Implementasi Algoritma BFS Dalam Pencarian Rute Teroptimal pada Aplikasi Navigasi *Waze*

Alson Cahyadi 13514035
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
alson.cahyadi@students.itb.ac.id

***Abstrak*— Dalam dunia digital di mana teknologi semakin hari berkembang semakin pesat ini, orang-orang secara perlahan menggantikan peta analog dengan peta berbasis internet. Peta berbasis internet ini jugalah yang merupakan batu loncatan untuk aplikasi-aplikasi navigasi berbasis GPS untuk menjadi pusat lampu sorot masyarakat yang membutuhkan arahan untuk mencapai tempat tujuan yang kurang familiar, dan sulit untuk dicapai. Makalah ini akan menjelaskan tentang sebuah aplikasi navigasi, Waze, dan algoritma pencarian jalur tercepatnya yang berlandaskan algoritma Dijkstra.**

***Kata Kunci* – Dijkstra Algorithm, Navigasi, Shortest-Path algorithm, Waze.**

I. PENDAHULUAN

Saat ini penggunaan GPS untuk mengetahui koordinat keberadaan diri sendiri sudah ada di hampir semua *smartphone*. Akurasi GPS yang tinggi dan waktu updatenya yang *realtime* membukakan jalan aplikasi-aplikasi berbasis peta online untuk menawarkan fitur berupa fitur navigasi. Fitur navigasi yang ditawarkan mengoptimalisasi perjalanan yang dibutuhkan berdasarkan waktu perjalanan.

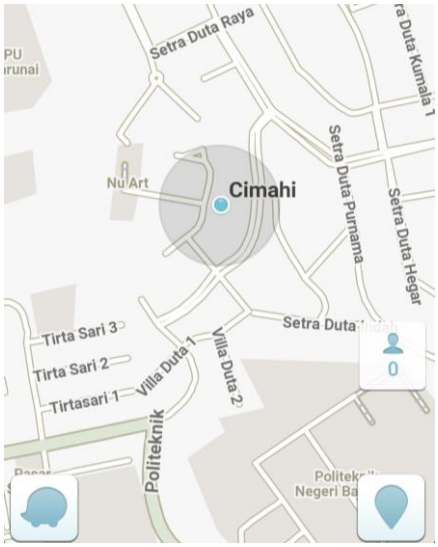
Dengan fitur navigasi ini, user dapat berpergian ke tempat yang bahkan sama sekali tidak user ketahu. Informasi input yang dibutuhkan oleh system navigasi ini hanyalah nama tempat yang ingin dituju, dan dengan algoritmanya fitur navigasi ini akan menunjukkan rute tercepat menuju tempat tersebut dengan waktu tempuh paling cepat, dan rute paling mangkus.

Salah satu aplikasi yang menyediakan fitur navigasi terbaik yang telah banyak diketahui orang-orang terutama masyarakat Indonesia adalah Waze.

II. WAZE

Gambar 2.1, Logo Waze dan tagline-nya.
Sumber: http://cdn.slashgear.com/wp-content/uploads/2015/03/waze.jpg

Waze merupakan sebuah piranti lunak navigasi bebas bayar untuk perangkat *mobile* seperti *smartphone* atau tablet, maupun PC yang memiliki fitur GPS. Sampai saat ini, Waze mendukung perangkat dengan system operasi iOS, Android, Windows Mobile, Symbian dan Blackberry.



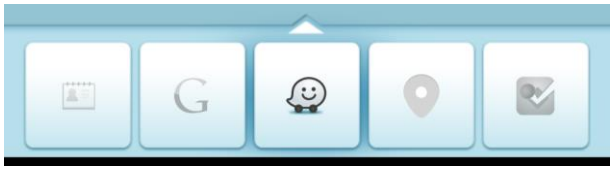
Gambar 2.2, Layour peta Waze
Sumber: Galeri penulis

Waze dapat diunduh dari negara mana saja di dunia termasuk Indonesia, tetapi peta dasar untuk negara

Indonesia belum tersedia, sehingga kontribusi *user* sangat diutamakan dalam pengembangan peta Indonesia ini.

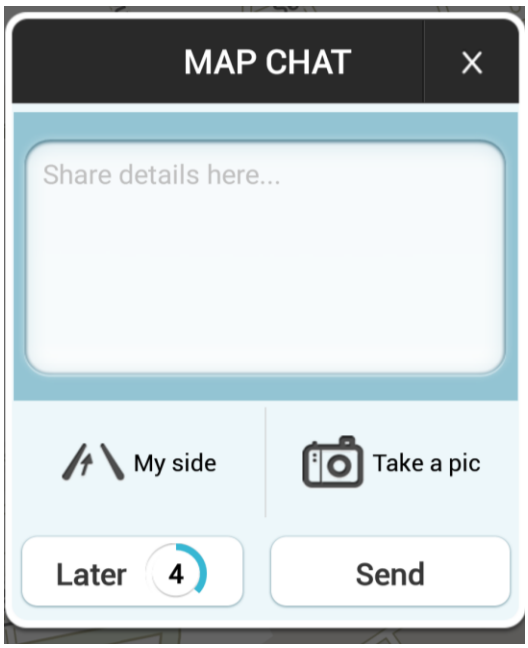
Gambar 2.3, Kontribusi yang *user* dapat berikan
Sumber: Galeri penulis

Waze berbeda dengan piranti lunak navigasi pada umumnya, dikarenakan peran *user* yang sangat besar dalam berkontribusi untuk informasi-informasi mengenai kecelakan, kemacetan jalan, polisi, maupun bahaya berdasarkan kondisi nyata yang dilaporkan para penggunanya. User juga dapat melakukan pemutakhiran peta, pemberian nomor rumah/bangunan, maupun penandaan lokasi secara pribadi dan langsung, seperti yang ada pada gambar 2.3. Gambar 2.4 menunjukkan contoh laporan jalan macet yang telah dilaporkan user-user yang ada di jalan raya. Informasi ini selanjutnya digunakan waze dalam algoritma pencarian rute teroptimalnya. Jalanan yang lebih macet akan menyebabkan estimasi waktu sampai yang lebih lama daripada jalan lancer.



Gambar 2.5, Pilihan search engine yang ditawarkan
Sumber: Galeri penulis

Keunggulan Waze yang lain adalah kemampuannya untuk mencari tempat tujuan di search engine yang berbeda, seperti Google Maps, Foursquare, Contacts, dan lain-lain, seperti yang ada pada gambar 2.5.



Gambar 2.6, Layour map chat
Sumber: Galeri penulis

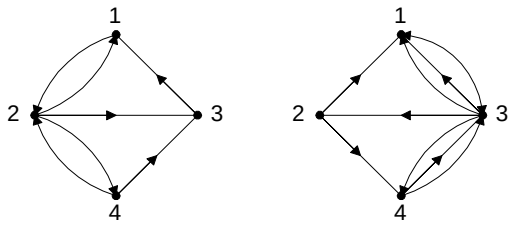
Waze juga mempunyai fasilitas *chat* yang memberikan poin untuk setiap kegiatan yang dilakukan *user* seperti menjelajah peta, kontribusi informasi, pemutakhiran peta dan peristiwa khusus lainnya. Dengan fasilitas ini, bukan hanya piranti navigasi, tetapi Waze juga adalah jejaring sosial dengan permainan online yang layoutnya ada pada gambar 2.6.

III. GRAF BERARAH

Graf berarah digunakan pada implementasi piranti navigasi karena adanya kemungkinan jalan yang satu arah di lapangan. Adapun penjelasan graf berarah sbb:

Definisi Graf adalah:
Graf G = (V, E), yang dalam hal ini:
V = himpunan tidak-kosong dari simpul-simpul
= { v1 , v2 , ... , vn }
E = himpunan sisi yang menghubungkan sepasang simpul
= {e1 , e2 , ... , en }

Graf yang setiap sisinya diberikan orientasi arah disebut sebagai graf berarah. Dua buah graf pada Gambar 3.1 adalah graf berarah.



(a)G_4 (b) G_5

Gambar 3.1 (a) graf berarah, (b) graf-ganda berarah
Sumber: Slide Kuliah Rinaldi Munir (Graf 2015)

Tabel jenis-jenis graf dapat dilihat pada Tabel 1

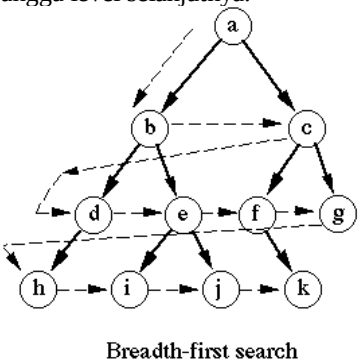
Tabel 1 Jenis-jenis graf [ROS99]

Jenis	Sisi	Sisi ganda dibolehkan?	Sisi gelang dibolehkan?
Graf sederhana	Tak-berarah	Tidak	Tidak
Graf ganda	Tak-berarah	Ya	Ya
Graf semu	Tak-berarah	Tidak	Ya
Graf berarah	Tak-berarah	Ya	Ya
Graf-ganda berarah	Bearah		
	Bearah		

Sumber: Slide Kuliah Rinaldi Munir (Graf 2015)

IV. BREADTH FIRST SEARCH

BFS adalah algoritma pencarian untuk menelusuri pohon atau graf. Algoritma dimulai dari akar pohon dan menelusuri tetangga simpulnya terlebih dahulu, sebelum menelusuri tetangga level selanjutnya.



***Gambar 4.1**, visualisasi BFS*
Sumber: Slide Kuliah Rinaldi Munir: BFS dan DFS (2015)

Pencarian Melebar ini memiliki algoritma singkat seperti berikut:

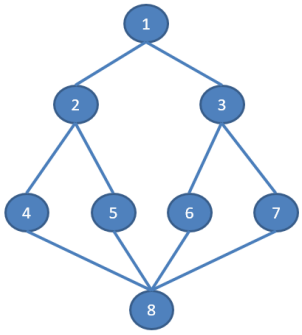
1. Kunjungi simpul akar
2. Kunjungi semua simpul yang bertetangga dengan simpul yang sedang dikunjungi
3. Kunjungi semua simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadinya dikunjungi, dan sedemikian seterusnya

Untuk memprogram algoritma BFS, dibutuhkan struktur data sebagai berikut:

1. Matriks ketetanggaan a[i][j] yang mencatat bobot dari i ke j, atau 1 dan 0 pada graf tidak berbobot (1 menunjukkan simpul i dan j bertetangga, sedangkan 0 menunjukkan simpul i dan j tidak bertetangga)
2. Queue q untuk menyimpan simpul yang telah dikunjungi dan akan ditinjau lagi
3. Tabel boolean d[i] yang bernilai true bila simpul i sudah dikunjungi dan false bila simpul i belum dikunjungi.

Algoritma Breadth First Search ini memiliki kompleksitas $O(|V| + |E|)$ dimana V adalah jumlah sisi (*vertex*) dan E adalah jumlah simpul (*edge*).

Dalam analisis algoritma, input dari pencarian melebar ini diasumsikan selalu terhingga. Pencarian ini komplit (pasti mendapatkan goal state bila ada pada graf), berbeda dengan depth first search yang mungkin akan hilang terlalu lama di pencarian cabang pohon yang tak terhingga panjangnya.



Iterasi	V	Q	dikunjungi							
			1	2	3	4	5	6	7	8
Inisialisasi	1	{1}	T	F	F	F	F	F	F	F
Iterasi 1	1	{2,3}	T	T	T	F	F	F	F	F
Iterasi 2	2	{3,4,5}	T	T	T	T	T	F	F	F
Iterasi 3	3	{4,5,6,7}	T	T	T	T	T	T	T	F
Iterasi 4	4	{5,6,7,8}	T	T	T	T	T	T	T	T
Iterasi 5	5	{6,7,8}	T	T	T	T	T	T	T	T
Iterasi 6	6	{7,8}	T	T	T	T	T	T	T	T
Iterasi 7	7	{8}	T	T	T	T	T	T	T	T
Iterasi 8	8	{}	T	T	T	T	T	T	T	T

***Gambar 4.2**, ilustrasi algoritma BFS*
Sumber: Slide Kuliah Rinaldi Munir: BFS dan DFS (2015)

V. ALGORITMA DIJKSTRA

A. Sejarah dan Penjelasan Algoritma

Algoritma Dijkstra merupakan algoritma yang mengimplementasikan BFS

Algoritma Dijkstra merupakan algoritma yang dikemukakan oleh seorang ilmuwan komputer bernama Edsger W. Dijstra pada tahun 1956 dan dipublisasikan tiga tahun kemudan.

Algoritma ini adalah sebuah algoritma yang digunakan untuk menemukan rute terpendek antara node-node dalam suatu graf yang dapat merepresentasikan jalan, jaringan, dan sebagainya, dan algoritma ini berdasar dari algoritma BFS (*Breadth First Search*).

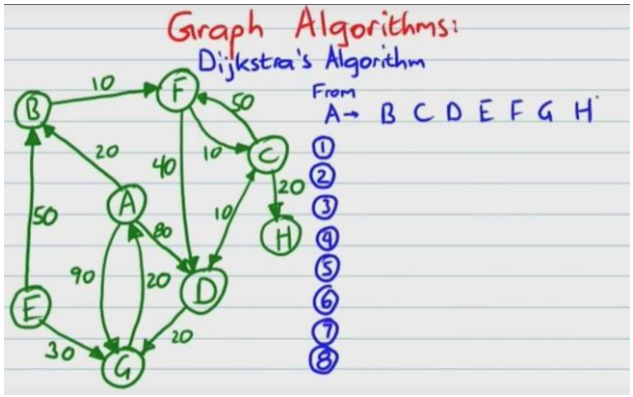
Algoritma Dijkstra yang asli memiliki tujuan untuk menemukan rute tercepat antara node asal dan node destinasi, sedangkan Algoritma Dijkstra yang lebih dikenal

mencari rute optimal antara node asal dan semua node lainnya yang ada di graf, yang menghasilkan pohon rute terpendek.

B. Penerapan Algoritma Dijkstra pada Graf Berarah

Berikut adalah langkah-langkah implementasi algoritma Dijkstra pada graf berarah, beserta contohnya.

1. Tentukan node "sumber"

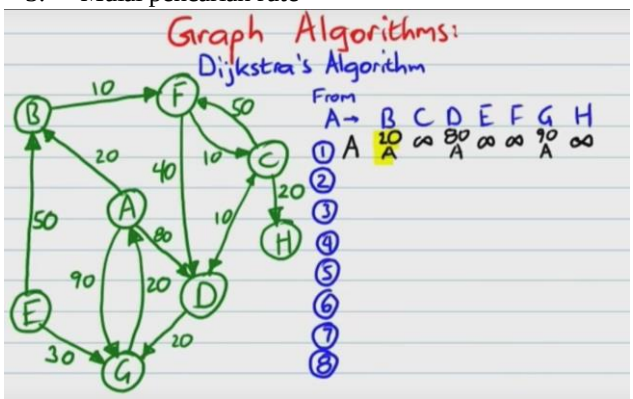


Gambar 5.1
Sumber: https://www.youtube.com/watch?v=8Ls1RqHCOPw

Tentukan node sumber, yang merepresentasikan posisi awal. Pada kasus ini, node sumbernya adalah A.

2. Gambar tabel pencarian
Gambarkan tabel N-1xN sesuai dengan gambar 3.1, dimana N adalah jumlah node yang ada pada graf, dengan node-node tujuan sebagai kolomnya, dan angka 1 sampai N sebagai barisnya.

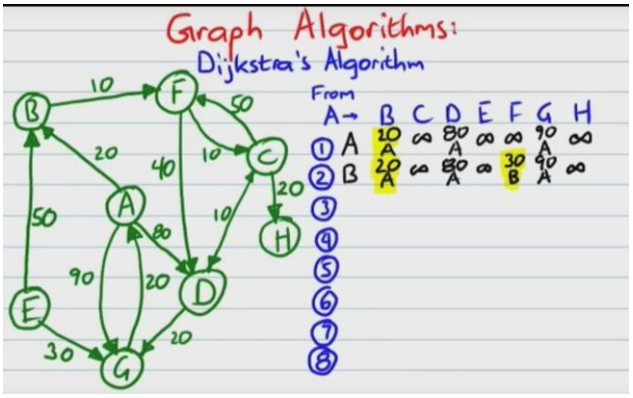
3. Mulai pencarian rute



Gambar 5.2
Sumber: *https://www.youtube.com/watch?v=8Ls1RqHCOPw*

Pencarian rute dilakukan dengan cara traversal baris dari 1 sampai N. Pada baris pertama, dipilih node sumber sebagai patokan. Beban yang ditempuh node sumber untuk menjelajah ke tetangganya ditulis di bawah kolom node tetangga tujuannya, seperti tertera pada Gambar 3.2.

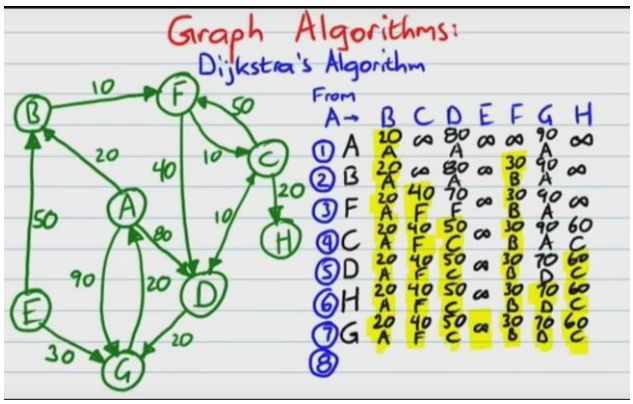
Huruf "A" di bawah besar beban merupakan node yang menjadi media untuk berpindah ke node tujuan. Dalam kasus ini, untuk traversal ke-1, B diraih dengan A sebagai medianya, begitu pula dengan node D dan G. Bila node bukan merupakan tetangga node sumber, tuliskan simbol tak hingga di bawah kolom node.
Node dengan beban terkecil diberi tanda sebagai penanda bahwa node tersebut telah dikunjungi.



Gambar 5.3
Sumber: https://www.youtube.com/watch?v=8Ls1RqHCOPw

Untuk iterasi ke-2, node patokan sekarang berpindah kepada node yang terakhir dikunjungi, dalam kasus ini adalah node B.
Beban tetangga yang dapat dikunjungi node patokan ditambahkan dengan beban node yang telah dikunjungi sebelumnya. Bila hasil penambahan lebih kecil daripada angka yang telah dicatat sekarang, maka angka yang lama diganti dengan angka yang baru, dengan media yang baru, dalam khasus ini, beban F menjadi 30, dengan media B.

Hal ini dilakukan terus menerus hingga menghasilkan pohon rute terpendek seperti yang dapat dilihat pada gambar 3.4.



Gambar 3.4
Sumber: https://www.youtube.com/watch?v=8Ls1RqHCOPw

VI. IMPLEMENTASI ALGORITMA DIJKSTRA PADA SISTEM NAVIGASI WAZE

Pada implementasinya dalam siistem navigasi, algoritma Dijkstra tidak secara gamblang dipakai untuk mencari rute teroptimal. Walaupun algoritma Dijkstra dipakai sebagai landasan teori pencarian rute teroptimal, banyak algoritma-algoritma yang ditambahkan pada algoritma tsb sehingga runtime yang tadinya 10 detik bila dijalankan dengan algoritma Dijkstra saja, dapat dijalankan hanya dengan satu millisecond [1]. Beberapa proses *speedup* yang ditambahkan adalah sbb:

1. Priority Queues

   Algoritma Djikstra dapat diimplementasikan dengan menggunakan *priority queue* dengan kompleksitas O(n). Meskipun demikian, pada praktiknya, peningkatan kinerja pada jalan yang sangat besar sangat terbatas karena *cache fault* untuk mengakses grafik menjadi kendala utama.
2. Bidirectional Search

   Dengan mengeksekusi algoritma Djikstra ke depan dari sumber dan ke belakang dari tujuan, algoritma pencarian rute dapat dipercepat. Ketika beberapa simpul telah dikunjungi oleh kedua arah, jalur terpendek dapat diturunkan berdasarkan informasi yang sudah didapat. *Bidirectional search* merupakan teknik *speedup* yang sangat penting karena dapat dikombinasikan dengan baik dengan teknik *speedup* lain, dan merupakan hal yang penting pada teknik yang paling modern dan efisien.
3. Geometric Goal Directed Search (*A**)

   Karena algoritma pencarian *A** menggunakan heuristik yang menunjukkan SLD (straight line distance) antara sumber dan tujuan, maka pemakaian algoritma *A** ini merupakan sesuatu yang logis yang dapat diterapkan pada teknik *speedup*, karena "seharusnya" rute tercepat dari sumber menuju target hampir selalu mengarah ke target.
4. Heuristics

   Pada dekade terakhir, aplikasi navigasi komersial harus berfikir tentang cara pemilihan rute tercepat dengan kekuatan prosesor yang relatif rendah. Hal ini menyebabkan aplikasi-aplikasi tsb untuk menggunakan heuristik: jangan mencari ke jalan yang tidak penting, kecuali jika dekat dengan target, gunakan jalan besar lebih banyak daripada jalan kecil, dll. Heuristik seperti demikian harus ditangani dengan *tuning* manual dari jalan-jalan yang ada, jika ingin mendapatkan hasil yang baik, dengan kecepatan pencarian yang rendah.

Beberapa proses *speedup* yang lebih maju adalah *Edge labels*, *Landmark A**, dan *Precomputed Cluster Distances (PCD)*, yang kebanyakan menggunakan data-data yang diproses sebelum algoritma dijalankan (*preprocessing*).

Bukan hanya dengan proses *speedup*, pencarian juga dipercepat secara drastis dengan pengeksploitasian hirearki jalan-jalan yang menjadi sisi. Hal ini dilakukan dengan cara memproses hirearki yang sudah ditentukan sendiri sebelum program mencari rute (*preprocessing*). Hal ini memungkinkan untuk membuat algoritma semakin mangkus karena jalan yang jauh dari start tidak akan diperhitungkan oleh algoritma, sehingga mengakibatkan pengingkatan performa dan akurasi hasil.

Beberapa hierarki jalan yang digunakan dalam algoritma pencarian rute terpendek adalah:

1. Small Seperators

   Jalan-jalan di kota hampir planar, maka dari itu dapat diasumsikan bahwa jalan-jalan dapat direpresentasikan dalam graf planar yang penyelesaiannya dapat dicapai dengan $O(n \log^2 n)$ waktu dan tempat *preprocessing* dan $O(\sqrt{n} \log n)$ *query time*.

2. Reach-Based Routing

   Pencarian rute terpendek dapat dihentikan bila *reach* terlalu kecil untuk sampai ke node tujuan dari node yang sekarang sedang diproses. Rumus reach adalah:

   $$R(v) := \max_{s,t \in V} R_{st}(v)$$

   di mana

   $$R_{st}(v) := \min(d(s, v), d(v, t))$$

3. Highway Hirearchies (HHs)

   Mengelompokkan simpul dan sisi dalam level hierarki dengan mengalternatifkan dua prosedur: kontraksi, menghilangkan simpul derajat rendah dengan mem-*bypass* mereka dengan jalan pintas baru.

4. Advanced Reach-Based Routing

   Mengimplementasikan HHs ke dalam *preprocess* reach pada Reach-Based Routing.

5. Highway-Node Routing

   Menggeneralisasi *multi-level routing* dengan graf overlay sehingga bekerja dengan simpul2 yang *arbitrary*, bukan hanya dengan *separator*.

6. Distance Tables

   Menggunakan tabel ini, kita dapat menghentikan pencarian HH saat mencapai level L (disaat level memiliki ukuran $\Theta(\sqrt{n})$).

7. Transit Node Routing

   Menghitung bukan hanya Distance Table untuk transit yang penting tetapi juga semua koneksi relevan antara simpul yang tersisa.

Dengan kombinasi antara proses-proses di atas dengan preprocessing hierarchy, maka dapat dirancang suatu algoritma yang kompleks namun memiliki runtime yang

jauh lebih baik daripada algoritma Dijkstra biasa..

VII. KESIMPULAN

Pengaplikasian algoritma Dijsktra dalam system pencarian rute teroptimal pada fitur navigasi piranti lunak Waze sangat penting. Modifikasi terhadap algoritma Dijsktra menyebabkan runtime yang jauh lebih cepat daripada algoritma Dijkstra yang biasa. Sistem pencarian rute teroptimal masih terus berkembang dan patut untuk diapresiasi.

UCAPAN TERIMAKASIH

Penulis mengucapkan terimakasih kepada Tuhan YME yang dengan berkat-Nya penulis dapat menyelesaikan makalah Strategi Algoritma ini. Penulis juga mengucapkan terimakasih kepada Dr.Ir. Rinaldi Munir, MT. selaku dosen mata kuliah Strategi Algoritma, untuk semua pelajaran yang telah diberikan, terutama pelajaran graf yang menjadi dasar penulisan makalah ini.

REFERENSI

[1] http://stackoverflow.com/questions/430142/what-algorithms-compute-directions-from-point-a-to-point-b-on-a-map
diakses 10 Desember 2015 21.00 WIB

[2] https://www.youtube.com/watch?v=8Ls1RqHCOPw
diakses 10 Desember 2015 20.00 WIB

[3] Slide Kuliah Rinaldi Munir Graf 2015

[4] Peter Sanders and Dominik Schultes
Universit¨at Karlsruhe (TH), 76128 Karlsruhe, Germany, “Engineering Fast Route Planning Algorithms”.

[5] C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.

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, 8 Mei 2016

Alson Cahyadi 13514035