Pencarian Lintasan Terpendek Pada Peta Digital Menggunakan Teori Graf

Erfandi Suryo Putra 13515145
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515145@std.stei.itb.ac.id

***Abstrak*—Banyak orang yang memakai peta digital. Peta digital tidak hanya bisa digunakan untuk melihat suatu wilayah, tetapi juga bisa menuntun pengguna ke lokasi tertentu dengan rute terdekat. Untuk melakukan hal tersebut peta harus direpresentasikan dalam bentuk graf dan untuk menentukan rute terdekat dapat menggunakan algoritma *brute force*, dijkstra, atau A*.**

***Kata Kunci*—algoritma, bobot, graf, jarak, simpul, sisi.**

I. PENDAHULUAN

Di zaman modern ini kita hampir tidak bisa lepas dari teknologi. Teknologi, terutama teknologi digital, hampir selalu ada dalam segala aspek kehidupan modern manusia. Salah satu contoh teknologi digital yang sudah ada antara lain adalah internet, komputer, *smartphone*, media social, dll. Kita dapat menggunakan teknologi digital dalam berbagai hal, seperti untuk berkomunikasi, melakukan suatu pekerjaan, mencari informasi atau berita, bahkan sekarang sudah banyak sekolah-sekolah yang menggunakan teknologi digital dalam kegiatan belajar-mengajar. Dalam beberapa hal, orang-orang lebih memilih menggunakan teknologi digital dibanding menggunakan barang atau teknologi yang sebelumnya sudah mereka miliki karena kelebihan-kelebihan yang dimiliki teknologi digital, diantaranya adalah dalam mencari informasi, kita dengan sangat mudah dapat dengan mudah mencari segala informasi dari berbagai sumber melalui internet, selain kemudahan dalam mencari informasi, informasi baru, misalkan berita, juga bisa langsung diakses. Kelebihan lain dari teknologi digital adalah penggunaannya yang jauh lebih mudah, contohnya penggunaan *smartphone* yang bisa digunakan di mana saja untuk berbagai hal seperti berkomunikasi melalui media sosial atau melalui *chatting*, untuk menikmati hiburan seperti menonton film, mendengarkan musik, dll., *smartphone* juga bisa digunakan untuk membantu maupun mengerjakan suatu pekerjaan.

Penggunaan peta dalam bentuk fisik sekarang ini sudah jarang sekali digunakan banyak, orang-orang lebih memilih menggunakan peta digital. Penggunaan peta digital memang jauh lebih mudah dan juga jauh lebih efisien dibandingkan peta fisik. Kelebihan yang dimiliki peta digital adalah kemudahan pengaksesannya dan kepraktisannya, hal ini juga didukung oleh *smartphone* yang sudah sangat praktis dan canggih sehingga kita dapat mengakses peta itu di mana saja dan kapan saja hanya dengan menggunakan *smartphone*.

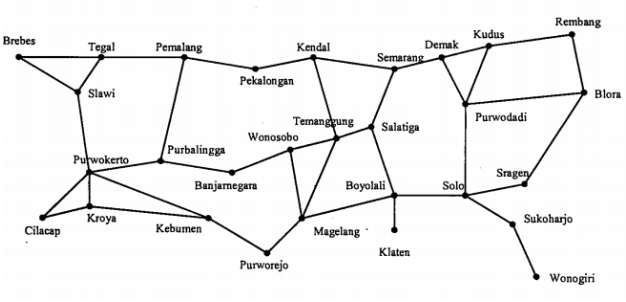
Penggunaan peta di *smartphone* tidak hanya bisa digunakan untuk melihat jalan saja, tetapi dengan *smartphone* yang dilengkapi dengan GPS (*Global Positioning System*), pengguna dapat melihat posisinya di peta itu secara *real-time*. Dengan adanya GPS, peta juga dapat menunjukkan jalan dari lokasi pengguna ke suatu tempat. Tidak hanya bisa menentukan jarak terdekat saja, peta juga bisa menentukan rute tercepat.

Untuk dapat memilih rute perjalanan terdekat, program peta harus dapat mengidentifikasi tempat-tempat yang ada di wilayah tersebut, peta juga harus mengetahui lintasan-lintasan yang dapat dilalui dan panjang setiap lintasan. Peta juga harus mengetahui data geografis suatu wilayah. Data geografis diperlukan jika lokasi pengguna sedang tidak berada di jalan yang tercatat di peta. Walaupun peta tidak mendeteksi suatu jalan, peta harus bisa mendeteksi jalan yang bisa dilalui pengguna. Hal ini diperlukan, terutama di Indonesia. Indonesia merupakan negara yang luas dan setiap daerahnya memiliki karateristik geografis yang berbeda-beda sehingga data geografis diperlukan untuk pemilihan retu perjalanan yang efektif dan aman.

Untuk menentukan rute perjalanan tercepat, peta memerlukan hal-hal yang sama dengan saat menentukan rute perjalanan terdekat. Akan tetapi saat menentukan rute perjalanan dengan jarak terdekat peta memerlukan data jarak setiap lintasan karena memerlukan jarak terdekat dari lokasi pengguna ke lokasi tujuan, maka untuk menentukan rute tercepat, peta memerlukan data perkiraan waktu yang diperlukan untuk melewati setiap lintasan. Data perkiraan waktu yang diperlukan untuk melalui suatu lintasan atau jalan bisa didapat dari tingkat kemacetan di jalan tersebut, ada tidaknya halangan atau rintangan, jenis lintasan yang dilalui, dll.

II. GRAF

Graf digunakan untuk mempresentasikan suatu objek diskrit dan hubungan antar objek-objek tersebut. Secara visual, suatu objek di graf direpresentasikan dengan sebuah titik, dan suatu hubungan antar objek direpresentasikan dengan sebuah garis yang menghubungkan kedua titik. Graf bisa digunakan untuk mepresentasikan berbagai hal, salah satu kegunaan graf adalah untuk mempresentasikan sebuah peta.



Gambar 1. Graf Peta

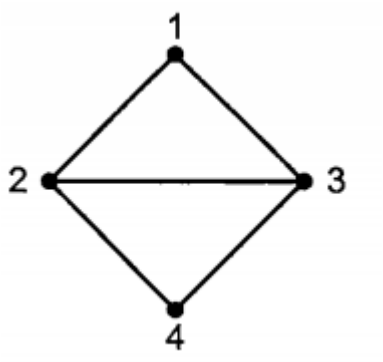
Pada gambar peta diatas, masing-masing kota direpresentasikan dengan sebuah titik dan lintasan-lintasan yang menghubungkan antar kota direpresentasikan dengan sebuah garis. Dengan adanya graf yang mempresentasikan suatu peta seperti graf di atas, kita dapat melihat apakah terdapat lintasan antara dua kota, selain itu kita juga dapat melihat panjang lintasan-lintasan antar kota, dan dengan diketahuinya panjang-panjang lintasan antar kota, kita juga dapat menentukan lintasan terpendek yang dapat dilalui dari satu kota ke kota lainnya.

Secara definisi, graf merupakan pasangan himpunan (V, E) dan dapat dinotasikan dengan G = (V, E) dengan V merupakan himpunan tidak kosong dari simpul-simpul yang mempresentasikan objek-objek, dan E merupakan merupakan himpunan sisi yang mempresentasikan hubungan antar objek atau simpul. Dari definisi tersebut dapat disimpulkan bahwa sebuah graf harus memiliki minimal satu buah objek atau simpul, tetapi boleh sama sekali tidak mempunyai sisi, yang berarti objek-objek yang dipresentasikan dengan simpul-simpul tidak mempunyai hubungan.

Menurut sudut pandang pengelompokannya, graf dapat dikelompokkan menjadi tiga kategori. Pengelompokan graf ini didasarkan pada ada tidaknya sisi ganda atau gelang, berdasarkan jumlah simpul, dan berdasarkan orientasi arah sisi.

Berdasarkan ada tidaknya gelang atau sisi ganda, graf dibagi menjadi dua jenis, yaitu graf sederhana dan graf tidak sederhana.

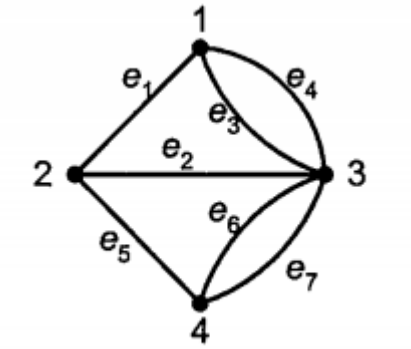
1. Graf sederhana



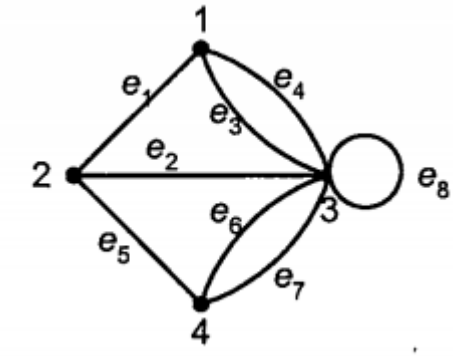
Gambar 2. Graf Sederhana

Graf sederhara merupakan graf yang tidak mempunyai sisi ganda maupun gelang

2. Graf Tidak Sederhana



Gambar 3. Graf Ganda



Gambar 4. Graf Semu

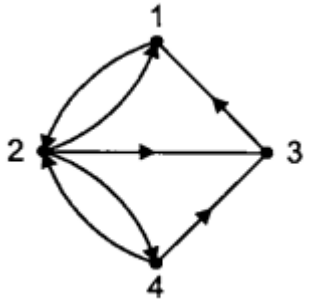
Graf tidak sederhana merupakan graf yang memiliki sisi ganda atau gelang. Graf tidak sederha juga dibagi menjadi dua, yaitu graf ganda dan graf semu. Graf ganda adalah graf yang mempunyai sisi ganda sementara graf semu adalah graf yang mempunyai gelang, walaupun juga mempunyai sisi ganda.

Berdasarkan orientasi sisinya, graf dibagi menjadi dua jenis yaitu tidak berarah dan graf berarah.

1. Graf Tidak Berarah

   Graf tidak berarah adalah graf yang sisinya tidak mempunyai orientasi arah. Pada graf berarah, (u, v) dan (v, u) merupakan dua sisi yang sama.

2. Graf Berarah



Gambar 5. Graf Berarah

   Graf berarah adalah graf yang setiap sisinya mempunyai orientasi arah. Pada graf berarah, (u, v) dan (v, u) merupakan dua sisi yang berbeda).

III. LINTASAN TERPENDEK GRAF

Dalam sebuah graf, terdapat beberapa jenis lintasan terpendek untuk persoalan-persoalan tertentu. Beberapa macam persoalan yang berkaitan dengan lintasan terpendek adalah sebagai berikut:

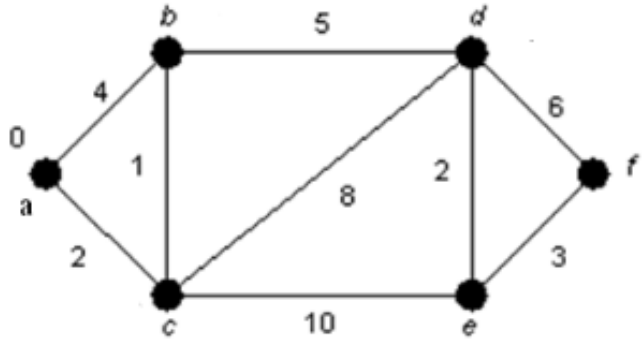
1. Lintasan terpendek antara dua buah simpul tertentu
2. Lintasan terpendek antara semua pasangan simpul
3. Lintasan terpendek antara simpul tertentu ke semua simpul lainnya
4. Lintasan terpendek antara dua buah simpul yang melalui beberapa simpul tertentu

Untuk menentukan lintasan terpendek dalam sebuah graf, dapat digunakan beberapa algoritma, antara lain algoritma *brute force*, algoritma dijkstra, dan algoritma A*.

A. Algoritma Brute Force

Algoritma *brute force* adalah algoritma yang digunakan dengan menyelesaikan masalah dengan pendekatan secara *straightforward*. Algoritma *brute force* ini didasarkan pada pernyataan pada persoalan dan definisi konsep yang yang dilibatkan, yang pada persoalan adalah pencarian jarak terdekat pada peta yang dipresentasikan sengan sebuah graf. Algoritma *brute force* menyelesaikan suaty nasalah dengan cara yang sangat sederhana, langsung, dan jelas.

Algoritma *brute force* ini adalah salah satu algoritma yang bisa digunakan untuk mencari jarak terdekat pada graf. Algoritma ini menentukan jarak terdekat dari satu simpul ke simpul lainnya dengan mengenumerasikan semua kemungkinan lintasan. Contoh penyelesaian pencarian jarak terdekat antara dua simpul di graf adalah sebagai berikut:



Gambar 6. Pencarian Jarak Terdekat Graf dengan Algoritma *Brute Force*

1. Misalnya kita ingin menentukan jarak dari a ke d
2. Enumerasi semua kemungkinan lintasan yang dapat dibentuk dari a ke d
3. Lintasan yang memiliki panjang atau bobot terkecil adalah lintasan terpendek dari a ke d

B. Algoritma Dijkstra

Algoritma dijkstra ditemukan oleh seorang ilmuwan komputer bernama Edsger Dijkstra yang berasal dari Belanda. Algoritma Dijkstra ini adalah salah satu algoritma *greedy* yang dipakai untuk mencari jarak terdekat dalam sebuah graf.

Graf yang digunakan dalam algoritma dijkstra harus merupakan graf berarah dan sisinya harus mempunyai bobot. Dalam graf ini, setiap kota dijadikan simpul dan setiap lintasan dijadikan sisi yang berbobot. Nilai dari bobot setiap sisi bisa berupa panjang lintasan, perkiraan waktu untuk melintasi lintasan tersebut, dll.

Algoritma ini memerlukan tempat asal dan tempat yang akan dituju. Algoritma ini akan menghasil sebuah lintasan dari tempat asal ke tempat tujuan dengan bobot terkecil.

Di bawah ini adalah *pseudocode* dari algoritma dijkstra.

```
function Dijkstra(Graph, source):
    for each vertex v in Graph:  // Initializations
     dist[v] := infinity ; // Unknown distance function from source to v
      previous[v] := undefined ; // Previous node in optimal path from source
    end for

    dist[source] := 0 ;  // Distance from source to source
    Q := the set of all nodes in Graph ; // All nodes in the graph are unoptimized - thus are in Q
    while Q is not empty:  // The main loop
     u := vertex in Q with smallest distance in dist[] ; // Start node in first case
```

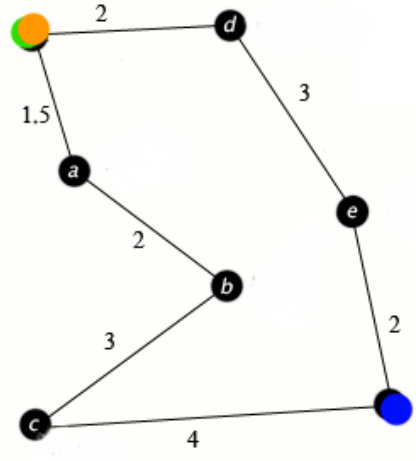
```
      remove u from Q ;
      if dist[u] = infinity:
       break ;  // all remaining vertices are inaccessible from source
      end if

      for each neighbor v of u: // where v has not yet been removed from Q.
        alt := dist[u] + dist_between(u, v) ;
        if alt < dist[v]: // Relax (u,v,a)
          dist[v] := alt ;
          previous[v] := u ;
          decrease-key v in Q;  // Reorder v in the Queue
        end if
      end for
    end while
  return dist;
```

*C. Algoritma A**

Selain kedua algoritma diatas, algoritma lain yang bisa digunakan untuk mencari jarak terdekat dua buah simpul atau dalam pembahasan ini jarak terdekat dua buah tempat, adalah algoritma A*. Algoritma ini pertama kali ditemukan oleh Peter Hart, Nils Nilsson, dan Bertram Raphael pada tahun 1968. Sama seperti algoritma dijkstra, algoritma A* juga memerlukan parameter tempat asal dan juga parameter tempat tujuan. Selain itu algoritma ini juga memerlukan simpul-simpul yang mempresentasikan tempat-tempat dalam peta. Simpul-simpul tersebut juga dihubungkan dengan sisi-sisi berbobot yang setiap sisi mempresentasikan lintasan yang bisa dilalui dari satu tempat ke tempat lainnya, dan nilai bobot bisa mempresentasikan panjang lintasan maupun lamanya waktu yang diperlukan untuk melewati lintasan tersebut.

Algoritma A* mencari semua kemungkinan lintasan yang dapat dilalui dari tempat awal ke tempat tujuan. Setelah algoritma membandingkan setiap kemungkinan lintasan dan memilih lintasan terpendek.

Di bawah ini adalah contoh pencarian jarak terpendek pada suatu graf menggunakan algoritma A*.



Gambar 7. Pencarian Jarak Terdekat Graf dengan Algoritma Brute Force

Misalnya titik kuning adalah tempat asal dan titik biru adalah tempat tujuan, maka algoritma akan mencari jarak terdekat dari titik kuning ke titik biru.

Algoritma ini akan melihat masing-masing lintasan

- Lintasan 1 (asal, d) = 2
- Lintasan 2 (asal, a) = 1,5
- Lintasan 1 > lintasan 2
- Lintasan yang dipilih adalah lintasan 2

- Lintasan 1 (asal, d) = 2
- Lintasan 2 (asal, a, b) = 1,5 + 2 = 3,5
- Lintasan 1 < lintasan 2
- Lintasan yang dipilih adalah lintasan 1

- Lintasan 1 (asal, d, e) = 2 + 3 = 5
- Lintasan 2 (asal, a, b) = 3,5
- Lintasan 1 > lintasan 2
- Lintasan yang dipilih adalah lintasan 2

- Lintasan 1 (asal, d, e) = 5
- Lintasan 2 (asal, a, b, c) = 3,5 + 3 = 6,5
- Lintasan 1 < lintasan 2
- Lintasan yang dipilih adalah lintasan 1

- Lintasan 1 (asal, d, e, tujuan) = 5 + 2 = 7
- Lintasan 2 (asal, a, b, c) = 6,5

Karena ada lintasan belum sampai ke tujuan, panjang lintasan tersebut perlu diperiksa terlebih dahulu sebelum kita bisa menyimpulkan panjang dan rute lintasan terpendek

- Lintasan 1 (asal, d, e, tujuan) = 7
- Lintasan 2 (asal, a, b, c, tujuan) = 6,5 + 4 = 10,5
- Lintasan 1 < lintasan 2

Karena setiap lintasan sudah mencapai tempat tujuan, kita sudah bisa menentukan lintasan terpendek dari tempat asal ke tempat tujuan, yaitu lintasan dengan nilai bobot terkecil. Dari contoh graf di atas, terlihat bahwa nilai bobot lintasan 1 lebih kecil dari lintasan 2, maka dapat disimpulkan bahwa rute terpendek dari tempat asal ke tempat tujuan adalah melalui lintasan dengan bobot bernilai 7, yang bobot tersebut bisa mempresentasikan jarak maupun waktu untuk melewati lintasan.

Di bawah ini adalah *pseudocode* dari algoritma A*.

```
function A*(start, goal)
    // The set of nodes already evaluated.
    closedSet := {}
    // The set of currently discovered nodes still to be evaluated.
    // Initially, only the start node is known.
    openSet := {start}
    // For each node, which node it can most efficiently be reached from.
    // If a node can be reached from many nodes, cameFrom will eventually contain the most efficient previous step.
    cameFrom := the empty map

    // For each node, the cost of getting from the start node to that node.
    gScore := map with default value of Infinity
    // The cost of going from start to start is zero.
    gScore[start] := 0
    // For each node, the total cost of getting from the start node to the goal by passing by that node. That value is partly known, partly heuristic.
    fScore := map with default value of Infinity
    // For the first node, that value is completely heuristic.
    fScore[start] := heuristic_cost_estimate(start, goal)

    while openSet is not empty
     current := the node in openSet having the lowest fScore[] value
      if current = goal
       return reconstruct_path(cameFrom, current)

      openSet.Remove(current)
      closedSet.Add(current)
      for each neighbor of current
       if neighbor in closedSet
        continue    // Ignore the neighbor which is already evaluated.
       // The distance from start to a neighbor
       tentative_gScore := gScore[current] + dist_between(current, neighbor)
       if neighbor not in openSet  // Discover a new node
        openSet.Add(neighbor)
       else if tentative_gScore >= gScore[neighbor]
        continue    // This is not a better path.

       // This path is the best until now. Record it!
       cameFrom[neighbor] := current
       gScore[neighbor] := tentative_gScore
       fScore[neighbor] := gScore[neighbor] + heuristic_cost_estimate(neighbor, goal)

    return failure

function reconstruct_path(cameFrom, current)
    total_path := [current]
    while current in cameFrom.Keys:
     current := cameFrom[current]
     total_path.append(current)
return total_path
```

IV. KESIMPULAN

Untuk mencari jarak terdekat pada sebuah peta, peta harus direpresentasikan dengan sebuah graf. Simpul-simpul di graf tersebut merupakan representasi suatu lokasi atau tempat di wilayah peta tersebut dan sisinya merupakan representasi lintasan antar tempat-tempat. Sisi-sisi graf ini harus merupakan sisi berbobot yang nilainya bisa mempresentasikan panjang lintasan maupun lama waktu yang diperlukan untuk melewati lintasan. Setelah peta sudah direpresentasikan dalam bentuk graf, pencarian rute terdekat dari suatu lokasi ke lokasi lainnya dapat dicari melalui algoritma *brute force*, dijkstra, atau A*.

REFERENSI

[1] Munir, Rinaldi, 2010, Matematika Diskrit edisi III. Bandung: Informatika Bandung.

[2] Munir, Rinaldi, 2004, Diktat Strategi Algoritmik. Bandung: Departemen Teknik Informatika Institut Teknologi Bandung

[3] http://www.boost.org/doc/libs/1_62_0/libs/graph/doc/dijkstra_shortest_paths.html diakses tanggal 7 Desember 2016.

[4] Hart, P. E.; Nilsson, N. J.; Raphael, B., 1968. A Formal Basis for the Heuristic Determination of Minimum Cost Paths, IEEE Transactions on Systems Science and Cybernetics.

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, 9 Desember 2016

Erfandi Suryo Putra 13515145