BAB II
TINJAUAN PUSTAKA DAN DASAR TEORI

2.1 Tinjauan Pustaka

Beberapa penelitian yang pernah dilakukan sebelumnya terkait dengan implementasi algoritma Dijkstra untuk pencarian rute terdekat antara lain :

Penelitian yang dilakukan oleh Muhammad Ikhsan Raharjo pada tahun 2019 berupa skripsi yang berjudul "Aplikasi Pencarian Rute Terdekat Wisata Kota Yogyakarta Menggunakan Algoritma Dijkstra". Aplikasi hasil penelitian ini dibangun dengan bahasa pemrograman Java dan dikembangkan menggunakan Android Studio sehingga aplikasi berjalan pada sistem operasi Android. Aplikasi menggunakan algoritma Dijkstra untuk menentukan rute terdekat dari lokasi pengguna munuju ke lokasi wisata di Kota Yogyakarta.

Penelitian yang dilakukan oleh Taiwo Gabriel Omomule, Basit Lolade Durodola, dan Segun Michael Orimoloye pada tahun 2019 yang berjudul "*Shortest Route Analysis for Road Accident Emergency using Dijkstra Algorithm and Fuzzy Logic*" dalam jurnal International Journal of Computer Science and Mobile Computing Volume 8. Hasil penelitian ini adalah sebuah aplikasi bernama "FindNearbyHospital" berbasis sistem operasi Android yang dibangun dengan bahasa pemrograman Java dan dikembangkan menggunakan Android Studio. Aplikasi tersebut menggunakan metode algoritma Dijkstra untuk mencari daftar rumah sakit terdekat dengan lokasi kecelakaan di jalan. Selanjutnya hasil daftar tersebut diseleksi melewati *Fuzzy Logic* dengan parameter tertentu yang telah ditentukan. Hasil akhir aplikasi adalah dapat menampilkan rekomendasi rumah sakit terdekat yang cocok untuk menangani jenis kecelakaan tersebut.

Penelitian yang dilakukan oleh Rakhmat Dedi Gunawan, Riduwan Napianto, Rohmat Indra Borman, dan Irma Hanifah pada tahun 2019 yang berjudul "*Implementation of Dijkstra's Algorithm in Determining the Shortest Path (Case Study: Specialist Doctor Search in Bandar Lampung)*" dalam jurnal International Journal Information System and Computer Science Volume 3. Penelitian ini

menghasilkan aplikasi yang berjalan pada basis sistem operasi Android. Aplikasi tersebut bertujuan untuk menerapkan algoritma Dijkstra sebagai metode pencarian rute terdekat ke rumah sakit yang memiliki dokter spesialis dan menampilkan hasil rutenya pada peta digital.

Penelitian yang dilakukan oleh Nita Dewi Setyawati pada tahun 2020 berupa skripsi yang berjudul "Algoritma Dijkstra untuk Pencarian Jarak Terdekat Tempat Wisata di Nabire Berbasis Peta Digital". Penelitian ini menghasilkan aplikasi yang berbasis *mobile web* yang menggunakan *Semantic UI* sebagai *framework* sistem dan OpenStreetMap sebagai peta digitalnya. Aplikasi tersebut mengimplementasikan algoritma Dijkstra sebagai metode untuk mencari jarak terdekat tempat-tempat wisata di Kabupaten Nabire.

Penelitian yang dilakukan oleh Muhammad Ilham Suwahyu pada tahun 2023 berupa skripsi yang berjudul "Implementasi Algoritma Dijkstra dalam Menentukan Jalur Terpendek Destinasi Wisata Kabupaten Tulungagung". Penelitian yang dilakukan adalah menganalisis dan menghitung masalah jarak terpendek destinasi wisata di Tulungagung menggunakan metode algoritma Dijkstra. Penelitian tersebut menggunakan titik persimpangan dan tikungan yang ada di Tulungagung sebagai titik awal dan titik lain pada graf dan titik destinasi wisata di Tulungagung sebagai titik tujuan pada graf perhitungan algoritma Dijkstra.

Adapun perbedaan pada penelitian yang dibuat ini dengan penelitian yang sudah dicantumkan di atas diantaranya ialah pembuatan aplikasi untuk implementasi algoritma Dijkstra untuk pencarian rute terdekat menggunakan *framework* bernama Laravel. Digunakan paket Laravel Livewire yang memungkinkan penggunaan *framework* Laravel sebagai *full-stack framework* sehingga aplikasi yang dibangun menjadi lebih dinamis. Sebagai penampil peta digital serta fungsi peta seperti menampilkan *marker* dan *polygon* menggunakan layanan Mapbox. Digunakan bantuan *geolocation API* guna mengambil lokasi geografis pengguna pada saat itu untuk menjadi salah satu cara masukan lokasi awal oleh pengguna sistem.

Tabel 2.1 Tinjauan Pustaka

No	Peneliti	Topik	Metode	Hasil
1	Muhammad Ikhsan Raharjo (2019)	Aplikasi Pencarian Rute Terdekat Wisata Kota Yogyakarta Menggunakan Algoritma Dijkstra	Dijkstra	Aplikasi berbasis Android untuk mencari rute terdekat dari lokasi pengguna ke lokasi wisata di Kota Yogyakarta
2	Taiwo Gabriel Omomule, Basit Lolade Durodola, dan Segun Michael Orimoloye (2019)	*Shortest Route Analysis for Road Accident Emergency using Dijkstra Algorithm and Fuzzy Logic*	Dijkstra, Fuzzy Logic	Aplikasi bernama FindNearbyHospital untuk mencari rute terdekat ke rumah sakit dan juga merekomendasikan rumah sakit yang cocok ketika terjadi kecelakaan di jalan
3	Rakhmat Dedi Gunawan, Riduwan Napianto, Rohmat Indra Borman, dan Irma Hanifah (2019)	*Implementation of Dijkstra's Algorithm in Determining the Shortest Path (Case Study: Specialist Doctor Search in Bandar Lampung)*	Dijkstra	Aplikasi berbasis Android dengan tujuan untuk mencari rute terdekat ke rumah sakit yang memiliki dokter spesialis
4	Nita Dewi Setyawati (2020)	Algoritma Dijkstra untuk Pencarian Jarak Terdekat Tempat Wisata di Nabire Berbasis Peta Digital	Dijkstra	Aplikasi berbasis *mobile web* untuk mencari jarak terdekat tempat-tempat wisata di Kabupaten Nabire

Tabel 2.1 (Lanjutan)

No	Peneliti	Topik	Metode	Hasil
5	Muhammad Ilham Suwahyu (2023)	Implementasi Algoritma Dijkstra dalam Menentukan Jalur Terpendek Destinasi Wisata Kabupaten Tulungagung	Dijkstra	Hasil perhitungan dari implementasi algoritma Dijkstra untuk memecahkan masalah jalur terpendek destinasi wisata Kabupaten Tulungagung

2.2 Dasar Teori

2.2.1 Definisi Graf

Suatu graf merupakan kumpulan dari simpul yang terhubung oleh sisi. Biasanya graf digambarkan dengan titik-titik (melambangkan simpul) yang saling dihubungkan oleh garis (melambangkan sisi). Graf juga dapat ditulis dengan notasi sebagai G=<V,E> yang berarti graf G terdiri atas himpunan V yang berisi kumpulan simpul dan himpunan E yang berisi kumpulan sisi. Simpul dapat dinotasikan dengan huruf atau bilangan atau keduanya, sedangkan sisi yang menghubungkan simpul *u* dan *v* dinyatakan dengan pasangan (*u*, *v*) atau dengan *e1*, *e2*, *e3*, dan seterusnya (Fauzi, 2011).

Beberapa istilah yang biasa terkait dengan graf antara lain :

1. Simpul yang biasa disebut juga dengan *node* atau *vertex*.
2. Sisi yang biasa disebut dengan *edge*.
3. Tetangga atau biasa disebut dengan *adjacent*.

Graf diklarifikasikan menjadi beberapa jenis. Berdasarkan orientasi arah pada sisinya maka graf dibagi menjadi dua jenis yaitu :

1. Graf berarah (*directed graph*) dengan setiap sisi pada graf memiliki orientasi arah (biasanya ditunjukkan dengan sisi memiliki petunjuk arah ke salah satu simpul) yang menyebabkan urutan pasangan simpul menjadi diperhatikan. Pada graf berarah sisi (*u,v*) dan sisi (*v,u*) merupakan sisi yang berbeda.

2. Graf tak berarah (*undirected graph*) dengan setiap sisi graf tidak memiliki orientasi arah. Pada graf tak berarah sisi (*u,v*) dan sisi (*v,u*) merupakan sisi yang sama.

Berdasarkan bobotnya graf dibagi menjadi dua yaitu :

1. Graf berbobot (*weighted graph*) merupakan graf yang setiap sisi graf memiliki suatu nilai atau bobot tertentu. Graf berbobot juga memiliki dua jenis yakni graf yang berbobot positf dan graf yang berbobot negatif.
2. Graf tak berbobot (*unweighted graph*) merupakan graf yang setiap sisi graf tidak memiliki nilai.

2.2.2 Rute Terdekat

Rute terdekat merupakan adalah sebuah malasah dalam pencarian rute dengan jarak yang paling dekat dari lokasi awal menuju ke lokasi tujuan. Masalah rute terdekat dapat dipecahkan dengan menggunakan graf berbobot. Graf merupakan kumpulan simpul (*vertex*) yang saling terhubung melalui kumpulan sisi (*edge*). Graf berbobot adalah sebuah graf dengan setiap garis atau sisi pada graf tersebut memiliki bobot atau biaya. Rute terdekat dari simpul S menuju ke simpul T merupakan sebuah rute terarah dari simpul S menuju simpul T yang memiliki bobot paling rendah dibandingkan bobot pada rute yang lain (Sedgewick & Wayne, 2011).

2.2.3 Algoritma Dijkstra

Algoritma Dijkstra merupakan salah satu algoritma yang dapat memecahkan permasalahan rute terdekat. Algoritma Dijkstra adalah algoritma yang mencari rute terdekat dari suatu simpul atau juga dikenal dengan simpul sumber menuju ke semua simpul yang ada pada graf. Algoritma ini menghasilkan daftar rute terdekat dari semua kemungkinan rute yang ada dan diawali dari simpul sumber atau simpul awalan menuju ke semua simpul yang ada di graf (Brilliant, 2016).

Algoritma Dijkstra mencari rute terdekatnya dengan membuat sebuah daftar dari semua simpul yang memiliki bobot paling kecil dari simpul sumber pada graf. Graf yang digunakan pada algoritma Dijkstra memiliki :

1. simpul, atau *node* atau *vertex*, yang dinyatakan dengan *u* atau *v*.
2. sisi yang menghubungkan dua simpul yakni simpul *u* dan simpul *v* dinyatakan dengan (*u*, *v*), dan bobot dari sisi tersebut dinyatakan dengan *w(u,v)*.

Algoritma Dijkstra dalam menjalankan fungsi untuk mencari rute terdekat menggunakan langkah penyelesaian yang diuraikan sebagai berikut.

a. Langkah awal.
    1) Membuat *array* atau larik *distances* yang merupakan sekumpulan jarak dari simpul sumber *s* menuju ke setiap simpul yang ada pada graf. Pada awal perhitungan dinyatakan *distances(s)=0* yakni jarak dari simpul sumber menuju simpul sumber adalah 0. Kemudian *distances(v)*= ∞ (tak hingga) sebagai penanda bahwa jarak dari simpul sumber menuju ke setiap simpul lain *v* adalah tak hingga. Ketika proses perhitungan algoritma berjalan jarak simpul sumber *s* menuju ke semua simpul lain *v* pada graf akan dihitung kembali dan hasil akhir perhitungan akan ditemukan setelah jarak terdekat ke simpul lain *v* telah ditemukan.
    2) Membuat larik *Q* yang berisi daftar semua simpul pada graf. Pada akhir perhitungan algoritma, larik *Q* akan menjadi kosong.
    3) Larik *V* merupakan larik kosong yang menyatakan kumpulan simpul yang telah dikunjungi. Pada akhir perhitungan algoritma, larik *V* berisi semua simpul yang ada pada graf.
b. Langkah repetisi
    1) Selama larik *Q* tidak kosong : ambil simpul *v* sebagai simpul sekarang, yang tidak berada pada larik *V*, dari larik *Q* yang memiliki *distances(v)* paling kecil. Pada repetisi perhitungan pertama, simpul sumber *s* dipilih sebagai simpul awalan perhitungan karena memiliki nilai *distances(s)* = 0. Pada repitisi selanjutnya simpul dengan nilai pada *distances* yang paling kecil yang dipilih sebagai simpul sekarang.
    2) Tambahkan simpul *v* ke larik *V* dan hapus simpul *v* dari larik *Q* untuk mengindikasikan bahwa simpul *v* telah dikunjungi.

3) Perbarui nilai *distances* dari tetangga simpul sekarang *v*. Setiap simpul tetangga dari simpul *v* (dinyatakan simpul *u*) :

   a) Apabila *distances(v) + weight(v, u) < distances(u)* maka perbarui nilai *distances(u)* dengan nilai terkecil yang baru saja dihitung tersebut.

      Penjelasan : Hitung total jarak dari jarak simpul sumber *s* menuju simpul *v* sekarang ditambah dengan bobot dari sisi yang menghubungkan simpul v dan simpul *u*. Lalu bandingkan dengan nilai jarak dari simpul sumber *s* menuju simpul *u* apakah lebih kecil atau tidak.

   b) Jika hasilnya adalah lebih kecil, maka ubah nilai jarak terdekat sebelumnya *distances(u)* dengan nilai jarak terdekat yang baru tersebut. Jika tidak, maka tidak ada perubahan pada *distances(u)*.

c. Pada akhir perhitungan larik *distances* berisi semua daftar simpul yang sudah dikunjungi algoritma dan didapatkan daftar jarak paling kecil dari simpul sumber *s* menuju ke setiap simpul yang ada pada graf.

Berikut pada gambar 2.1 adalah *pseudocode* dari algoritma Dijkstra (Brilliant, 2016*)*:

```
function Dijkstra(Graph, source):
    // Distance from source to source is set to 0
    dist[source]  := 0
    for each vertex v in Graph:      // Initializations
        if v ≠ source
        // Unknown distance function from source to each node
set to infinity
            dist[v]  := infinity
        add v to Q       // All nodes initially in Q

        while Q is not empty:  // The main loop
            // In the 1st run, this vertex is the source node
            v := vertex in Q with min dist[v]
            remove v from Q

            // where neighbor u has not yet been removed from Q
            for each neighbor u of v:
                alt := dist[v] + length(v, u)
                // A shorter path to u has been found
                if alt < dist[u]:
                    dist[u]  := alt    // Update distance of u

    return dist[]
end function
```

Gambar 2.1 Pseudocode Algoritma Dijkstra

2.2.4 Laravel

Laravel adalah salah satu *framework* untuk pengembangan web yang berbasis PHP dan bersifat gratis untuk digunakan siapa saja. Laravel menawarkan banyak fitur untuk membantu pengembangan web seperti *dependency injection*, antrian, pengujian *unit*, *real-time events* dan lain-lain (Laravel, 2023). Laravel memungkinkan untuk dijadikan *full stack framework* yang dapat digunakan untuk mengatur dan memproses permintaan ke aplikasi web atau *backend* dan juga memproses penampilan web dengan *blade templates* dari Laravel atau *frontend* pada satu lingkungan aplikasi web yang sama.

2.2.5 Laravel Livewire

Laravel Livewire merupakan *framework* untuk mengembangkan *frontend* dengan lingkungan Laravel. Paket Livewire memungkinkan *framework* Laravel

untuk berfungsi sebagai *full stack framework* sehingga aplikasi yang dikembangkan dengan Laravel dapat menikmati keuntungan antarmuka pengguna dari *JavaScript* seperti aplikasi halaman tunggal.

2.2.6 PHP

PHP atau singkatan dari "*PHP: Hypertext Preprocessor*" adalah bahasa skrip yang ditanam pada *HTML*, bertujuan untuk memungkinkan pengembang web melakukan penulisan halaman web dinamis yang lebih cepat (Bakken *et al*., 2001). PHP dikembangkan sebagai bahasa pemrograman yang berfokus pada pengembangan web, diciptakan oleh Rasmus Lerdorf pada tahun 1994 dan kemudian dirilis pada tahun 1995. PHP berfokus pada proses pengembangan web yang bersifat *server-side* dan kode PHP ini dieksekusi oleh *web server*.

2.2.7 Javascript

Javascript adalah bahasa scripiting serbaguna yang digunakan untuk aplikasi web interaktif, mendukung baik pengembangan *client-side* dan *server-side*, serta memiliki integrasi yang tinggi dengan *HTML* dan *CSS* (GeeksforGeeks, 2020). *JavaScript* merupakan bahasa *scripting* berorientasi objek yang berjalan pada *browser* pengguna tanpa perlu berkomunikasi dengan *server*. Bahasa pemrograman ini dapat digunakan untuk berbagai hal dari yang simpel sampai kompleks seperti membuat animasi dan merespon aksi dari pengguna secara *real-time*.

2.2.8 Mapbox

Mapbox merupakan salah satu penyedia terbesar dalam pelayanan desain peta digital untuk situs web dan aplikasi seluler. Mapbox menggunakan *OpenStreetMap* sebagai basis petanya dan memfasilitasi pengembang untuk mendesain peta tersebut seperti menambahkan *markers*, *lines*, *polylines*, dan *polygons*, serta juga memfasilitasi penambahan lapisan peta dari sumber luar seperti dari *GeoJSON* (Bulatovych, 2020).

2.2.9 Haversine Formula

Metode h*aversine formula* merupakan salah satu rumus untuk menghitung jarak antara dua titik pada permukaan bumi, yang memperhitungkan bahwa bumi memiliki permukaan melengkung seperti bola (Sari, 2023). Seperti contoh apabila ingin mengetahui jarak dari lokasi A ke lokasi B, maka terlebih dahulu dicari garis bujur (*longitude*) dan garis lintang (*latitude*) dari dua lokasi tersebut, kemudian dengan menggunakan *haversine formula* maka dapat ditemukanlah jarak dari dua lokasi tersebut.

Berikut adalah rumus dari *haversine formula* :

$$\Delta lat = lat2 - lat1$$

$$\Delta lon = lon2 - lon1$$

$$a = sin^2\left(\frac{\Delta lat}{2}\right) + \cos(lat1) \times \cos(lat2) \times sin^2\left(\frac{\Delta lon}{2}\right)$$

$$c = 2.atan2(\sqrt{a}, \sqrt{1-a}$$

$$d = R.c$$

Keterangan :

R = jari-jari bumi yaitu sebesar 6371 kilometer

Δlat = perbedaan nilai *latitude* titik dua dengan titik satu

Δlon = perbedaan nilai *longitude* titik dua dengan titik satu

c = kalkulasi perpotongan sumbu

d = jarak dalam satuan kilometer

1 derajat = 0.0174532925 radian

2.2.10 Geolocation API

Geolokasi adalah metode untuk mengetahui keberadaan lokasi suatu objek atau seseorang dengan menggunakan teknologi internet. *Geolocaton API* memungkinkan proses pengambilan informasi lokasi pengguna sehingga dapat digunakan pada aplikasi web seperti untuk keperluan menampilkan lokasi pengguna ke peta digital atau keperluan lain yang berhubungan dengan lokasi pengguna tersebut (Mozilla, 2023).