Pengaplikasian Algoritma Greedy Untuk Mencari Rute Terpendek

Christian Jonathan / 13516092
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516092@std.stei.itb.ac.id

***Abstract*— Seiring dengan berkembangnya teknologi, salah satu teknologi yang sangat membantu masyarakat adalah teknologi GPS yang memudahkan masyarakat dalam berpergian tanpa perlu takut menyasar. Teknologi navigasi GPS ini dipadukan dengan peta berbasis internet sehingga memungkinkan memberi informasi terhadap suatu jalan dengan akurat, arah serta cara menuju ke suatu alamat yang dituju. Makalah ini akan menjelaskan tentang algoritma Dijkstra yang memakai pendekatan algoritma Greedy untuk mencari rute terpendek. Algoritma ini kerap dipakai oleh aplikasi-aplikasi navigasi seperti google maps.**

***Keywords*— Dijkstra Algorithm, Greedy Algortihm, Google Maps, Shortest-Path.**

I. PENDAHULUAN

Tidak bisa dipungkiri lagi bahwa teknologi sudah kian berkembang dengan pesat selama satu dekade belakangan ini. Tidak hanya itu, teknologi juga sudah melekat dengan erat pada kehidupan manusia saat ini, bahkan untuk urusan yang sifatnya kecil. Teknologi juga memungkinkan manusia untuk berkomunikasi kapan pun, dimana pun dan dengan siapa pun. Hal ini tentunya merupakan suatu hal yang sangat revolusioner ketika kita membandingkan dengan masa lalu yang mengandalkan komunikasi lewat surat.

Namun, seiring dengan berkembangnya teknologi, tingkat mobilisasi pun meningkat dengan sangat pesat. Tidak kurang dari seratus ribu penerbangan terjadi setiap harinya, mobilisasi pada area darat dan perairan pun meningkat dengan signifikan. Mobilisasi-mobilisasi itu tentunya memerlukan tenaga, sehingga sangat penting bagi kita untuk menentukan rute terpendek agar mobilisasi efisien, cepat dan tepat.

Kalkulasi jalur terpendek, arah serta perkiraan waktu tempuh dapat dilakukan dengan menerapkan berbagai macam algoritma. Pada makalah ini, akan dijelaskan dengan pendekatan algoritma dijkstra yang menerapkan algorita *Greedy*, dimana algoritma ini akan dijelaskan pada sub-bab berikutnya.

Algoritma Dijkstra ini merupakan algoritma yang paling kerap dipakai oleh aplikasi-aplikasi navigasi terkenal yang dipakai oleh banyak masyarakat, seperti contoh nya google maps. Aplikasi ini tidak hanya memperlihatkan peta secara *online/offline*, namun dapat memberikan navigasi dari daerah asal menuju daerah tujuan dengan memanfaatkan teknologi GPS untuk menentukan posisi pengguna saat ini, dan memberikan rute yang terpendek. Hal ini sangat membantu mobilisasi manusia yang sudah meningkat ini sehingga pengguna tidak perlu takut untuk berpergian ke tempat yang belum atau tidak mereka ketahui karena akan diberikan arah sehingga mereka tidak tersesat.

II. LANDASAN TEORI

1. Graf

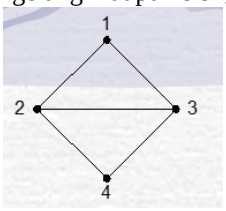
Graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut (contoh : graf peta jalan raya yang menghubungkan antar kota).

Sebuah graf direpresentasikan dari himpunan tidak kosong vertex (jika melihat dari contoh representasi peta antar kota maka vertex merupakan kota-kota nya) dan edge yang menjadi representasi hubungannya (jalan rayanya). Notasi umum graf adalah G = (V,E) dimana G adalah graf, V adalah himpunan tidak kosong vertex dan E adalah himpunan edge.

Graf dapat dikelompokkan menjadi berbagai jenis bergantung dengan dasar pengelompokkannya. Jika didasarkan dari ada atau tidaknya gelang atau sisi ganda, graf dikelompokkan mnejadi dua yaitu :

- Graf Sederhana :

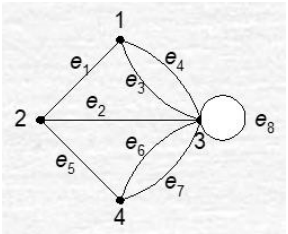
  Graf sederhana adalah graf yang tidak memiliki sisi gelang maupun sisi ganda



Sumber (Slide perkuliahan matdis Rinaldi Munir – Graf)

- Graf Tak-sederhana :

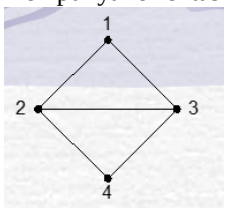
  Graf tak-sederhana adalah graf yang mengandung sisi ganda atau gelang



Sumber (Slide perkuliahan matdis Rinaldi Munir – Graf)

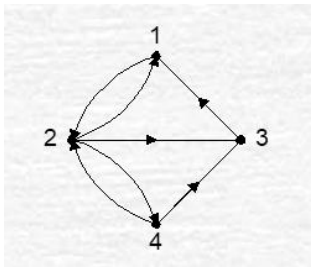
Jika didasarkan dari orientasi arah pada sisi(vertex), maka secara umum graf dibedakan menjadi 2 jenis:

- Graf tak-berarah :
  Graf tak-berarah adalah graf yang sisinya tidak mempunyai orientasi arah.



Sumber (Slide perkuliahan matdis Rinaldi Munir – Graf)

- Graf berarah
  Graf berarah adalah graf yang sisinya memiliki orientasi arah



Sumber(Slide perkuliahan matdis Rinaldi Munir – Graf)

Untuk mempelajari Graf secara lebih, perlu diketahui beberapa terminologi sebagai berikut :

- Bertetangga (Adjacent)
  Dua buah simpul pada graf tak-berarah jika keduanya terhubung dikatakan bertetangga.
- Bersisian (incident)
  Untuk sembarang sisi e = (vj,vk) sisi e dikatakan bersisian dengan simpul vj dan vk.
- Simpul terpencil (Isolated Vertex)
  Simpul yang tidak mempunyai sisi yang bersisian dengannya dikatan simpul terpencil.
- Graf Kosong (Null Graph)
  Graf yang himpunan sisinya merupakan himpunan kosong.
- Derajat (Degree)
  Derajat suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut.
- Lintasan (Path)
  Lintasan yang panjangnya n dari simpul awal va menuju simpul vk dengan melewati berbagai sisi dan simpul secara bergantian
- Siklus/Sirkuit (Cycle/Circuit)
  Lintasan yang berawal dan berakhir pada simpul yang sama
- Terhubung (Connected)
  Suatu Graf tak-berarah merupakan graf terhubung jika untuk setiap simpul pada graf terdapat lintasan yang menuju simpul tersebut
- Upagraf (Subgraph)
  Upagraf merupakan bagian dari suatu graf (subset suatu graf)
- Upagraf Merentang (Spanning Subgraph)
  Suatu upagraf yang upagraf tersebut terdapat semua simpul dari graf utama.
- Cut-Set
  Cut-Set dari suatu graf adalah apabila beberapa anggota dari himpunan sisi dibuang menyebabkan graf tidak lagi terhubung
- Graf Berbobot (Weighted Graph)
  Graf berbobot adalah graf yang setiap sisinya diberikan nilai / bobot (cost).

2. Algoritma Greedy

Algoritma greedy merupakan metode yang paling popular untuk memecahkan persoalan optimasi. Persoalan optimasi adalah persoalan mencari solusi optimum, ada dua persoalan optimasi yaitu maksimasi dan minimasi.

Prinsip greedy adalah “take what you can get now!” atau berarti algoritma ini membentuk solusi langkah per langkah namun pada setiap langkah terdapat banyak pilihan yang perlu dievaluasi oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan.

Pada setiap langkah, kita membuat pilihan optimum lokal dengan harapan bahwa langkah sisanya mengarah ke solusi optimum global. Elemen-elemen algoritma greedy:

- Himpunan kandidat, C.
- Himpunan solusi, S
- Fungsi seleksi (selection function)
- Fungsi kelayakan (feasible)
- Fungsi obyektif

Dengan kata lain algoritma greedy melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat **C** yang dalam hal ini, **S** harus memenuhi beberapa kriteria yang ditentukan, yaitu menyatakan suatu solusi dan **S** dioptimisasi oleh fungsi obyektif.

Strategi algoritma greedy terpakai oleh algoritma Dijkstra dimana pada setiap langkah, ambil sisi yang nilainya paling kecil yang menghubungkan vertex yang sudah dilewati dengan vertex lain yang belum dilewati. Lintasan dari vertex asal ke vertex yang baru harus merupakan lintasan yang terpendek diantara semua lintasannya ke vertex-vertex yang belum dilewati.

Berikut adalah skema umum algoritma greedy:

```
function greedy(input C: himpunan_kandidat)→ himpunan_kandidat
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy
  Masukan: himpunan kandidat C
  Keluaran: himpunan solusi yang bertipe himpunan_kandidat
}
Deklarasi
    x : kandidat
    S : himpunan_kandidat

Algoritma:
    S ← {}   { inisialisasi S dengan kosong }
    while (not SOLUSI(S)) and (C ≠ {} ) do
       x ← SELEKSI(C)          { pilih sebuah kandidat dari C}
       C ← C - {x}             { elemen himpunan kandidat berkurang satu }
       if LAYAK(S ∪ {x}) then
           S ← S ∪ {x}
       endif
    endwhile
    {SOLUSI(S) or C = {} }

    if SOLUSI(S) then
        return S
    else
        write('tidak ada solusi')
    endif
```

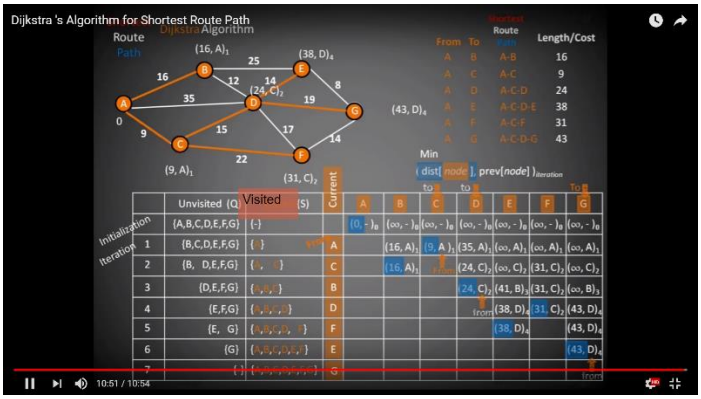
Sumber (Slide perkuliahan Rinaldi Munir – Greedy).

3. Algoritma Dijkstra

Algoritma Dijkstra ditemukan oleh Edsger W. Dijkstra pada tahun 1959. Algoritma ini merupakan salah satu algoritma yang populer untuk masalah optimasi pencarian rute/lintasan terpendek (*shortest path*). Algoritma ini mencari lintasan terpendek dengan menghitung lintasan dari verteks a ke f (misal) dalam graf berbobot berarah yang tidak diperbolehkan bernilai negatif.

Algoritma ini bertujuan untuk menemukan jalur terpendek berdasarkan nilai bobot terkecil dari satu titik ke titik lainnya. Misalkan titik(vertex) menunjukan gedung dan garis(edge) menunjukan jalan, maka algoritma Dijkstra akan melakukan semua kemungkinan bobot terkecil dari setiap titik. Dari vertex awal (pada kasus ini vertex a), diberi bobot jarak dari vertex pertama menuju ke vertex terdekat satu persatu, setelah itu algoritma ini akan mengembangkan pencarian dari satu titik ke titik lainnya dan selanjutnya secara bertahap. Berikut cara kerja algoritma Dijkstra:

1. Beri nilai untuk setiap vertex ke vertex lainnya, lalu set nilai 0 pada vertex awal dan nilai tak hingga untuk vertex lain
2. Set semua vertex “belum didatangi” dan vertex awal sebagai “start vertex”
3. Dari start vertex, dicek vertex tetangga yang belum didatangi dan hitung jaraknya dari start vertex. Sebagai contoh jika start vertex A ke B memiliki jarak 6 dan dari B ke vertex C berjarak 2, maka jarak ke C melewati B menjadi 6+2 = 8. Jika jarak ini lebih kecil dari jarak sebelumnya(yang sudah terekam), hapus data lama, simpan ulang data jarak dengan jarak yang baru.
4. Saat selesai cek setiap jarak terhadap vertex tetangga, tandai vertex yang telah didatangi sebagai “sudah didatangi”. Vertex yang sudah didatangi tidak akan di cek kembali, jarak yang disimpan adalah jarak terakhir yang paling minimal bobotnya.
5. Set vertex “belum didatangi” dengan jarak terkecil (dari start vertex) sebagai start vertex selanjutnya dan kembali dilakukan langkah ke 3.



Sumber (https://www.youtube.com/watch?v=dS1Di2ZHl4k).

Algoritma Dijstra dapat dijelaskan dengan pseudo-code dibawah ini:

```
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
    end for                                     // from source

    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
        end if                                  // inaccessible from source

        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;
```

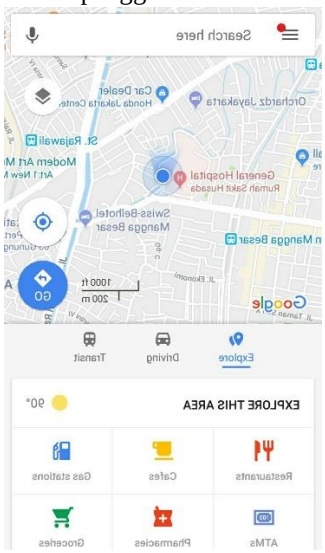
Sumber (https://id.wikipedia.org/wiki/Algoritme_Dijkstra).

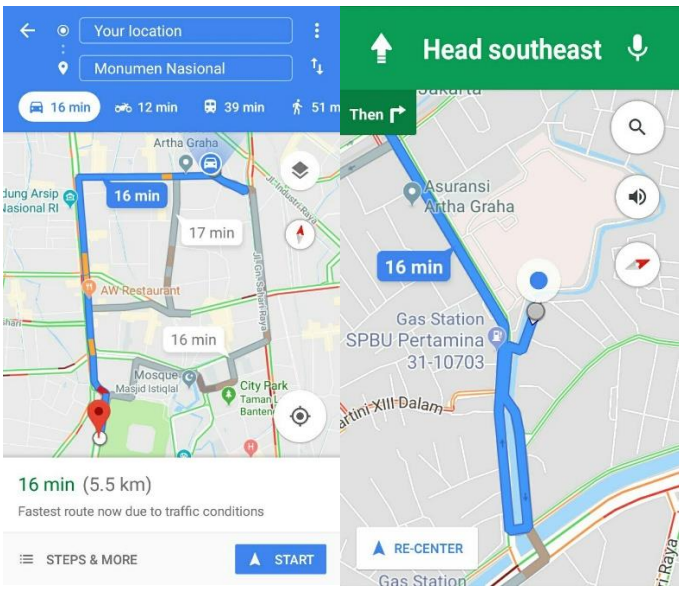
III. PENJELASAN PRINSIP KERJA

Walaupun tidak dapat dipastikan seratus persen hasil dari algoritma Dijkstra merupakan hasil yang paling optimum seperti exhaustive search secara brute force, algoritma dijkstra merupakan algoritma yang populer untuk permasalahan mencari rute terpendek dengan menggunakan strategi algoritma greedy.

Penggabungan algoritma dijkstra dengan GPS ini, memampukan aplikasi-aplikasi navigasi seperti google maps untuk mencari rute terpendek sekaligus dengan mengarahkan pengguna aplikasi menuju tempat tujuan.

Dengan menggunakan gedung-gedung / beberapa titik di jalan raya serta persimpangan jalan sebagai node dan jalanan sebagai edgenya. Dengan memakai titik pengguna yang diambil dari GPS sebagai node awal, dapat dicari rute terpendek yang dapat dilalui pengguna dari posisi pengguna ke posisi tujuan. Tentunya bobot sisi yang digunakan oleh google maps tidak hanya berdasarkan jarak nya, namun diperhitungkan juga tingkat kemacetannya, sehingga dapat tercapai rute perjalanan paling efisien yang dapat dilalui oleh pengguna.





Sumber (Koleksi pribadi penulis)

Tentunya pada implementasi sistem navigasi terkenal seperti google maps, tidak secara seratus persen hanya memakai algoritma Dijkstra. Algoritma ini hanya dipakai sebagai dasar/landasan teori untuk mencari rute terpendek. Berbagai algoritma tambahan dipakai agar mempercepat proses aplikasi sehingga algoritma lebih mangkus seperti:

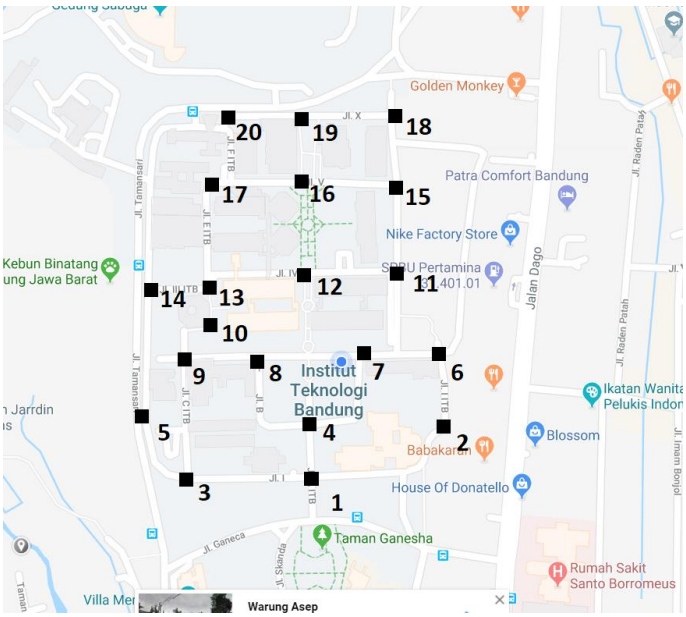
1. Bidirectional Search
2. Priority Queue
3. Geometric Goal Directed Search (A*)
4. Heuristic

Pencarian juga dipercepat dengan melakukan eksploitasi hierarki jalan-jalan yang menjadi sisi. Hal ini dapat dilakukan dengan melakukan preprocessing atau memproses hirearki yang ditentukan sendiri sebelum program mencari rute. Ini membuat algoritma menjadi lebih mangkus karena jalan yang jauh dari node awal tidak akan diperhitungkan oleh algoritma, sehingga performa aplikasi dan akurasi hasil dari aplikasi meningkat.

Dengan itu semua, maka terancang suatu algoritma yang kompleks namun memiliki runtime yang sangat jauh lebih cepat dan efisien dibandingkan dengan algoritma Dijkstra biasa yang harus memperhitungkan semua kemungkinan dari rute jalan yang ada.

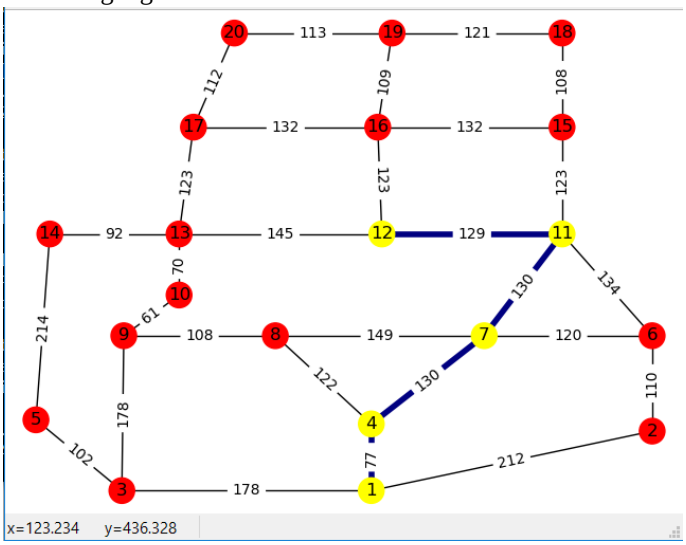
IV. CONTOH PENERAPAN ALGORITMA

Dengan memakai peta itb yang diambil dari google map dan diberi titik titik secara manual sebagai node akan dijelaskan dengan rinci bagaimana cara kerja algoritma Dijkstra pada sub-bab ini, pada gambar ini akan diperlihatkan peta itb beserta titik titik dan penomoran node serta jarak dalam satuan meter antar node sebagai cost sisinya.



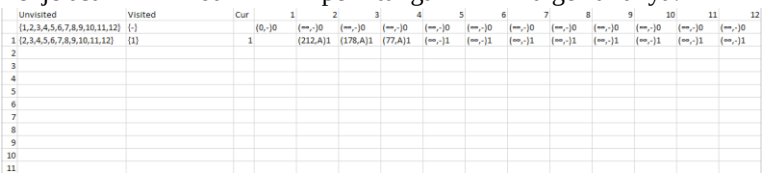
Sumber (Koleksi pribadi penulis dari tugas kecil ke-3)

Jika Start awal kita gerbang utama ITB yang berada pada node 1 dan tujuan kita adalah daerah gedung TVST dan PLN yang berada di node 12, maka jarak terdekat yang dilewati adalah sebagai gambar berikut



Sumber (Koleksi pribadi penulis dari tugas kecil ke-3)

Penjelasan dan perhitungan algoritmanya:



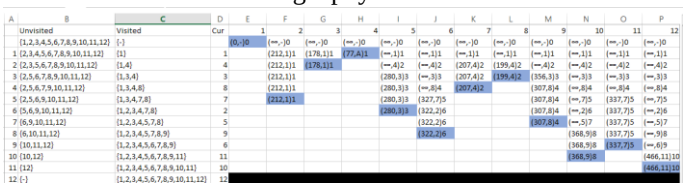
Sumber (koleksi penulis).

Gambar diatas merupakan inisiasi awal tabel yang menghitung serta menyimpan nilai jarak minimal yang akan dipakai oleh algoritma Dijkstra, pada setiap iterasi akan di cek misal mau ke node 9 dari 1 bisa melewati 3 lalu ke 9 atau dari 1 melewati 4,8 lalu 9. Dari kedua kemungkinan itu akan di cek mana yang bobotnya lebih minimal, dan akan menyimpan nilai tersebut. Iterasi dilakukan terus menerus sampai semua node

nya terkunjungi (untuk membatasi permasalahan, penulis hanya menghitung sampai node ke 12 karena mencontohkan jarak terpendek dari node 1 ke node 12).

Cara perhitungan tabel adalah dengan mencari nilai minimum lalu menuliskan di tabel dengan format (dist[node],prev[node])iteration-I (contoh (121,A)1 artinya berbobot 121 dari node A dan didapat dari iterasi pertama).

Berikut adalah Tabel lengkapnya:



	Unvisited	Visited	Cur	1	2	3	4	5	6	7	8	9	10	11	12
	{1,2,3,4,5,6,7,8,9,10,11,12}	{-}		(0,-)0	(∞,-)0	(∞,-)0	(∞,-)0	(∞,-)0	(∞,-)0	(∞,-)0	(∞,-)0	(∞,-)0	(∞,-)0	(∞,-)0	(∞,-)0	(∞,-)0
1	{2,3,4,5,6,7,8,9,10,11,12}	{1}	1	(212,1)1	(178,1)1	(77,A)1	(∞,-)1	(∞,-)1	(∞,-)1	(∞,-)1	(∞,-)1	(∞,-)1	(∞,-)1	(∞,-)1	(∞,-)1	(∞,-)1
2	{2,3,5,6,7,8,9,10,11,12}	{1,4}	4	(212,1)1	(178,1)1		(∞,-)4	(∞,-)2	(207,4)2	(199,4)2	(∞,-)4	(∞,-)2	(∞,-)2	(∞,-)2	(∞,-)2	
3	{2,5,6,7,8,9,10,11,12}	{1,3,4}	3	(212,1)1			(280,3)3	(∞,-)3	(207,4)2	(199,4)2	(356,3)3	(∞,-)3	(∞,-)3	(∞,-)3	(∞,-)3	(∞,-)3
4	{2,5,6,7,9,10,11,12}	{1,3,4,8}	8	(212,1)1			(280,3)3	(∞,-)4	(207,4)2		(307,8)4	(∞,-)4	(∞,-)4	(∞,-)4	(∞,-)4	
5	{2,5,6,9,10,11,12}	{1,3,4,7,8}	7	(212,1)1			(280,3)3	(327,7)5			(307,8)4	(∞,-)5	(337,7)5	(∞,-)5	(∞,-)5	
6	{5,6,9,10,11,12}	{1,2,3,4,7,8}	2				(280,3)3	(322,2)6			(307,8)4	(∞,-)6	(337,7)5	(∞,-)6		
7	{6,9,10,11,12}	{1,2,3,4,5,7,8}	5					(322,2)6			(307,8)4	(∞,-)5	(337,7)5	(∞,-)5		
8	{6,10,11,12}	{1,2,3,4,5,7,8,9}	9					(322,2)6				(368,9)8	(337,7)5	(∞,-)8		
9	{10,11,12}	{1,2,3,4,5,6,7,8,9}	6									(368,9)8	(337,7)5	(∞,-)9		
10	{10,12}	{1,2,3,4,5,6,7,8,9,11}	11									(368,9)8		(466,11)10		
11	{12}	{1,2,3,4,5,6,7,8,9,10,11}	10											(466,11)10		
12	{-}	{1,2,3,4,5,6,7,8,9,10,11,12}	12													

Sumber (Koleksi Penulis)

Berikut penjelasan cara kerja dan pengisian tabel :

1 Beri nilai untuk setiap vertex ke vertex lainnya, lalu set nilai 0 pada vertex awal dan nilai tak hingga untuk vertex lain

2 Set semua vertex “belum didatangi” dan vertex awal sebagai “start vertex”

3 Dari start vertex, dicek vertex tetangga yang belum didatangi dan hitung jaraknya dari start vertex. Sebagai contoh jika start vertex A ke B memiliki jarak 6 dan dari B ke vertex C berjarak 2, maka jarak ke C melewati B menjadi 6+2 = 8. Jika jarak ini lebih kecil dari jarak sebelumnya(yang sudah terekam), hapus data lama, simpan ulang data jarak dengan jarak yang baru.

4 Saat selesai cek setiap jarak terhadap vertex tetangga, tandai vertex yang telah didatangi sebagai “sudah didatangi”. Vertex yang sudah didatangi tidak akan di cek kembali, jarak yang disimpan adalah jarak terakhir yang paling minimal bobotnya.

5 Set vertex “belum didatangi” dengan jarak terkecil (dari start vertex) sebagai start vertex selanjutnya dan kembali dilakukan langkah ke 3.

Pengisian tabel dimulai dengan setting semua nilai menjadi tak hingga terlebih dahulu dan start awal menjadi (0,-)0 seperti pada gambar di iterasi ke-0. Karena nilai 0 itu adalah nilai paling minimum (nilai minimum disini diberikan warna kotak biru muda), maka dimulai dari node tersebut (node 1).

Dari node 1 dilakukan pengecekan terhadap hubungan hubungannya, jika berhubungan, tuliskan nilainya jika tidak, tetap berikan nilai tak hingga tetapi node sebelumnya diganti menjadi node current nya (∞,1)1. Setelah itu dicek lagi hasil iterasi pertama mana yang paling minimum, dari tabel diatas didapati node 4, sehingga ganti current node menjadi node 4 dan lakukan iterasi kedua. Dari node 4 dicek hubungannya, jika tidak berhubungan dan nilainya berhingga, ambil nilai sebelumnya (dari baris atasnya), namun jika nilainya tak hingga, lakukan seperti node 1 yaitu mengubah nilainya menjadi (∞,4)2.

Jika dicek ada hubungan dan sudah ada nilainya, lakukan pengecekan apakah dengan menggunakan nilai bobot yang melewati node 4 lebih kecil dari nilai yang didapati dari melewati node sebelumnya, jika lebih kecil, maka nilai yang tercatat adalah nilai yang lebih kecil, namun jika lebih besar, maka ambil nilai sebelumnya yang ada pada baris atasnya.

Hal ini dilakukan secara terus-menerus sampai semua node terkunjungi. Hal ini akan menghasilkan semua shortest path dari node awal ke node-node lain yang akan dijabarkan sebagai berikut:

Node awal	Node Tujuan	Lintasan	Cost
1	2	1-2	212
1	3	1-3	178
1	4	1-4	77
1	5	1-3-5	280
1	6	1-2-6	322
1	7	1-4-7	207
1	8	1-4-8	199
1	9	1-4-8-9	307
1	10	1-4-8-9-10	368
1	11	1-4-7-11	337
1	12	1-4-7-11-12	466

Dari hasil yang didapat, menurut perhitungan manual secara pribadi dan menurut perhitungan secara algoritma Dijkstra didapati hasil yang sama.

V. KESIMPULAN

Pengaplikasian algoritma Dijkstra untuk mencari rute terpendek masih sangat populer dibandingkan dengan algoritma-algoritma lain. Dalam sistem pencarian rute terpendek pada fitur navigasi perangkat lunak seperti Google maps, modifikasi algoritma Dijkstra sangat diperlukan untuk pencapaian runtime yang jauh lebih cepat dan efisien serta lebih tepat. Meski demikian, sistem pencarian rute teroptimal kedepannya pasti lebih akan berkembang dan patut untuk dipelajari. Karena menurut Andrew Goldberg, peneliti utama di Microsoft Research Silicon Valley, beliau mengatakan “Jalan terpendek adalah masalah optimasi yang relevan untuk berbagai macam aplikasi, seperti jaringan routing, game, desain sirkuit, dan pemetaan” (dikutip dari http://pendi.web.id/algoritma-dijkstra/).

VI. UCAPAN TERIMA KASIH

Pertama saya sangat berterima kasih dan bersyukur kepada Tuhan Yang Maha Esa, kepada orang tua saya atas kesempatan yang diberikan untuk membuat makalah ini dan atas suksesnya pengerjaan makalah ini tanpa adanya halangan sehingga saya dapat menyelesaikan makalah ini dengan cukup baik. Saya juga mengucapkan terima kasih kepada dosen pengajar mata kuliah Strategi Algoritma IF2211 atas pemberian materi Algoritma Greedy sehingga saya dapat menemukan inspirasi pengerjaan makalah ini dan dapat belajar lebih banyak hal dari tugas makalah ini. Yang terakhir tidak lupa saya ucapkan terima kasih kepada teman-teman yang turut mendukung dan mendorong motivasi saya dalam pembuatan makalah ini sampai selesai dengan baik.

REFERENSI

[1] http://www.ms.unimelb.edu.au/~moshe/620-261/dijkstra/dijkstra.html Diakses pada tanggal 9 Mei 2018 pukul 20.00

[2] http://algs4.cs.princeton.edu/44sp/ Diakses pada tanggal 9 Mei 2018 pukul 21.00

[3] http://interactivepython.org/RkmcZ/courselib/static/pythonds/Graphs/graphshortpath.html Diakses pada tanggal 9 Mei 2018 pukul 23.00

[4] https://mti.binus.ac.id/2017/11/28/algoritma-dijkstra/ Diakses pada tanggal 10 Mei 2018 pukul 18.00

[5] http://pendi.web.id/algoritma-dijkstra/ Diakses pada tanggal 10 Mei 2018 pukul 19.00

[6] Slide perkuliahan Matematika Diskrit Rinaldi Munir – Graf (2015), Diunduh dari http://informatika.stei.itb.ac.id/~rinaldi.munir pada tanggal 11 Mei 14.00

[7] Slide perkuliahan Strategi Algoritma Rinaldi Munir – Algoritma Greedy (2016), Diunduh dari http://informatika.stei.itb.ac.id/~rinaldi.munir pada tanggal 11 Mei 15.00

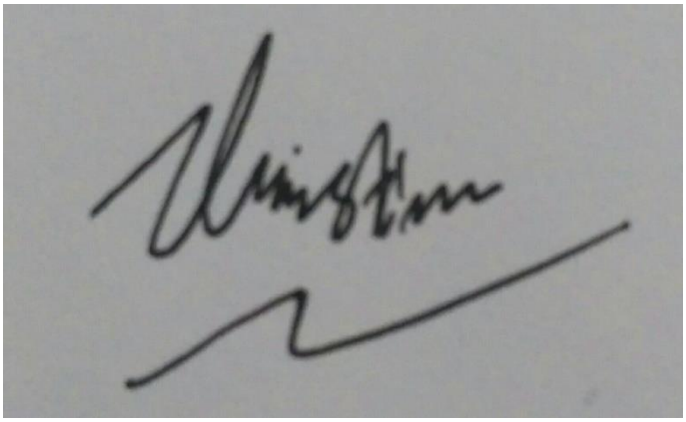
[8] https://www.geeksforgeeks.org/greedy-algorithms-set-6-dijkstras-shortest-path-algorithm/ diakses pada tanggal 12 Mei 2018 pukul 13.00

[9] https://motherboard.vice.com/en_us/article/4x3pp9/the-simple-elegant-algorithm-that-makes-google-maps-possible Diakses pada tanggal 12 Mei 2018 pukul 17.00

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, 11 Mei 2018



Christian Jonathan / 13516092