Algoritma Brute-Force dan Greedy dalam Pemrosesan Graf

Marvin Jerremy Budiman / 13515076[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13515076@std.stei.itb.ac.id

***Abstrak*—Graf merupakan struktur data yang unik. Graf diaplikasikan dalam berbagai hal, seperti perancangan rangkaian listrik, *social network analysis*, perancangan jaringan komputer, dan sebagainya. Karena struktur data graf berbeda dengan struktur data lain, maka dalam memproses graf diperlukan algoritma yang berbeda dibanding dengan algoritma untuk memproses struktur data lain, seperti tabel atau matriks. Beberapa algoritma untuk mengelola graf menggunakan pendekatan *brute-force* dan *greedy*.**

***Kata Kunci*—Algoritma, *Brute-Force*, Graf, *Greedy*.**

I. Pendahuluan

Algoritma adalah sekumpulan intruksi terurut yang digunakan untuk menyelesaikan masalah. Selain algoritma, untuk menyelesaikan masalah tersebut diperlukan struktur data, yang digunakan untuk menyimpan data yang berkaitan dengan masalah tersebut. Misal, dalam mencari bilangan bulat terbesar dari N buah bilangan bulat, setiap bilangan bulat disimpan dalam larik (*array*). Kemudian akan dipilih suatu bilangan pertama X dalam larik, yang akan dibandingkan dengan N-1 bilangan yang lain dalam larik tersebut. Perbandingan tersebut dilakukan secara sekuensial, mulai dari indeks larik kedua, hingga indeks ke-N. Larik tersebut menjadi struktur data yang dipakai untuk menyelesaikan masalah tersebut.

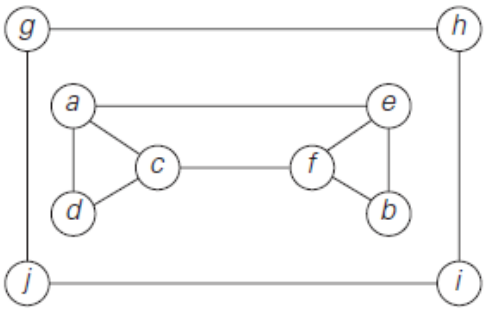
Struktur data graf, merupakan struktur data yang berbeda dengan larik. Graf terdiri dari himpunan simpul (*vertex*) dan himpunan sisi (*edge*) yang menghubungkan simpul-simpul tersebut. Suatu sisi yang menghubungkan 2 buah simpul biasanya melambangkan suatu hal/hubungan. Graf digunakan untuk merepresentasikan banyak hal, seperti hubungan antarindividu dalam suatu komunitas, hubungan antarorganisme dalam rantai makanan, hubungan antarkomponen elektronik dalam suatu rangkaian listrik, hubungan antarkota dalam suatu negara, dan sebagainya.

Beberapa masalah umum yang dapat direpresentasikan dengan struktur data graf, diantaranya adalah mencari lintasan antara 2 simpul, mencari lintasan terpendek antara 2 simpul, pewarnaan graf, dan lain-lain. Masalah-masalah tersebut memerlukan algoritma yang berbeda dengan algoritma untuk mengelola larik, ataupun struktur data lain.

II. Teori Dasar Graf

A. Pengertian Graf

Graf *G*, terdiri atas himpunan simpul *V* dan himpunan sisi *E*, umumnya ditulis *G = (V, E)*. Graf *H* pada Gambar 1, terdiri dari himpunan simpul *V = {a, b, c, d, e, f, g, h, i, j}* dan himpunan sisi *E = {(a,c), (a,d), (a,e), (c,f), (c,d), (e,f), (e,b), (b,f), (g,h), (g,j), (h,i), (i,j)}*.



Gambar 1. Graf H merupakan graf sederhana dan tak berarah. (Sumber: Levitin, 2011)

B. Penggolongan Graf

- *Graf sederhana dan graf tak sederhana*

Graf disebut graf sederhana (*simple*) jika graf tersebut tidak memiliki sisi ganda (misal, simpul *a* dan *b* dihubungkan oleh 2 sisi berbeda) dan tidak memiliki *loop* (misal, simpul *a* memiliki sisi yang menghubungkannya dengan diri sendiri). Graf yang di dalamnya terdapat sisi ganda atau *loop*, disebut graf tak sederhana (*multigraph*).

- *Graf berarah dan graf tak berarah*

Graf yang setiap sisinya tidak memiliki arah tertentu disebut graf tak berarah (*undirected*). Graf yang setiap sisinya memiliki arah tertentu (dilambangkan dengan anak panah), disebut graf berarah (*directed*). Jika terdapat sisi berarah *(a, b)* dalam suatu graf berarah, maka sisi tersebut berawal dari simpul *a* dan berakhir di simpul *b*.

C. Terminologi Graf

- *Bertetangga dan Bersisian*

Dua buah simpul *a* dan *b* dikatakan bertetangga (*adjacent*) jika terdapat sisi *e* yang mengubungkan *a* dan *b*. Sisi *e* dikatakan bersisian (*incident*) dengan simpul *a* dan

simpul *b*. Pada Gambar 1 simpul *c* dan simpul *f* dikatakan bertetangga, dan sisi *(c,f)* dikatakan bersisian dengan simpul c dan simpul f.

- *Derajat*

Derajat adalah jumlah sisi yang bersisian dengan suatu simpul. Derajat simpul *a* dinyatakan dengan *deg(a)*, pada Gambar 1, *deg(a) = 3*. *Lemma* jabat tangan (*handshaking lemma*) menyatakan bahwa jumlah derajat seluruh simpul dalam suatu graf *G*, sama dengan jumlah dua kali jumlah sisi dalam graf tersebut.

$$2e = \sum_{v \epsilon G} \deg(v)$$

- *Lintasan dan Sirkuit*

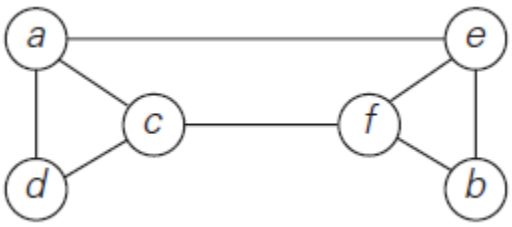
Lintasan (*path*) dengan panjang *n* dari simpul *a* menuju *b* adalah kumpulan *n* sisi berurutan yang menguhubungkan simpul *a* dan simpul *b*. Lintasan disebut sirkuit (*circuit*), jika berawal dan berakhir pada simpul yang sama, dan panjangnya lebih besar dari 0. Lintasan atau sirkuit dikatakan sederhana, jika tidak terdapat 2 atau lebih sisi yang sama pada lintasan tersebut. Pada Gambar 1, lintasan dari *a* ke *b* terdiri dari sisi *(a,c)*, *(c,f)* dan *(b,f)* dengan panjang 3. Salah satu sirkuit yang melalui simpul *a* adalah *(a,c)*, *(c,d)*, *(a,d)* dengan panjang 3. Tidak terdapat lintasan dari simpul *a* menuju simpul *g*.

- *Keterhubungan dalam Graf*

Graf tak berarah *G* dikatakan terhubung (*connected*), jika terdapat lintasan antara *a* dan *b*, dengan *a* dan *b* merupakan pasangan setiap simpul pada graf *G*. Jika terdapat 2 simpul yang terhubung dengan suatu lintasan, maka graf disebut graf tak terhubung (*unconnected*). Graf H pada Gambar 1 merupakan graf tak terhubung.

- *Upagraf*

Graf $H = (V_1, E_1)$ merupakan upagraf (*subgraph*) dari graf $G = (V, E)$, jika V_1 merupakan himpunan bagian dari *V*, dan E_1 merupakan himpunan bagian dari *E*. Graf *J* pada Gambar 2 merupakan upagraf dari *H* pada Gambar 1.



Gambar 2. Graf J. (Sumber: Levitin, 2011)

- *Graf Berbobot*

Graf berbobot (*weighted graph*) merupakan graf, yang setiap sisinya diberi bobot (nilai) tertentu.

D. *Representasi Graf*

- *Matriks Ketetanggaan*

Setiap baris dan kolom pada matriks ketetanggan (*adjacency matrix*) melambangkan simpul-simpul pada graf. Jika simpul v_i bertetangga dengan simpul v_j pada graf, maka isi elemen X_{ij} pada matriks diisi dengan 1, jika tidak bertetangga, diisi dengan 0.

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

Gambar 3. Matriks ketetanggaan dari graf J pada Gambar 2. Urutan baris dari atas ke bawah dan urutan kolom dari kiri ke kanan adalah a, b, c d, e, f

- *Matriks Bersisian*

Baris pada matriks bersisian (*incidency matrix*) melambangkan simpul-simpul pada graf. Kolom melambangkan sisi-sisi yang ada pada graf. Jika simpul v_i bersisian dengan sisi e_j pada graf, maka isi elemen X_{ij} pada matriks diisi dengan 1, jika tidak bersisian, diisi dengan 0.

- *Senarai Ketetanggaan*

Senarai ketetanggan (*adjacency list*) berisi daftar semua simpul beserta simpul-simpul yang bertetangga dengan simpul tersebut.

Simpul	Simpul Tetangga
a	*c, d, e*
b	*e, f*
c	*a, d, f*
d	*a, c*
e	*a, b, f*
f	*b, c, e*

Tabel 1. Senarai ketetanggaan dari graf J pada Gambar 2.

III. ALGORITMA PEMROSESAN GRAF

A. *Algoritma Brute-Force*

Algoritma yang menggunakan pendekatan *brute-force*, merupakan algoritma yang sederhana, namun langsung menyelesaikan masalah, tidak membutuhkan banyak pemikiran dari *programmer*. Algoritma *brute-force* relatif lebih mudah diaplikasikan.

Contoh algoritma *brute-force* yang digunakan untuk memproses graf, antara lain menggunakan pendekatan *depth-first search* dan *breadth-first search*.

- *Depth-First Search*

Pemrosesan dengan pendekatan *depth-first search* diawali dengan mengunjungi suatu simpul *a* dari graf *G*, tandai setiap simpul yang sudah dikunjungi. Dalam setiap kali pengulangan, algoritma akan memproses simpul yang bertetangga dengan simpul yang sedang dikunjungi pada saat itu. Proses mengunjungi akan terus berlangsung hingga tiba di simpul yang semua tetangganya sudah pernah dikunjungi. Ketika tiba di simpul tersebut, misal simpul *b*, pemrosesan akan kembali ke simpul yang dikunjungi sebelum *b*, dan mencari lagi tetangga *b* yang belum dikunjungi.

Pemrosesan ini dapat dibantu dengan struktur data lain, yaitu *stack*. Setiap kali mengunjungi suatu simpul, maka simpul tersebut di-*push* ke dalam *stack*. Ketika simpul tersebut tidak memiliki tetangga yang belum dikunjungi, maka simpul tersebut di-*pop* dari *stack*.

ALGORITMA

```
a ← 0 // a variabel global
Depth_First_Search (G)
{ I.S. G = (V, E) terdefinisi
  F.S. Semua simpul pada G, ditandai dengan n, n
  adalah urutan dikunjunginya simpul tersebut
  secara DFS}

  tandai setiap simpul pada V dengan 0
  for v ∈ V do
    if v ditandai dengan 0
      DFS(v)

DFS(v)
{ I.S. v terdefinisi
  F.S. v ditandai dengan suatu nilai }

  a ← a + 1; tandai v dengan a
  for w ∈ V and w bertetangga dengan v do
    if w ditandai dengan 0
      DFS(w)
```

Algoritma 1. Pseudocode algoritma dengan pendekatan DFS

- *Breadth-First Search*

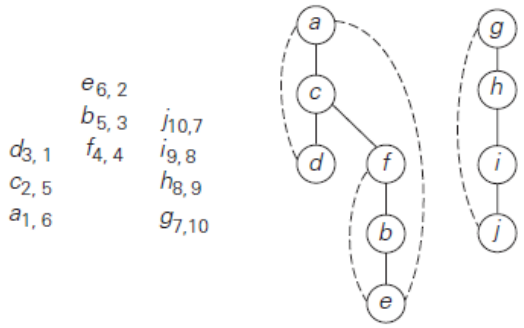
Pemrosesan dengan pendekatan *breadth-first search* dibantu dengan menggunakan struktur data *queue*. Pemrosesan diawali dengan mengunjungi suatu simpul *a* dari graf *G*, simpul tersebut ditandai dan ditambahkan ke *queue* (a menjadi simpul terdepan dalam *queue*). Dalam setiap kali pengulangan, semua simpul yang bertetangga dengan simpul terdepan ditandai dan dimasukkan ke dalam *queue*, lalu simpul terdepan dihapus dari *queue*. Proses mengunjungi akan terus berlangsung hingga *queue* kosong.

ALGORITMA

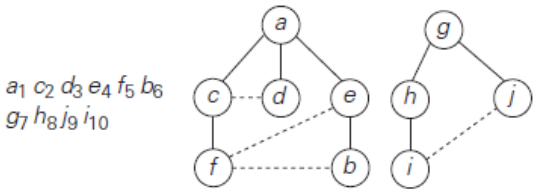
```
a ← 0 // a variabel global
Breadth_First_Search (G)
{ I.S. G = (V, E) terdefinisi
  F.S. Semua simpul pada G, ditandai dengan n, n
  adalah urutan dikunjunginya simpul tersebut
  secara BFS }

  tandai setiap simpul pada V dengan 0
  for v ∈ V do
    if v ditandai dengan 0
      BFS(v)

BFS(v)
{ I.S. v terdefinisi
  F.S. v ditandai dengan suatu nilai }

  a ← a + 1; tandai v dengan a
  tambahkan v ke queue
  while queue tidak kosong do
    for w ∈ V and w bertetangga dengan simpul
    terdepan do
      if w ditandai dengan 0
        a ← a + 1; tandai w dengan a
        tambahkan w ke queue
    hapus simpul terdepan dari queue
```

Algoritma 2. Pseudocode algoritma dengan pendekatan BFS

Simpul simpul yang telah diproses oleh algoritma ini dapat disusun menjadi DFS *forest* dan BFS *forest*. Setiap DFS *forest* maupun BFS *forest* tersusun dari 1 atau lebih pohon (*tree*). Simpul awal yang dikunjungi akan menjadi akar (*root*) dari pohon. Setiap simpul yang ditandai akan ditambahkan dalam pohon tersebut sebagai anak dari simpul sebelumnya yang dikunjungi. Jika pada suatu pohon hasil DFS atau BFS , semua simpulnya sudah tidak memiliki tetangga yang belum dikunjungi, namun pada graf tersebut masih ada simpul yang belum dikunjungi, maka akan dibuat pohon baru, dengan akarnya adalah salah satu simpul yang belum dikunjungi.



Gambar 4. Isi stack (gambar kiri) dan DFS forest (gambar kanan) untuk DFS pada graf H pada gambar 1. Indeks pertama pada isi stack melambanakan urutan di-push-nya simpul tersebut, indeks kedua melambangkan urutan di-pop-nya simpul tersebut. (Sumber: Levitin, 2011)



Gambar 5. Isi queue (gambar kiri) dan BFS forest (gambar kanan) untuk BFS pada graf H pada gambar 1. Indeks pertama pada isi queue melambanakan urutan ditambahkannya simpul tersebut pada queue. (Sumber: Levitin, 2011)

Kedua pendekatan ini dapat digunakan untuk menyelidiki keterhubungan suatu graf. Simpul-simpul yang terhubung dalam graf akan berada dalam pohon yang sama. Jika terdapat lebih dari 1 pohon yang terbentuk, maka graf tidak terhubung. Jumlah pohon yang terbentuk merupakan jumlah komponen terhubung dari graf (*connected component*), untuk graf *H* pada Gambar 1, jumlah komponen terhubungnya adalah 2. Pendekatan DFS dapat digunakan untuk mencari simpul yang jika dihapus akan memecah graf menjadi bagian-bagian yang *disjoint*. Pendekatan BFS dapat digunakan untuk mencari jumlah sisi paling sedikit antara 2 simpul.

Algoritma dengan pendekatan DFS dan BFS sama-sama memiliki kompleksitas asimptotik $\theta(|V| + |E|)$, jika graf diimplementasi dengan senarai ketetanggaan, dengan $|V|$ merupakan jumlah simpul pada graf dan $|E|$ merupakan jumlah sisi pada graf.

B. *Algoritma Greedy*

Selain masalah keterhubungan dalam suatu graf, persoalan lain yang sering dicari solusinya adalah bobot minimum dari suatu lintasan pada graf berbobot. Algoritma yang menggunakan pendekatan *greedy*,

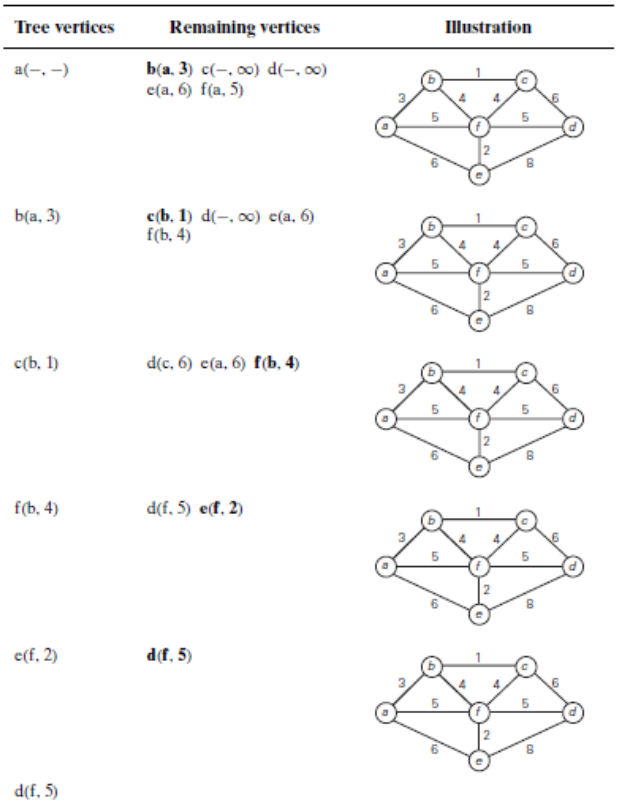
merupakan algoritma yang mencari solusi paling optimal diantara seluruh solusi yang ada. Algoritma dengan pendekatan *greedy* bisa diterapkan dalam pencarian bobot lintasan minimum dalam suatu graf.

- *Pohon Merentang, Algoritma Prim dan Algoritma Kruskal*

Pohon merentang (*spanning tree*) dari suatu graf *G*, adalah upagraf terhubung tanpa sirkuit dari *G*, yang berisi seluruh simpul pada *G*. Pohon merentang minimum dari suatu graf berbobot *W*, adalah pohon merentang dari *W* yang memiliki bobot minimum. Pohon merentang minimum dapat dicari dengan menggunakan algoritma Prim dan Kruskal.

○ *Algoritma Prim*

Algoritma Prim diawali dengan memilih suatu simpul *a* pada graf *G* dan memasukkannya pada suatu subgraf *T*. Dalam setiap pengulangan akan ditambahkan suatu simpul, misal simpul *b* dan suatu sisi *e* yang menghubungkan *b* dengan dengan suatu simpul pada *T*. Sisi *e* yang dipilih merupakan sisi dengan bobot minimum. Pengulangan akan berhenti ketika semua simpul pada *G*, telah ditambahkan pada *T*. Algoritma Prim diilustrasikan pada Gambar 6.



Gambar 6. Ilustrasi algoritma Prim. (Sumber: Levitin, 2011)

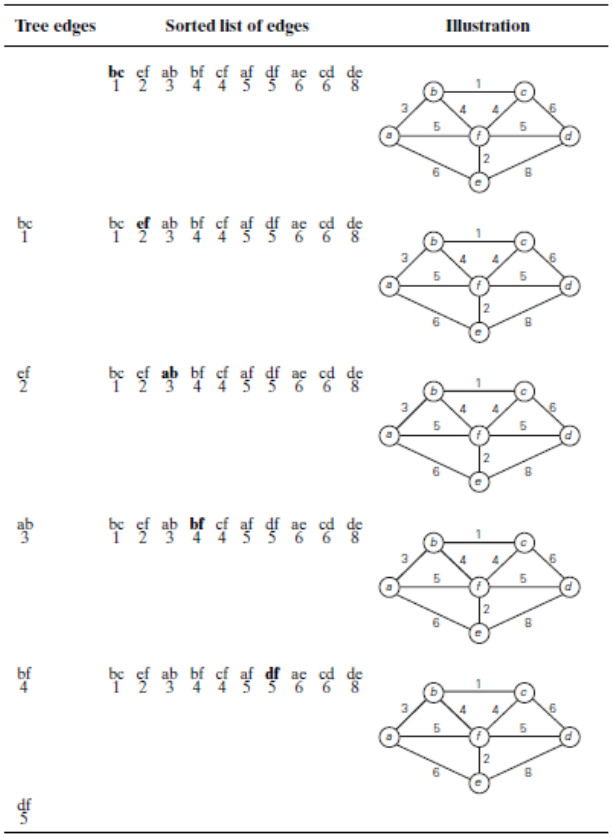
ALGORITMA

```
Prim (G)
{ I.S. G = (V, E) terdefinisi
  F.S. Terdapat graf T = (Vt, Et) yang menyusun
  pohon merentang minimum dari G }

  Vt ← {a} // a salah satu simpul dalam G
  Et ← Ø
  for i ← 1 to |V| - 1 do
    cari sisi berbobot minimum e = (j,k)
    diantara semua sisi (b,c)
    // b ∈ Vt, c ∈ V - Vt
    Vt ← Vt ∪ {k}
    Et ← Et ∪ {e}
return Et
```

Algoritma 3. Pseudocode algoritma Prim

○ *Algoritma Kruskal*

Algoritma Kruskal diawali dengan mengurutkan secara naik seluruh sisi dalam suatu graf *G* menurut bobotnya. Kemudian dilakukan penambahan simpul pada subgraf *T*, diawali dengan urutan pertama dari sisi yang telah terurut. Jika penambahan sisi tersebut membentuk sirkuit pada *T*, maka sisi tersebut tidak ditambahkan ke dalam *T*, dan sisi berikutnya akan diproses. Algoritma Kruskal diilustrasikan pada gambar 7.



Gambar 7. Ilustrasi algoritma Kruskal. (Sumber: Levitin, 2011)

ALGORITMA

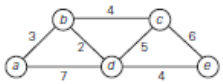
```
Kruskal (G)
{ I.S. G = (V, E) terdefinisi
  F.S. Terdapat graf T = (Vt, Et) yang menyusun
  pohon merentang minimum dari G }

  urutkan E secara menaik berdasarkan bobot
  // e1, e2, ... dst.
  Et ← Ø; counter ← 0; k ← 0
  while counter < |V| - 1 do
    k ← k + 1
    if Et ∪ {ek} tidak memiliki sirkuit
      Et ← Et ∪ {ek}; counter ← counter + 1
return Et
```

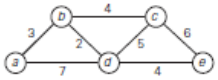
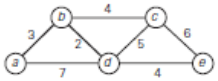
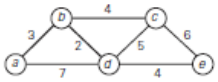
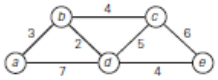
Algoritma 4. Pseudocode algoritma Kruskal

- *Persoalan Lintasan Terpendek dan Algoritma Dijkstra*

Lintasan terpendek dari simpul *a* ke simpul *b* dalam suatu graf *G* adalah sekumpulan sisi dengan bobot minimum yang menghubungkan *a* dan *b*. Berbeda dengan mencari pohon merentang minimum, sekarang tidak semua sisi dalam *G* harus berada dalam lintasan dari *a* ke *b*. Salah satu algoritma yang digunakan untuk mencari lintasan terpendek ini adalah algoritma Dijkstra. Algoritma Dijkstra mencari jarak terpendek dari suatu simpul dalam graf, ke semua simpul lain dalam graf tersebut. Algoritma Dijkstra dapat digunakan jika tidak ada simpul yang berbobot negatif pada graf.

Pada algoritma Dijkstra, setiap simpul diberi 2 label. Label pertama merupakan panjang lintasan dari simpul awal ke simpul yang diberi label. Label kedua merupakan nama simpul terakhir sebelum simpul yang diberi label. Algoritma Dijkstra menggunakan *priority queue* dengan simpul-simpul pada *G*, sebagai elemen *queue*. *Priority* pada elemen *queue*, merepresentasikan label jarak.



Tree vertices	Remaining vertices	Illustration
a(–, 0)	**b(a, 3)** c(–, ∞) d(a, 7) e(–, ∞)	
b(a, 3)	c(b, 3 + 4) **d(b, 3 + 2)** e(–, ∞)	
d(b, 5)	**c(b, 7)** e(d, 5 + 4)	
c(b, 7)	**e(d, 9)**	
e(d, 9)		

Gambar 8. Ilustrasi algoritma Dijkstra. (Sumber: Levitin, 2011)

ALGORITMA

```
Dijkstra (G, s)
{ I.S. G = (V, E) dan a simpul pada G terdefinisi
  F.S. dv merupakan jarak terpendek dari a ke
  v, pv merupakan simpul paling akhir sebelum v,
  v adalah seluruh simpul pada V }

  Initialisasi(Q) // Q dikosongkan
  for v ∈ V do
    dv ← ∞; pv ← null;
    Insert(Q, v, dv)
    // masukkan v dalam Q dengan priority ∞

  ds ← 0; Update(Q, s, ds)
  // update priority s dengan ds
  Vt ← Ø

  for i ← 0 to |V| - 1 do
    k ← DeleteMin(Q)
    // hapus elemen Q dengan priority paling kecil
    Vt ← Vt ∪ {k}
  for u ∈ V - Vt and u bertetangga dengan k do
    if dk + w(k,u) < du
      du ← dk + w(k,u); pu ← k
      // w(k,u) adalah bobot sisi (k,u)
      Update(Q, u, du)
```

Algoritma 5. Pseudocode algoritma Dijkstra

IV. SIMPULAN

Pada graf tak berarah, terdapat algoritma-algoritma unik yang digunakan untuk memproses graf tersebut, diantaranya algoritma DFS dan BFS, algoritma Prim, algoritma Kruskal, serta algoritma Dijkstra. Penanganan akan berbeda jika yang ditangani merupakan graf berarah, kecuali untuk algoritma Dijkstra yang dapat diaplikasikan juga pada graf berarah.

V. UCAPAN TERIMA KASIH

Saya mengucapkan terima kasih kepada Tuhan Yesus Kristus, karena oleh campur tangan-Nya, saya bisa menyelesaikan perkuliahan di semester ini, beserta semua tugas dan ujian yang harus saya lalui. Saya mengucapkan terima kasih kepada Ibu Harlili dan Bapak Rinaldi Munir, sebagai dosen-dosen dari mata kuliah IF2120 Matematika Diskrit, atas bimbingan dan ilmu yang sudah diberikan selama 1 semester ini.

REFERENSI

[1] Levitin, Anany. 2012. *Introduction to the Design and Analysis of Algorithm 3rd Edition*. Addison-Wesley.
[2] Rosen, Kenneth. 2012. *Discrete Mathematics and Its Application 7th Edition*. McGraw-Hill.
[3] https://www.tutorialspoint.com/data_structures_algorithms/, diakses tanggal 8 Desember 2016, pukul 19.17.

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, 8 Desember 2016

Marvin Jerremy Budiman/13515076