lain. Setelah melakukan filtereisasi, didapat satu hasil yang paling mungkin untuk direalisasikan, yaitu:
Rute terpendek dari IPDN ke Belokan SPBU adalah: ['IPDN', 'Belokan IKOPIN', 'Belokan SPBU']

Hasil ini berbeda dengan kondisi rute sekarang, yang mana jika ingin ke belokan SPBU dari IPDN, rute akan seperti IPDN – Belokan IKOPIN – ITB – Belokan Masjid Salman – Jatos – Belokan SPBU.

Hasil rute terpendek melalui algoritma *greedy* jauh lebih baik dari kondisi sekarang, di mana hanya membutuhkan 2 simpul untuk menuju IPDN ke belokan SPBU, yang sebelumnya membutuhkan untuk mengunjungi 5 simpul.

Maka dari itu, dibutuhkan belokan yang membuat pengguna jalan dapat melakukan putar balik di belokan IKOPIN dari arah IPDN, belokan ini ada pada Gambar 3.

IV. KESIMPULAN

Dari hasil analisis yang disajikan pada Gambar 8, dapat disimpulkan bahwa diperlukan beberapa langkah filterisasi untuk menghilangkan rute-rute yang tidak memungkinkan, seperti rute yang melawan arah, rute yang memerlukan pembangunan jembatan, dan lain sebagainya. Setelah melakukan filterisasi, ditemukan satu rute yang paling mungkin untuk direalisasikan, yaitu:

Rute terpendek dari IPDN ke Belokan SPBU adalah: ['IPDN', 'Belokan IKOPIN', 'Belokan SPBU'].

Hasil ini berbeda dengan kondisi rute saat ini, di mana jika pengguna jalan ingin menuju Belokan SPBU dari IPDN, rutenya adalah: IPDN – Belokan IKOPIN – ITB – Belokan Masjid Salman – Jatos – Belokan SPBU.

Dari perbandingan ini, terlihat bahwa hasil rute terpendek melalui algoritma greedy jauh lebih efisien dibandingkan dengan kondisi rute saat ini. Rute optimal yang dihasilkan hanya membutuhkan 2 simpul untuk mencapai tujuan dari IPDN ke Belokan SPBU, sedangkan rute yang ada saat ini membutuhkan 5 simpul.

Oleh karena itu, diperlukan penambahan belokan yang memungkinkan pengguna jalan untuk melakukan putar balik di Belokan IKOPIN dari arah IPDN. Penambahan belokan ini diilustrasikan pada Gambar 3.

REFERENSI

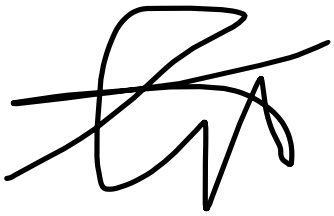
[1] Munir, Rinaldi. 2021. “Algoritma greedy (Bagian 1)” https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf (Diakses 12 Juni 2024)

[2] Munir, Rinaldi. 2024. “Pengantar Strategi Algoritma” https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Pengantar-Strategi-Algoritma-(2024).pdf (Diakses 12 Juni 2024)

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, 12 Juni 2024



Mohammad Akmal Ramadan - 13522161