Login : Jurnal Teknologi Komputer
ISSN : 2302-9692 (print) | 2723-8695 (online)
Vol. 17, No. 1, 2023, pp. 9-22
http://login.seaninstitute.org/index.php/Login

Perbandingan Algoritma Djikstra dan Floydwarshall untuk Mencari Jalur Terpendek dengan Contoh Kasus Mencari Rumah Sakit Terdekat di Kota Medan

Fredy Sitinjak, S.Kom
RSUP H. Adam Malik, Sumatera Utara, Indonesia.

Article Info

Keywords:

Algoritma Dijkstra,
Algoritma Floydwarshall,
Jalur terpendek.

ABSTRACT

Saat ini banyak sekali algortima-algoritma yang dapat digunakan untuk menyelesaikan persoalan penentuan jalur terpendek (shortest path problem) dari suatu jalur. Ada dua algortima yang cukup terkenal yang bisa digunakaan untuk menyelesaikan persoalan lintasan terpendek, yaitu Algoritma Dijkstra dan Algoritma Floydwarshall. Algoritma Dijkstra ini menggunakan prinsip greedy yang menyatakan bahwa pada setiap langkah kita memilih sisi yang berbobot minimum dan memasukkannya ke dalam himpunan solusi sedangan algoritma Floyd-Warshall menggunakan prinsip dinamis yang melakukan pemecahan masalah dengan memandang solusi yang akan diperoleh sebagai suatu keputusan yang saling terkait. Artinya solusi-solusi tersebut dibentuk dari solusi yang berasal dari tahap sebelumnya dan ada kemungkinan solusi lebih dari satu.

Corresponding Author:

Fredy Sitinjak,
RSUP H. Adam Malik, Sumatera Utara, Indonesia,
Jl. Bunga Lau No.17, Kemenangan Tani, Kec. Medan Tuntungan, Kota Medan, Sumatera Utara 20136.
Email: fredysitinjak@gmail.com

1. INTRODUCTION

Salah satu persoalan optimasi yang sering ditemui dalam kehidupan sehari-hari adalah pencarian lintasan terpendek (*shortest path*). Persoalan ini bisa dimodelkan kedalam suatu graf berbobot dengan nilai pada masing-masing sisi yang mewakili persoalan yang akan dipecahkan. Kata "terpendek" pada kata "jalur terpendek" ini tidak berarti hanya bisa diartikan jarak secara fisik, namun hal itu tergantung dari tipe persoalan yang akan dipecahkan. Bisa jadi kata tersebut memiliki makna tingkat akses suatu simpul dalam graf dari simpul yang lain. Persoalan jalur terpendek yang akan dibahas di makalah ini adalah lintasan terpendek antara dua buah simpul tertentu di dalam graf (*single pair shortest path*).

Algoritma *Dijkstra* merupakan salah satu generasi dari algoritma *greedy*, yaitu salah satu bentuk algoritma populer dalam pemecahan persoalan yang terkait dengan masalah optimasi. Sifatnya sederhana dan mudah (*straightforward*). Sesuai dengan artinya yang berarti tamak atau rakus – namun tidak dalam konteks negatif , algoritma greedy ini hanya memikirkan solusi terbaik yang akan diambil pada setiap langkah tanpa memikirkan konsekuensi ke depan. Prinsipnya, ambillah apa yang bisa Anda dapatkan saat ini (*take what you can get now?*), dan keputusan yang telah diambil pada setiap langkah tidak akan bisa diubah kembali. Intinya algoritma *greedy* ini berupaya membuat pilihan nilai optimal lokal pada setiap langkah dan berharap agar nilai optimum lokal ini mengarah kepada nilai optimum global.

Algoritma *Floydwarshall* adalah salah satu varian dari pemrograman dinamis, yaitu suatu metode yang melakukan pemecahan masalah dengan memandang solusi yang akan diperoleh sebagai suatu keputusan yang saling terkait. Artinya solusi-solusi tersebut dibentuk dari solusi yang berasal dari tahap sebelumnya dan ada kemungkinan solusi lebih dari satu. Hal yang membedakan pencarian solusi menggunakan pemrograman dinamis dengan algoritma *greedy* adalah bahwa keputusan yang diambil pada tiap tahap pada algoritma *greedy* hanya berdasarkan pada informasi yang terbatas sehingga nilai optimum yang diperoleh pada saat itu Jadi pada algoritma *greedy*, kita tidak memikirkan konsekuensi yang akan terjadi seandainya kita memilih suatu keputusan pada suatu tahap. Dalam beberapa kasus, algoritma *greedy* gagal memberikan solusi terbaik karena kelemahan yang dimilikinya. Di sinilah peran pemrograman dinamis yang mencoba untuk memberikan solusi yang memiliki pemikiran terhadap konsekuensi yang ditimbulkan dari pengambilan keputusan pada suatu tahap. Pemrograman dinamis mampu mengurangi keputusan yang tidak mengarah ke solusi. Prinsip yang dipegang oleh pemrograman dinamis adalah prinsip optimasi, yaitu jika solusi total optimal, maka bagian solusi sampai suatu tahap (misalnya tahap ke-i) juga optimal.

Kota Medan sebagai kota terbesar nomor tiga di Indonesia dan juga Ibukota propinsi Sumatera Utara, memiliki perkembangan yang pesat dalam pembangunan rumah sakit dan perbaikan pelayanan, maka hal yang wajar apabila pasien dari daerah-daerah atau dari luar kota Medan datang untuk berobat ke rumah sakit yang ada di kota Medan, selain itu tidak banyak masyarakat yang tinggal di kota Medan yang mengetahui keberadaan atau alamat rumah sakit terdekat dari lokasi tempat tinggalnya, selain sekedar mengetahui alamat rumah sakit maka diperlukan juga jalur terdekat yang harus di tempuh untuk sampai ke rumah sakit terdekat. Kebutuhan akan perjalanan ini menuntut adanya pemilihan jalur terpendek dari suatu lokasi ke rumah sakit terdekat sehingga dapat mengefisiensikan jarak, waktu dan biaya yang dibutuhkan untuk mencapai rumah sakit tujuan tersebut.

Dalam melakukan perjalanan, setiap pelaku perjalanan akan mencoba mencari jalur terbaik yang meminimkan biaya perjalanannya. Selain untuk mengefisiensikan jarak, waktu dan biaya yang dibutuhkan untuk menuju rumah sakit terdekat ataupun sebaliknya bagi pengguna/pelaku perjalanan, juga dapat mengurangi dampak kemacetan dengan pendistribusian/sebaran pergerakan perjalanan mengingat bahwa dewasa ini jaringan jalan di kota Medan mengalami permasalahan transportasi yang sangat kritis seperti kemacetan lalu lintas.

2. RESEARCH METHOD

1. Graph

Teori *graph* merupakan pokok bahasan yang sudah tua usianya namun memiliki banyak terapan sampai saat ini. *Graph* digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Representasi visual dari *graph* adalah dengan menyaakan objek sebagai noktah, bulatan atau titik, sedangkan hubungan antara objek dinyatakan dengan garis. Sebagai contoh, peta jaringan jalan raya yang menghubungkan sejumlah kota di propinsi jawa tengah. Sesungguhnya peta tersebut adalah sebuah *graph*, dimana kota disebut sebagai bulatan sedangkan jalan dinyatakan sebagai garis. (Rinaldi Munir, 2005, 535) jaringanjalan raya di propinsi Jawa Tengah dapat digambarkan dalam bentuk *graph* pada Figure 1.

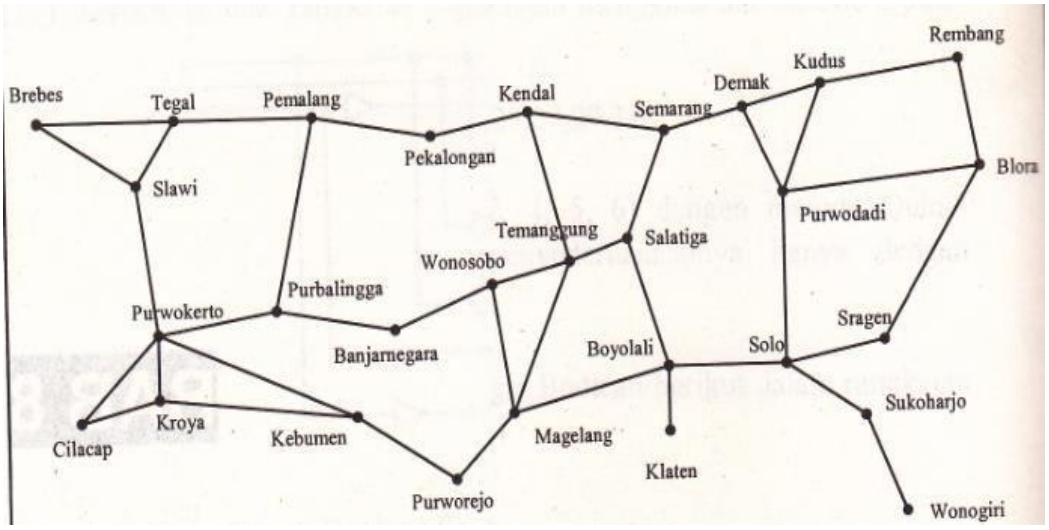


Figure 1. Jaringan jalan raya di propinsi Jawa Tengah

2. Definisi *Graph*

Secara matematis, *graph G* dapat didefenisikan sebagai berikut: *Graph G* didefenisikan sebagai pasangan himpunan *(V, E)*, ditulis dengan *G= (V, E),*yang dalam hal ini *V* adalah himpunan tidak kosong dari simpul-simpul (*vertices* atau *nodes*) dan *E* adalah himpunan sisi (*edges* atau *arcs*) yang menghubungkan sepasang simpul, (Rinaldi Munir, 2005, 356)

Defenisi ini menyatakan bahwa *V* tidak boleh kosong, sedangkan *E* boleh kosong. Jadi, sebuah *graph* dimungkinkan tidak mempunyai sisi satu buah pun , tetapi simpulnya harus ada , minimal satu. *Graph* yang hanya mempunyai satu buah simpul tanpa sebuah sisi pun dinamakan *graph* (Rinaldi Munir, 2005, 356)

Simpul pada *graph* dapat dinimori dengan huruf, seperti *a, b, c, ... v, w,...* atau dengan bilangan asli1, 2, 3, … ataupun dengan gabungan keduanya. Sedangkan sisi yang menghubungkan simpul *u* dengan simpul *v* dinyatakan dengan pasangan (*u, v*) atau dinyatakan dengan lambing *e1, e2, ...* dengan kata lain , jika *e* adalah sisi yang menghubungkan simpul *u* dengan simpul *v*, maka *e* dapat ditulis sebagai :

$$e = (u,v)$$

secara geometri *graph* digambarkan sebagai sekumpulan noktah (simpul) di dalam bidang dua dimensi yang dihubungkan dengan sekumpulan garis (sisi). (Rinaldi Munir, 2005,356) Beberapa contoh *graph* dapat dilihat pada Figure 2.

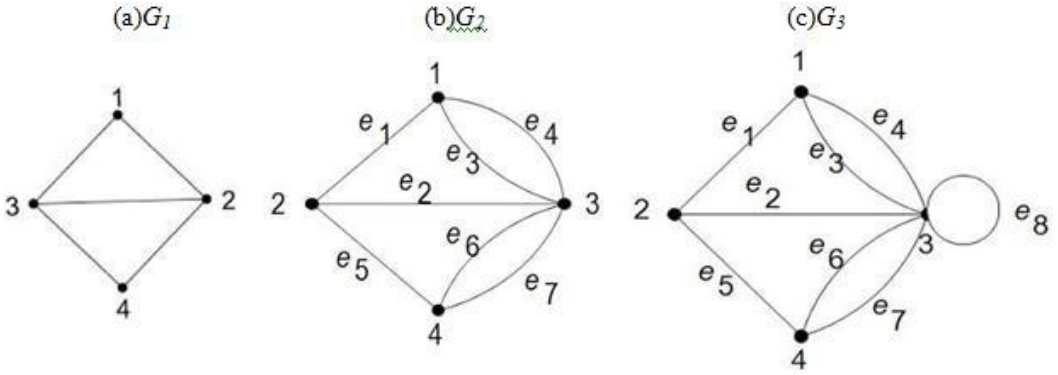


Figure 2. Tiga buah graph (a)graph sederhana (b) graph ganda (c)graph semu

Figure 2 memperlihatkan tiga buah *graph, G_1, G_2, G_3, G_1* adalah *graph* dengan himpunan simpul *V* dan himpunan sisi *E* adalah:

G_1 adalah *graph* dengan himpunan simpul *V* dan himpunan sisi *E*, yaitu:
V = {1,2,3,4 }
E = {(1,2), (1,3), (2,4), (3,4) }
G_2 adalah *graph* dengan himpunan simpul *V* dan himpunan sisi *E*, yaitu:
V = {1,2,3,4 }
E = {(1,2), (2,3), (1,3), (1,3),(2,4), (3,4), (3,4) }→ himpunan ganda
= {e_1, e_2, e_3, e_4, e_5, e_6, e_7}
G_3 adalah *graph* dengan himpunan simpul *V* dan himpunan sisi *E*, yaitu:
V = {1,2,3,4 }
E = {(1,2), (2,3), (1,3), (1,3),(2,4), (3,4), (3,4) }→ himpunan ganda
= {e_1, e_2, e_3, e_4, e_5, e_6, e_7}
Pada *graph* G_2 sisi e_3 = {1,3} dan sisi e_4 = {1,3} dinamakan sisi ganda (*multiple edges* atau *pararel edges*) karena kedua sisi ini menghubungi dua buah simpul yang sama, yaitu simpul 1 dan simpul 3. Pada *graph* G_3, sisi e_8 dinamakan gelang atau kalung (*loop*) karena berawal dan berakhir pada simpul yang sama. (Rinaldi Munir, 2005, 356-357)

3. Representasi *Graph*

Bila *graph* akan dip roses dengan program computer, maka *graph* harus direpresentasikan di dalam memory. Terdapat beberapa refpresentasi yang mungkin untuk *graph*, yaitu :

1) Matriks ketetanggaan (*adjacency matriks*)

Matriks ketetanggaan adalah representasi *graph* yang paling umum. Misalkan G = (V, E) adalah *graph* dengan n simpul, n ≥ 1. Matriks ketetanggaan G adalah matriks dua dimensi yang berukuran n x n. Bila matriks tersebut dinamakan A = [a_{ij}] maka a_{ij} = 1 jika simpul i dan j bertetangga, sebaliknya a_{ij} = 0, jika simpul I dan j tidak bertetngga.

Karena matriks bertetanggaan hanya berisi 0 dan 1, maka matriks tersebut dinamakan juga matriks nol-satu (*zero-one*). Selain dengan angka 0 dan 1, elemen matriks dapat juga dinyatakan dengan nilai *false* (menyatakan 0) dan *true* (menyatakan 1). Matriks ketetanggaan didasarkan pada pengurutan nomor simpul. Matriks ketetanggaan untuk *graph* sederhana dan tidak berarah selalu simetri, sedangkan untuk *graph* berarah, matriks ketetanggaannya belum tentu simetri (akan simetri jika berupa *graph* berarah lengkap). Selain itu, diagonal utamanya selalu nol karena tidak ada sisi gelang.

Sayangnya, matriks ketetanggaan nol-satu tidak dapat digunakan untuk merepresentasikan *graph* yang mempunyai sisi ganda (*graph* ganda). Untuk menyiasatinya, maka elemen a_{ij} pada matriks ketetanggaan maka sama dengan jumlah sisi yang berasi=osiasi dengan (v_i, v_j). matriks ketetanggaannya tentu bukan lagi matriks nol-satu. Untuk *graph* semu, gelang pada simpul v_i dinyatakan dengan nilai 1 pada posisi (i, i) di matriks ketetanggaannya.

Keuntungan representasi dengan matriks ketetanggaan adalah elemen matriksnya dapat diakses lalu melalui indeks. Selain itu, juga dapat menentukan dengan langsung apakahsimpul i dan simpul j bertetangga

2) Matriks Beririsan (Incidency Matrikx)

Bila matriks ketetanggaan menyatakan ketetangaan simpul-simpul di dalam *graph*, maka matriks beririsan menyatakan keberirisan simpul dengan sisi, Misalkan G= (V, E) adalah *graph* dengan n simpul dan m buah sisi. Matriks beririsan G adalah matriks dua dimensi yang berukuran n x m. Baris menunjukkan label simpul, sedangkan kolom menunjukkan label sisinya. Bila matriks tersebut dinamakan A= [a_{ij}], maka a_{ij} = 1 jika simpul i beririsan dengan sisi j, sebaliknya a_{ij} = 0 jika simpul I tidak beririsan dengan sisi j. Matriks beririsan dapat digunakan untuk merepresentasikan *graph* yang mengandung sisi ganda atau sisi gelang.

3) Senarai Ketetangaan (*Adjacecy List*)

Kelemahan matriks ketetanggaan adaah bila *graph* memiliki jumlah sisi relatif sedikit, karena matriksnya bersifat jarang (*sparse*), yaitu mengandung banyak elemen nol, sedangkan elemen yang bukan nol sedikit. Ditinjau dari implementasinya didalam computer, kebutuhan ruang memori untuk matriks jarang boros karena komputer menyimpan elemen 0 yang tidak perlu. Untuk mengatasi masalah ini, dapat menggunakan representasi yang ketiga yaitu senarai ketetanggaan. Senarai

ketetanggaan mengenumerasi simpul-simpul yang bertetangga dengan setiao simpul di dalam *graph*. (Rinaldi Munir, 2005, 381-385)

4. Contoh Terapan *Graph*

Graph dipakai diberbagai disiplin ilmu maupun dalam kehidupan sehari-hari. Penggunaan *graph* di berbagai bidang tersebut adalah untuk memodelkan persoalan. Dibawah ini dikemukakan contoh trepan *graph* dalam bidang kelistrikan, kimia, ilmu komputer dan pertandingan olahraga.

1) Rangkaian listrik
   Kirchoff (1987) menggunakan *graph* untuk memodelkan rangkaian listrik, berdasarkan *graph* tersebut Kirchoff menurunkan persamaan arus yang masuk dan keluar pada tiap simpul. Dari system persamaan linier simultan yang diperoleh dapat dihitung arus listrik yang mengalir pada setiap komponen.
2) Isomer senyawa kimia karbon
   Arthur Cayley (1857) menggunakan *graph* dalam memodekan molekul senyawa alkana C_2H_{2a+2} untuk menghitung jumlah isomernya. Atom karbon (C) dan atom hydrogen (H) dinyatakan sebagai simpul, sedangkan ikatan antara atom C dan H dinyatakan sebagai sisi. Isomer adalah senyawa kimia yan mempunyai rumus molekul sama tetapi rumus bangun (bentuk *graph*) berbeda.
3) Transaksi konkuren pada basis data terpusat
   Ini adalah peneraan *graph* pada bidang computer. Basis data (database) terpusat melayani beberapa transaksi (T) yang dilakukan secara konkuren adalah *deadlock*, yaitu keadaanyang dilakukan secara konkuren (kebersamaa). Transaksi terhadap basis datadapat berupa operasi pembacaan dan operasi penulisan terhadap data yang sama. Persoalan kritis pada proses konkuren adalah *deadlock*, yaitu keadaan yang timbul karena beberapa transaksi saling menunggu transaksi lainnya, sehingga system menjadi *hang*.
4) Pengujian programdalam bidang rekayasa perangkat lunak, sebuah program harus mengalami tahap pengujian untuk menemukan kesalahan (*bugs*). Salah satu pengujian program adalah pengujian eksekusi. Aliran kendali program harus diperiksa untuk memastikan apakah aliran tersebut sudah benar untuk berbagi kasus data uji. Aliran kendali program dimodelkan dengan *graph* berarah yang dinamakan *graph* alir (*flow graph*). Pada *graph* berarah tersebut, simpul menyatakan pernyataan atau kondisi yang di evaluasi, sedangkan busur menyatakan aliran kendali program ke pernyataan atau kondisi berikutnya.
5) Terapan *graph* di dalam teori automata
   Terapan *graph* dalam teori automatadapat dilihat sangat jelas pada konstruksi *Non-Deterministic Finite Automata* dan *Deterministic Finite Automata* pada bidang ilmu teori Automata (Teknik Kompilasi)
6) Turnamen Round-Robin
   Turnamen dimana setiap tim bertanding dengan tim lainnya hanya sekali disebut turnamen round robin. Turnamen semacm itu dimodelkan dengan *graph* berarah, yang dalam hal ini simpul menyatakan tiap tim yang bertanding dan busur menyatakan pertandingan. (Rinaldi Munir, 2005, 359-364)
7) Lintasan terpendek (shorted path)
   Persoalan lintasan terpendek merupakan salah satu persoalan optimasi yang menggunakan *graph* berbobot, dimana bobot pada setiap sisi*graph* tersebut dapat digunakan untuk jarak antar kota, waktu pengiriman pesan, ongkos pembangunan, dan sebagainya. Sedangkan, vertex melambangkan item yang terdapat dalam persoalan (seperti kota, node komputer, dan sebagainya) dan *edge* melambangkan hubungan dari item tersebut (seperti jalan, dan sebagainya) dan edge melambangkan hubungan dari item tersebut (seperti jalan, hubungan kabel, dan sebagainya).

5. Algoritma

Algoritma adalah suatu kumpulan sehingga (*finite set*) dari instruksi yang terdefenisi dengan baik (*well-defined instructions*) untuk menyelesaikan beberapa pekerjaan dimana diberikan state awal (initial state) dan akan dihentikan pada saat ditemukan state akhir (end-state) yang dikenal.

Algoritma dapat diimplementasikan dalam pembuatan program komputer. Kesalahan dalam merancang algoritma untuk menyelesaikan suatu problema dapat menyebabkan program gagal dalam implementasinya.

Konsep dari suatu algoritma sering di ilustrasikan dengan mengambil contoh sebuah resep, walaupun banyak algoritma yang jauh lebih kompleks. Algoritma sering memiliki beberapa langkah perulangan (iterasi) atau memerlukan pengambilan keputusan seperti logika (logic) atau perbandingan (*comparison*) sampai pekerjaan diselesaikan. Menerapkan suatu algoritma secara benar belum tentu dapat menyelesaikan problema. Hal ini dikarenakan adanya kemungkinan algoritma tersebut rusak atau cacat, atau penerapannya tidak cocok (tidak tepat) untuk menyelesaikan problema. Sebagai contoh, sebuah algoritma hipotesis untuk membuat sebuah salad kentang akan gagal jika tidak terdapat kentang.

Suatu pekerjaan dapat diselesaikan dengan menggunkan algoritma yang berbeda dengan kumpulan instruksi (*set of instructions*) yang berbeda dengan perbedaan waktu akses, efisiensi tempat, usaha dan sebagainya. Sebagai contoh, dierikan dua buah resep yang berbeda untuk membuat salad kentang, resep pertama mengupas kulit kentas terlebih dahulu sebelum memasak kentang tersebut, sementara resep akan mengulangi kedua langkah tersebut dan akan dihentikan pada saat salad kentang siap untuk dimakan.

Algoritma adalah hal yang mendasar untuk komputer dalam memproses informasi, karena sebuah program computer adalah sebuah algoritma yang memberitahukan kepada komputer langkah-langkah spesifik yang akan dijalankan (dalam urutan spesifik) untuk melakukan pekerjaan tertentu, misalnya menghitung gaji karyawan untuk mencetak rapor murid. Oleh karena itu, algoritma dapat dianggap sebagai beberapa operasi sekuensial (terurut) yang dapat dijalankan oleh sebuah system lengkap *Turing*. (Rinaldi Munir,2005, 495-497)

6. Shortest Path

Seorang pengendara sepeda motor ingin menemukan rute terpendek yang dapat dilalui dari Chicago ke Boston. Diberikan sebuah peta jalan dari amerika Serikat dimana setiap daerah yang dapat dilalui dengan jalur darat dengan jarak tertentu ditandai / dihubungkan, bagaimana cara menentukan rute terpendek ini?

Salah satu cara yang dapat dilakukan adalah dengan menjabarkan semua rute yang mungkin dari Chicago ke boston, menghitung jarak dari setiap rute tersebut dan mengambil rute dengan jarak terpendek dari kumpulan rute terpendek tersebut. Cara ini sangat tidak efisien dan akan memakan waktu yang sangat lama dalam mencari solusi rute terpendek tersebut karena bias terdapat banyak sekali rute yang dapat dilalui dari Chicago ke boston.

Cara lainnya yang lebih efisien adalah dengan menerapkan algoritma *shortest path* pada *graph.* Dalam sebuah problema *shortestpath, diberikan sebuah graph* berbobot dan berarah G = (V, E), dengan fungsi bobot w : E → R memetakan sisi (edge) dengannilai bobot dari *path* p = {v_0, v_1,…, v_q} adalah penjumlahanbobot dari setiap sisinya seperti ditunjukkan oleh rumusan berikut:

$$w(p) = \sum_{i-1}^{k}(v_{i-1}, v_1)$$

Kemudian, didefenisikan berbobot *shortest path* dari u ke v dengan rumusan berikut:

$$\&(u,v) = \begin{cases} \min\{w(p) : u \rightarrow v\} & : \text{jika terdapat } \textit{path} \text{ u ke v} \\ \infty & : \text{jika tidak terdapat } \textit{path} \end{cases}$$

Shotest pathdari simpul u ke simpul vdapat didefenisikan sebagai sembarang *path* p denganbobot w(p) = &(u, v).

Dalam contoh pencarian rute dari Chicago ke boston, peta jalan dapat dimodelkan sebagai sebuah *graph* dimana simpul memiliki daerah sisi mewakili jalan antar daerah yang dapat dilalui dan bobot sisi mewakili jarak jalan. Sasarannya adalah menemukan*shortest path* dari Chicago ke boston. (Thomas H. Cormen, Charles Eleiserson, Ronald L. Rivest, 1990,514)

Algoritma *shortest path* ini dapat diterapkan untuk mencari solusi dari beberapa variasi dari *shortest path* seperti:

a. *Single-destination shortest path*
   Problem ini ditunjukkan untuk mencari *shortest path* dari sebuah simpul tujuan t yang telah ditentukan dari setiap simpul v. problema ini dapat diselesaikan dengan membalikkanarah dari setiap sisi pada *graph*, sehingga penyelesaian problema ini sama persis dengan problema *single-source shortest path.*
b. *Single-pairs shortest path*
   Problema ini ditunjukkan untuk mencari *shortest path* dari u ke v untuk simpul u dan v yang ditentukan. Problema ini disebut juga degan problema *single-source shortest path.*
c. *All-pairs shortest path*

Problem ini ditunjukkan untuk mencari *shortest path* dari u ke v untuk setiap pasangan simpul u dan v. problema ini dapat diselesaikan dengan menerapkan algoritma *single-source shortest path* sekali dari setiap simpul, tetapi ada algoritma spesifik lain yang dapat menyelesaikan problema ini denganlebih cepat.(Thomas H. Cormen, Charles Eleiserson, Ronald L. Rivest, 1990,514)

7. Single Source Shortest path

Problema *single-source shortest path* ini dapat dideskripsikan sebagai berikut: Diberikan pasangan simpul u dan v dimana simpul sebagai simpul awal dan simpul v sebagai simpul tujuan. Sasarannya adalah mencari *shortest path* dari simpul u ke simpul v. solusi yang ingin dicari mencakup bobot dari *shortest path* dan sisi-sisi yang terdapat dalam jalur terpendek (*shortest path*) tersebut.

Dalam menyelesaikan problema *shortest path* ini, sering juga ditemui bahwa bobot dari sisi *graph* bernilai negative. Bobot negatif ini sring ditemukan pada penerapan algoritma *shortest path* untuk mencaari solusi biaya minimum pada ilmu eonomi. Jika *graph* G = (V,E) tidak memiliki siklus berbobot negative yang dapat dilalui dari simpul awal s, maka untuk semua v € V, bobot *shortest path* $\delta(s, v)$ dapat ditentukan, sekalipun *graph* tersebut memiliki bobot negative. Jika terdapat siklus berbobot negative yang dapa dilalui dari simpul awal s, maka *shortest path*-nya tidak dapat ditentukan. Jika terdapat siklus berbobot negative pada beberapa jalur (*path*) dari s ke v, maka *shortest path* $\delta(s, v) = -\infty$.(Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest,1990,518)

Beberapa algoritma yang dapat diterapkan untuk mencari solusi dari problema *shortest path* adalah:

a. Algoritma Dijkstra
b. Algorima Bellman-Ford
c. Algoritma Johnson
d. Algoritma Floydwarshall

8. Algoritma Dijkstra

Algoritma ini menerapkan konsep algoritma greedy dalam mencari solusi dari problema *shortest path.* Algoritma Dijkstra menyelesaikan problema *single-source shortest path* pada sebuah *graph* berbobot dan beaeah G = (V, E) untuk kasus dimana semua bobot sisi adalah positif aaupun mengambil asumsi bahwa $w(u, v) \geq 0$ untuk semua sisi (u, v) € E. Algoritma Dijkstr ini memiliki kompleksitas waktu sebesar $O(V^2)$.

Algoritma dijkstra secara berulang memilih simpul u ϵ V –S dengan mencari bobot sisi paling minimum, memasukkan u ke dalam S, dan menyusun kembali *graph* V – S tanpa memasukkan simpul u. Jumlah perulangan dari algoritma ini adalah sebesar jumlah simpul dari *graph.* Salah satu contoh implementasi algoritma dijkstra dengan menggunakan representasi *adjacency list* dalah sebagai berikut (Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest,1990,518)

Dijkstra (G, w, s)
1. *Initialize-Single-Source(G,s)*
2. *S* ← *Ø*
3. *Q* ← *V[G]*
4. ***While*** *Q* ≠ *Ø*
5. ***do*** *u* ← *Extract-Min(Q)*
6. *S* ← *S* ∪ *(u)*

7. ***For*** *setiap sisi v € Adj[u]*
8. ***do*** *Relax(u, v, w)*

9. Algoritma Bellman-Ford

Algoritma Bellman-Ford mnyelesaikan problema *single-source path* untuk kasus yang lebih umum dimana bobot dari sisi dapat berupa bilangan negatif. Diberikan sebuah *graph* berbobot dan berarah G = (V,E) dengan simpul awal s dan fungsi bobot w : E → R, algoritma Bellman-Ford mengembalikan sebuah nilai *Boolean* yang mengindikasikan apakah terdapat siklus berbobot negatif yang dapat dilalui oleh simpul awal atau tidak. Jika terdapat siklus, algoritma akam mengindikasikan bahwa tidak terdapat solusi *shortest path.* Jika tidak, algoritma akan menghasilkan *shortest path* beserta bobotnya.

Algoritma bellman-ford ini memiliki kompleksitas waktu sebesar O(VE). Algoritma bellman-Ford ini dapat didefenisikan sebagai berikut (Thomas H. Cormen, Chaarles E.Leiserson, Ronald L. Rivest, 1990, 532-533)

Bellman-Ford(G, w, s)
1. *Initialize-Single-Source(G,s)*
2. ***for I*** ← *1 to |V[G]| - 1*
3. ***do for*** *setiap sisi (u, v) ϵ E[G]*
4. ***do*** *Relax (u,v,w)*
5. ***for*** *setiap sisi (u, v) ϵ E[G}*
6. ***do if*** *d[v] > d[u] + w(u, v)*
7. ***then*** *return FALSE*
8. ***Return*** *TRUE*

10. All-Pairs shrortest Path

Problema ini ditujukan untuk mencari *shortest path* dari semua pasangan simpul pada sebuah *graph.* Salah satu contoh penerapan problema ini yaitu dalam pembuatan table jarak terpendek antara semua pasangan kota dari peta jalan. Seperti pada problema *sigle-source path,* diberikan sebuah *graph* berbobot dan berarah G = (V, E) dengan fungsi bobot w : E → R yang memetakan sisi ke sebuah nilai rill. Sasarannya adalah untuk setiap pasangan simpul u, v ϵ V, dicari sebuah jalur terpendek (berbobot paling minimum) dari u ke v, dimana bobot dari sebuah jalur terpendek (berbobot paling minimum) dari u ke v, dimana obot dari sebuah jalur adalah penjumlahan dari bobot sisi-sisinya. Hasil dari problema ini berupa sebuah table dimana nilai pada baris u dan kolom v merepresentasikan bobot *shortest path* dari u ke v.

Cara termudah untuk menyelesaikan problema ini adalah dengan menjelaskan algoritma *single-source shortest path* sebanyak V kali, sekali untuk setiap simpul sebaga simpul awal. Jika semua sisi berbobot positif, makan algoritma *Dijkstra* dapat digunakan. Sedangkan, jika ada sisi berbobot negative maka algoritma *Dijkstra* tidak dapat digunakan. Untuk menyelesaikannya, dapat menggunakan algoritma Bellman-Ford yang lebih lambat.

Cara lainnya yang memiliki waktu eksekusi yang lebih baik adalah dengan menggunakan algoritma Johnson dan *floydwarshall*. Tidak seperti algritma *single shortest path* yang kebanyakan menggunakan representasi *adjacency list,* algoritma *all-pairs shortest path* ini menggunakan representasi *adjacency* matriks. Input dari algoritma beberapa matriks W berukuran n x n yang merepresentasikan nilai bobot sisi dari sebuah *graph* berarah G = (V,E) dengan n buah simpul. (Thomas H. Cormen, Chaarles E.Leiserson, Ronald L. Rivest, 1990, 532-533).

11. Algoritma Johnson

Algortima Johnson menemkan *shortest path* diantara semua pasangan simpul pada *graph* dengan kompleksias waktu $O(V^2 \lg V + VE)$. Algoritma Johnson ini memiiki waktu eksekusi yang lebih bagus dari *floydwarshall* untuk *input* yang berupa *graph* jarang.

Algoritma Johnson menggunakan kombinasi dai algoritma *dijkstra* dan algoritma bellman-ford. Prosedur kerja dari algoritma Johnson ini dpat dilihat sperti berikut

Jhonson(G)

1. *Compute G', where V[G'] = V[G] ∪ {s} and E[G'] = E[G] ∪ {(S, V) : v ∈ V[G]}*
2. ***if*** *bellman-Ford(G', w, s) = False* ***then***
3. *print "input graph memiliki negative-weight cycle"*
4. ***Else***
5. ***for*** *setiap vertex v ∈ V[G']*
6. *$\bar{\delta}$(s, v) ← h(v) (hitung dengan algoritma Bellman-Ford)*
7. ***for*** *setiap edge(u, v) ∈* E[G']
8. *W(u, v)← w(u, v) + h(u) – h(v)*
9. ***for*** *setiap vertex u ∈* V[G']
10. *Jalankan algoritma Dijksrtra(G, w, u) untuk mengitung $\bar{\delta}$(u, v) untuk semua v ∈ V[G]*
11. ***for*** *setiap vertex v ∈ V[G]*
12. *d(u, v) ← $\bar{\delta}$(u, v) + h(v) – h(u)*
13. ***return*** *D*

(Thomas H. Cormen, Chaarles E.Leiserson, Ronald L. Rivest, 1990, 532-533).

12. Floydwarshall

Algoritma ini merupakan konsep *dynamic programming* dalam mencari *all-pairs shortest-path* dari sebuah *graph.* Algoritma ini memiliki kompleksitas waktu sebesar $O(V^3)$. Proses kerja dari algoritma ini adalah sebagai berikut:

1. Algoritma proses perhitungan *shortest path.*

Floyd-Warshall(d,n)

1. ***for k*** ← *1 to n*
2. ***for j*** ← *1 to n*
3. ***for*** ← *1 to n*
4. *$d_{ij}(k) = min(d_{ij}(k-1), d_{ij}(k-1))$*
5. ***if*** *choose $d_{ij}(k-1) + d_{ij}(k-1)$* ***then*** *pred(ij) = k*

2. Algoritma proses penentuan rute dari *shortest path.*

Path (I, j)

1. ***If*** *pred(I, j) = nil* ***then***
2. ***Return*** *output(I, j)*
3. ***Else***
4. *Path(i, pred(I, j))*
5. *Path(pred(I, j), j)*

3. RESULTS AND DISCUSSION

1. Menu Utama

Menu utama dalam aplikasi ini adalah berfungsi sebagai tampilan utama dalam aplikasi pencarian jalur terpendek. Dalam hal ini menu utama menyajikan gambar titik rumah sakit yang ada di kota medan, pada tampilan ini juga telah disediakan titik asal yang ditentukan dan titik tujuan yang akan dicari, dan juga pilihan algoritma yang akan digunakan untuk melakukan pencarian jalur terpendek untuk menempuh salah satu rumah sakit yang akan dituju, interface untuk menu utama dapat dilihat pada Figure 3

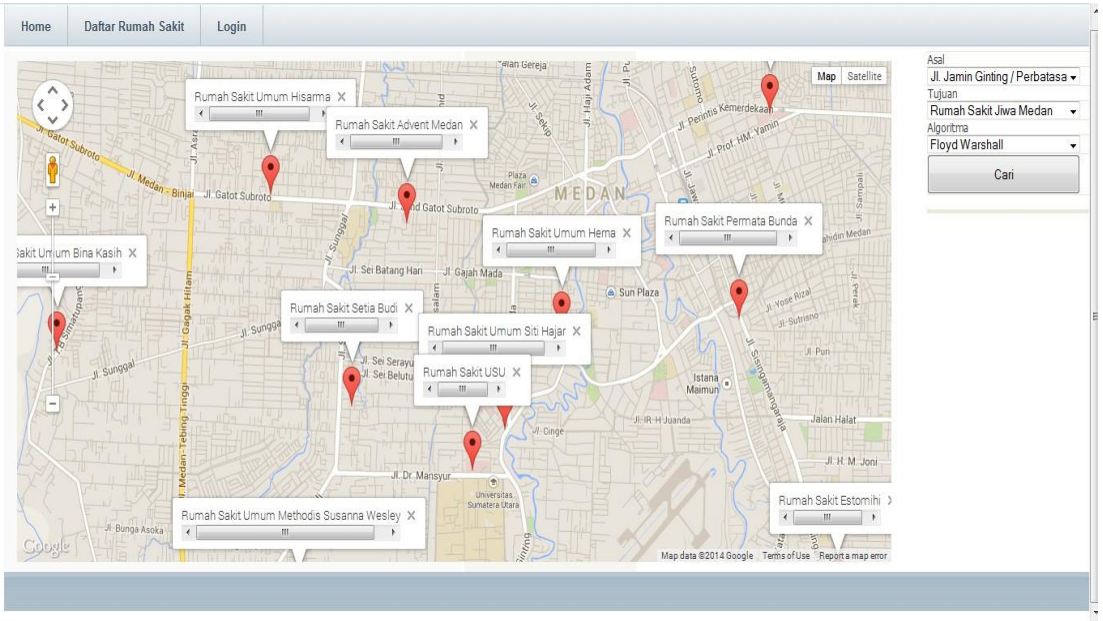


Figure 3. Menu Utama

2. Menu Daftar Rumah Sakit

Menu daftar rumah sakit adalah untuk membantu user melihat daftar rumah sakit yang disediakan oleh sistem, pada daftar rumah sakit juga disediakan data rumah sakit seperti jalan, nomor telepon dan website (jika ada) untuk mempermudah user masuk ke link website rumah sakit yang di inginkan. Interface untuk menu daftar rumah sakit dapat dilihat pada Figure 4

No	Nama Rumah Sakit	Alamat	No Telp	Website
1	Rumah Sakit Umum Pusat H Adam Malik	Jalan Bunga Lao No. 17, Medan	(061) 83 64 581	http://rsham.co.id/
2	Rumah Sakit Jiwa Medan	Jl. Tali air NO 21		
3	Rumah Sakit Umum Methodis Susanna Wesley	Jl. Setia Budi	061-7861771	http://www.rsu-methodist-susannawesley.or.id/
4	Rumah Sakit Umum Siti Hajar	JL. Djamin Ginting, No. 2,	+62 61 821387	http://www.rsi-sitihajar-sda.com/
5	Rumah Sakit Setia Budi	Jl. Mesjid No.3 Tanjung Rejo Medan	+62 61 8220995	http://rssetiabudi.com
6	Rumah Sakit USU	Jl. Dr. Mansyur Medan		
7	Rumah Sakit Umum Mitra Sejati	Jl Jend AH Nasution 7-A Medan		
8	aaaa	aaaaa	0987654	http.sss.com

Figure 4. Menu daftar rumah sakit

3. Tampilan Login

Tampiln login merupakan tampilan khusus admin untuk masuk kedalam sistem dan mengubah maupun memperbaiki sistem. Interface untuk tampilan login dapat dilihat pada Figure 5

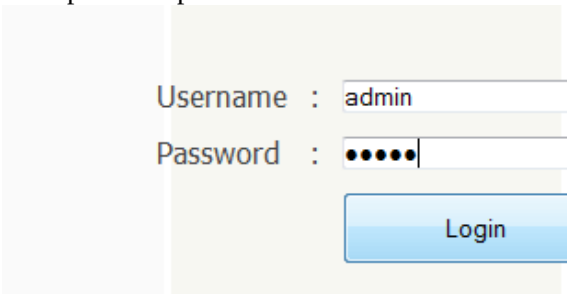


Figure 5. Tampilan Login

4. **Tampilan daftar titik**

Tampilan daftar titik merupaka tampila untuk admin setelah login kedalam sistem, pada tampilan ini admin dapat memasukkan titik baru, melakukan edit titik maupun penghapusan titik. Interface untuk tampilan daftar titik dapat dilihat pada Figure 6

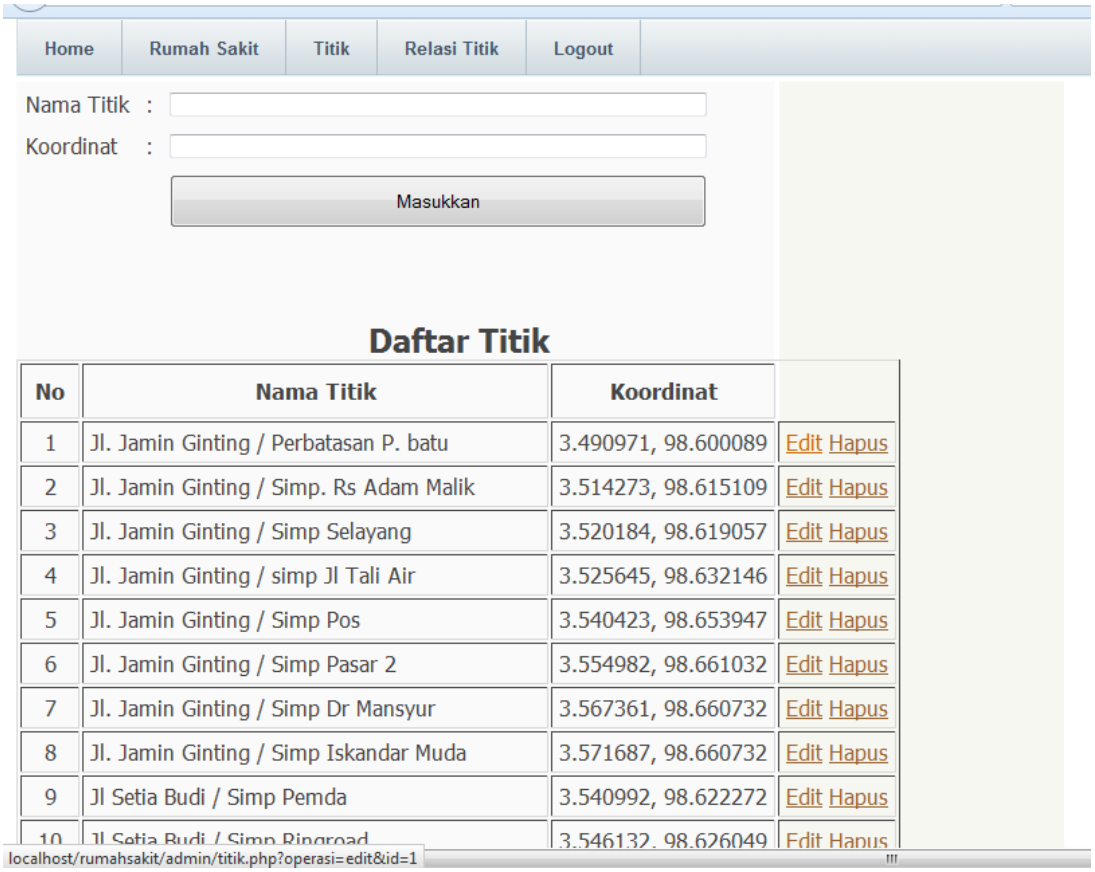
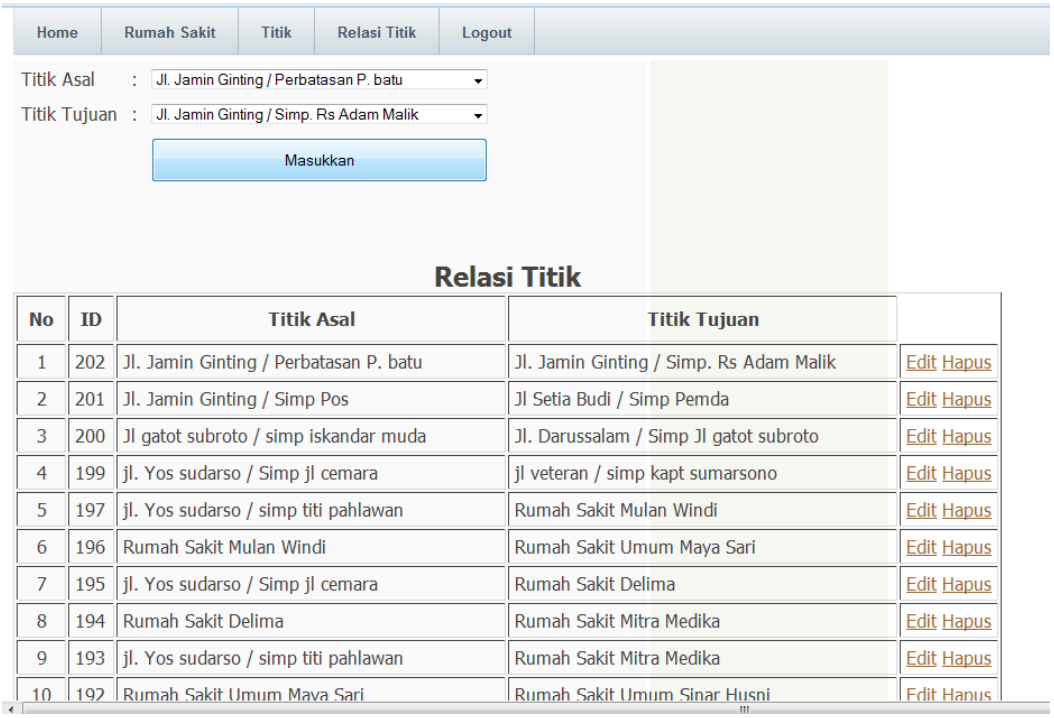


Figure 6. Tampilan daftar titik

5. Tampilan relasi titik

Tampilan relasi titik merupakan tampilan untuk membuat relasi titik baru setelah login sebagai admin, admin dapat membuat relasi titik baru, mengedit dan menghapus relasi titik yang ada pada sistem. Interface untuk tampilan relasi titik dapat dilihat pada Figure 7



No	ID	Titik Asal	Titik Tujuan	
1	202	Jl. Jamin Ginting / Perbatasan P. batu	Jl. Jamin Ginting / Simp. Rs Adam Malik	Edit Hapus
2	201	Jl. Jamin Ginting / Simp Pos	Jl Setia Budi / Simp Pemda	Edit Hapus
3	200	Jl gatot subroto / simp iskandar muda	Jl. Darussalam / Simp Jl gatot subroto	Edit Hapus
4	199	jl. Yos sudarso / Simp jl cemara	jl veteran / simp kapt sumarsono	Edit Hapus
5	197	jl. Yos sudarso / simp titi pahlawan	Rumah Sakit Mulan Windi	Edit Hapus
6	196	Rumah Sakit Mulan Windi	Rumah Sakit Umum Maya Sari	Edit Hapus
7	195	jl. Yos sudarso / Simp jl cemara	Rumah Sakit Delima	Edit Hapus
8	194	Rumah Sakit Delima	Rumah Sakit Mitra Medika	Edit Hapus
9	193	jl. Yos sudarso / simp titi pahlawan	Rumah Sakit Mitra Medika	Edit Hapus
10	192	Rumah Sakit Umum Maya Sari	Rumah Sakit Umum Sinar Husni	Edit Hapus

Figure 7 Tampilan relasi titik

6. Hasil

Untuk melihat hasil dan perbandingan pencarian jalur tependek dari algoritma *dijkstra* dan *floydwarshall*l maka aka dilakikan pencarian dari titik asal yang sama menuju rumah sakit yang sama, yaitu yang menjadi titik asal adalah Jl Sm Raja / simp HM joni menuju Rumah sakit Methodist Susanna Wesley.

7. Algoritma *Dijkstra*

Setelah melakukan uji coba dengan menggunakan algoritma *dijkstra* maka jalur yang dilalui dari jl Sm Raja / simp Hm Joni adalah : Jl SM Raja / Simp HM Joni →Jl SM Raja / Simp jl Halat → jl. H juanda / Simp Brigjen Katamso →jl. H juanda / Simp bandara polonia → jl. H juanda / Simp jl slamet ryadi → Jl. Mongonsidi / Bundaran depan pardede hotel →Jl. Mongonsidi / simp jl pattimura → Jl. Jamin Ginting / Simp Iskandar Muda → Rumah Sakit Umum Siti Hajar → Jl. Jamin Ginting / Simp Dr Mansyur → Jl. Jamin Ginting / Simp Pasar 2 →Rumah Sakit Umum Methodis Susanna Wesley, dengan waktu eksekusi yang dilakukan sebanyak tiga kali berulang ulang adalah 1.11 detik, 1.12 detik, 1.13 detik, maka dapat diambil rata-rata waktu eksekusi nya adalah 1.11 detik. Interface untuk hasil pencarian dengan menggunakan algoritma *dijkstra* dapat dilihat pada Figure 8

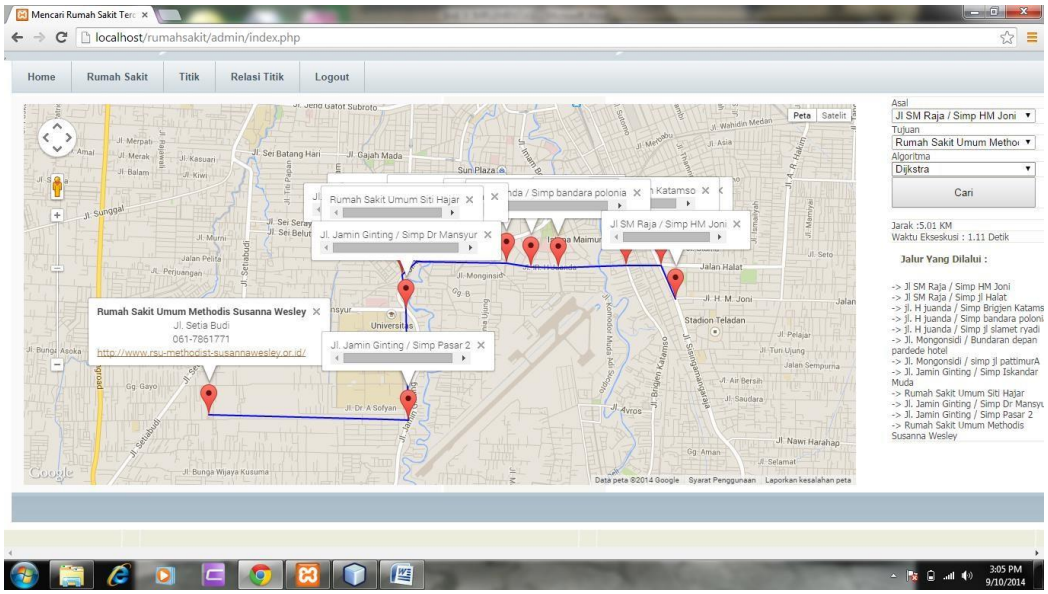


Figure 8. Hasil eksekusi algoritma dijkstra

8. Algoritma *Floydwarshall*

Setelah melakukan uji coba dengan menggunakan algoritma *floydwarshall*l maka jalur yang dilalui dari jl Sm Raja / simp Hm Joni adalah : Jl SM Raja / Simp HM Joni→Jl SM Raja / Simp jl Halat → jl. H juanda / Simp Brigjen Katamso →jl. H juanda / Simp bandara polonia → jl. H juanda / Simp jl slamet ryadi → Jl. Mongonsidi / Bundaran depan pardede hotel →Jl. Mongonsidi / simp jl pattimura → Jl. Jamin Ginting / Simp Iskandar Muda → Rumah Sakit Umum Siti Hajar → Jl. Jamin Ginting / Simp Dr Mansyur → Jl. Jamin Ginting / Simp Pasar 2 →Rumah Sakit Umum Methodis Susanna Wesley, dengan waktu eksekusi yang dilakukan sebanyak tiga kali berulang ulang adalah 6.27 detik, 6.50 detik, 6.70 detik, maka dapat diambil rata-rata waktu eksekusi nya adalah 6.49 detik. Interface untuk hasil pencarian dengan menggunakan algoritma *floydwarshall* dapat dilihat pada Figure 9

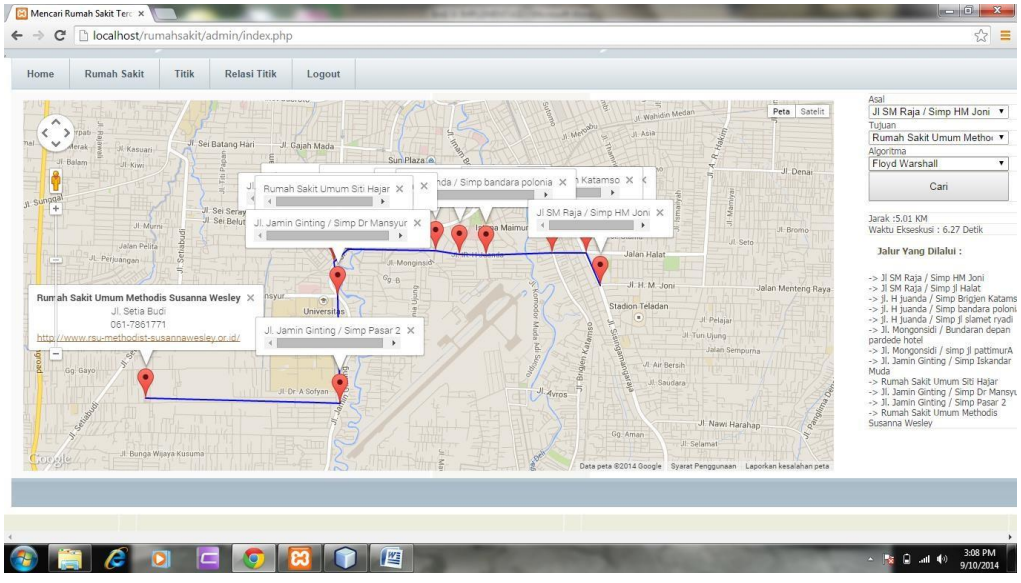


Figure 9. hasil eksekusi algoritma floydwarsahll

9. CONCLUSION

Setelah melakukan uji coba sistem maka dapat disimpulkan untuk mencari jalur terpendek dalam pencarian rumah sakit terdekat dikota medan dengan menggunakan algoritma dijkstra dan floydwarshall maka dapat disimpulkan dalam pengambilan jalur terpendeknya selalu menghasilkan jalur yang sama, adapun yang menjadi perbedaannya dalam studi kasus ini adalah waktu eksekusi program dimana algoritma dijkstra lebih cepat dalam eksekusi program saat user melakukan pencarian titik.

REFERENCES

[1] Barus, B dan U. S. Wiradisastra. 2000. *Sistem Informasi Geografi Sarana Manajemen Sumberdaya*. Laboratorium Penginderaan Jauh dan Kartografi. Jurusan Tanah. Fakultas Pertanian. IPB. Bogor.

[2] Fakhri. 2014. penerapan algoritma dijkstra dalam pencarian solusi maximum flow problem. http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2007-2008/Makalah2008/MakalahIF2251-2008-039.pdf (diakses 4 april 2014).

[3] Rinaldi. 2005. Matematika Diskrit. Bandung. Informatika.

[4] Raden Aprian Diaz Novandi. Perbandingan Algoritma Dijkstra dan Algoritma Floyd-Warshall dalam Penentuan Lintasan Terpendek (Single Pair ShortestPath). http://webmail.informatika.org/~rinaldi/Stmik/2006-2007/Makalah_2007/MakalahSTMIK2007-021.pdf (diakses 4 april 2014).

[5] Sugiyono, 2009, Metode Penelitian Kuantitatif, Kualitatif dan R&D, Bandung : Alfabeta

[6] Thomas H. Cormen, Charles Eleiserson, Ronald L. Rivest. 1990. Introduction to Algorithms. USA. MIT press.