Penerapan Algoritma Dijkstra pada Link State Routing Protocol untuk Mencari Jalur Terpendek

Muhammad Ghufron Mahfudhi / 13508020
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if18020@students.if.itb.ac.id

***Abstrak*— Makalah ini membahas penerapan algoritma *dijkstra* pada *link state routing protocol* untuk mencari jalur terpendek. Dalam penerapan tersebut, kita harus dapat memahami jaringan merupakan salah satu aplikasi dari teori graf. Untuk menentukan jalur terpendek dalam jaringan tersebut diperlukan suatu *routing protocol* yang dapat menentukan tabel *routing*. Dalam menentukan tabel *routing* tersebut, sebuah *router* harus tahu di mana saja *router* tersebut terhubung dan bagaimana mencapai tujuan proses *routing*. Oleh karena itu, diperlukan suatu algoritma yang sesuai untuk menentukan jalur terpendek tersebut. Contoh dari cara *routing* yang digunakan secara umum yang dibahas dalam makalah ini yaitu proses *link state routing protocol*. Dalam *routing protocol* ini diterapkan algoritma *shortest path first* atau yang kita kenal dengan algoritma *dijkstra* yang merupakan penerapan dari algoritma *greedy*.**

***Kata Kunci*—Dijkstra, Graf, Greedy, Link State Routing Protocol.**

I. PENDAHULUAN

Graf merupakan pokok bahasan yang sudah tua usianya, tetapi memiliki banyak terapan sampai saat ini. Graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Representasi visual dari graf adalah dengan objek dinyatakan sebagai noktah, bulatan, atau titik, sedangkan hubungan antar objek dinyatakan dengan garis.

Salah satu contoh penerapan graf yaitu dalam jaringan komputer dan yang menjadi objek-objeknya adalah router. Jaringan berfungsi sebagai media penyampaian data atau informasi dari suatu pihak yang merupakan pengirim dan pihak lain yang menjadi tujuan pengiriman.

Pada pengiriman informasi dalam jaringan, kita pasti menginginkan waktu yang paling cepat. Untuk itu, harus dicari terlebih dahulu jalur yang paling pendek dan efektif dari pengirim sampai tujuan. Dalam penentuan jalur tersebut kita membutuhkan suatu aturan yang disebut *routing protocol*.

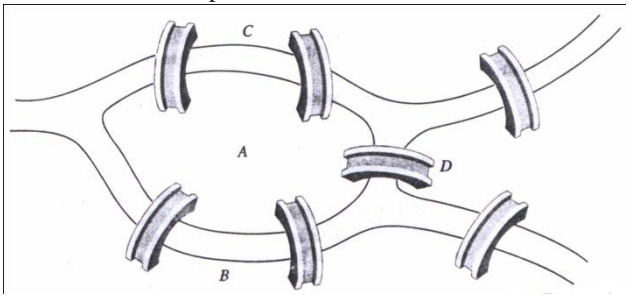
Salah satu jenis *routing protocol* yang sering diterapkan yaitu *link state routing protocol*. Dengan *routing protocol* ini, kita dapat memperoleh tabel *routing* dan menentukan jalur terpendek. Untuk membuat tabel *routing* tersebut, kita dapat menerapkan prinsip algoritma *greedy*. Dalam makalah ini akan dijelaskan bagaimana penerapan algoritma *greedy* (*dijkstra*) dalam menentukan tabel *routing* pada *link state routing protocol*.

II. DASAR TEORI

A. Graf

Graf adalah himpunan benda-benda yang disebut simpul atau noktah (*vertex*) yang terhubung oleh sisi (*edge*). Biasanya graf digambarkan sebagai kumpulan titik-titik yang melambangkan simpul dan dihubungkan oleh garis-garis yang melambangkan sisi. Suatu sisi dapat menghubungkan suatu simpul dengan simpul yang sama. Sisi yang demikian dinamakan gelang (*loop*).

Orang yang pertama kali menerapkan konsep graf ini adalah L. Euler, yang merupakan matematikawan Swiss, pada tahun 1736. Dia menerapkan konsep graf ini untuk memodelkan masalah jembatan Konigsberg. Masalah ini berasal dari keadaan tujuh buah jembatan di Kota Konigsberg, Jerman, yang menghubungkan sungai Pregal yang mengalir mengitari Pulau Kneiphof lalu bercabang menjadi dua anak buah sungai. Tujuan dari masalah Konigsberg ini yaitu untuk mengetahui apakah mungkin melalui ketujuh jembatan masing-masing tepat satu kali dan kembali ke tempat semula.

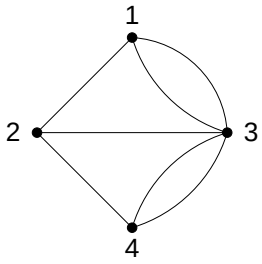


Gambar 1. Masalah Jembatan Konigsberg

Jawaban yang dikemukakan oleh Euler yaitu orang tidak mungkin melalui ketujuh jembatan itu masing-masing satu kali dan kembali ke tempat asal keberangkatan jika derajat setiap simpul tidak seluruhnya genap. Yang dimaksud dengan derajat adalah banyaknya garis yang bersisian dengan noktah. Sebagai contoh, simpul C memiliki derajat tiga karena tiga buah garis yang bersisian dengannya, simpul B dan D, juga berderajat dua, sedangkan simpul A berderajat lima.

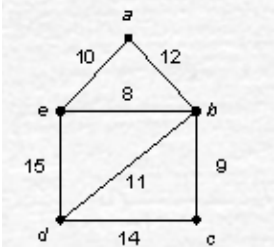
Karena tidak semua simpul berderajat genap, maka tidak mungkin dilakukan perjalanan berupa sirkuit pada graf tersebut.

Pada suatu graf, lintasan merupakan barisan berselang-seling dari simpul-simpul dan sisi-sisi. Jika setiap sisi hanya dilalui sekali (setiap simpulnya berbeda), maka disebut lintasan sederhana. Dua buah simpul dikatakan bertetangga jika keduanya terhubung langsung dengan sebuah sisi atau busur. Dua buah simpul dikatakan terhubung jika terdapat lintasan dari simpul yang satu ke simpul yang lain.



Gambar 2. Contoh Graf

Graf dapat direpresentasikan dengan menggunakan matriks ketetanggaan (*adjacency matrix*). Matriks ketetanggaan merepresentasikan hubungan tiap simpul dan bobot yang ada dalam suatu graf. Keuntungan representasi ini yaitu elemen matriksnya dapat diakses langsung melalui indeks. Selain itu, kita juga dapat menentukan dengan langsung apakah suatu simpul bertetangga dengan simpul yang lain.

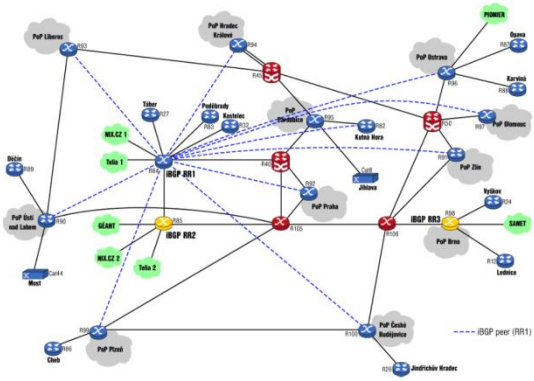


	a	b	c	d	e
a	∞	12	∞	∞	10
b	12	∞	9	11	8
c	∞	9	∞	14	∞
d	∞	11	14	∞	15
e	10	8	∞	15	∞

Gambar 3. Contoh Representasi Matriks Ketetanggan

B. Jaringan dan Routing Protocol

Seperti yang telah disebutkan dalam pendahuluan, jaringan merupakan salah satu contoh aplikasi graf berarah. Dalam jaringan, simpul dinyatakan dalam bentuk *router*, busur dinyatakan dalam bentuk keterhubungan antar *router*, dan bobot dinyatakan dalam bentuk jarak antar *router* yang terhubung.



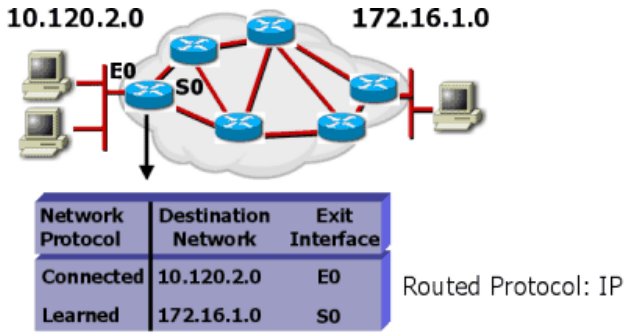
Gambar 4. Contoh Jaringan

Routing adalah sebuah proses untuk meneruskan paket-paket jaringan dari satu jaringan ke jaringan lainnya sehingga menjadi rute tertentu. Untuk melakukan *routing* dalam suatu jaringan, kita membutuhkan suatu alat yang disebut *router* yang berfungsi untuk meneruskan paket-paket dari sebuah jaringan ke jaringan yang lainnya sehingga *host-host* yang ada pada suatu jaringan bisa berkomunikasi dengan *host-host* yang ada pada jaringan yang lain.

Sebuah *router* mempelajari informasi *routing* dari mana sumber dan tujuannya yang kemudian ditempatkan pada tabel *routing*. *Router* akan berpatokan pada tabel ini untuk memberitahu *port* yang akan digunakan untuk meneruskan paket ke alamat tujuan. Jika jaringan tujuan terhubung langsung dengan *router*, maka *router* dapat langsung mengetahui *port* yang harus digunakan untuk meneruskan paket. Jika jaringan tujuan tidak terhubung langsung di badan *router*, maka *router* harus mempelajari rute terbaik yang akan digunakan untuk meneruskan paket tersebut. Informasi ini dapat dipelajari dengan cara manual oleh *network administrator* atau pengumpulan informasi melalui proses dinamik dalam jaringan.

Setiap organisasi yang diberikan sebuah *address network* dari ISP dianggap sebagai suatu *autonomous system* (AS). Setelah itu, organisasi tersebut bisa secara bebas membentuk satu jaringan yang besar atau membaginya ke dalam bagian-bagian tertentu. *Router-router* di dalam suatu *Autonomous System* digunakan untuk membagi suatu jaringan berdasarkan *subnet*. *Router-router* tersebut juga bisa digunakan untuk menghubungkan beberapa AS secara bersama. *Router* menggunakan *routing protocol* untuk secara dinamis menemukan jalur atau rute, membangun *routing table*, dan membuat keputusan tentang bagaimana harus mengirim paket melalui jaringan internet.

Routing protocol merupakan protokol atau suatu aturan untuk berkomunikasi di antara *router-router*. *Routing protocol* mengijinkan *router-router* untuk berbagi informasi tentang jaringan dan koneksi antar *router*. *Router* menggunakan informasi ini untuk membangun dan memperbaiki tabel *routing*-nya.



Gambar 5. Contoh Tabel *Routing*

Secara umum *routing protocol* mempunyai beberapa tujuan seperti berikut ini:

- Secara dinamis mempelajari dan mengisi tabel *routing* dengan sebuah lintasan bagi semua *subnet* yang ada dalam jaringan.
- Jika ada lebih dari satu lintasan untuk sebuah *subnet*, maka *routing protocol* menempatkan lintasan terbaik ke dalam tabel routing.
- Memberitahukan jika lintasan dalam tabel *routing* tidak lagi valid dan menghapus lintasan tersebut dari tabel *routing*.
- Jika suatu lintasan di dalam tabel *routing* dihapus dan lintasan lain yang dipelajari dari router sekitarnya tersedia, maka akan ditambahkan ke tabel routing.
- Untuk menambahkan lintasan baru atau memperbarui lintasan secepat mungkin. Waktu antara hilangnya lintasan dan usaha mendapatkan lintasan baru penggantinya disebut waktu konvergen.
- Yang terakhir adalah mencegah terjadinya *routing loops*.

Pada saat semua *router* dalam jaringan pengetahuannya sudah sama semua, berarti dapat dikatakan jaringan berada dalam keadaan konvergen. Keadaan konvergen yang cepat sangat diharapkan karena dapat menekan waktu pada saat *router* meneruskan untuk mengambil keputusan *routing* yang tidak benar.

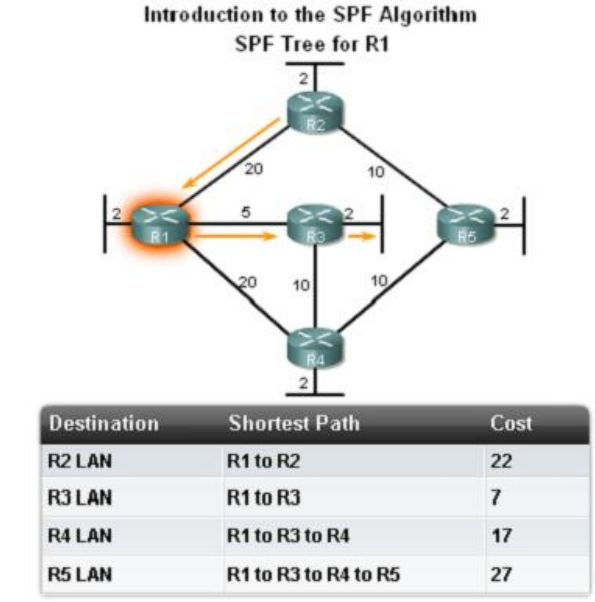
Routing protocol bisa diklasifikasikan berdasarkan apakah mereka melewatkan *traffic* di dalam atau antara *Autonomous System* menjadi dua jenis, yaitu:

- *Interior Gateway Protocol* (IGP)
  IGP merupakan protokol yang melewatkan *traffic* di dalam *Autonomous System*. IGP terdapat dua jenis, yaitu *link state routing protocol* dan *distance vector routing protocol*.
- *Exterior Gateway Protocol* (EGP)
  EGP merupakan protokol yang melewatkan *traffic* keluar atau antar *Autonomous System*. Contoh dari protocol ini yaitu *Border Gateway Protocol* (BGP).

C. Link State Routing Protocol

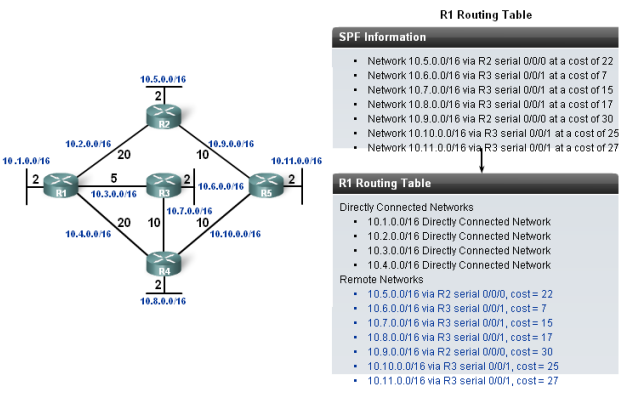
Link state routing protocol merupakan salah satu jenis *routing protocol* yang digunakan dalam pemilihan paket jaringan untuk komunikasi antar jaringan komputer. Fitur-fitur yang dimiliki oleh *routing link-state* yaitu:

- *Link-state advertisement* (LSA): adalah paket kecil dari informasi routing yang dikirim antar router.
- *Topological database*: adalah kumpulan informasi dari LSA-LSA.
- Algoritma SPF (*shortest path first*): adalah hasil perhitungan pada *database* sebagai hasil dari pohon SPF.
- Tabel *routing*: adalah daftar rute dan interface.



Gambar 6. Gambaran Link State Routing Protocol

Agar mencapai keadaan konvergen, setiap *router* mencari jaringan yang langsung terhubung dengan *router* tersebut. Kemudian l*ink state router* menukar paket *hello* untuk bertemu dengan *link state router* lain yang langsung terhubung. *Link state routing protocol* menggunakan *hello protocol* yang bertujuan untuk menemukan tetangga yang terhubung. Caranya yaitu *interface* terhubung yang menggunakan *link state routing protocol* yang sama akan bertukar paket *hello*, kemudian saat router tahu kalau mempunyai tetangga maka dia akan membentuk suatu hubungan dan dua tetangga yang terhubung akan saling bertukar paket *hello* sebagai penanda kalau sistem berfungsi dengan baik. Setiap *router* membuat *link state packet* (LSP)-nya masing-masing yang terdiri dari informasi mengenai tetangga, seperti ID tetangga, tipe hubungan, dan *bandwidth*. Dengan begitu, setiap *router* dapat mengetahui setiap jalur yang ada di dalam jaringan.



Gambar 7. Contoh Hasil Tabel *Routing*

D. Algoritma Greedy

Algoritma *greedy* merupakan metode yang paling populer untuk memecahkan persoalan optimasi atau pencarian solusi yang optimum karena algoritma ini sederhana dan lempang. Dari bahasanya, *greedy* berarti rakus atau tamak. Orang yang tamak biasanya akan mengambil sebanyak mungkin barang yang tersedia tanpa memikirkan kosekuensi ke depannya. Prinsip algoritma ini yaitu "*take what you can get now!*" yang artinya "ambil apa yang dapat anda peroleh sekarang!". Prinsip inilah yang digunakan dalam pemecahan masalah optimasi.

Algoritma *greedy* membentuk solusi langkah per langkah (*step by step*). Terdapat banyak langkah yang perlu dieksplorasi pada setiap langkah solusi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan. Keputusan yang diambil pada suatu langkah tidak dapat diubah lagi pada langkah selanjutnya. Sebagai contoh, jika kita menggunakan algoritma *greedy* untuk menempatkan komponen di atas papan sirkuit (*circuit board*), sekali sebuah komponen telah ditetapkan posisinya, komponen tersebut tidak dapat dipindahkan lagi.

Pendekatan yang digunakan dalam algoritma *greedy* adalah membuat pilihan yang "tampaknya" memberikan perolehan terbaik, yaitu dengan membuat pilihan optimum lokal pada setiap langkah dengan harapan bahwa sisanya mengarah ke solusi optimum global.

Persoalan optimasi dalam algoritma *greedy* disusun oleh elemen-elemen sebagai berikut:

1. Himpunan kandidat, yang berisi elemen-elemen pembentuk solusi.
2. Himpunan solusi, yang berisi kandidat-kandidat yang terpilih sebagai solusi persoalan.
3. Fungsi seleksi, yaitu fungsi yang pada setiap langkah memilih kandidat yang paling memungkinkan mencapai solusi optimal.
4. Fungsi kelayakan (*feasible*), yang memeriksa apakah suatu kandidat yang telah dipilih dapat memberikan solusi yang layak, yaitu kandidat tersebut bersama-sama dengan himpunan solusi yang sudah terbentuk tidak melanggar kendala (*constraints*) yang ada.
5. Fugsi obyektif, yaitu fungsi yang memaksimumkan atau meminimumkan nilai solusi.

Secara umum, *pseudo-code* dari algoritma *greedy* adalah sebagai berikut:

```
function SELEKSI (C : himpunan_kandidat) → kandidat
{mengembalikan sebuah kandidat yang dipilih dari C berdasarkan kriteria tertentu}

function SOLUSI (S : himpunan_kandidat) → boolean
{bernilai true jika S adalah solusi dari persoalan; sebaliknya bernilai false jika S belum menjadi solusi}

function LAYAK (S : himpunan_kandidat) → boolean
{bernilai true jika S merupakan solusi yang tidak melanggar kendala; sebaliknya bernilai false jika S melanggar kendala}

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
```

Dari *pseudo-code* tersebut dapat kita ketahui bahwa algoritma *greedy* tidak selalu menghasilkan solusi yang benar-benar optimum. Seringkali algoritma ini menjadi basis untuk pendekatan heuristik. Selain itu, jika persoalan dapat dipecahkan secara eksak dengan algoritma *greedy*, maka pembukian kebenaran dari algoritma *greedy* merupakan proses yang *non-trivial*.

E. Algoritma Dijkstra

Algoritma *dijkstra* merupakan algoritma yang dipakai untuk memecahkan permasalahan jarak terpendek (*shortest path problem*) untuk sebuah graf berarah dengan bobot-bobot sisi (*edge weights*) yang bernilai tak-negatif. Walaupun algoritma ini ditujukan pada graf berarah, algoritma *dijkstra* tetap benar untuk graf tak berarah.

Algoritma ini mencari lintasan terpendek dalam sejumlah langkah. Algoritma ini menggunakan strategi algoritma *greedy*. Pada setiap langkah, ambil sisi yang berbobot minimum yang menghubungkan sebuah simpul yang sudah terpilih dengan sebuah simpul lain yang belum terpilih. Lintasan dari simpul asal ke simpul yang baru merupakan lintasan yang terpendek di antara semua lintasan ke simpul-simpul yang belum dipilih.

Secara garis besar, *pseudo-code* algoritma *dijkstra* yaitu:

```
function Dijkstra (input M : matriks, a : simpul awal) → tabel
{ Mencari lintasan terpendek dari impul awal a ke semua simpul alinnya
   Masukan: matriks ketetanggaan (M) dari graf berbobot G dan simpul awal a
   Keluaran: tabel D yang berisi panjang lintasan terpendek dari a ke semua simpul
```

```
   lainnya
}
Deklarasi
   D, S : tabel
   i : integer
Algoritma:
   {Langkah 0 inisialisasi}
   for i ← 1 to n do
      S[i] ← 0
      D[i] ← m[a,i]
   endfor

   {Langkah 1}
   S[a] ← 1    {karena simpul a adalah simpul
                asal lintasan terpendek, jadi
                simpul a sudah pasti terpilih
                dalam lintasan terpendek}
   D[a] ← ∞    {tidak ada lintasan terpendek
                dari simpul a ke a}

   {Langkah 2, 3, ..., n-1}
   for i ← 2 to n-1 do
      Cari j sedemikian hingga S[j] = 0 dan D[j]
         = Minimum{D[1], D[2], ..., D[n]}
      S[j] ← 1 {Simpul j sudah terpilih ke dalam
                lintasan terpendek}
      Hitung D[i] yang baru dari a ke simpul i ∉
         S dengan cara sebagai berikut:
         D[i] ← Minimum{D[i], (D[j] + M[j,i])}
   endfor
   return D
```

Misalkan sebuah graf berbobot dengan n buah simpul dinyatakan degan matriks ketetanggan M = [m_{ij}], yang dalam hal ini,

m_{ij} = bobot sisi (i, j)
m_{ii} = 0
m_{ij} = ∞, jika tidak ada sisi dari simpul i ke simpul j

Selain matriks M, kita juga menggunakan tabel S = [s_i] yang dalam hal ini,

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

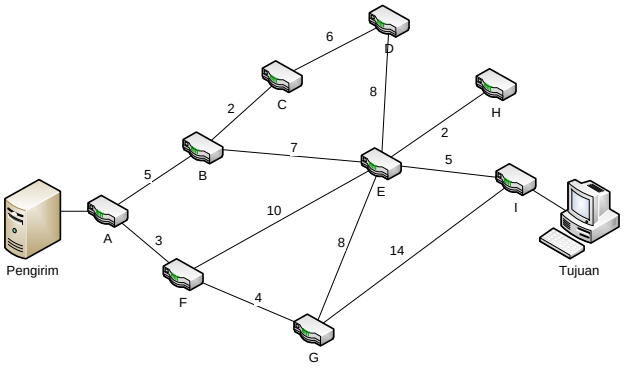
dan keluarannya adalah sebuah tabel D = [d_i], yang dalam hal ini,
d_i = panjang lintasan dari simpul awal ke simpul i

Pada akhir algoritma, d_i akann berisi panjang lintasan terpendek dari simpul asal ke simpul i.

III. METODE PENERAPAN

Dari proses pada dasar teori, kita dapat mengetahui bahwa *link state routing protocol* menggunakan masalah lintasan terpendek (*shortest path*) untuk menentukan jalur terbaik dari pengirim sampai tujuan. Masalah tersebut merupakan salah satu persoalan optimasi dan graf yang digunakan dalam pencarian lintasan terpendek adalah graf berbobot. Untuk memecahkan masalah tersebut dapat digunakan algoritma *dijkstra* yang merupakan penerapan dari algoritma *greedy*.

Misalnya, kita mempunyai graf seperti di bawah ini.



Gambar 8. Graf Masalah

Pada graf tersebut, proses *routing* dilakukan dari simpul A menuju simpul I. Kemudian graf tersebut kita representasi dengan menggunakan matriks ketetanggaan sehingga menjadi:

Tabel 1. Matriks Ketetanggaan dari Graf Masalah

Simpul	A	B	C	D	E	F	G	H	I
A	∞	5	∞	∞	∞	3	∞	∞	∞
B	5	∞	2	∞	7	∞	∞	∞	∞
C	∞	2	∞	6	∞	∞	∞	∞	∞
D	∞	∞	6	∞	8	∞	∞	∞	∞
E	∞	7	∞	8	∞	10	8	2	5
F	3	∞	∞	∞	10	∞	4	∞	∞
G	∞	∞	∞	∞	8	4	∞	∞	14
H	∞	∞	∞	∞	2	∞	∞	∞	∞
I	∞	∞	∞	∞	5	∞	14	∞	∞

Dengan menggunakan *pseudo-code* algoritma *dijkstra* pada bab dasar teori, kita dapat memecahkan permasalahan ini dengan langkah-langkah sebagai berikut:

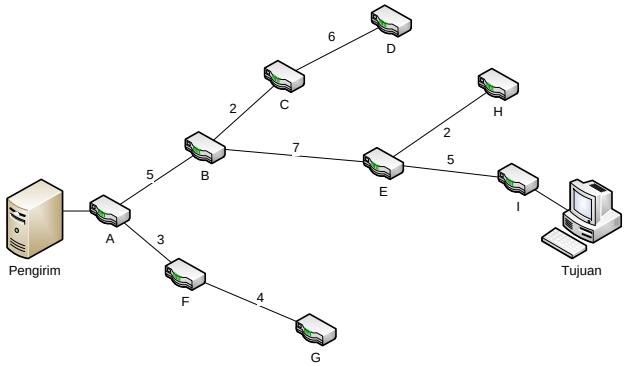
Tabel 2. Langkah Pencarian Jalur Terpendek

Langkah	Simpul yang dipilih	Lintasan	S								D								
			A	B	C	D	E	F	G	H	I	A	B	C	D	E	F	G	H	I
0	-	-	0	0	0	0	0	0	0	0	0	0	5 (A, B)	∞	∞	∞	3 (A, F)	∞	∞	∞
1	A	A	1	0	0	0	0	0	0	0	∞	5 (A, B)	∞	∞	∞	∞	3 (A, F)	∞	∞	∞
2	F	AF	1	0	0	0	0	1	0	0	∞	5 (A, B)	∞	∞	13 (A,F ,E)	3 (A, F)	7 (A,F ,G)	∞	∞	
3	G	AFG	1	0	0	0	0	1	1	0	∞	5 (A, B)	∞	∞	13 (A,F ,E)	3 (A, F)	7 (A,F ,G)	∞	21 (A,F, G,I)	
4	E	AFE	1	0	0	0	1	1	1	0	∞	5 (A, B)	∞	21 (A,F, E,D)	13 (A,F ,E)	3 (A, F)	7 (A,F ,G)	15 (A,F, E,H)	18 (A,F, E,I)	
5	B	AB	1	1	0	0	1	1	1	0	∞	5 (A, B)	7 (A,B ,C)	21 (A,F, E,D)	12 (A, B,E)	3 (A, F)	7 (A,F ,G)	15 (A,F, E,H)	18 (A,F, E,I)	
6	C	ABC	1	1	1	0	1	1	1	0	∞	5 (A, B)	7 (A,B ,C)	13 (A,B, C,D)	12 (A, B,E)	3 (A, F)	7 (A,F ,G)	15 (A,F, E,H)	18 (A,F, E,I)	
7	D	ABC D	1	1	1	1	1	1	1	0	∞	5 (A, B)	7 (A,B ,C)	13 (A,B, C,D)	12 (A, B,E)	3 (A, F)	7 (A,F ,G)	15 (A,F, E,H)	18 (A,F, E,I)	
8	E	ABE	1	1	1	1	1	1	1	0	∞	5 (A, B)	7 (A,B ,C)	13 (A,B, C,D)	12 (A, B,E)	3 (A, F)	7 (A,F ,G)	14 (A,B, E,H)	17 (A,B, E,I)	
8	H	ABE H	1	1	1	1	1	1	1	1	0	∞	5 (A, B)	7 (A,B ,C)	13 (A,B, C,D)	12 (A, B,E)	3 (A, F)	7 (A,F ,G)	14 (A,B, E,H)	17 (A,B, E,I)
9	I	ABE I	1	1	1	1	1	1	1	1	0	∞	5 (A, B)	7 (A,B ,C)	13 (A,B, C,D)	12 (A, B,E)	3 (A, F)	7 (A,F ,G)	14 (A,B, E,H)	17 (A,B, E,I)

Untuk contoh program penerapannya dapat diunduh pada pranala http://students.if.itb.ac.id/~if18020/Kuliah/Stima/.

IV. Hasil Penerapan dan Pembahasan

Dari tabel 8 yang merupakan hasil tabel *routing*, kita bisa menentukan jalur terpendek dari simpul A ke simpul yang lain. Untuk menuju simpul B dapat menggunakan jalur A → B, menuju simpul C menggunakan jalur A → B → C, menuju simpul D menggunakan jalur A → B → C → D, menuju simpul E menggunakan jalur A → B → E, menuju simpul F menggunakan jalur A → F, menuju simpul G menggunakan jalur A → F → G, menuju simpul H menggunakan jalur A → B → E → H, dan menuju simpul I menggunakan jalur A → B → E → I. Dapat digambarkan pada gambar di bawah ini:



Gambar 9. Lintasan Terpendek Graf pada Gambar 8

Dengan begitu, didapatkan jalur *routing* terpendek dari pengirim sampai tujuan melalui *router* A, B, E, dan I.

V. Simpulan

Di dalam suatu jaringan, kita menentukan *routing protocol* untuk menentukan jalur yang terbaik sesuai dengan aturan yang digunakan dalam *routing protocol* tersebut. Untuk membuat *routing protocol* dapat menggunakan algoritma yang telah dipelajari dalam mata kuliah Strategi Algoritma. Kita harus dapat memilih algoritma yang sesuai dengan *routing protocol* tersebut dengan memahami konsep dan aturan dalam algoritma dan *routing protocol* yang diterapkan. Di dalam proses *routing* yang merupakan implementasi dari graf berarah, masalah yang biasanya muncul adalah bagaimana menentukan jalur atau rute yang terpendek agar didapatkan proses *routing* yang efisien dan efektif. Contoh algoritma yang digunakan dalam proses *link state routing protocol* yaitu algoritma *dijkstra* yang merupakan penerapan dari algoritma *greedy*.

Kebanyakan orang mungin tidak menyadari bahwa saat sedang terhubung dengan jaringan internet, terdapat proses tersebut. Akan tetapi, proses tersebut memang ada di dalam kehidupan kita dan penggunaan algoritma yang menjadi dasar konsepnya. Oleh karena itu, sangatlah penting untuk mempelajari berbagai macam algoritma dan strategi pemilihannya untuk dapat diterapkan dalam kehidupan sehari-hari.

Daftar Pustaka

[1] Rinaldi Munir, "Diktat Kuliah IF2091, Struktur Diskrit", Program Studi Teknik Informatika, STEI, ITB, 2008, halaman VIII-1 sampai VIII-48.

[2] Rinaldi Munir, "Diktat Kuliah IF3051, Strategi Algoritma", Program Studi Teknik Informatika, STEI, ITB, 2008, halaman 26-31 dan 56-63.

[3] Wikipedia, http://en.wikipedia.org/wiki/Dijkstra's_algorithm (7 Desember 2010, 15.00).

[4] Wikipedia, http://en.wikipedia.org/wiki/Greedy_algorithm (7 Desember 2010, 15.15)

[5] http://www-b2.is.tokushima-u.ac.jp/~ikeda/suuri/dijkstra/Dijkstra.shtml (8 Desember 2010, 19.00)

[6] Wikipedia, http://en.wikipedia.org/wiki/Link-state_routing_protocol (8 Desember 2010, 19.30)

[7] Wikipedia, http://id.wikipedia.org/wiki/Teori_graf (8 Desember 2010, 19.45)

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, 29 April 2010

Muhamma Ghufron Mahfudhi
13508020