PERBANDINGAN ALGORITMA DIJKSTRA DAN ALGORITMA ANT COLONY DALAM PENENTUAN JALUR TERPENDEK

SKRIPSI

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

disusun oleh :

FINSA FERDIFIANSYAH

NIM. 0710630014 - 63

KEMENTERIAN PENDIDIKAN NASIONAL
UNIVERSITAS BRAWIJAYA
FAKULTAS TEKNIK
MALANG
2013

LEMBAR PERSETUJUAN

PERBANDINGAN ALGORITMA DIJKSTRA DAN ALGORITMA ANT COLONY DALAM PENENTUAN JALUR TERPENDEK

SKRIPSI

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

disusun oleh :

FINSA FERDIFIANSYAH

NIM. 0710630014 - 63

Telah mengetahui dan disetujui oleh:

Dosen Pembimbing I,	Dosen Pembimbing II,
Adharul Muttaqin ,ST.,MT.	**Ir. Muhammad Aswin, MT.**
NIP. 19760121 200501 1 001	**NIP. 19650626 19902 1 001**

LEMBAR PENGESAHAN

PERBANDINGAN ALGORITMA DIJKSTRA DAN ALGORITMA ANT COLONY DALAM PENENTUAN JALUR TERPENDEK

SKRIPSI
JURUSAN TEKNIK ELEKTRO

Diajukan Untuk Memenuhi Persyaratan
Memperoleh Gelar Sarjana Teknik

Disusun oleh:

FINSA FERDIFIANSYAH
NIM. 0710630014-63

Skripsi ini telah diuji dan dinyatakan lulus pada
Tanggal 23 April 2013

DOSEN PENGUJI

Waru Djuriatno, ST., MT.
NIP.19690725 199702 1 001

R. Arief Setyawan, ST., MT.
NIP.19750819 199903 1 001

Rusmi Ambarwati, S.T., M.T.
NIP. 19720204 200003 2 002

Mengetahui,
Ketua Jurusan Teknik Elektro

Dr. Ir. Sholeh Hadi Pramono, MS
NIP. 19580728 198701 1 001

KATA PENGANTAR

Puji syukur kepada Allah SWT yang telah memberikan segala nikmat dan rahmat-Nya sehingga penyusunan skripsi ini dapat diselesaikan. Karena hanya dengan pertolongan-Nya semata penulis mampu melewati segala kendala yang ada selama penyusunan skripsi ini.

Skripsi berjudul "*Perbandingan Algoritma Dijkstra Dan Algoritma Ant Colony Dalam Penentuan Jalur Terpendek*" ini disusun sebagai salah satu syarat untuk mendapatkan gelar Sarjana Teknik di Jurusan Teknik Elektro, Fakultas Teknik, Universitas Brawijaya. Terselesaikannya skripsi ini tentunya tidak lepas juga dari bantuan berbagai pihak. Oleh sebab itu, dengan segala kerendahan hati penulis menyampaikan terima kasih kepada:

1. Kedua Orang Tua serta seluruh keluarga penulis yang senantiasa memberikan do'a, nasihat, motivasi serta segalanya yang tulus dalam menyelesaikan skripsi ini.
2. Bapak Dr. Ir. Sholeh Hadi Pramono, MS selaku Ketua Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya.
3. Bapak M. Azis Muslim, ST.,MT.,Ph.D selaku Sekretaris Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya.
4. Bapak Waru Djuriatno, ST., MT selaku KKDK Teknik Informatika dan Komputer Teknik Elektro yang telah memberikan arahan dalam pengambilan judul skripsi ini.
5. Bapak Adharul Muttaqin ST.,MT selaku dosen pembimbing I atas bantuan dan motivasi serta bimbingannya selama ini.
6. Bapak Ir. Muhammad Aswin, MT selaku dosen pembimbing II atas bantuan dan motivasi serta bimbingannya selama ini.
7. Bapak dan Ibu Dosen serta karyawan jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya atas kesediannya membagi ilmu.
8. Bapak Hadi Suyono, ST., MT., Ph.D selaku dosen pembimbing akademik atas nasehatnya selama masa perkuliahan.

9. Nuning Dwi Puspa, Arvian MN, Agung Pramudhita, Akhmad Adiasta, Galih Candra, penghuni Kos 348A serta anggota KIASS atas bantuan, nasehat, dukungan dan persahabatannya.
10. Teman-teman Teknik Elektro khususnya CORE 2007 dan seluruh staff PT PATTINDO khususnya tim IT yang setia menemani, memberikan ilmu dan kesempatan untuk belajar.
11. Semua pihak yang telah banyak membantu penyelesaian skripsi ini.

Penulis menyadari sepenuhnya bahwa skripsi ini masih banyak kekurangan. Segala kritik dan saran yang membangun akan penulis terima demi kesempurnaan skripsi ini. Harapan penulis semoga skripsi ini bermanfaat bagi pembaca dan khususnya mahasiswa jurusan teknik elektro dimasa yang akan datang.

Malang, April 2013

Penyusun

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

FINSA FERDIFIANSYAH, Jurusan Teknik Elektro, Fakultas Teknik Universitas Brawijaya, April 2013, *Perbandingan Algoritma Dijkstra Dan Algoritma Ant Colony Dalam Penentuan Jalur Terpendek.* Dosen Pembimbing : Adharul Muttaqin, ST., MT dan Ir. Muhammad Aswin, MT.

Pencarian jalur terpendek merupakan pencarian sebuah jalur pada graf berbobot yang meminimalkan jumlah bobot sisi pembentuk jalur tersebut. Dengan begitu jalur yang dihasilkan merupakan jalur yang memiliki bobot atau jarak yang paling sedikit. Salah satu penerapan pencarian jalur terpendek terdapat pada aktivitas maskapai penerbangan yang mana jalur-jalur antar kota yang dilewatinya akan membentuk suatu graf berarah dan berbobot. Dari graf yang terbentuk inilah akan diproses menggunakan algoritma Dijkstra dan Ant Colony untuk menentukan jalur terpendek dari suatu kota ke kota yang lain.

Pada proses Algoritma, Dijkstra memerlukan data jarak setiap kota terlebih dahulu sebelum memulai proses algortimanya. Sedangkan pada Algoritma Ant Colony, tidak memerlukan jarak setiap kota karena pada Ant Colony jarak antar kota dihitung setelah semut menyelesaikan perjalanannya. Sehingga Algoritma Dijkstra hanya bisa berjalan jika terlebih dahulu diketahui jarak tiap kota, sedangkan pada Algoritma Ant Colony tidak memerlukan jarak tiap kota untuk menjalankan prosesnya. Dari hasil proses kedua algoritma diketahui jalur yang dihasilkan oleh algoritma Dijkstra lebih konsisten dan tepat daripada algoritma Ant Colony yang mana memberikan hasil yang belum tentu sama dalam setiap prosesnya. Rata-rata memory yang digunakan pada algoritma Dijkstra sebesar 82,204 KB dan algoritma Ant Colony sebesar 90,404 KB. Sedangkan dari analisa kompleksitas waktu pada algoritma Dijkstra diperoleh persamaan $\boldsymbol{f(n) = O(2n + 13)}$ dan pada Algoritma Ant Colony persamaannya $\boldsymbol{f(n) = O(5n + 3)}$

BAB I

PENDAHULUAN

1.1 Latar Belakang

Pencarian jalur terpendek merupakan pencarian sebuah jalur pada graf berbobot yang meminimalkan jumlah bobot sisi pembentuk jalur tersebut. Dengan begitu jalur yang dihasilkan merupakan jalur yang memiliki bobot atau jarak yang paling sedikit.

Salah satu penerapan pencarian jalur terpendek terdapat pada aktivitas maskapai penerbangan. Maskapai penerbangan dalam menyediakan jasa perjalanannya tidak sepenuhnya dari kota asal ke kota tujuan secara langsung, melainkan mereka memiliki jalur perjalanannya sendiri yang mana jalur tersebut menghubungkan antara kota atau bandara dengan kota terdekat lainnya. Maka, untuk mencari jalur terpendek dalam aktivitasnya itu harus diketahui terlebih dahulu jalur-jalur antar kota yang akan dilewati dan kemudian merepresentasikannya dalam sebuaf graf berarah dan berbobot. Dari graf yang terbentuk inilah akan diproses menggunakan suatu algoritma untuk menentukan jalur terpendek dari suatu kota ke kota yang lain.

Dalam ilmu komputer, untuk menentukan jalur terpendek diperlukan suatu algoritma. Algoritma yang bisa digunakan untuk menentukan jalur terpendek adalah algoritma Dijkstra dan Ant Colony.

1.2 Rumusan Masalah

Perumusan masalah dalam tulisan ini adalah perbandingan algoritma *Dijkstra* dan *Ant Colony* dalam penentuan jalur terpendek adalah :

1. Mengetahui bagaimana karakteristik masing-masing Algoritma dalam penentuan jalur terpendek.
2. Bagaimana penerapan Algoritma Dijkstra dan Ant Colony dalam penentuan jalur terpendek.

1.3 Batasan Masalah

Mengacu pada permasalahan yang telah dirumuskan, maka hal-hal yang berkaitan dengan sistem informasi yang akan dibuat, diberi batasan sebagai berikut:

1. Data yang dibandingkan adalah jarak yang diperoleh, memory yang dibutuhkan dan kecepatan proses berdasarkan kompleksitas algoritma pada kedua algoritma tersebut.
2. Data lokasi kota serta jalur antar kota berasal dari database yang kemudian disinkronasikan dengan Google Map API sebagai media untuk menampilkannya.
3. Pada Algoritma Dijkstra, jarak antar titik atau kota diketahui terlebih dahulu sebelum proses Algoritma berjalan.

1.4 Tujuan

Tujuan dari penulisan tugas akhir (skripsi) ini adalah memberikan informasi mengenai karakteristik serta penerapan algortima *Dijkstra* dan *Ant Colony* dalam penentuan jalur terpendek.

1.5 Manfaat

Manfaat yang diharapkan dapat diperoleh melalui pengerjaan skripsi ini adalah :

1. Lebih memahami karakteristik dan penerapan algoritma Dijkstra dan *Ant Colony* dalam penentuan jalur terpendek.
2. Menerapkan ilmu yang telah diperoleh selama menjadi mahasiswa Teknik Elekto Konsentrasi Teknik Informatika dan Komputer Universitas Brawijaya.
3. Sebagai bahan referensi dan bahan kajian lebih lanjut untuk peneliti maupun pengembang berikutnya.

1.6 Sistematika Penulisan

Sistematika penulisan dan gambaran untuk setiap bab pada laporan skripsi ini adalah sebagai berikut :

BAB I Pendahuluan

Menjelaskan latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika penulisan skripsi.

BAB II Dasar Teori

Menjelaskan kajian pustaka dan dasar teori yang digunakan.

BAB III Metodologi

Menjelaskan metode yang digunakan dalam pengerjaan skripsi.

BAB IV Perancangan

Menjelaskan analisis kebutuhan dan perancangan perangkat lunak serta menerapkan teknologi yang akan digunakan.

BAB V Implementasi

Menjelaskan langkah-langkah pembuatan aplikasi untuk menetukan jalur atau lintasan yang terpendek dengan algoritma *Dijkstra* dan *Ant Colony*.

BAB VI Pengujian

Menjelaskan langkah-langkah pengujian dari sistem yang telah dibuat dan membahas hasil pengujiannya.

BAB VII Kesimpulan dan Saran

BAB II

DASAR TEORI

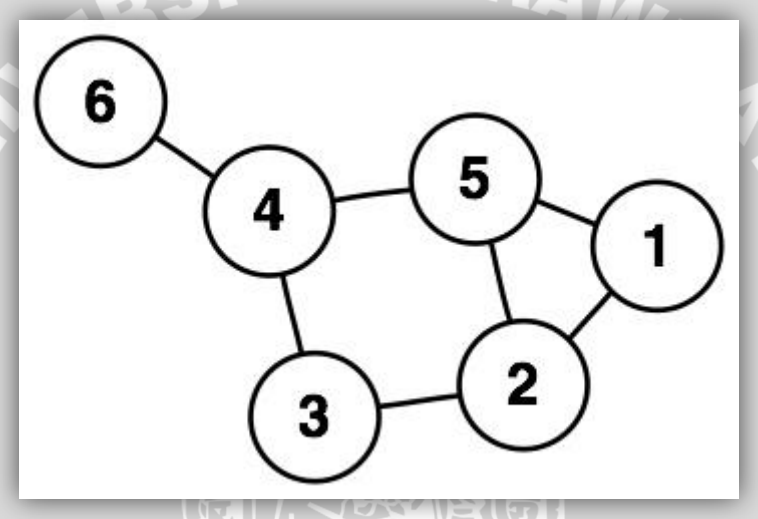
Pada bab ini dibahas mengenai dasar teori yang digunakan untuk menunjang penulisan skripsi mengenai Perbandingan Algoritma *Dijkstra* dan *Ant Colony* Dalam Penentuan Jalur Terpendek. Beberapa dasar teori yang dimaksud diantaranya adalah Teori Graf, Algoritma *Dijkstra*, Algoritma *Ant Colony*, Google Map API, dan Kompelsitas Algoritma.

2.1 Teori Graf

Dalam matematika dan ilmu komputer, teori graf adalah cabang kajian yang mempelajari sifat-sifat graf. Secara informal, suatu graf adalah himpunan benda-benda yang disebut simpul (*vertex* atau *node*) yang terhubung oleh sisi (*edge*) atau busur (*arc*). Biasanya graf digambarkan sebagai kumpulan titik-titik (melambangkan simpul) yang dihubungkan oleh garis-garis (melambangkan sisi) atau garis berpanah (melambangkan busur). Suatu sisi dapat menghubungkan suatu simpul dengan simpul yang sama. Sisi yang demikian dinamakan gelang (*loop*).

Banyak sekali struktur yang bisa direpresentasikan dengan graf, dan banyak masalah yang bisa diselesaikan dengan bantuan graf. Jaringan persahabatan pada jejaring sosial bisa direpresentasikan dengan graf: simpul-simpulnya adalah para pemakai jejaring sosial dan ada sisi antara A dan B jika dan hanya jika A berteman (berkoinsidensi) dengan B. Perkembangan algoritma untuk menangani graf akan berdampak besar bagi ilmu komputer.

Sebuah struktur graf bisa dikembangkan dengan memberi bobot pada tiap sisi. Graf berbobot dapat digunakan untuk melambangkan banyak konsep berbeda. Sebagai contoh jika suatu graf melambangkan jaringan jalan maka bobotnya bisa berarti panjang jalan maupun batas kecepatan tertinggi pada jalan tertentu. Ekstensi lain pada graf adalah dengan membuat sisinya berarah, yang secara teknis disebut graf berarah atau digraf (*directed graph*). Digraf dengan sisi berbobot disebut jaringan.



Gambar 2.1 Graf dengan 6 simpul dan 7 sisi.

Jaringan banyak digunakan pada cabang praktis teori graf yaitu analisis jaringan. Perlu dicatat bahwa pada analisis jaringan, definisi kata "jaringan" bisa berbeda, dan sering berarti graf sederhana (tanpa bobot dan arah).

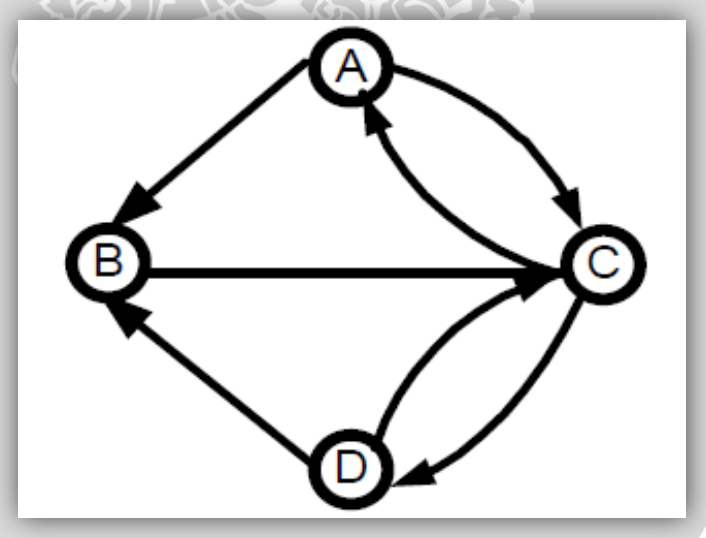
2.1.1 Representasi Graf Dalam Komputer

Untuk mengimplementasikan suatu algoritma graf dalam bahasa pemrograman komputer, dibutuhkan suatu cara untuk menterjemahkan bentuk graf ke dalam bentuk lain yang dikenal oleh komputer, sebab komputer tidak

dapat mengenal graf dalam bentuk gambar graf biasa. Matriks dapat digunakan untuk menyatakan suatu graf, jika graf dinyatakan sebagai matriks maka perhitungan-perhitungan yang diperlukan dapat dilakukan dengan mudah. Representasi graf pada komputer dapat dilakukan dengan beberapa cara, yaitu:

1. Matriks *Adjacency*

Matriks *adjacency* didefinisikan sebagai berikut, misalkan A matriks berordo $n \times n$ (n baris dan n kolom). Jika antara dua *vertex* terhubung (*adjacent*) maka elemen matriks bernilai 1, dan sebaliknya jika tidak terhubung bernilai 0. Matriks *adjacency* dari graf berarah dibawah ini adalah:



Gambar 2.2. Contoh Graf Berarah

$$Adjacency = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Jika graf yang diberikan adalah graf berbobot maka elemen matriks yang terhubung antara *vertex* adalah bobot graf.

2. Matriks *Incidency*

Matriks *incidency* atau matriks bersisian adalah matriks yang merepresentasikan hubungan antara *vertex* dan *edge*. Misalkan B adalah matriks dengan m baris untuk setiap *vertex* dan n kolom untuk setiap *edge*. Jika vertex terhubung dengan *edge*, maka elemen matriks bernilai 1. Sebaliknya, jika *vertex* tidak terhubung dengan *edge* maka elemen matriks bernilai 0. Matriks bersisian dari graf Gambar 2.2 adalah :

$$Incidency = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Seperti halnya matriks kedekatan untuk matriks berbobot, untuk matriks bersisian elemen dari matriks juga merupakan bobot dari setiap edge dari graf.

2.1.2 Lintasan Terpendek

Lintasan terpendek merupakan bagian dari teori graf. Jika diberikan sebuah graf berbobot, masalah jarak terpendek adalah bagaimana kita mencari sebuah jalur pada graf yang meminimalkan jumlah bobot sisi pembentuk jalur tersebut.

Contoh-contoh terapan pencarian lintasan terpendek :

1. Misalkan simpul pada graf dapat merupakan kota, sedangkan sisi menyatakan jalan yang menghubungkan dua buah kota. Bobot sisi

graf dapat menyatakan jarak antara dua buah kota atau rata-rata waktu tempuh antara dua buah kota. Apabila terdapat lebih dari satu lintasan dari kota A ke kota B, maka persoalan lintasan terpendek di sini adalah menentukan jarak terpendek atau waktu tersingkat dari kota A ke kota B.

2. Misalkan simpul pada graf dapat merupakan terminal komputer atau simpul komunikasi dalam suatu jaringan, sedangkan sisi menyatakan saluran komunikasi yang menghubungkan dua buah terminal. Bobot pada graf dapat menyatakan biaya pemakaian saluran komunikasi antara dua buah terminal, jarak antara dua buah terminal, atau waktu pengiriman pesan (*message*) antara dua buah terminal. Persoalan lintasan terpendek di sini adalah menentukan jalur komunikasi terpenek antara dua buah terminal komputer. Lintasan terpendek akan menghemat waktu pengiriman pesan dan biaya komunikasi.

Ada beberapa macam persoalan lintasan terpendek, antara lain:

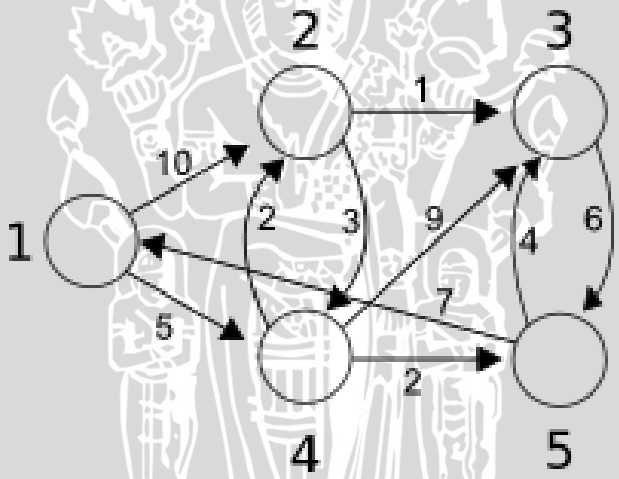
1. Lintasan terpendek antara dua buah simpul tertentu (*a pair shortest path*).
2. Lintasan terpendek antara semua pasangan simpul (*all pairs shortest path*).
3. Lintasan terpendek dari simpul tertentu ke semua simpul yang lain (*single-source shortest path*).
4. Lintasan terpendekan antara dua buah simpul yang melalui beberapa simpul tertentu (*intermediate shortest path*).

2.2 Algoritma *Dijkstra*

Algoritma *Dijkstra*, (dinamai menurut penemunya, seorang ilmuwan komputer, Edsger Dijkstra), adalah sebuah algoritma rakus (*greedy algorithm*) yang dipakai dalam memecahkan permasalahan jarak terpendek (*shortest path problem*) untuk sebuah graf berarah (*directed graph*) dengan bobot-bobot sisi (*edge weights*) yang bernilai tak-negatif. Algoritma Dijkstra adalah salah satu solusi yang paling umum untuk masalah ini.

2.2.1 Skema Algoritma Dijkstra

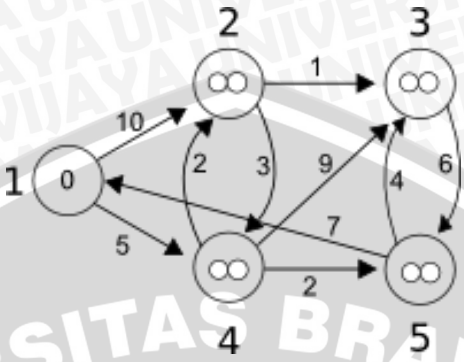
Pada graf berikut akan menentukan jalur terpendek dari *node* 1 ke *node* 3.



Gambar 2.3 Graf menentukan jalur terpendek dari *node* 1 ke *node* 3

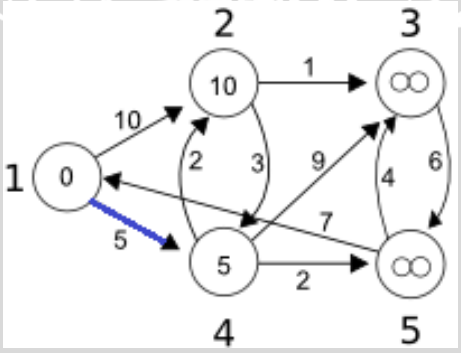
Ada beberapa jalur yang mungkin misalnya **1 -> 4 -> 3, 1 -> 2 -> 3** dan lain-lain, tetapi cara yang paling efektif adalah :

a. Menentukan *node* awal dengan memberikan nilai 0 pada *node* tersebut dan memberikan nilai tak hingga (∞) pada *node* yang lainnya



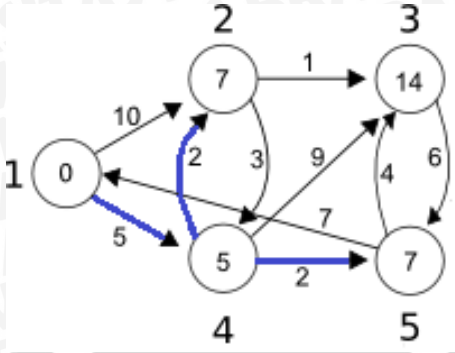
Gambar 2.4 *node* awal (1) bernilai 0 dan lainnya bernilai tak hingga (∞)

b. Menentukan bobot paling kecil untuk melangkah ke *node* selanjutnya. Pada gambar diketahui bahwa untuk melangkah ke *node* 5 memiliki bobot terkecil, kemudian memberikan nilai atau label permanen pada *node* 4 yaitu bernilai 5.



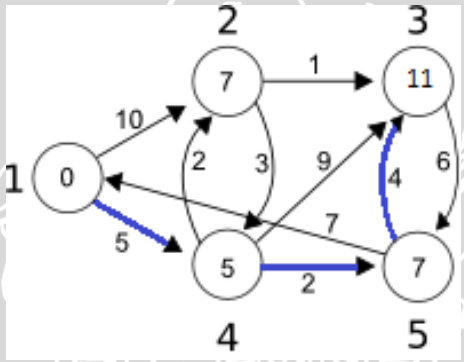
Gambar 2.5 Menentukan bobot minimum

c. Mencari *node* terpendek berikutnya yaitu yang memiliki bobot minimum dengan membandingkan nilai biaya menuju node tersebut atau melalui *node* yang telah memiliki nilai permanen.

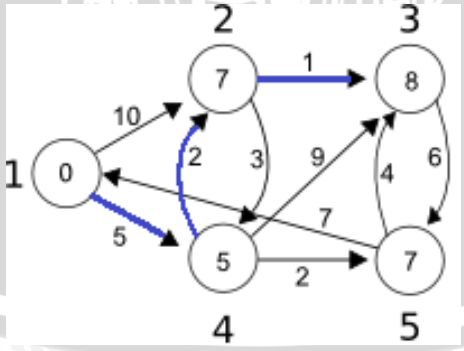


Gambar 2.6 Menentukan bobot minimum untuk menuju *node* selanjutnya

d. *Node* 2 dan *node* 5 memiliki nilai yang sama, maka selanjutnya dilakukan pencarian jalur yang paling efektif dari kedua *node* tersebut



Gambar 2.7 Menentukan jalur yang paling efektif dari *node* 5



Gambar 2.8 Menentukan jalur yang paling efektif dari *node* 2

e. Setelah dibandingkan, ternyata jalur dari *node* 2 memiliki nilai paling minimum. Sehingga bisa ditentukan jalur terpendek dari *node* 1 ke *node* 3 adalah **1 » 4 » 2 » 3**

2.2.2 Pseudocode Algoritma Dijkstra

```
1  function Dijkstra(Graph, source):
2      for each vertex v in Graph:
3          dist[v] := infinity ;
4          previous[v] := undefined ;
5      end for
6
7      dist[source] := 0 ;
8      Q := the set of all nodes in Graph ;
9
10     while Q is not empty:
11         u := vertex in Q with smallest distance in dist[] ;
12         remove u from Q ;
13         if dist[u] = infinity:
14             break ;
15         end if
16
17         for each neighbor v of u:
18             alt := dist[u] + dist_between(u, v) ;
19             if alt < dist[v]:
20                 dist[v] := alt ;
21                 previous[v] := u ;
22                 decrease-key v in Q;
23             end if
24         end for
25     end while
26 return dist;
```

2.3 Algoritma Semut (*Ant Colony*)

Algoritma Semut (*Ant Colony*) diadopsi dari perilaku koloni semut yang dikenal sebagai sistem semut (Dorigo, 1996). Secara alamiah koloni semut mampu menemukan rute terpendek dalam perjalanan dari sarang ke tempat-tempat sumber makanan. Koloni semut dapat menemukan rute terpendek antara sarang dan sumber makanan berdasarkan jejak feromon pada lintasan yang telah dilalui. Semakin banyak semut yang melalui suatu lintasan, maka akan banyak jalur yang dibuat dan memungkinkan semakin besar untuk menentukan dari jalur terpendek yang dibuat oleh semut-semut tadi.

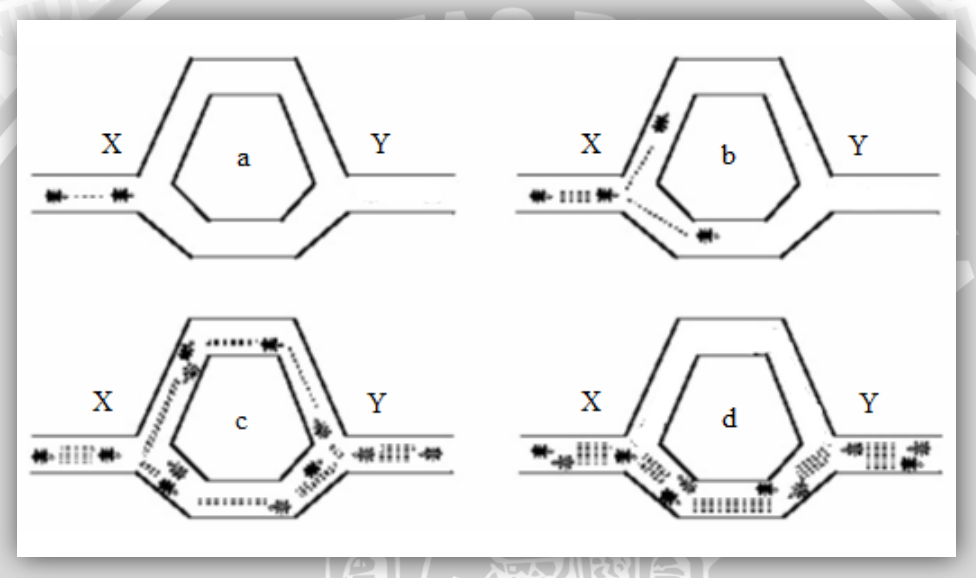
Perilaku semut itu telah menginspirasikan sebuah metodologi baru dalam mencari solusi terhadap suatu permasalahan kombinatorial dengan cara bertukar informasi melalui feromon yang diletakkan pada *edge-edge* sebuah graf. Telah diamati bahwa pada saat berjalan semut meninggalkan sejumlah feromon pada jalur yang dilaluinya untuk menandai jalur tersebut, sehingga semut lain yang mengikuti jalur tersebut dapat mengidentifikasi feromon yang diletakkan semut sebelumnya, memutuskan untuk mengikutinya dan menguatkan jalur tersebut dengan feromon miliknya. Perilaku inilah yang menjelaskan bagaimana semut mampu menemukan jalur terpendek. Bentuk komunikasi yang diperantarai feromon ini di sebut *stigmergy*.

2.3.1 Skema Algoritma *Ant Colony*

Secara logika dan matematik semut-semut dari titik asal yang sama, misalnya sarang, bergerak sendiri-sendiri melewati jalur masing-masing menuju makanan sebagai titik tujuannya. Pada saat berjalan semut tersebut meninggalkan

feromon sebagai jejaknya. Jejak inilah yang kemudian diikuti oleh semut yang lainnya, semakin banyak semut yang melalui jalur tersebut maka jejaknya akan semakin kuat hingga semua semut melewati jejak tersebut. Setelah semua semut melewati jejak tersebut bisa disimpulkan jalur itulah sebagai jalur terpendek yang dilalui oleh semut-semut tersebut.

Berikut adalah tahapan-tahapan algoritma Semut dalam graf :



Gambar 2.9 Perjalanan semut menemukan sumber makanan.

1. Pada gambar 2.9.a menunjukkan semut yang akan melakukan perjalanan mencari tempat dimana ada makanan, dari X menuju Y.
2. Semut akan melakukan gerakan secara acak menuju tempat makanan dengan jalur masing-masing. Seperti pada gambar 2.9.b semut berjalan dengan jalurnya masing-masing.

3. Setelah berjalan secara acak berdasarkan jalurnya masing-masing kemudian semut-semut tersebut akan bertemu lagi dimana tempat makanan berada seperti pada gambar 2.9.c
4. Pada saat melakukan perjalanan melalui jalurnya masing-masing, semut meninggalkan feromon sebagai jejak yang akan diikuti oleh semut yang lainnya. Semakin banyak semut dan semakin dekat jarak yang ditempuh maka feromon juga semakin kuat sehingga semut yang lainnya akan mengikuti jalur tersebut.
5. Pada optimisasi algoritma semut, proses tadi akan dilakukan secara berulang sesuai dengan siklus maksimum yang telah ditentukan.

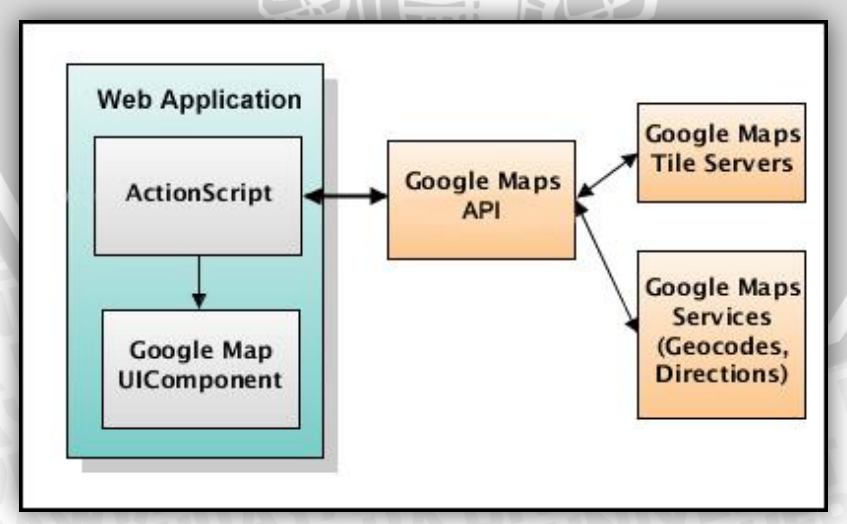
2.3.2 Pseudocode *Algoritma Ant Colony*

```
1  Inisialisasi parameter α,β,τ0,m,Q
2  Inisialisasi titik pertama atau asal semut
3  begin
4      η(a,b) = 1/D(a,b)
5      τ(a,b) = τ0
6
7      // calculate next path, nect feromon and distance
8      function Probabilistik (from)
9          for m = 1; m++ : m
10             for t = 1; t++ : siklus
11                 P_ab,m(t) = [τ_ab(t)]^α [η_ab(t)]^β / Σ_tϵb_ak [τ_ab(t)]^α [η_ab(t)]^β
12             end for
13         end for
14     return dist, path;
15
16     function updateFeromon ()
17         Δτ_ab = Q/Lk
18         next τ = ρ.τ_ab + Δτ_ab
19     return next τ
20
```

```
21      // choice distance & shortpath
22      for x = 1; x++
23           if (min (dist(x)))
24                shortpath = path;
25           end if
26      end for
27  end;
```

2.4 Google Map API

Google Maps adalah layanan gratis Google yang cukup popular, sebuah jasa peta globe virtual gratis dan online disediakan oleh Google dapat ditemukan di http://maps.google.com Anda dapat menambahkan fitur Google Maps dalam web Anda sendiri dengan Google Maps API. Google Maps API adalah library JavaScript. Menggunakan/memprogram Google Maps API sangat mudah. Yang Anda butuhkan adalah pengetahuan tentang HTML dan JavaScript, serta koneksi Internet. Dengan menggunakan Google Maps API Anda dapat menghemat waktu dan biaya Anda untuk membangun aplikasi peta digital yang handal, sehingga Anda dapat focus hanya pada data-data Anda.



Gambar 2.10 Blok diagram Google Map API

Dalam suatu program yang sederhana, dibutuhkan setidaknya ribuan *system calls* per detik. Oleh karena itu Kebanyakan *programmer* membuat aplikasi dengan menggunakan *Application Programming Interface*(API). Dalam API itu terdapat fungsi-fungsi/perintah-perintah untuk menggantikan bahasa yang digunakan dalam *system calls* dengan bahasa yang lebih terstruktur dan mudah dimengerti oleh *programmer*. Fungsi yang dibuat dengan menggunakan API tersebut kemudian akan memanggil *system calls* sesuai dengan sistem operasinya. Tidak tertutup kemungkinan nama dari *system calls* sama dengan nama di API.

Keuntungan memprogram dengan menggunakan API adalah:

- **Portabilitas.** Programmer yang menggunakan API dapat menjalankan programnya dalam sistem operasi mana saja asalkan sudah ter- *install* API tersebut. Sedangkan *system call* berbeda antar sistem operasi, dengan catatan dalam implementasinya mungkin saja berbeda.
- **Lebih Mudah Dimengerti.** API menggunakan bahasa yang lebih terstruktur dan mudah dimengerti daripada bahasa *system call*. Hal ini sangat penting dalam hal editing dan pengembangan.

System call interface ini berfungsi sebagai penghubung antara API dan *system call* yang dimengerti oleh sistem operasi. *System call interface* ini akan menerjemahkan perintah dalam API dan kemudian akan memanggil *system calls* yang diperlukan.

2.5 Kompleksitas Algoritma

Algoritma adalah salah satu konsep matematis dan analisis untuk membantu seseorang dalam menyelesaikan suatu masalah atau persoalan.

Algoritma merupakan urutan langkah yang tepat dan pasti dalam memecahkan suatu masalah secara logis. Beberapa masalah dapat diselesaikan dengan algoritma bermacam-macam asal hasilnya sama. Algoritma dapat dianalisis efisiensi dan kompleksitasnya.

2.5.1 Kompleksitas Waktu

Efisiensi di dalam algoritma sangat dipertimbangkan karena suatu masalah dapat diselesaikan dengan berbagai macam cara. Algoritma yang bagus adalah algoritma yang efisien yang mana algoritma tersebut dikatakan bagus karena dinilai dari aspek kebutuhan waktu dan ruang membutuhkan jumlah yang sedikit.

Kinerja algoritma dapat diukur dari orde yang terdapat di dalam persamaan kompleksitas waktu. Kompleksitas waktu diukur dari jumlah tahapan komputasi di yang dibutuhkan dalam menjalankan algoritma dimana kompleksitas waktu tersebut merupakan fungsi dari jumlah masukan (n).

2.5.2 Notasi Big-O

Tugas yang dilakukan oleh komputer untuk menyelesaikan masalah biasanya berupa tugas yang serupa, tetapi dilakukan berulang-ulang (iterasi). Banyaknya perulangan yang harus dilakukan oleh komputer menentikan lama

waktu proses (running time). Sering kali jumlah perulangan yang harus dilakukan dipengaruhi oleh jumlah data yang harus diproses.

Seperti kata pepatah "banyak jalan menuju Roma" sering kali didapati beberapa algoritma yang berbeda untuk menyelesaikan suatu masalah tertentu dalam komputer. Sebagai contoh, pengurutan sejumlah data (sorting). Lama pengurutan dipengaruhi oleh banyaknya data yang diurutkan (di samping faktor-faktor lain). Meskipun demikian, ada metode pengurutan yang memproses lebih cepat dibandingkan metode-metode lain meskipun jumlah datanya sama. Jika jumlah data (biasanya disimbolkan dengan n) sedikit, perbedaan tersebut tidaklah menjadi persoalan. Akan tetapi, untuk n yang besar, perbedaan itu akan terasa karena perbedaan tersebut ada dala skala jam bahkan hari.

Perbedaan waktu proses sebagai fungsi jumlah data yang diproses sangat erat hubungannya dengan laju pertumbuhan (rate of growth) agoritma yang bersangkutan. Laju pertumbuhan menunjukkan faktor kelipatan waktu proses seiring dengan kenaikan jumlah data jika jumlah data dilipatgandakan, beberapa faktor perubahan lama waktu proses yang dibutuhkan?

Dalam komputer, laju pertumbuhan dinyatakan dalam notasi Big-O. Notasi Big-O merupakan suatu niotasi matematika untuk menjelaskan batas atas dari magnitude suatu fungsi dalam fungsi yang lebih sederhana. Dalam dunia ilmu komputer, notasi ini sering digunakan dalam nalisa kompleksitas algoritma.

Notasi Big-O pertama kali diperkenalkan pakar teori bilangan Jerman, Paul Bachman pada tahun 1894, pada bukunya yang berjudul Analytische

Zahlentheorie edisi kedua. Notasi tersebut kemudian dipopulerkan oleh pakar teori bilangan Jerman lainnya, Edmund Landau, dan oleh karena itu terkadang disebut sebagai simbol Landau.

Notasi Big-O sangat berguna menganalisis efisiensi suatu algoritma. Sebagai contoh waktu atau jumlah langkah yang diperlukan oleh suatu algoritma untuk menyelesaikan tugas dengan ukuran n adalah $T(n) = 4n^2 - 2n + 2$. Untuk n yang besar, hasil perhitungan n^2 menjadi dominan, sehingga perhitungan yang lain dapat diabaikan (misalnya saat $n = 500$, $4n^2$ bernilai 1000 kali lebih besar dari $2n$, sehingga mengabaikan $2n + 2$ yang tidak akan membawa efek yang besar pada tujuan utama umumnya). Kemudian koefisien pada polinominal pun dapat dihilangkan dengan alasan yang sama, sehingga dengan notasi Big-O dapat disimpulkan : $T(n) \in O(n^2)$ bahwa algoritma diatas memiliki kompleksitas waktu dengan orde $O(n^2)$.

Untuk membandingkan kompleksitas algoritma yang satu dengan yang lain, dapat digunakan tabel jenis kompleksitas dibawah ini.

Tabel 2.1 Notasi *Big-O*

Notasi	Nama
$O(1)$	Konstan
$O(log^* n)$	Logaritma iterative
$O(\log n)$	Logaritmik
$O([\log\ n]^c)$	Polilogaritmik
$O(n)$	Linier
$O(n \log n)$	Linierithmik, loglinier

$O(n^2)$	Kuadratik
$O(n^c), c < 1$	Polinomial
$O(c^n)$	Eksponensial
$O(n!)$	Faktorial atau kombinatorial
$O(n^{2n})$	-

BAB III

METODE PENELITIAN

3.1 Studi Literatur

Studi literatur berguna untuk memperoleh data dan menjelaskan dasar teori yang digunakan untuk menunjang penulisan skripsi. Teori-teori pendukung tersebut meliputi:

1. Mempelajari teknik–teknik Lintasan terpendek dengan Algoritma *Dijkstra* dan *Ant Colony*
2. Mempelajari teknik – teknik yang ada pada Google Map API
3. Mempelajari metode pencarian data, pencocokan data, pengambilan data dan pengolahan data pada *database*

3.2 Analisa Kebutuhan

Analisis kebutuhan bertujuan untuk mendapatkan semua kebutuhan yang diperlukan untuk membandingkan agoritma *Dijkstra* dan *Ant Colony* dala menentukan jalur terpendek.

3.3 Perancangan

Perancangan aplikasi penentuan jalur terpendek menggunakan algoritma *Dijkstra* dan *Ant Colony* dilakukan setelah semua kebutuhan sistem didapatkan melalui tahap analisis kebutuhan. Diawali dengan melakukan desain database dari aplikasi tersebut serta perancangan antarmuka (Graphical User Interface) dari

aplikasi. Sedangkan perancangan aplikasi berdasarkan blok diagram secara umum serta menjelaskan secara detail mengenai sistem dan langkah-langkahh yang ditempuh oleh tiap algoritma yang digunakan dalam proses menentukan jalur terpendek. Setelah itu, jalur terpendek yang dihasilkan oleh kedua algoritma tersebut akan dibandingkan.

3.4 Implementasi

Implementasi dilakukan dengan mengacu kepada perancangan perangkat lunak. Implementasi perangkat lunak dilakukan dengan menggunakan pemrograman web (PHP, Javascript, HTML, CSS), database MySQL, termasuk pemrograman pada Google Map API serta penerapan algoritma *Dijkstra* dan *Ant Colony* pada pencarian jalur terpendek. Sistem dapat mengalami perubahan dari rancangan sebelumnya pada tahapan ini, bila ternyata diperlukan pergantian.

3.5 Pengujian Sistem

Pengujian dilakukan untuk menjamin dan memastikan bahwa sistem yang telah dirancang memiliki tingkat akurasi yang tinggi. Pengujian dilakukan pada tiap algoritma yang kemudian membandingkan proses dan hasil yang dihasilkan algoritma *Dijkstra* dan *Ant Colony*. Parameter yang digunakan sebagai perbandingan kedua algoritma adalah jalur terpendek yang dihasilkan, memory yang dibutuhkan serta kecepatan proses berdasarkan kompleksitas algoritma yang dalam hal ini menggunakan notasi *Big-O*.

3.6 Kesimpulan dan Saran

Pada tahap ini, diambil dari hasil pengujian dan analisa terhadap aplikasi yang telah dibuat. Tahap Selanjutnya adalah membuat saran untuk perbaikan terhadap penelitian maupun pengembang selanjutnya sehingga dapat menyempurnakan kekurangan-kekurangan yang dari skripsi ini.

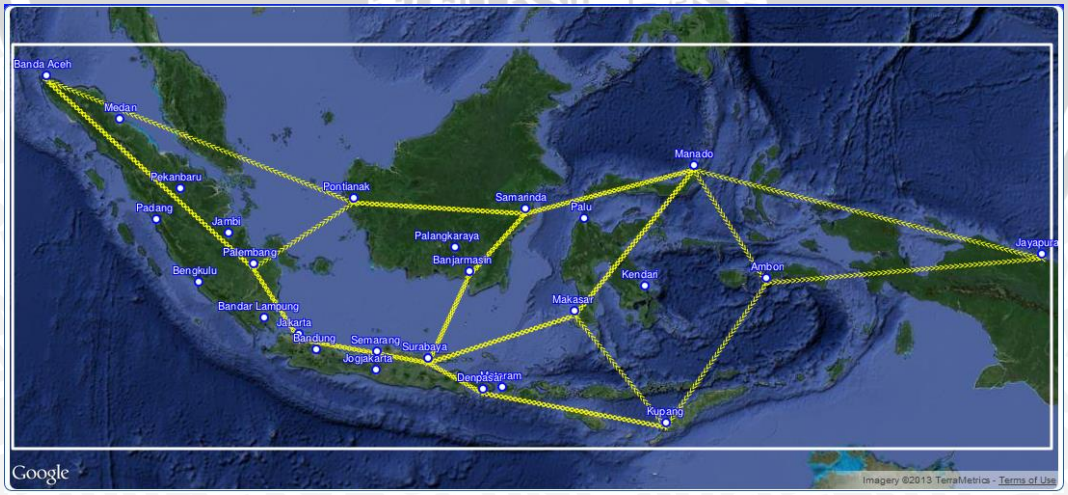
BAB IV

PERANCANGAN DAN IMPLEMENTASI

4.1 Analisa Permasalahan

Pencarian jalur terpendek merupakan pencarian sebuah jalur pada graf berbobot yang meminimalkan jumlah bobot sisi pembentuk jalur tersebut. Dengan begitu jalur yang dihasilkan merupakan jalur yang memiliki bobot atau jarak yang paling sedikit. Salah satu penerapan pencarian jalur terpendek terdapat pada aktivitas maskapai penerbangan yang mana jalur-jalur antar kota yang dilewatinya akan membentuk suatu graf berarah dan berbobot.

Dari graf yang terbentuk inilah akan diproses menggunakan algoritma Dijkstra dan Ant Colony untuk menentukan jalur terpendek dari suatu kota ke kota yang lain. Setelah mendapatkan data graf, kemudian graf tersebut divisualisasikan menggunakan Google Map API seperti pada gambar dibawah ini.



Gambar 4.1 Visualisasi Graf pada Google Map API

4.2 Kebutuhan Sistem

Perangkat lunak ini menggunakan Microsoft© Windows 7 sebagai sistem operasi utamanya. Kebutuhan sistem itu sendiri dapat dibedakan menjadi dua, yaitu kebutuhan perangkat lunak dan perangkat keras.

4.2.1 Perangkat Keras

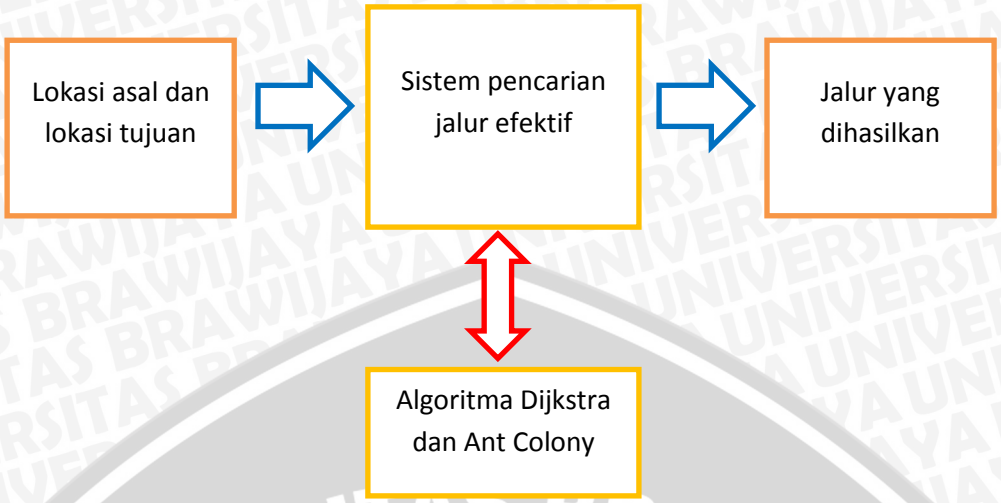
Untuk perangkat kerasnya, membutuhkan komputer dengan spesifikasi minimum Pentium IV dengan RAM 512 Mbytes. Penulis mengaplikasikan perangkat lunak ini pada sebuah notebook Intel® Core™i5-2450M CPU @2.50 GHz 62-bit dengan RAM 4,00 GB.

4.2.2 Perangkat Lunak

Untuk perangkat lunak yang digunakan dalam pembuatan aplikasi ini ialah sistem operasi Microsoft© Windows 7, XAMPP, dan Adobe Dreamweaver CS5. Selain itu juga dibutuhkan web browser dan jaringan internet untuk bisa mengkses atau menjalankan apikasi.

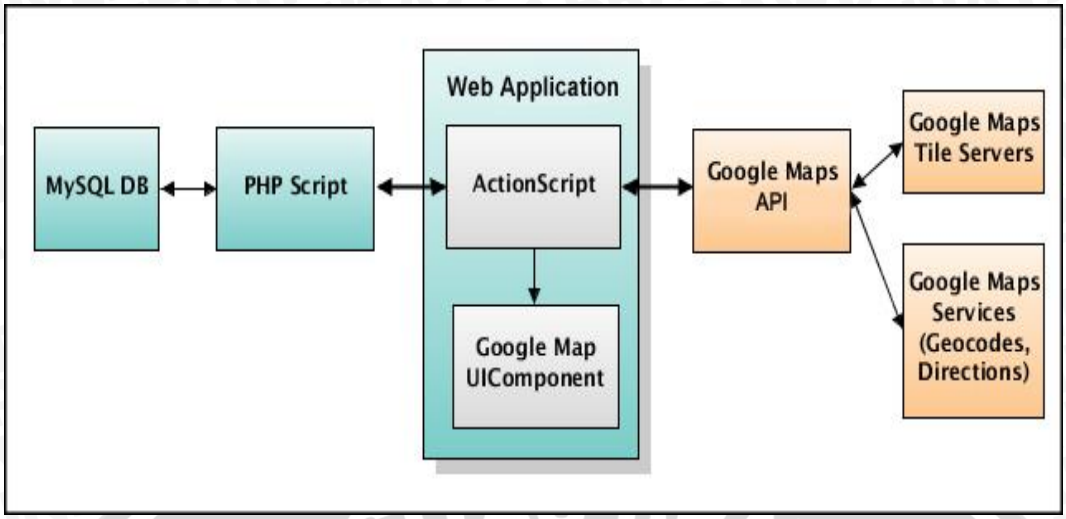
4.3 Perancangan Aplikasi Secara Umum

Sistem ini menerima input berupa lokasi asal dan lokasi tujuan yang kemudian di proses didalam sistem yang terkoneksi dengan database sehingga dapat dihasilkan output berupa jalur yang akan dilewati ke lokasi tujuan. Berikut adalah gambar dari diagram sistem tersebut.



Gambar 4.2 Diagram menentukan jalur terpendek

Secara keseluruhan, sistem dibedakan menjadi dua sisi, yaitu sisi *client* dan sisi *server*. Di sisi *server* terdapat data yang mengandung informasi peta atau lokasi yang tersimpan dalam database server dan akan diolah menggunakan algoritma *Dijkstra* dan *Ant Colony* untuk mendapatkan rute dengan jarak terpendek. Hasilnya dapat ditampilkan pada sistem informasi berbasis website Setelah itu, jalur terpendek yang dihasilkan oleh kedua algoritma tersebut akan dibandingkan. Dalam sistem ini terdapat beberapa komponen utama, yaitu database, web design yang merupakan GUI (*Graphical User Interface*) dari sistem serta web programming, Google Map API dan implementasi dari algoritma *Dijkstra* dan *Ant Colony* yang semuanya akan saling bekerja sama dalam sistem informasi ini. Cara kerja dari sistem itu sendiri dapat dijelaskan sebagai berikut:



Gambar 4.3 Blok diagram cara kerja sistem

4.4 Pembuatan Aplikasi Jalur Terpendek

Dalam pembuatan aplikasi ini terdapat tiga komponen, yang pertama adalah database dari sistem itu sendiri kemudian form pada web yang merupakan GUI (Graphical User Interface) dari aplikasi penentuan jalur terpendek. Dan terakhir adalah proses penentuan jalur terpendek menggunakan algoritma Dijkstra dan Ant Colony serta integrasinya dengan database. Cara kerja dari aplikasi ini sendiri dapat dijelaskan sebagai berikut

4.4.1 Database sistem

Pada *database* dibuat tabel-tabel untuk mendukung penentuan jalur terpendek seperti tabel *point* dan *line*.

- Tabel point berfungsi untuk menentukan atau menampilkan koordinat suatu titik pada peta.

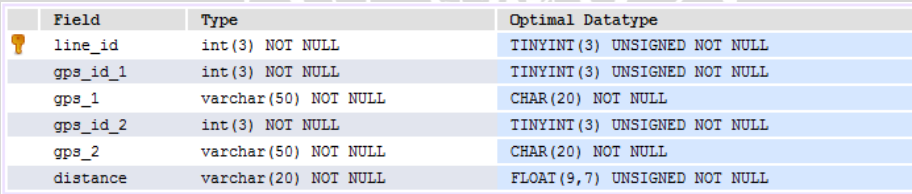
	Field	Type	Optimal Datatype
🔑	id	int(5) NOT NULL	TINYINT(3) UNSIGNED NOT NULL
	gps	varchar(50) NOT NULL	CHAR(20) NOT NULL
	nama	text NOT NULL	CHAR(1) NOT NULL
	jenis	int(1) NOT NULL	TINYINT(1) UNSIGNED NOT NULL
	keterangan	text NOT NULL	VARCHAR(20) NOT NULL

Gambar 4.4 Tabel *Point*

Tabel 4.1 Keterangan *field* tabel *Point*

Field	**Keterangan**
id	ID point atau titik
gps	Menyatakan koordinat titik
nama	Nama titik
jenis	Jenis titik yang ditampilkan di peta
keterangan	Keterangan tambahan

- Tabel *Line* berisi daftar jarak antar satu titik dengan titik yang lain dengan menggunakan rumus pitagoras dalam menentukan jarak antara dua titik koordinat. Tabel ini berfungsi untuk menghitung jarak pada penerapan kedua algoritma. Pada algoritma *Ant Colony* tabel ini juga berfungsi menentukan nilai visibilitas antar kota (η_{ij}), yang mana $\eta_{ij} = 1/d_{ij}$ dan d_{ij} adalah jarak antar titik seperti pada tabel *line*



	Field	Type	Optimal Datatype
🔑	line_id	int(3) NOT NULL	TINYINT(3) UNSIGNED NOT NULL
	gps_id_1	int(3) NOT NULL	TINYINT(3) UNSIGNED NOT NULL
	gps_1	varchar(50) NOT NULL	CHAR(20) NOT NULL
	gps_id_2	int(3) NOT NULL	TINYINT(3) UNSIGNED NOT NULL
	gps_2	varchar(50) NOT NULL	CHAR(20) NOT NULL
	distance	varchar(20) NOT NULL	FLOAT(9,7) UNSIGNED NOT NULL

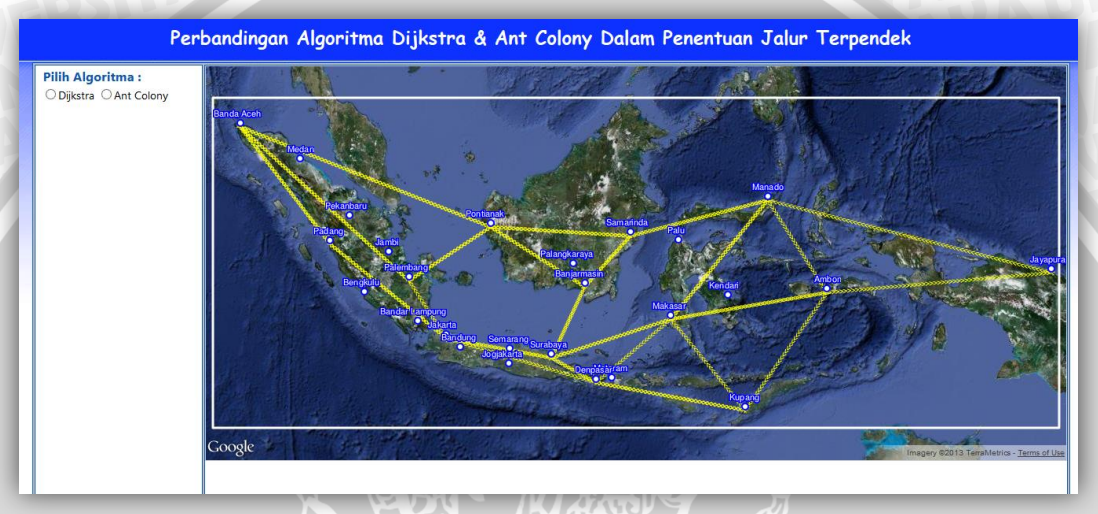
Gambar 4.5 Tabel *Line*

Tabel 4.2 Keterangan *field* tabel *Line*

Field	**Keterangan**
line_id	ID line
gps_id_1	ID point atau titik pertama
gps_1	Koordinat pada titik pertama
gps_id_2	ID point atau titik kedua
gps_2	Koordinat pada titik kedua
distance	Jarak antara titik pertama dengan titik kedua

4.4.2 GUI

Graphical User Interface atau antarmuka yang digunakan akan berupa Google Map API sebagai media lintasan atau peta dan web design yang menggunakan pemrograman HTML, CSS, PHP serta Java Script. GUI aplikasi ini dapat dilihat pada gambar berikut :



Gambar 4.6 Tampilan Antarmuka Aplikasi

4.4.3 Menentukan Jalur Terpendek Dengan Algoritma Dijkstra

Input algoritma *Dijkstra* berupa graf berarah yang memiliki nilai atau bobot (*wighted directed graph)* dan sebuat titik asal *vertex* yang merupakan hipunan dari graf tersebut. Algoritma *Dijkstra* dimulai dari sebuah simpul yang merupakan titik asal dan dalam setiap iterasinya menambahkan sebuah simpul lain ke lintasan terpendeknya. Berikut adalah langkah-langkah menentukan jalur terpendek dengan algoritma *Dijkstra.*

1. Masukan (*input)* algoritma *Dijkstra*

   a. Matriks ketetanggaan $M[m_{ij}]$

   Matriks ini diambil dari database yang berisi data jalur antara satu kota ke kota yang lain.

line_id	gps_id_1	gps_1	gps_id_2	gps_2	distance
21	13	-7.280741,112.729797	9	-6.200629,106.841125	666.72841978871
22	21	-5.134715,119.414978	24	1.493971,124.835815	953.6128176412
23	24	1.493971,124.835815	21	-5.134715,119.414978	953.6128176412
24	13	-7.280741,112.729797	17	-3.329043,114.590034	486.40091451285
25	17	-3.329043,114.590034	13	-7.280741,112.729797	486.40091451285
26	17	-3.329043,114.590034	20	-0.499872,117.151794	425.0388284534
27	17	-3.329043,114.590034	19	0,109.329529	693.28614858856
28	20	-0.499872,117.151794	17	-3.329043,114.590034	425.0388284534
29	19	0,109.329529	17	-3.329043,114.590034	693.28614858856
30	13	-7.280741,112.729797	14	-8.651626,115.223694	316.9259557957
31	14	-8.651626,115.223694	13	-7.280741,112.729797	316.9259557957
32	13	-7.280741,112.729797	21	-5.134715,119.414978	781.90997173912
33	9	-6.200629,106.841125	6	-2.997899,104.752807	425.79284364806
34	6	-2.997899,104.752807	19	0,109.329529	609.29412035311
35	6	-2.997899,104.752807	1	5.57225,95.31372	1419.8138684129
36	19	0,109.329529	6	-2.997899,104.752807	609.29412035311
37	1	5.57225,95.31372	6	-2.997899,104.752807	1419.8138684129
38	9	-6.200629,106.841125	4	-0.955766,100.34729	929.59873370416

Gambar 4.7 Data tabel jalur antar kota

- $m_{ij} = bobot\ sisi\ (i,j)$ → pada graf tak-berarah $m_{ij} = m_{ji}$
- $m_{ii} = 0$
- $m_{ij} = \infty$; jika tidak ada sisi dari simpul i ke simpul j

b. Data atau tabel jarak, $D = [d_i]$ yang dalam hal ini,

- d_i = *panjang lintasan dari simpul awal s ke simpul i*

  Panjang lintasan atau jarak antar kota juga terdapat dalam satu tabel dengan jalur antar kota

c. Data simpul $S = [s_i]$

Dalam hal ini simpul direpresentasikan dalam bentuk posisi koordinat kota yang terdapat pada tabel seperti gambar 4.7.

8	-5.427351,105.242729	Bandar Lampung	14B
9	-6.200629,106.841125	Jakarta	7B
10	-6.904614,107.604675	Bandung	7B
11	-7.798079,110.367737	Jogjakarta	10B
12	-6.964597,110.411682	Semarang	8B
13	-7.280741,112.729797	Surabaya	8B
14	-8.651626,115.223694	Denpasar	8B
15	-8.581021,116.113586	Mataram	7B
16	-10.168967,123.573303	Kupang	6B
17	-3.329043,114.590034	Banjarmasin	11B

Gambar 4.8 Data tabel simpul atau kota

- $s_i = 1$, jika simpul i termasuk ke dalam lintasan terpendek
- $s_i = 0$, jika simpul i tidak termasuk ke dalam lintasan terpendek

2. Mencari Lintasan Terpendek

   Berikut adalah langkah-langkah untuk mencari lintasan terpendek dari kota $\boldsymbol{a}$ ke kota yang lainnya.

   a. Inisialisasi titik awal dan jarak antar titik

      Pada proses inisialisasi, kota $\boldsymbol{a}$ diinialisasi sebagai $\boldsymbol{s_i}$ dengan nilai jarak antar kota adalah **0** dan memberinya label sementara

      - Inisialisasi $\boldsymbol{s_i = 0}$ dan $\boldsymbol{d_i = m_{ai}}$ untuk $\boldsymbol{i = 1, 2, \ldots, n}$

   b. Langkah 1 :

      Menentukan jarak minimum dari label sementara

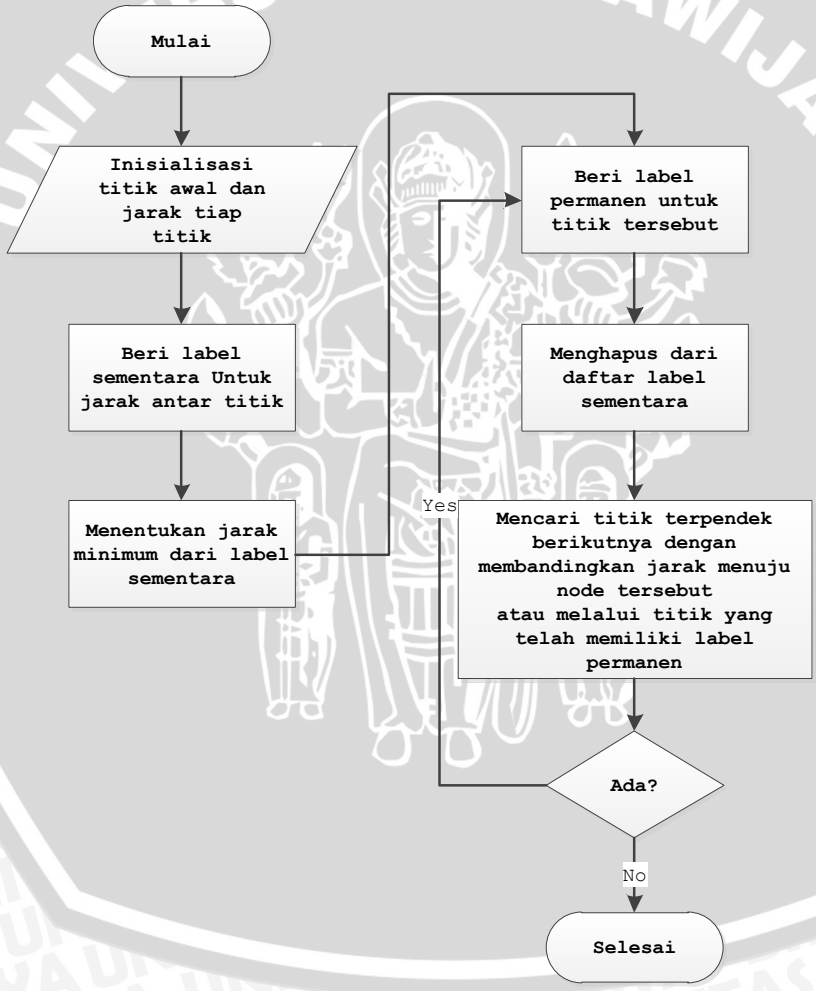
      - Isi $\boldsymbol{s_a}$ dengan **1**, karena simpul $\boldsymbol{a}$ adalah simpul asal lintasan terpendek, jadi sudah pasti terpilih.
      - Isi $\boldsymbol{d_a}$ dengan ∞, tidak ada lintasan terpendek dari simpul $\boldsymbol{a}$ ke a.

c. Langkah $\mathbf{2, 3, \ldots, n-1}$ **:**

- Cari $\boldsymbol{j}$ sehingga $\boldsymbol{s_j = 0}$ dan $\boldsymbol{d_j = min\,\{d_1, d_2, \ldots, d_n\}}$
- Isi $\boldsymbol{s_j}$ dengan $\boldsymbol{1}$
- Perbarui $\boldsymbol{d_i}$ **untuk** $\boldsymbol{i = 1, 2, 3, \ldots, n}$ dengan :

$$\boldsymbol{d_i(baru) = min\,\{d_i(lama),\ d_j + m_{ji}}$$

3. Diagram alir algoritma *Dijkstra*



Gambar 4.9 Diagram alir algoritma *Dijkstra*

4.4.4 Menentukan Jalur Terpendek Dengan Algoritma *Ant Colony*

Dalam algoritma Semut, diperlukan beberapa variabel dan langkah-langkah untuk menentukan jalur terpendek, yaitu

1. Inisialisasi parameter yang digunakan dalam penjarian jalur terpendek.
   a. Intensitas jejak antar kota (τ_{ij}) digunakan dalam persamaan probabilitas kota yang akan dikunjungi.
   b. Banyak kota (n) termasuk koordinat (x,y) atau jarak antar kota (d_{ij}). Nilainya tergantung ada banyaknya jumlah kota yang ada di *database.*
   c. Tetapan pengendali intensitas jejak semut (α), digunakan dalam persamaan probabilitas kota yang akan dikunjungi yang berfungsi sebagai pengendali intensitas jejak semut.
   d. Tetapan pengendali visibilitas (β) yang digunakan dalam persamaan probabilitas kota yang akan dikunjungi, berfungsi sebagai pengendali visibilitas.
   e. Tetapan penguapan jejak feromon (ρ), nilai $\rho > 0$ dan < 1 untuk mencegah jejak feromon yang tak terhingga
   f. Visibilitas antar kota (η_{ij}) digunakan dalam persamaan probabilitas kota yang akan dikunjungi. Nilai η_{ij} merupakan hasil dari $1/d_{ij}$ (jarak kota).
   g. Banyak semut (m) merupakan banyak semut yang akan melakukan siklus dalam algoritma semut. Nilai m ditentukan oleh pengguna. Semakin banyak nilai semut.
   h. Tetapan siklus semut (Q), siklus atau banyak langkah yang dilakukan oleh setiap semut dalam melakukan perjalananya.

i. Jumlah siklus maksimum (NCmax)

NCmax adalah jumlah maksimum siklus yang akan berlangsung pada setiap semut. Siklus akan berhenti sesuai dengan NCmax yang telah ditentukan atau telah menemukan titik/kota tujuan.

2. Menghitung Probabilitas antar kota

Perhitungan probabilitas antar kota bertujuan untuk mencari kemungkinan kemana semut akan berjalan. Probabilitas antar kota dapat dicari dengan menggunakan rumus :

$$P_{ab,m}(t) = \frac{[\tau_{ij}(t)]^{\alpha}\,[\eta_{ij}(t)]^{\beta}}{\sum_{t\epsilon j_{ik}}[\tau_{ij}(t)]^{\alpha}\,[\eta_{ij}(t)]^{\beta}}$$

Setelah mendapatkan probabilitas antar kota, dihitung probabilitas komulatifnya dan kemudian membangkitkan bilangan acak. Bilangan acak bernilai antara 0 sampai 1. Kota yang terpilih untuk melanjutkan perjalanan semut adalah kota dengan nilai probabilitas komulatifnya lebih besar dari bilangan acak. Jika ada lebih dari satu kota yang memiliki nilai probabilitas komulatif lebih besar dari bilangan acak, maka dari kota tersebut ambil kota yang probabilitas komulatifnya paling kecil.

Tabel 4.3 Probabilitas Komulatif

Probabilitas Komulatif	0.00	0.36	0.56	0.73	1.00
Kota	A	B	C	D	E

Misalkan bilangan acak yang diperoleh adalah 0,61 maka kota yang memiliki probabilitas komulatif lebih besar dari bilangan acak adalah kota C, D, dan E. Karena terdapat tiga kota yang memiliki probabilitas

komulatif lebih besar dari bilangan acak, maka kota yang dipilih adalah kota E.

Ketika kota E menjadi kota terpilih, maka dihitung kembali probabilitas antar kota dari kota E dan seterusnya hingga menemukan kota tujuan atau siklus telah mencapai nilai maksimum.

3. Menentukan Jarak Tiap Rute

Perhitungan panjang jarak tiap rute tertutup (length closed tour) atau Lk setiap semut dilakukan setelah satu siklus diselesaikan oleh semua semut. Perhitungan ini dilakukan berdasarkan daftar semut masing-masing. setelah Lk setiap semut dihitung, akan didapat harga minimal panjang rute tertutup setiap siklus. Kemudian akan dihitung perbaikan jejak feromon atau perubahan nilai feromon antar kota.

$$\Delta\tau_{ab} = \frac{Q}{Lk}$$

$\Delta\tau_{ab}$ merupakan perubahan feromon antar kota setiap semut untuk $(a, b) \in$ titik atau kota dalam daftar semut.

4. Perhitungan Jejak Feromon

Perhitungan jejak feromon antar titik dilakuka untuk siklus selanjutnya. Nilai jejak feromon pada semua lintasan antar kota ada kemungkinan berubah karena adanya perbedaan jumlah semut dan penguapan feromon yang melewati setiap rutenya. Selanjutnya nilai feromon dihitung dengan persamaan

$$next\ \tau = \rho.\tau_{ab} + \Delta\tau_{ab}$$

5. Menentukan Jalur Terpendek

Setelah menghitung probabilitas antar kota serta perubahan feromonnya maka akan didapatkan kota terpilih untuk melanjutkan perjalanan.

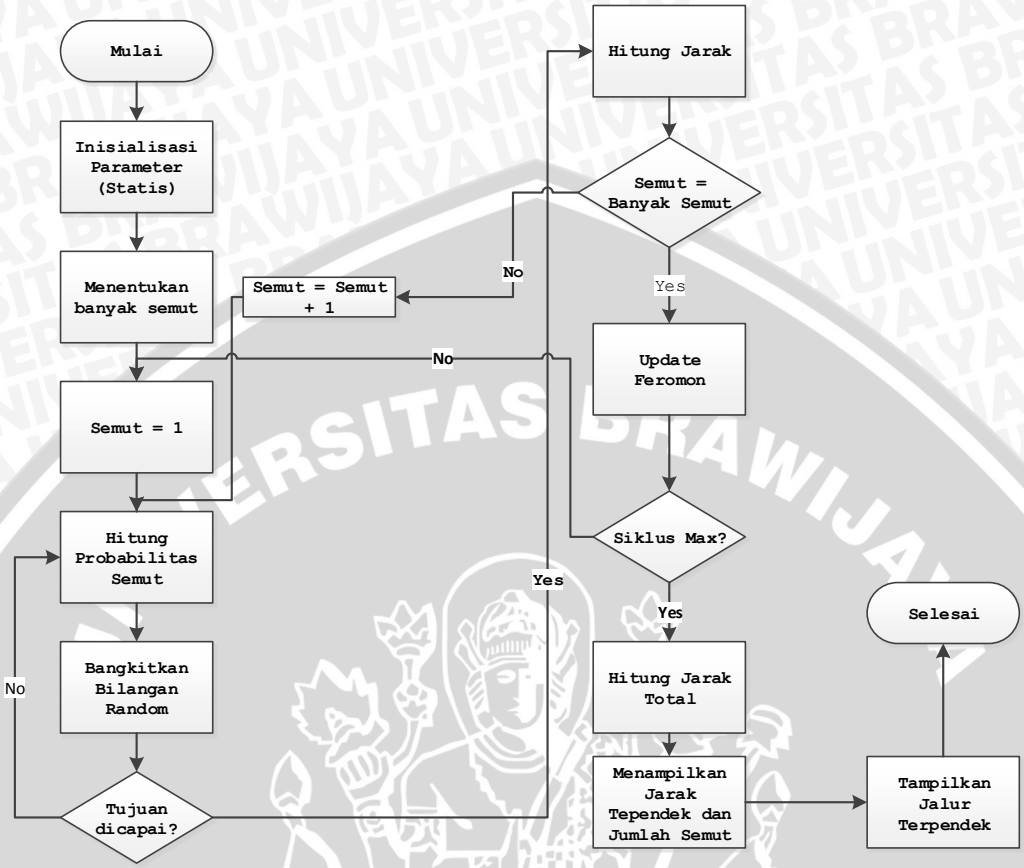
Dari jalur yang telah dikunjungi oleh tiap semut akan diambil jalur dengan kota asal dan tujuan yang sama, kemudian dari jalur tersebut pilih jalur yang memiliki jarak paling dekat. Sehingga jalur yang memiliki jarak terdekat itulah yang merupakan jalur terpendek yang didapat oleh algoritma *Ant Colony.*

Tabel 4.4 Data Perjalanan Semut

Semut	Jalur	Jarak (km)
1	A-B-C-E-F	121
2	A-C-D-F	217
3	A-B-G-I-F	174
4	A-C-D-G-E-E	108
5	A-H-D-C-E	98

Misalkan dari data tabel diatas merupakan data perjalanan semut yang akan ditentukan jalur mana yang merupakan jalur terpendek dari kota A menuju kota F. Maka, jalur terpendek yang terpilih adalah jalur yang ditempuh oleh semut ke-3 dengan jalur A-B-G-I-F dan jaraknya sejauh 174 km.

6. Diagram alir algoritma Ant Colony



Gambar 4.10 Diagram alir algoritma Ant Colony

BAB V

PENGUJIAN SISTEM

Pengujian dan analisis dilakukan untuk mengetahui apakah sistem telah bekerja sesuai perancangan.

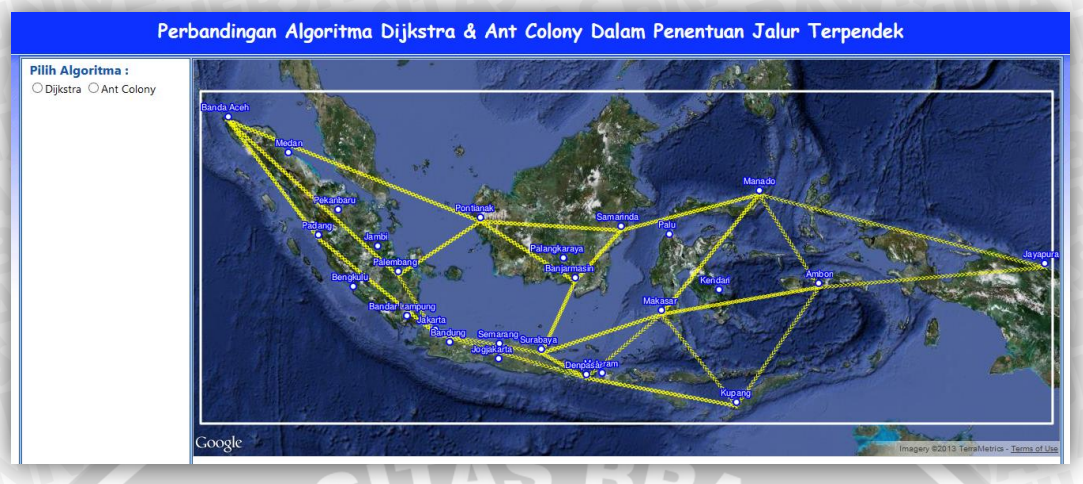
5.1 Metode Pengujian

Metode pengujian terdiri dari 5 tahap, yaitu :

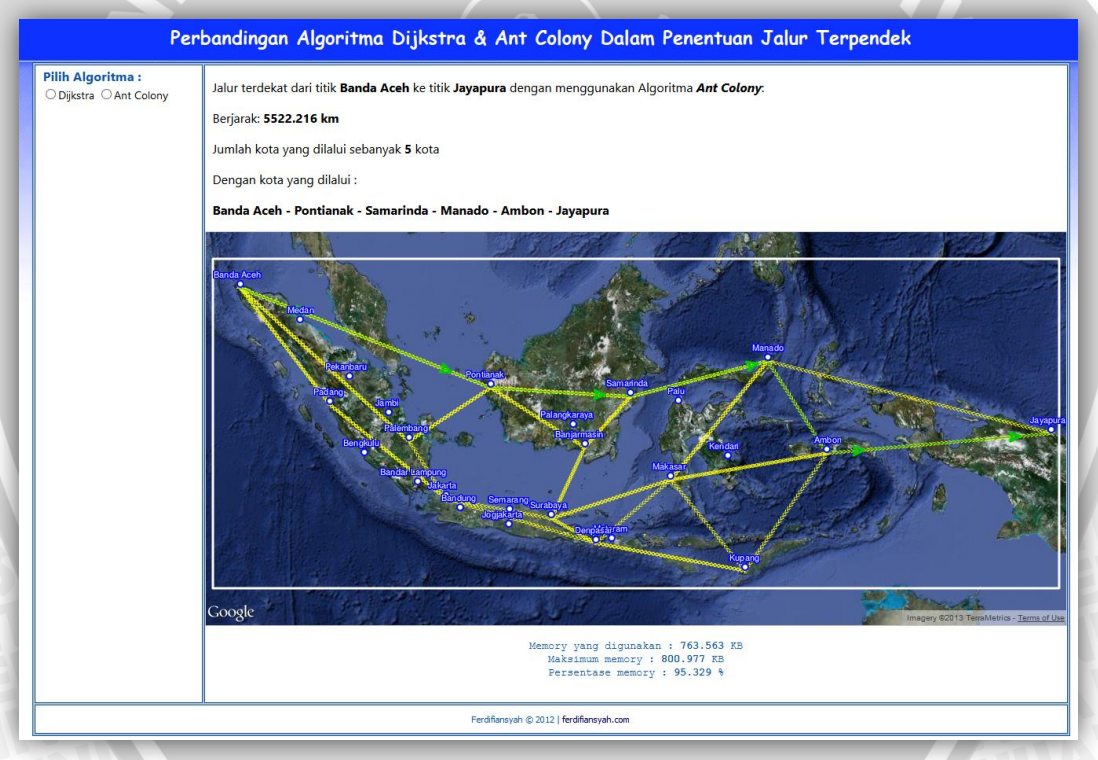
- Pengujian jalur terpendek menggunakan algoritma Dijkstra
- Pengujian jalur terpendek menggunakan algoritma Ant Colony
- Membandingkan jalur terpendek yang dihasilkan algoritma Dijkstra dan Ant Colony
- Membandingkan memory yang dibutuhkan oleh kedua algoritma dalam proses pencarian jalur terpendek
- Mempandingkan kecepatan proses berdasarkan kompleksitas algoritma.

5.2 Hasil Pengujian

Pada saat mengakses aplikasi akan menampilkan halaman utama yang berisi pilihan algortima dan jalur yang ada pada peta. Setelah pemilihan algoritma, mengisi data masukan dan memerintahkan apikasi untuk meprosesnya, maka hasil dari proses tersebut akan muncul pada halaman seperti pada gambar 5.2.



Gambar 5.1 Tampilan halaman utama

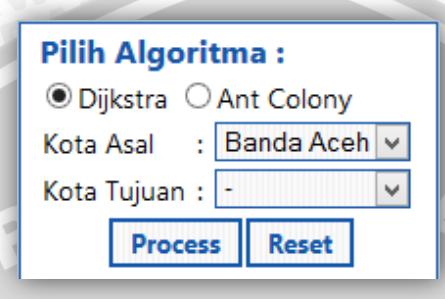


Gambar 5.2 Tampilan hasil proses algoritma

Berikut ini merupakan hasil pengujian pada perbandingan algoritma Dijkstra dan Ant Colony berdasarkan beberapa metode diatas.

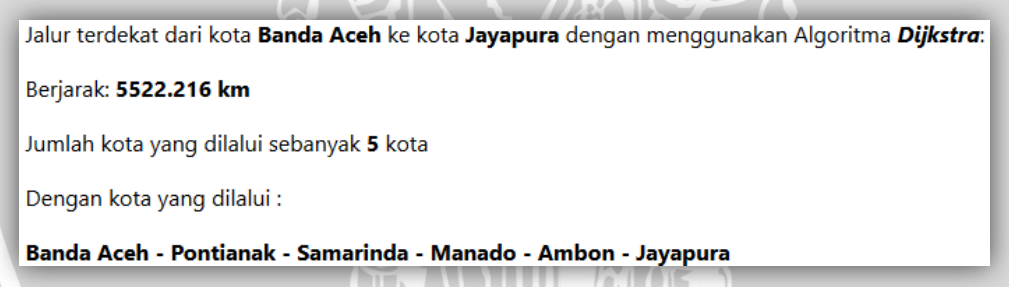
5.2.1 Pengujian jalur terpendek menggunakan algoritma Dijkstra

Untuk mengetahui hasil pengujian jalur terpendek menggunakan algoritma Dijkstra, masukkan titik awal dan titik tujuan pada menu algoritma Dijkstra berdasarkan titik yang ada pada peta.

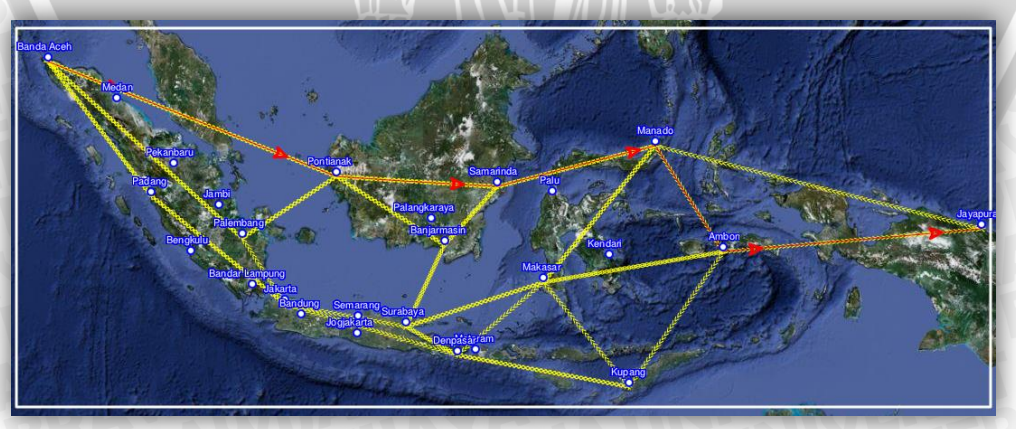


Gambar 5.3 Memilih Algoritma dan Menentukan titik asal dan titik tujuan

Kemudian tekan proses dan akan muncul hasil dari pengujian jalur terpendek Algoritma Dijkstra berupa jarak, titik yang dilalui serta jalur pada peta.



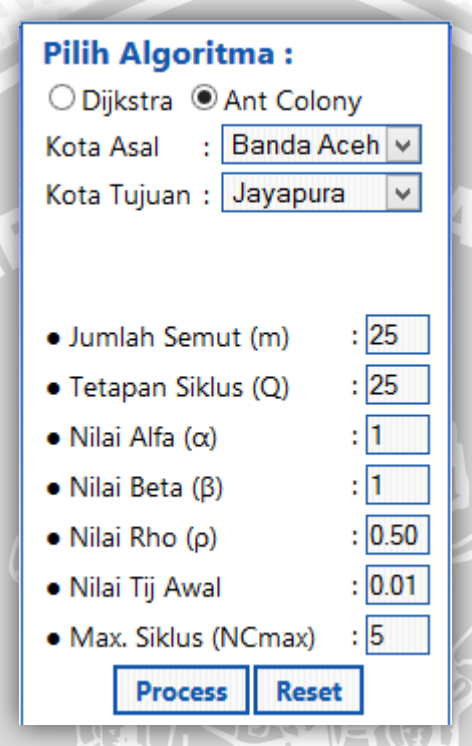
Gambar 5.4 Hasil Pengujian Algoritma Dijkstra



Gambar 5.5 Jalur yang dihasilkan oleh Algoritma Dijkstra

5.2.2 Pengujian jalur terpendek menggunakan algoritma Ant Colony

Untuk mengetahui hasil pengujian jalur terpendek menggunakan algoritma Ant Colony, masukkan titik awal, tujuan serta parameter yang ada pada menu algoritma Ant Colony berdasarkan titik yang ada pada peta.



Gambar 5.6 Pilih Algoritma dan Menentukan titik asal, tujuan serta parameter

Kemudian tekan proses dan akan muncul hasil dari pengujian jalur terpendek Algoritma Ant Colony berupa jarak, titik yang dilalui serta jalur pada peta.

Jalur terdekat dari titik **Banda Aceh** ke titik **Jayapura** dengan menggunakan Algoritma ***Ant Colony***:

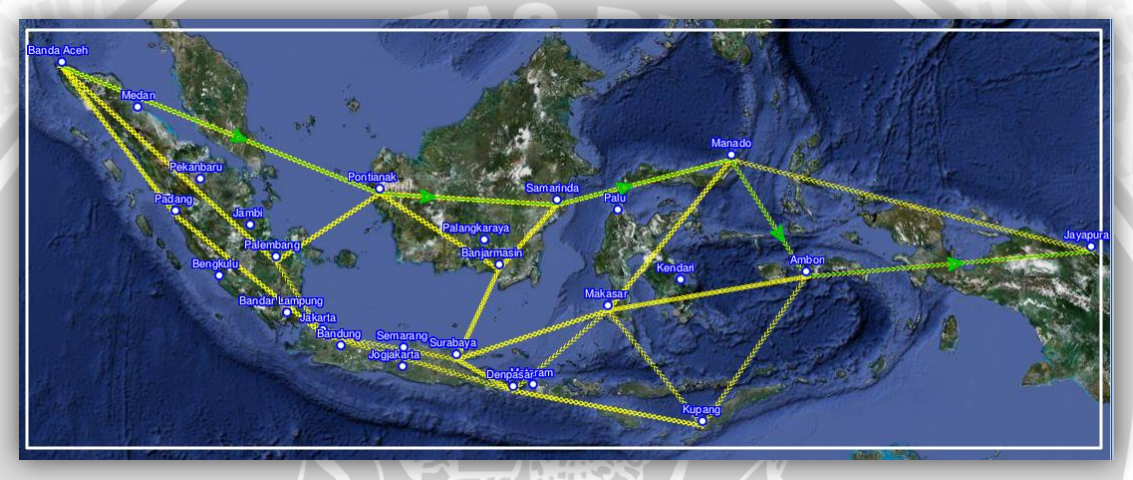
Berjarak: **5522.216 km**

Jumlah kota yang dilalui sebanyak **5** kota

Dengan kota yang dilalui :

Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura

Gambar 5.7 Hasil Pengujian Algoritma Ant Colony



Gambar 5.8 Jalur yang dihasilkan oleh Algoritma Ant Colony

5.2.3 Membandingkan jalur terpendek yang dihasilkan algoritma Dijkstra dan Ant Colony

Dalam pengujian perbandingan jalur terpendek yang dihasilkan oleh kedua algoritma, penulis menguji kedua algoritma dengan tiga data masukkan yang berbeda dan setiap masukkan dilakukan pengujian sebanyak sepuluh kali. Langkah- langkah pengujian untuk setiap algoritma sama seperti pengujian pada masing-masing algoritma pada penjelasan sub-bab sebelunya. Pada algoritma Ant Colony nilai parameter yaitu sebagai berikut :

a. Jumlah Semut (m) : 50

b. Tetapan Siklus (Q) : 50

c. Nilai Alfa (α) : 1

d. Nilai Beta (β) : 1

e. Nilai Rho (ρ) : 0.5

f. Nilai τ_{ab} Awal : 0.01

g. Max. Siklus (NCmax) : 5

h. Pengujian dengan data masukkan kota asal **Banda Aceh** dan kota tujuan **Jayapura.**

Tabel 5.1 Pengujian Algoritma *Dijkstra* Dari Banda Aceh ke Jayapura

Algoritma : Dijkstra

No	Input		Output		
	Kota Asal	Kota Tujuan	Jarak Total (km)	Jumlah Kota	Jalur
1	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura
2	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura
3	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura
4	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura
5	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura
6	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura
7	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura
8	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura
9	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura
10	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura

Tabel 5.2 Pengujian Algoritma *Ant Colony* Dari Banda Aceh ke Jayapura

Algoritma : Ant Colony

No	Input		Output		
	Kota Asal	Kota Tujuan	Jarak Total (km)	Jumlah Kota	Jalur
1	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura
2	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura
3	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura
4	Banda Aceh	Jayapura	Tidak Ditemukan		
5	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura
6	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura
7	Banda Aceh	Jayapura	Tidak Ditemukan		
8	Banda Aceh	Jayapura	Tidak Ditemukan		
9	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura
10	Banda Aceh	Jayapura	5522,216	5	Banda Aceh - Pontianak - Samarinda - Manado - Ambon - Jayapura

i. Pengujian dengan data masukkan kota asal **Surabaya** dan kota tujuan **Manado**

Tabel 5.3 Pengujian Algoritma *Dijkstra* Dari Surabaya ke Manado

Algoritma : Dijkstra

No	Input		Output		
	Kota Asal	Kota Tujuan	Jarak Total (km)	Jumlah Kota	Jalur
1	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
2	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
3	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
4	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
5	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
6	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
7	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
8	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
9	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
10	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado

Tabel 5.4 Pengujian Algoritma *Ant Colony* Dari Surabaya ke Manado

Algoritma : Ant Colony

No	Input		Output		
	Kota Asal	Kota Tujuan	Jarak Total (km)	Jumlah Kota	Jalur
1	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
2	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
3	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
4	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
5	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
6	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
7	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
8	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
9	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado
10	Surabaya	Manado	1735,523	2	Surabaya - Makasar - Manado

j. Pengujian dengan data masukkan kota asal **Jakarta** dan kota tujuan **Pontianak**

Tabel 5.5 Pengujian Algoritma *Dijkstra* Dari Jakarta ke Pontianak

Algoritma : Dijkstra

No	Input		Output		
	Kota Asal	Kota Tujuan	Jarak Total (km)	Jumlah Kota	Jalur
1	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
2	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
3	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
4	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
5	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
6	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
7	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
8	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
9	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
10	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak

Tabel 5.6 Pengujian Algoritma *Ant Colony* Dari Jakarta ke Pontianak

Algoritma : Dijkstra

No	Input		Output		
	Kota Asal	Kota Tujuan	Jarak Total (km)	Jumlah Kota	Jalur
1	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
2	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
3	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
4	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
5	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
6	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
7	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
8	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
9	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak
10	Jakarta	Pontianak	1035,087	2	Jakarta - Palembang - Pontianak

Dari semua pengujian diatas dapat diketahui bahwa penggunaan Algoritma *Dijkstra* dan *Ant Colony* menghasilkan jalur dengan jalur terpendek yang sama. Pada Algoritma *Dijkstra* menghasilkan jalur dan jarak yang sama untuk setiap prosesnya, tetapi pada Algoritma *Ant Colony* jarak dan jalur yang dihasilkan untuk setiap prosesnya tidak selalu sama walaupun pada algoritma ini juga dapat menemukan jalur dan jarak terpendek.

Proses dan hasil algortima *Ant Colony* juga dipengeruhi oleh jumlah semut (m) dan sikusnya. Semakin besar jumlah semut dan siklusnya makan hasil dari algoritma *Ant Colony* akan semakin maksimal. Dengan rincian semakin banyak semut yang melakukan perjalanan, maka semakin besar kemungkinan untuk menemukan jarak terpendek. Sedangkan siklus perjalanan mempengaruhi banyaknya jalur yang ditempuh oleh semut. Dalam hal ini juga akan mempengaruhi jika dalam graf terdapat banyak simpul atau semakin banyak kota.

5.2.4 Membandingkan *memory* yang dibutuhkan oleh kedua algoritma dalam proses pencarian jalur terpendek

Penggunaan *memory* pada bahasa pemrograman PHP bisa dipantau dan ditampilkan menggunakan fungsi PHP `memory_get_usage()`. Fungsi ini bertujuan untuk mengetahui *memory* yang dibutuhkan selama proses. Untuk melihat berapa jumlah penggunaan *memory* maksimum bisa ditambahkan fungsi PHP `memory_get_peak_usage()`.

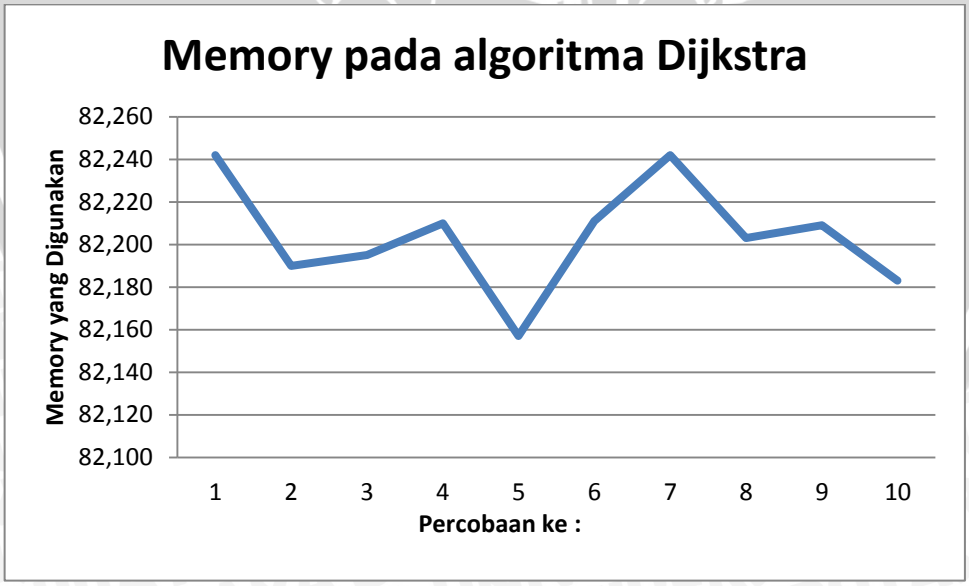
Untuk membaca atau mengetahui penggunaan *memory*, fungsi PHP `memory_get_usage()` diletakkan pada awal dan akhir source code aplikasi yang prosesnya ingin diketahui. Fungsi yang diletakkan diawal proses aplikasi disebut sebagai *memory* awal, sedangkan fungsi yang diletakkan pada akhir proses aplikasi disebut *memory* akhir. Untuk mengetahui *memory* yang digunakan pada aplikasi maka dilakukan pengurangan antara memory akhir dengan *memory* awal.

Dalam membandingkan *memory* yang dibutuhkan oleh kedua algoritma penulis melakukan pembacaan *memory* awal, memory akhir dan memory yang digunakan dari hasil pengurangan *memory* akhir dan *memory* awal. Pada masing-masing algoritma dilakukan pembacaan *memory* dengan data masukkan secara acak sebanyak sepuluh kali. Berikut adalah hasil pembacaan penggunaan memory kedua algoritma.

a. Hasil pembacaan memory pada algoritma *Dijkstra*

Tabel 5.7 Pembacaan Memory Pada Algoritma *Dijkstra*

Pembacaan ke- :	Memory yang digunakan (KB)
1	82,242
2	82,190
3	82,195
4	82,210
5	82,157
6	82,211
7	82,242
8	82,203
9	82,209
10	82,183
Jumlah	822,042
Rata-rata	82,204

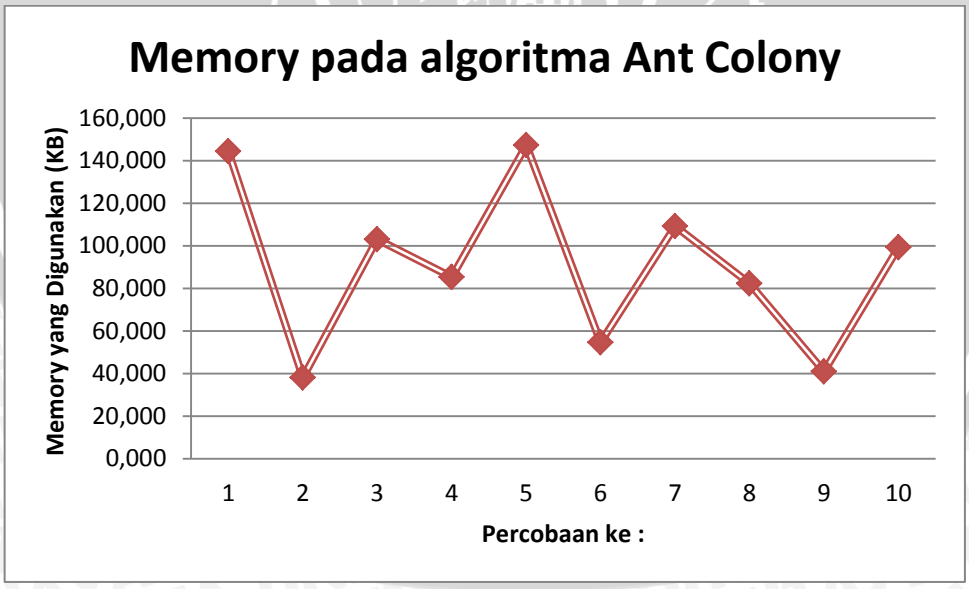


Gambar 5.9 Grafik Pemakaian Memory Pada Algoritma *Dijkstra*

b. Hasil pembacaan memory pada algoritma *Ant Colony*

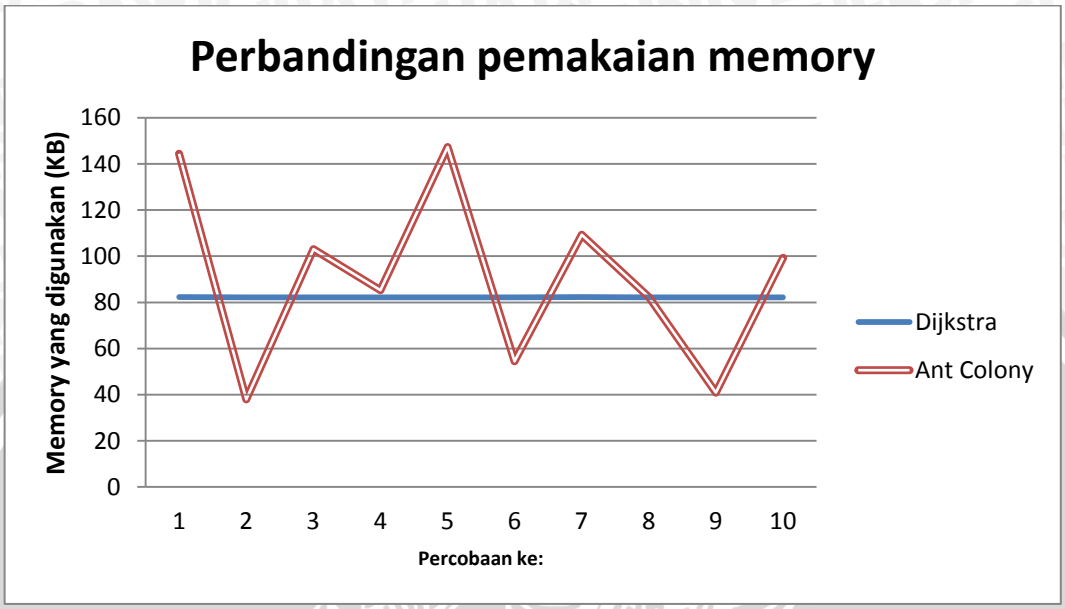
Tabel 5.8 Pembacaan Memory Pada Algoritma *Ant Colony*

Pembacaan ke- :	Memory yang digunakan (KB)
1	144,326
2	37,993
3	102,962
4	85,321
5	147,258
6	54,554
7	109,217
8	82,203
9	40,875
10	99,334
Jumlah	904,043
Rata-rata	90,404



Gambar 5.10 Grafik Pemakaian Memory Pada Algoritma *Ant Colony*

Jika kedua hasil pengujian tersebut digabungkan, maka akan terlihat jelas perbedaan penggunaan memory antar kedua algoritma tersebut seperti terlihat pada gambar 5.11



Gambar 5.11 Grafik perbandingan pemakaian memory antara algoritma *Dijkstra* dan *Ant Colony*

Dari hasil pengujian diatas dapat diketahui bahwa penggunaan memory pada Algoritma D*ijkstra* memiliki rata-rata lebih rendah daripada algoritma *Ant Colony*. Pada algoritma *Dijkstra* memory yang digunakan antara proses yang satu dengan yang lain nilainya terlihat relatif lebih stabil dan konstan seperti pada gambar 5.11 yang mana grafiknya hampir lurus. Sedangkan penggunaan memory pada algoritma *Ant Colony* terdapat perbedaan yang besar antar proses yang satu dengan yang lainnya.

Penggunaan memory pada Algoritma *Ant Colony* terjadi perubahan yang relatif lebih besar dibandingkan algoritma *Ant Colony,* hal dipengaruhi oleh masukan nilai parameternya.

5.2.5 Membandingkan kecepatan proses berdasarkan kompleksitas algoritma.

Untuk mangetahui kecepatan proses berdasarkan kompleksitas algoritma, penulis menggunakan notasi *Big-O* dalam menganalisa *Pseudocode* dari algoritma *Dijkstra* dan *Ant Colony*.

a. Analisa pada algoritma *Dijkstra*

```
for each vertex v in Graph:
    dist[v] := infinity ;
    previous[v] := undefined ;
end for
```

Kondisi diatas memiliki nilai kompleksitas :

- **vertex v** dieksekusi sebanyak **1** kali
- **dist[v]** dieksekusi sebanyak **n**
- **previous[v]** dieksekusi sebanyak **n**

Total nilai kompleksitasnya adalah :

$$1 + n + n = 2n + 1$$

```
dist[source] := 0 ;
Q := the set of all nodes in Graph ;
```

Kondisi diatas memiliki nilai kompleksitas :

- **dist[source]** dieksekusi sebanyak **1** kali
- **Q** dieksekusi sebanyak **1** kali

Total nilai kompleksitasnya adalah :

$$1 + 1 = 2$$

```
while Q is not empty:
    u := vertex in Q with smallest distance in dist[];
    remove u from Q ;
    if dist[u] = infinity:
        break ;
    end if

    for each neighbor v of u:
        alt := dist[u] + dist_between(u, v) ;
        if alt < dist[v]:
                dist[v] := alt ;
                previous[v] := u ;
                decrease-key v in Q;
        end if
    end for
end while
```

Persamaan diatas memiliki nilai kompleksitas :

- **Q** dieksekusi sebanyak **1** kali
- **u** dieksekusi sebanyak **1** kali
- **remove u** dieksekusi sebanyak **1** kali
- **if dist[u] = infinity:** dieksekusi sebanyak **1** kali
- **for each neighbor v of *u*** dieksekusi sebanyak **1** kali
- **alt** dieksekusi sebanyak **1** kali
- **if alt < dist[v]:** dieksekusi sebanyak **1** kali
- **dist[v] := alt** dieksekusi sebanyak **1** kali
- **previous[v] := u** dieksekusi sebanyak **1** kali
- **decrease-key v in Q** dieksekusi sebanyak **1** kali

Total nilai kompleksitasnya adalah : **10**

Dari analisa masing-masing fungsi atau proses yang ada pada Algoritma *Dijkstra* dapat diketahui nilai kompleksitas untuk Algoritma tersebut, yaitu :

$$(2n + 1) + 2 + 10 = 2n + 13$$

$$f(n) = O(2n + 13)$$

b. Analisa pada algoritma *Ant Colony*

$$\eta(i,j) = 1/D(i,j)$$
$$\tau(i,j) = \tau_0$$

Persamaan diatas dieksekusi masing-masing sebanyak satu kali karena tidak ada perulangan, sehingga nilai kompleksitasnya :

$$1 + 1 = 2$$

```
function Probabilistik (from)
    for m = 1; m<(jml m); m++ : m
        for t = 1; t<(jml t); t++ : siklus
            P_ab,m(t) = [τ_ab(t)]^α [η_ab(t)]^β / Σ_tϵb_ak [τ_ab(t)]^α [η_ab(t)]^β
        end for
    end for
return dist, path;
```

Fungsi diatas akan diakses sebanyak jumlah semut dan siklus yang telah ditentukan, sehingga nilai kompleksitasnya

- **Probabilistik** dieksekusi sebanyak n kali

Total nilai kompleksitasnya adalah : n

```
function updateFeromon ()
    Δτ_ab = Q / Lk
    next τ = ρ.τ_ab + Δτ_ab
return next τ
```

Fungsi updateFeromon diatas akan diakses sebanyak jumlah parameter siklus maksumum yang telah ditentukan, sehingga nilai kompleksitasnya

- **updateFeromon** dieksekusi sebanyak ***n*** kali

Total nilai kompleksitasnya adalah : ***n***

```
for x = 1; x++
    if (min (dist(x)))
        shortpath = path;
    end if
end for
```

Fungsi penentuan jalur terpendek diatas memiliki nilai kompleksitas

- **x=1** dieksekusi sebanyak **1** kali
- **x++** dieksekusi sebanyak **n**
- **if (min (dist(x)))** dieksekusi sebanyak **n**
- **sorthpath** dieksekusi sebanyak **n**

Total nilai kompleksitasnya adalah :

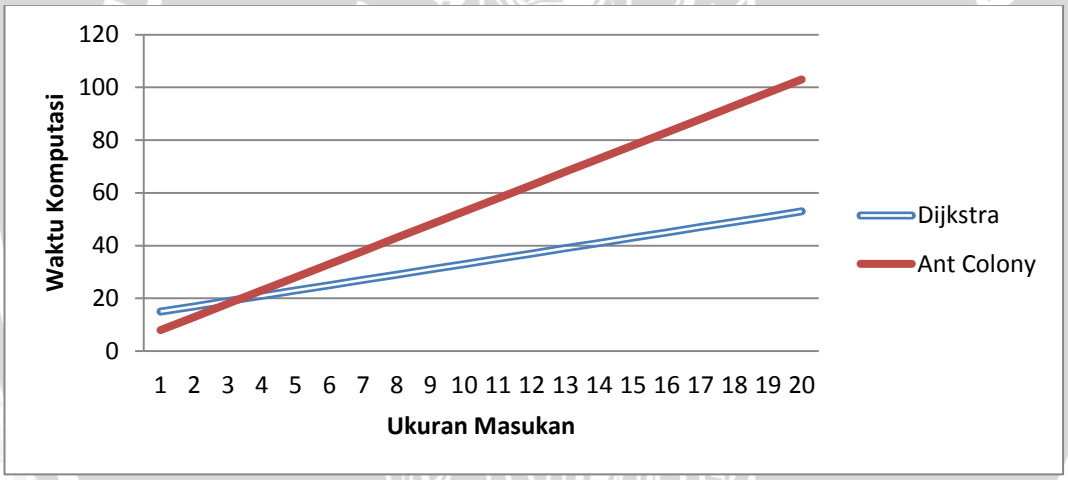
1 + n + n + n = 3n + 1

Dari analisa masing-masing fungsi atau proses yang ada pada Algoritma *Ant Colony* dapat diketahui nilai kompleksitas untuk Algoritma tersebut, yaitu :

$$\mathbf{2 + n + n + (3n + 1) = 5n + 3}$$

$$\boldsymbol{f(n) = O(5n + 3)}$$

Dari analisa kedua algoritma diatas dapat diketahui bahwa Algoritma *Dijkstra* memiliki kompleksitas waktu $\boldsymbol{f(n) = O(2n + 13)}$ yang berarti lebih lebih rendah daripada Algoritma *Ant Colony* $\boldsymbol{f(n) = O(5n + 3)}$. Kedua kompleksites tersebut dapat dilihat dalam bentuk grafik seperti pada gambar 5.12



Gambar 5.12 Grafik perbandingan kompleksitas waktu

Dari grafik diatas dapat dilihat bahwa penggunaan algoritma *Dijkstra* memiliki pertambahan nilai komputasi waktu yang semakin besar tetapi tidak sebesar pada algoritma *Ant Colony* yang memiliki nilai waktu komputasi yang jauh semakin besar ketika ukuran data masukannya semakin besar.

BAB VI
PENUTUP

6.1 Kesimpulan

Berdasarkan hasil perancangan, implementasi, pengujian dan analisis sistem maka dapat diambil kesimpulan sebagai berikut:

1. Perancangan aplikasi menentukan jalur terpendek dengan menggunakan algoritma *Dijkstra* tidak perlu menentukan parameter-parameter yang banyak seperti pada algoritma *Ant Colony*.
2. Pada proses Algoritma, Dijkstra memerlukan data jarak setiap kota terlebih dahulu sebelum memulai proses Algortimanya. Sedangkan pada Algoritma Ant Colony, tidak memerlukan jarak setiap kota karena pada Ant Colony jarak antar kota dihitung setelah semut menyelesaikan perjalanannya. Sehingga Algoritma Dijkstra hanya bisa berjalan jika terlebih dahulu diketahui jarak tiap kota, sedangkan pada Algoritma Ant Colony tidak memerlukan jarak tiap kota untuk menjalankan prosesnya.
3. Penggunaan memory pada algoritma *Dijkstra* menggunakan memory yang lebih rendah dengan rata-rata 82,204 KB daripada algoritma *Ant Colony* yang mana selamat pengujian memiliki rata-rata memory sebesar 90,404 KB. Pada algoritma *Dijkstra*, penggunaan memory untuk setiap prosesnya relatif sama, sedangkan pada algoritma Ant Colony penggunaan memory antar proses terdapat perbedaan yang besar karena tergantung pada parameter masukannya terutama banyak semut dan siklusnya.
4. Dari analisa kompleksitas waktu terhadap pseudocode masing-masing algoritma, Dijkstra menghasilkan persamaan $\boldsymbol{f(n) = O(2n + 13)}$ dan Ant Colony menghasilkan persamaan $\boldsymbol{f(n) = O(5n + 3)}$ yang mana bisa diketahui dari kedua fungsi tersebut waktu yang dibutuhkan algortima Dijkstra untuk memproses data lebih pendek daripada algoritma Ant Colony.

6.2 Saran

Dalam perancangan dan pembuatan aplikasi serta perbandingan algoritma *Dijkstra* dan *Ant Colony* dalam menentukan jalur terpendek masih terdapat kekurangan dan kelemahan, oleh karena itu masih diperlukan adanya penyempurnaan dalam rangka pengembangan kedepannya. Adapun hal yang dapat disempurnakan antara lain.

1. Menambahkan kota dan jalur pada database agar lebih mengetahui performa algoritma jalur terpendek.
2. Menambahkan atau membandingkan kedua algoritma yang telah dibahas dengan algoritma yang lainnya dalam menentukan jalur terendek.

DAFTAR PUSTAKA

Jong Jek Siang. 2004. *Matematika Diskrit dan Aplikasinya pada Ilmu Komputer*, Yogyakarta: Penerbit Andi.

Abdul Kadir.2003. *Dasar Pemrograman Web Dinamis Menggunakan PHP*, Yogyakarta: Penerbit Andi.

Prahasta, E., 2001. *Konsep-konsep Dasar Sistem Informasi Geografis*. Bandung: CV. Informatika

Wardy, Ibnu Sina, 2006. *Penggunaan Graf dalam Algoritma Semut untuk Melakukan Optimisasi*. Bandung : Institut Teknologi Bandung.

Google Developers. *Getting Started - Google Maps JavaScript API v3* https://developers.google.com/maps/documentation/javascript/tutorial 17 April 2012