BAB II

LANDASAN TEORI

2.1 Graf

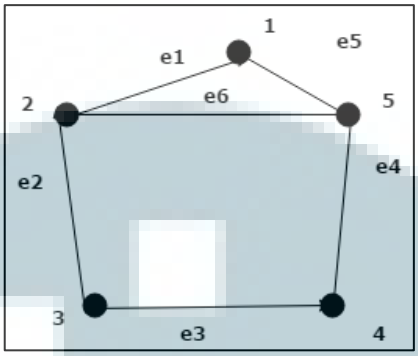
Graf merupakan suatu cabang ilmu yang memiliki banyak terapan. Banyak sekali struktur yang bisa direpresentasikan dengan graf, dan banyak masalah yang bisa diselesaikan dengan bantuan graf. Seringkali graf digunakan untuk merepresentasikan suatu jaringan. Misalkan jaringan jalan raya dimodelkan graf dengan kota sebagai simpul (*vertex/node*) dan jalan yang menghubungkan setiap kotanya sebagai sisi (*edge*) yang bobotnya (*weight*) adalah panjang dari jalan tersebut. Dalam beberapa model persoalan dimungkinkan bahwa bobot dari suatu sisi bernilai negatif. Misalkan simpul merepresentasikan kota, sisi merepresentasikan perjalanan yang memungkinkan, dan bobot dari setiap sisi adalah biaya yang dikeluarkan dalam perjalanan yang memungkinkan, dan bobot dari setiap sisi adalah jarak yang ditempuh dalam perjalanan tersebut. (Ristono, 2011).

2.1.1. Definisi Graf

Secara matematis graf didefinisikan sebagai pasangan himpunan (V,E), ditulis dengan notasi G = (V,E), yang dalam hal ini V adalah himpunan tidak kosong dari simpul-simpul (vertex atau node) dan E adalah himpunan sisi (edge) yang menghubungkan sepasang simpul (Munir, 2005).

Simpulan (vertex) pada graf dapat dinyatakan dengan huruf, bilangan atau gabungan keduanya. Sedangkan sisi-sisi yang menghubungkan simpul u dengan simpul v diyatakan dengan pasangan (u,v) atau dinyatakan dengan lambang e1, e2, e3 dan seterusnya. Dengan kata lain, jika e adalah sisi yang menghubungkan

simpul u dengan simpul v, maka e dapat dituliskan sebagai e = (u,v) (Fauzi, 2011).



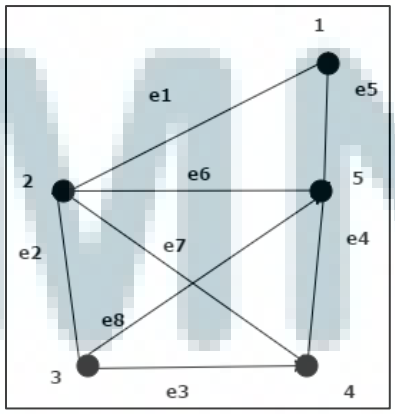
Gambar 2.1 Graf Sederhana
(Fauzi, 2011)

2.1.2. Jenis-jenis Graf

Klasifikasi pada graf cukup luas, klasifikasi tersebut bergantung pada faktor-faktor yang membedakannya. Bedasarkan orientasi arah pada sisinya, maka secara umum graf dibedakan atas dua jenis sebagai berikut (Fauzi, 2011):

1. Graf tidak berarah (*undirected graph*)

   Graf yang sisinya tidak memiliki orientasi arah disebut graf tidak berarah. Pada graf tidak berarah, urutan pasangan simpul yang dihubungkan oleh sisi tidak diperhatikan. Jadi, (u,v) = (v,u) adalah sisi yang sama.

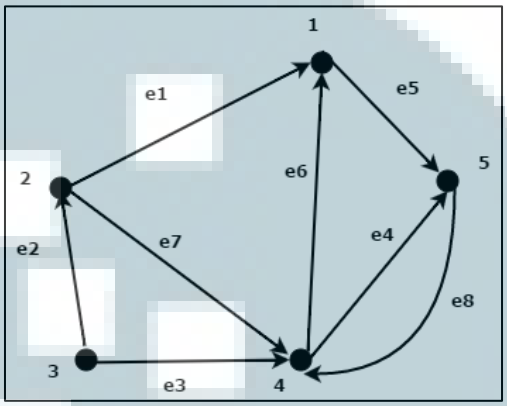


Gambar 2.2 Graf Tidak Berarah
(Fauzi, 2011)

2. Graf berarah (*directed graph*)

   Graf yang setiap sisinya diberikan orientasi arah disebut graf berarah, pada graf berarah (u,v) dan (v,u) menyatakan dua buah sisi yang berbeda.

   Dengan kata lain dapat ditulis (u,v) ≠ (v,u).



Gambar 2.3 Graf Berarah
(Fauzi, 2011)

Adapun graf yang memiliki nilai bobot pada setiap sisinya disebut graf berbobot (*weighted graph*) (Fauzi, 2011).

2.2 Lintasan Terpendek (Shortest Path)

Persoalan mencari lintasan terpendek di dalam graf merupakan salah satu persoalan optimasi. Graf yang digunakan dalam pencarian lintasan terpendek adalah graf berbobot (weighted graph), yaitu graf yang setiap sisinya diberikan suatu nilai atau bobot. Bobot pada sisi graf dapat menyatakan jarak antar kota, waktu pengiriman pesan, ongkos pembangunan, dan sebagainya. Asumsi yang digunakan di sini adalah bahwa semua bobot bernilai positif. Lintasan terpendek adalah jalur yang dilalui dari suatu node ke node lain dengan besar atau nilai pada sisi yang jumlah akhirnya dari node awal ke node akhir paling kecil. Lintasan terpendek adalah lintasan minimum yang diperlukan untuk mencapai suatu tempat dari tempat lain. Lintasan *minimum* yang dimaksud dapat dicari dengan

menggunakan graf. Graf yang digunakan adalah graf yang berbobot yaitu graf yang setiap sisinya diberikan suatu nilai atau bobot (Hayati, 2014).

2.3 Algoritma Dijkstra

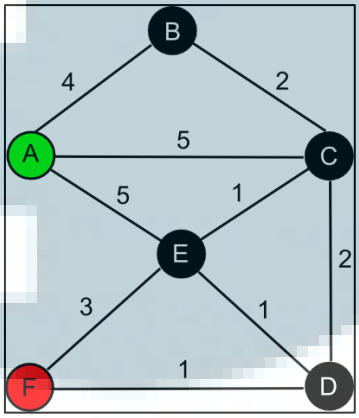
Algoritma Dijkstra merupakan algoritma yang paling sering digunakan dalam pencarian rute terpendek sederhana, penggunaannya dengan menggunakan simpul-simpul sederhana pada jaringan jalan yang tidak rumit (Chamero, 2006). Adapun nama algoritma Dijkstra sendiri berasal dari penemunya yaitu Edger Dijkstra. Dalam mencari solusi, algoritma Dijkstra menggunakan prinsip greedy, yaitu mencari solusi optimum pada setiap langkah yang dilalui, dengan tujuan untuk mendapatkan solusi optimum pada langkah selanjutnya yang akan mengarah pada solusi terbaik. Hal ini membuat kompleksitas waktu algoritma Dijkstra menjadi cukup besar, yaitu sebesar O (V * Log (v + e)), dimana v adalah simpul dan e adalah sisi pada graf yang digunakan.

Input dari algoritma Dijkstra berupa sebuah graf berbobot G(e,v), sedangkan outputnya berupa rute terpendek dari simpul awal (start) ke masing-masing simpul yang ada pada graf. Dengan demikian algoritma Dijkstra dapat menemukan solusi terbaik. Cara kerja algoritma Dijkstra hampir sama dengan cara kerja algoritma BFS yaitu dengan menggunakan prinsip antrian (queque), akan tetapi antrian yang digunakan algoritma Dijkstra adalah antrian berprioritas (priority queue). Jadi hanya simpul yang memiliki prioritas tertinggi yang akan ditelusuri. Dalam menentukan simpul yang berprioritas, algoritma ini membandingkan setiap nilai (bobot) dari simpul yang berada pada satu level. Selanjutnya nilai (bobot) dari setiap simpul tersebut disimpan untuk dibandingkan

dengan nilai yang akan ditemukan dari rute yang baru ditemukan kemudian, begitu seterusnya sampai ditemukan simpul yang di cari (Fauzi, 2011).

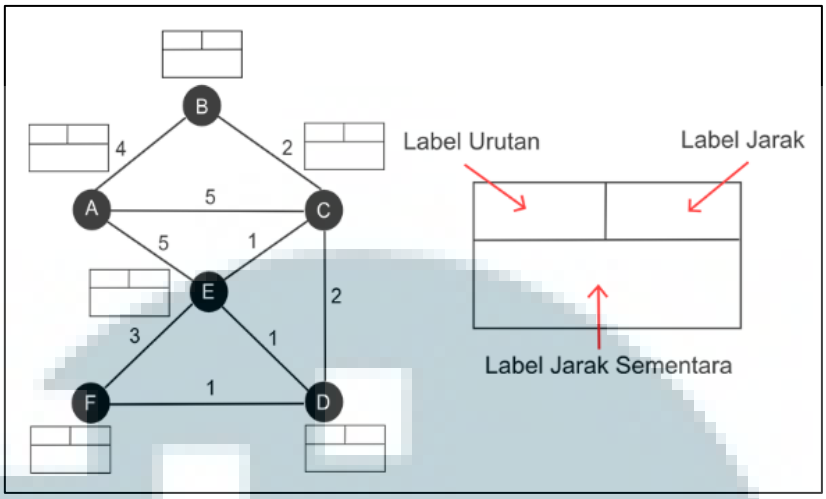
Menurut (Arsori, 2013) untuk menerapkan algoritma Dijkstra dibutuhkan beberapa data yang harus disiapkan, yaitu:

1. Beberapa titik/simpul/daerah, titik/simpul/daerah yang bisa dijangkau secara langsung (titik/simpul/daerah yang terhubung langsung) dan juga jarak antara mereka (*cost*).
2. Titik/simpul/daerah awal.
3. Titik/simpul/daerah tujuan, contoh gambar graf dapat dilihat pada Gambar 2.4.



Gambar 2.4 Graf Tidak Berarah Beserta Jarak Antara Titik
(Arsori, 2013)

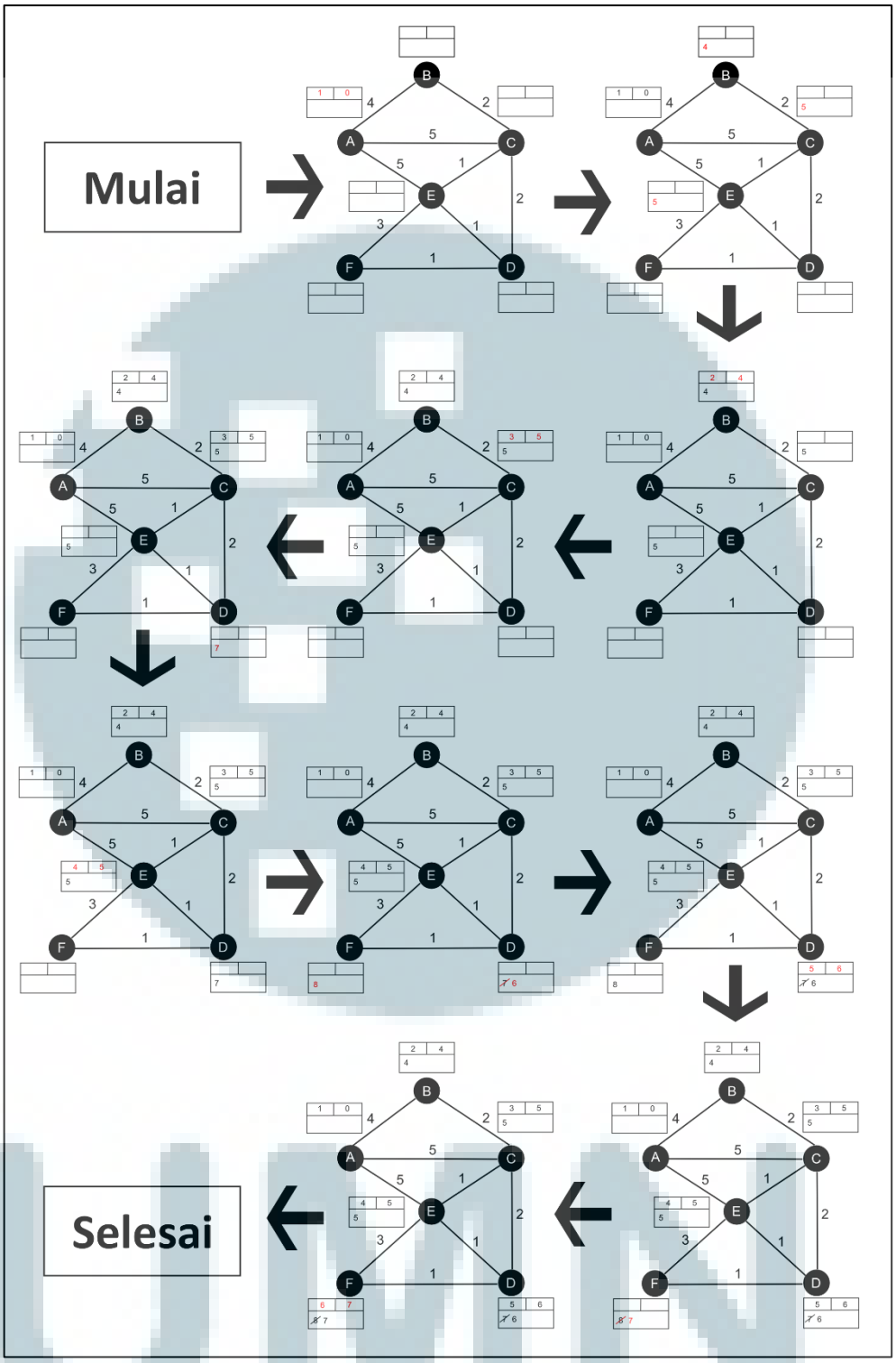
Gambar2.4 merupakan contoh gambar graf beserta dengan jarak antar titik, langkah selanjutnya adalah menentukan titik awal dan titik akhir, misal: titik awal adalah A, titik F adalah titik tujuan dan dilanjutkan dengan mencari rute yang harus dilalui dengan total jarak *minimum*. Untuk bisa mendapatkan rute tersebut, maka ditambahkan beberapa kotak untuk mengisi label, seperti Gambar 2.5



Gambar 2.5 Graf Dengan Kotak Untuk Mengisi Label

(Arsori, 2013)

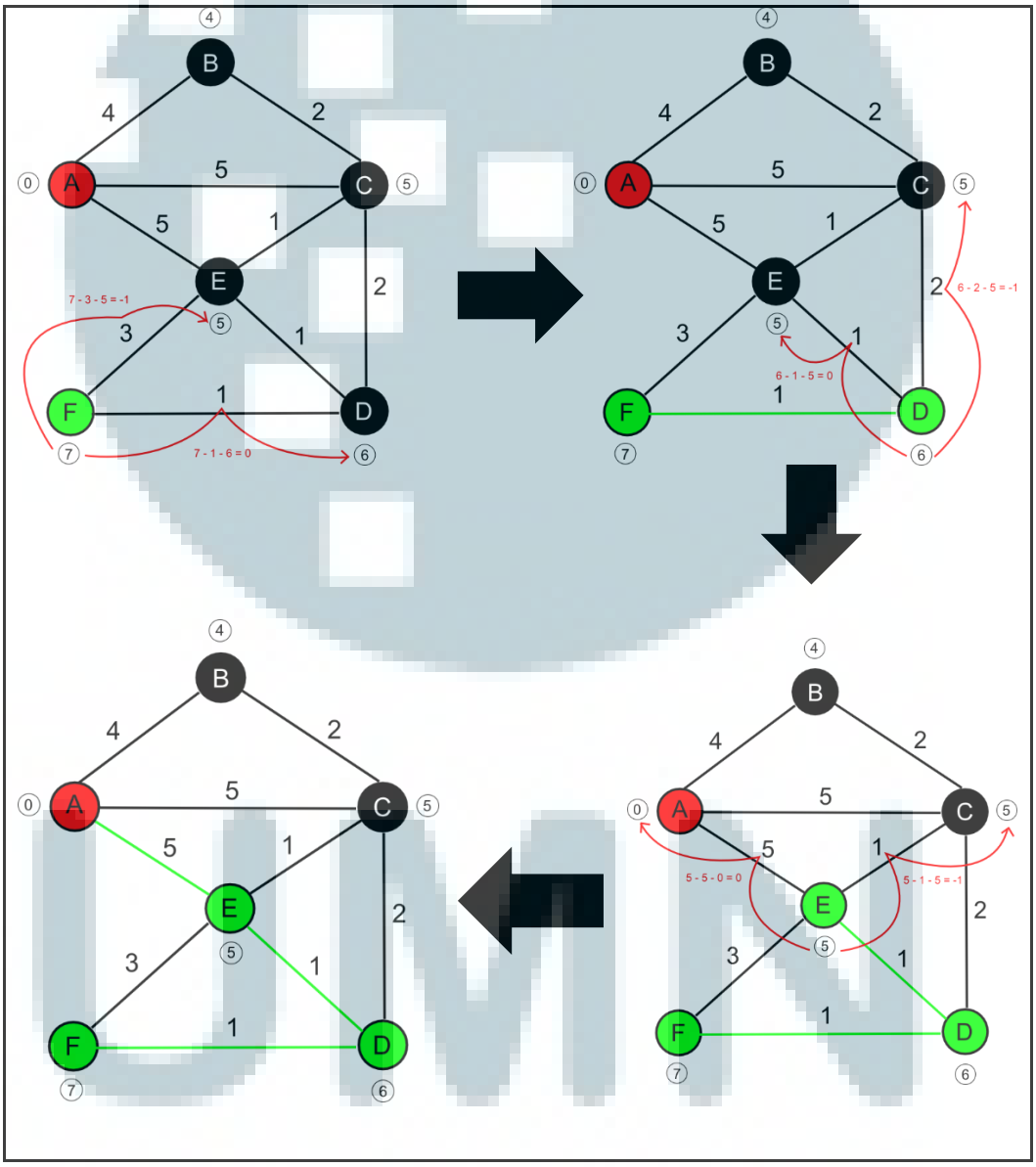
Untuk mengetahui jarak minimum yang dituju dari titik A ke titik lainnya, dapat dilihat pada Gambar 2.6.



Gambar 2.6 Penerapan Algoritma Dijkstra Dalam Menentukan Jarak Minimum (Arsori, 2013)

Setelah mendapatkan jarak *minimum* dengam menggunakan label, langkah selanjutnya yaitu mencari rute yang harus dilalui untuk sampai ke titik F, dapat

dilihat pada Gambar 2.7. Cara mengetahui rute yang akan dilalui hingga sampai ke titik F adalah dengan menelusuri dari titik tujuan ke titik awal dan melakukan pengurangan label jarak dari titik F dengan jarak titik tujuan, seperti pada Gambar 2.7 jika hasil kurang dari 0 maka titik tersebut tidak layak dilewati dan jika hasilnya lebih dari 0 serta lebih mendekati 0 maka titik tersebut yang akan dilewati.



Gambar 2.7 Cara Menentukan Rute Yang Dilalui
(Arsori, 2013)

Dengan begitu, rute dengan jarak *minimum* yang dilalui dari titik A menuju titik F dengan jarak terpendek adalah A → E → D → F.

Source code algoritma Dijkstra yang digunakan dalam penelitan ini dapat dilihat pada Gambar 2.8, variabel yang dibutuhkan dalam function dijkstra adalah relasi antar titik dan titik start.

```
function dijkstra($neighbors, $start) {
  $closest = $start;
  while (isset($closest)) {
    $marked[$closest] = 1;
    /* loop through each neighbor for this $closest node and
     * and store the distance and route in an array */
    foreach ($neighbors[$closest] as $vertex => $distance) {
      /* if we already have the route and distance, skip */
      if (isset($marked[$vertex]))
        continue;
      /* distance from $closest to $vertex */
      $dist = (isset($paths[$closest][0]) ? $paths[$closest][0]
: 0) + $distance;
      /* if we dont have a path to $vertex yet, create it.
       * If this distance is shorter, override the existing one
*/
      if (!isset($paths[$vertex]) || ($dist <
$paths[$vertex][0])) {
        if (isset($paths[$closest])) {
      $paths[$vertex]= $paths[$closest];
    }
        $paths[$vertex][] = $closest;
        $paths[$vertex][0] = $dist;
      }
    }
    unset($closest);
    /* Find the next node we should create a path for */
    foreach ($paths as $vertex => $path) {
      if (isset($marked[$vertex]))
        continue;
      $distance = $path[0];
      if ((!isset($min) || $distance < $min) ||
!isset($closest)) {
        $min = $distance;
        $closest = $vertex;
      }
    }
  }
  return $paths;
}
```

Gambar 2.8 Source Code Algoritma Dijkstra
(Wouter, 2006)

2.4 Web

World Wide Web secara luas dikenal dengan istilah *Web*. *Web* pertama kali diperkenalkan pada tahun 1992. Hal ini sebagai hasil usaha pengembangan yang dilakukan CERN di Swiss. Internet dan merupakan TCP/IP sebagai protocol operasionalnya, sedangkan *web* menggunakan HTTP (*Hyper Text Transfer Protocol*) (Mulyanto, 2008).

2.4.1 Web Statis dan Web Dinamis

Halaman *web* dapat digolongkan menjadi *web* statis dan *web* dinamis. Pengertian *web* statis dan web dinamis seringkali mengundang perdebatan. Sebagian pengguna internet menyatakan jika pada halaman-halaman *web* dilengkapi dengan animasi yang bergerak maka disebut web dinamis sedangkan jika halaman-halaman *web* tersebut hanya berisi teks dan gambar yang tidak bergerak maka disebut *web* statis. Namun berdasarkan kesepakatan maka pengertian statis dan dinamis tidak ditentukan oleh ada atau tidaknya animasi bergerak pada halaman-halaman *web*, tetapi ditentukan oleh isi atau informasi yang ada pada halaman-halaman tersebut. Data dan informasi yang ada pada *web* statis tidak berubah-ubah. Dokumen *web* yang dikirim kepada client akan sama isinya dengan apa yang ada di web server. Sedangkan *web* dinamis, memiliki data dan informasi yang berbeda-beda tergantung input apa yang disampaikan client. Dokumen yang sampai di client akan berbeda dengan dokumen yang ada di *web* server (Mulyanto, 2008).

2.5 Cara Kerja Gogle Maps

Google maps dibuat dengan menggunakan kombinasi dari gambar peta, *database*, serta objek-objek yang interaktif yang dibuat dengan bahasa pemrograman HTML, *javascripts,* dan *AJAX* serta beberapa bahasa pemrograman lainnya. Gambaran-gambaran peta yang muncul pada layar merupakan hasil komunikasi dari pengguna dengan *database* pada *web server google* untuk menampilkan gabungan dari potongan-potongan gambar yang diminta. Keseluruhan citra yang ada diintegrasikan ke dalam suatu *database* pada Goggle *sever,* yang nantinya akan dapat dipanggil sesuai dengan kebutuhan permintaan (Firman, 2014).

2.5.1 Google Maps Application Programming Interface (API)

Google maps adalah layanan gratis yang diberikan oleh Google dan sangat popular. *Google maps* adalah suatu peta dunia yang dapat digunakan untuk melihat suatu daerah. Dengan kata lain, *google maps* merupakan suatu peta yang dapat dilihat dengan menggunakan suatu *browser*. Pengguna dapat menambahkan fitur *google maps* dalam aplikasi yang telah pengguna buat. *Google maps ap*i adalah suatu *library* yang berbentuk JavaScript (Wardana, 2015).

Cara memasang *google maps* untuk ditampilkan pada suatu aplikasi sangat mudah hanya membutuhkan pengetahuan mengenai *javascript* serta koneksi internet yang stabil. Dengan menggunakan *google maps api,* pengguna dapat menghemat waktu dan biaya untuk membangun aplikasi peta digital yang handal, sehingga pengguna dapat fokus hanya pada data-data yang akan ditampilkan. Dengan kata lain, pengguna hanya membuat suatu data sedangkan peta yang akan

ditampilkan adalah milik Google, sehingga pengguna tidak dipusingkan dengan membuat peta suatu lokasi.

Pada *google maps* api terdapat 4 jenis pilihan model peta yang disediakan oleh Google, diantaranya adalah sebagai berikut (Wardana, 2015).

1. Roadmap, untuk menampilkan peta biasa dalam dua dimensi.
2. Satelit, untuk menampilkan foto satelit.
3. Terrain, untuk menunjukkan relief fisik permukaan bumi dan menunjukkan gunung dan sungai.
4. Hybrid, akan menunjukkan foto satelit yang di atasnya tergambar pula apa yang tampil pada roadmap (jalan dan nama kota).

Terdapat beberapa urutan yang perlu dilakukan dalam menggunakan Google Maps API, yaitu:

1. Memasukkan Maps API JavaScript ke dalam HTML.
2. Membuat element div dengan nama map_canvas untuk menampilkan peta.
3. Membuat beberapa objek literal untuk menyimpan property-properti pada peta.
4. Menuliskan fungsi JavaScript untuk membuat objek peta.
5. Menginisiasi peta dalam tag body HTML dengan event onload, atau dapat dilihat pada Gambar 2.9 (Shodiq, 2009).

```
<html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-
scalable=no" />

<!-- Langkah 1 -->
<script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=true&amp;key=ABQIAA
AA8tt4eKTuBZMVnLJfP2BZrBT2yXp_ZAY8_ufC3CFXhHIE1NvwkxS4Rz1LFzG0odNP
tk8VLkdrQF5grA">
</script>
<script type="text/javascript">

// Langkah 4
function initialize() {
var latlng = new google.maps.LatLng(-6.4, 106.8186111);
var myOptions = {
zoom: 13,
center: latlng,
mapTypeId: google.maps.MapTypeId.ROADMAP
};

// Langkah 3
var map = new
google.maps.Map(document.getElementById("map_canvas"), myOptions);
}
</script>
</head>

<!-- Langkah 5 -->
<body onload="initialize()">

<!-- Langkah 2 -->
<div id="map_canvas" style="width:600px; height:600px"></div>
</body>
</html>
```

Gambar 2.9 Sorce Code Google Maps API

2.6 Taman Margasatwa Ragunan

Taman Margasatwa Ragunan terletak di daerah Pasar Minggu, sekitar 20 km dari pusat kota Jakarta. Berdiri di atas tanah seluas 147 hektar dan berpenghuni lebih dari 2.009 ekor satwa serta ditumbuhi lebih dari 20.000 pohon membuat suasana lingkungannya sejuk dan nyaman. Berkunjung ke Taman Margasatwa Ragunan berarti memasuki sebuah hutan tropis mini, di dalamnya terdapat keanekaragaman hayati yang memiliki nilai konservasi tinggi dan menyimpan harapan untuk masa depan. Lahannnya tertata dan terbangun serta

sebagian lagi masih dikembangkan menuju suatu kebun binatang yang modern sebagai identitas kota Jakarta (Ragunan Zoo, 2016).