Penerapan Algoritma BFS dan DFS pada Topological Sorting

Patrick Lumban Tobing - 13510013
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13510013@std.stei.itb.ac.id

***Abstraksi*—Makalah ini menjabarkan dan menjelaskan mengenai penggunaan algoritma Breadth-First Search (BFS) dan Depth-First Search (DFS) pada Topological Sorting. Topological Sorting adalah suatu metode sorting pada Directed Acylic Graph untuk menentukan keterurutan setiap simpul dari simpul yang paling diutamakan sampai yang paling tidak diutamakan. Pada makalah ini pengenalan mengenai algoritma BFS dan DFS akan terlebih dahulu diperkenalkan, dilanjutkan dengan penjelasan mengenai topological sorting. Dasar teori yang telah dijelaskan ini mendasari implementasi yang dilakukan di bagian berikutnya dalam penggunaan BFS serta DFS pada Topological Sorting.**

***Kata kunci*—BFS, DFS, Topological Sorting, Directed Acylic Graph**

I. PENDAHULUAN

Graf merupakan kumpulan dari objek, di mana beberapa bahkan dapat semua objek tersebut saling terhubung melalui konektor. Objek-objek tersebut di dalam graf direpresentasikan sebagai simpul, sedangkan konektor yang menghubungkan dua simpul disebut dengan sisi. Graf dapat memiliki sisi yang berarah atau tidak berarah, pada makalah ini graf yang dibahas adalah graf dengan sisi berarah sehingga dapat diketahui keterurutan antara setiap simpul.

Proses manipulasi yang dilakukan terhadap suatu graf bermacam-macam, salah satunya adalah penelusuran terhadap simpul-simpul di graf. Pada makalah ini akan dibahas dua metode penelusuran atau traversal pada graf, yaitu Breadth-First Search dan Depth-First Search.

Breadth-First Search (BFS) adalah salah satu metode penelusuran pada pohon atau graf dengan tujuan menemukan objektif yang ditentukan di awal pencarian. Algoritma BFS pada umumnya menggunakan struktur data queue sebagai penampung simpul selama mengunjungi seluruh simpul tetangga dari simpul yang sedang dikunjungi saat tersebut.

Metode penelusuran graf lainnya yang terkenal adalah Depth-First Search (DFS). Algoritma ini juga bertujuan untuk menemukan objektif yang diinginkan pada suatu pohon atau graf. Berbeda dari BFS, DFS secara umum menggunakan konsep rekursi dan backtracking untuk menelusuri setiap simpul pada suatu graf sampai ditemukan objektif yang diinginkan.

Selain manipulasi untuk melakukan penelusuran di dalam graf, terdapat juga proses manipulasi untuk mengurutkan simpul-simpul pada graf berdasarkan kepentingannya jika dibandingkan dengan simpul yang lain. Proses manipulasi ini secara khusus dilakukan terhadap graf berarah. Salah satu metode yang digunakan yang akan dibahas pada makalah ini adalah Topological Sorting. Metode topological sorting secara umum adalah metode yang bertujuan untuk mengurutkan simpul-simpul pada suatu graf dari simpul yang paling penting atau diutamakan sampai simpul yang paling dapat diabaikan atau prioritas terakhir. Pada proses topological sorting, terdapat penggunaan metode manipulasi graf yang lain yaitu BFS dan DFS. Tentu tidak keduanya digunakan dalam satu waktu, tergantung pengembang algoritma atau aplikasi yang menentukan algoritma yang mau digunakan.

Makalah ini akan membahas mengenai penerapa algoritma Breadth-First Search dan Depth-First Search dalam proses pengembangan metode Topological Sorting yang bertujuan untuk menentukan keterurutan simpul-simpul di dalam suatu graf. Metode topological sorting ini nantinya dapat dipakai dalam berbagai kebutuhan di dunia nyata, seperti membuat jadwal pengerjaan proyek, pembuatan diagram untuk optimasi query database, dan multiplikasi matriks. Penerapan-penerapan ini tidak akan dibahas secara lanjut pada makalah ini, hanya bagaimana algoritma Breadth-First Search dan Depth-First Search dapat diimplementasikan pada metode topological sorting.

II. DASAR TEORI

A. BREADTH-FIRST SEARCH

Breadth-First Search (BFS) adalah salah satu strategi pencarian pada suatu graf yang terdiri dari dua proses utama, yaitu mengunjungi serta mengecek suatu simpul dari graf dan mengunjungi lagi setiap simpul tetangga dari simpul yang sedang

dicek sekarang. Penelusuran pada BFS dimulai dari simpul akar/awal, kemudian pengecekan dilakukan untuk setiap simpul tetangga yang belum dikunjungi, begitu seterusnya sampai semua simpul sudah dikunjungi atau objektif dari penelusuran sudah dicapai, misal mencari suatu simpul dengan kondisi tertentu. Algoritma ini menggunakan struktur data queue/antrian untuk menyimpan simpul sementara yang akan dikunjungi dan sudah dikunjungi.

Untuk lebih mempermudah, ilustrasikan ide dari Breadth-First Search sebagai berikut: suatu gelombang dikirimkan dari sumber, yaitu simpul akar/awal. Gelombang tersebut akan mengenai semua simpul yang berjarak satu sisi dari simpul akar. Dari sini, gelombang itu akan mengenai semua simpul yang berjarak duat sisi dari simpul akar yang bertetangga dengan semua simpul yang sebelumnya sudah dikenai, dan begitu seterusnya. Untuk mempertahankan gelombang di bagian depan dari proses, ilustrasi ini menggunakan antrian dengan aturan FIFO (First-In-First-Out), di mana suatu simpul akan berada di antrian apabila gelombang sudah mengenainya tetapi belum keluar dari simpul tersebut.

Berikut adalah penjelasan mengenai strategi secara umum dari Breadth-First Search:

Setiap simpul pada graf memiliki salah satu warna atau state sebagai berikut:
1.Belum ditemukan
2.Ditemukan tetapi belum dikunjungi
3.Sudah dikunjungi

Kondisi dari suatu simpul, *u*, ditandai dengan satu warna khusus:
1.warna[*u*] = putih – untuk kondisi pertama
2.warna[*u*] = abu-abu – untuk kondisi kedua
3.warna[*u*] = hitam – untuk kondisi ketiga

Algoritma BFS ini mengimplementasikan pohon breadth-first search dengan simpul awal/sumber adalah *s*, sebagai akar. Simpul ayah yang mendahului suatu simpul adalah simpul yang menghasilkan simpul anak yang membuat simpul anak tersebut ditemukan pertama kali. Untuk setiap simpul, *v*, simpul ayah dari *v* diletakkan di variabel π[*v*]. Variabel lain, d[*v*] memiliki jumlah sisi yang dibutuhkan untuk mencapai simpul *v* dari simpul *s* (akar). Algoritma ini menggunakan antrian FIFO, Q, untuk menyimpan simpul yang berwarna abu-abu.

Berikut adalah algoritma untuk mengimplementasikan BFS seperti yang sudah dijelaskan di atas:

BFS(V, E, *s*)

```
  for each u in V − {s}   ▷ for each simpul u in V[G] except s.
        do warna[u] ← putih
        d[u] ← ∞
        π[u] ← NULL
    warna[s] ← abu-abu          ▷ simpul akar ditemukan
    d[s] ← 0                    ▷ inisialisasi
    π[s] ← NULL                 ▷ inisialisasi
    Q ← {}                      ▷ bersihkan antrian Q
    ENQUEUE(Q, s)
    while Q not-empty
      do u ← DEQUEUE(Q)    ▷ u = head[Q]
        for each v bertetangga to u      ▷ iterasi untuk setiap simpul yang bertetangga
            do if warna[v] ← putih    ▷ belum pernah ditemukan sebelumnyajika putih
                  then  warna[v] ← abu-abu
                        d[v] ← d[u] + 1
                        π[v] ← u
                        ENQUEUE(Q, v)
        DEQUEUE(Q)
    warna[u] ← hitam
```

B. DEPTH-FIRST SEARCH

Seperti algoritma Breadth-First Search, algoritma Depth-First Search menelusuri komponen-komponen yang saling terhubung pada graf dan mendefinisikan suatu pohon merentang. Ide dasar dari DFS adalah sebagai berikut: algoritma ini secara bertahap akan mengeksplorasi setiap sisi. Penelusuran dimulai dari simpul akar, segera setelah sebuah simpul ditemukan dari simpul akar, algoritma ini akan mulai mengeksplorasi dari situ, begitu seterusnya sampai buntu, setelah tidak ada jalan, proses akan melakukan backtrack untuk kembali ke simpul di atasnya dan proses lagi ke bawah, begitu seterusnya (Tidak seperti BFS yang menaruh simpul-simpul di dalam antrian agar dapat dikunjungi belakangan).

Berikut adalah penjelasan mengenai strategi secara umum dari Depth-First Search:
Pilih simpul akar *s* dari graf, dan tandai "dikunjungi". Simpul *s* ini menjadi simpul *u* yang saat ini dikunjungi. Lalu, lakukan penelusuran pada graf untuk menemukan suatu sisi (*u*,*v*) dari simpul *u*. Apabila sisi (*u*,*v*) merujuk kepada simpul *v* yang sudah dikunjungi, maka lakukan backtrack kembali ke simpul *u*. Tetapi, jika sisi (*u*,*v*) merujuk ke simpul yang belum dikunjungi, maka tandai simpul tersebut sudah dikunjungi dan jadikan simpul *v* sebagai simpul saat ini, dan ulangi langkah-langkah di atas. Suatu saat proses ini akan menemui jalan

buntu, yaitu keadaan di mana setiap sisi dari simpul *u* saat ini merujuk kepada simpul yang sudah dikunjungi. Untuk keluar dari situasi deadlock ini, proses harus berjalan mundur melalui sisi yang membawa ke simpul *u* saat ini dan menuju ke simpul *v* yang sebelumnya sudah dikunjungi. Simpul *v* ini kemudian dijadikan simpul *u* saat ini dan lakukan proses seperti di atas untuk sisi-sisi yang masih belum diproses. Apabila algoritma ini sudah melakukan proses mundur atau backtrack sampai ke simpul akar *s*, maka telah terbentuk pohon DFS dengan semua simpul sudah ditelusuri dari simpul akar *s*.

Seperti BFS, setiap simpul dari pohon pada DFS berada dalam salah satu kondisi di bawah ini:

1. Belum ditemukan
2. Sudah ditemukan (belum semua simpul di bawah simpul ini ditelusuri dari simpul ini)
3. Selesai (semua simpul di bawah simpul ini sudah ditelusuri)

Kondisi atau state dari simpul disimpan dalam satu variabel warna seperti di bawah ini:

1. color[*u*] = putih – untuk kondisi satu
2. color[*u*] = abu-abu – untuk kondisi dua
3. color[*u*] = hitam – untuk kondisi tiga

Seperti BFS, Depth-First Search menggunakan $\pi[v]$ untuk mencatat simpul ayah dari simpul v. $\pi[v]$ = NULL jika dan hanya jika simpul *v* adalah simpul akar.

Berikut adalah algoritma untuk mengimplementasikan BFS seperti yang sudah dijelaskan di atas:

DFS (V, E)

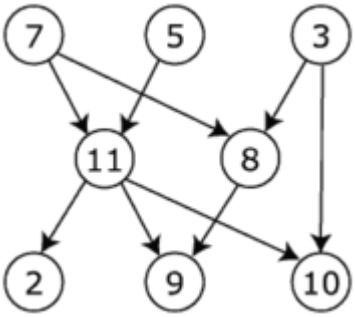
```
for each simpul u in V[G]
  do warna[u] ← putih
     π[u] ← NULL
for each simpul u in V[G]
  do if warna[u] ← putih
     then DFS-Visit(u) ▷ membuat pohon DFS baru dari simpul u
```

DFS-Visit(u)

```
color[u] ← abu-abu      ▷ u ditemukan
for each simpul v yang bertetanggadengan u   ▷ telusuri sisi (u, v)
  do if color[v] ← putih
     then π[v] ← u
          DFS-Visit(v)
warna[u] ← hitam
```

C. TOPOLOGICAL SORTING

Topological sorting atau topological ordering adalah algoritma untuk melakukan pengurutan secara linier terhadap semua simpul dari sebuah graf berarah, di mana setiap sisi *uv*, simpul *u* berada sebelum *v* pada hasil pengurutan. Setiap simpul dari graf tersebut dapat saja dimisalkan sebagai suatu pekerjaan yang harus dilakukan. Sisi yang ada menandakan urutan pengerjaan, bahwa suatu pekerjaan harus dilakukan terlebih dahulu sebelum pekerjaan yang lain. Dalam hal ini, topological sorting atau topological ordering adalah keterurutan yang sahih dari penjadwalan pekerjaan ini. Topological sorting dapat dilakukan jika dan hanya jika graf yang bersangkutan tidak mempunyai siklus berarah, dengan kata lain graf itu adalah Directed Acyclic Graph (DAG) atau graf berarah yang tidak mempunyai siklus. Setiap DAG pasti mempunyai minimal satu keterurutan secara topologi dan algoritma topological sorting digunakan untuk menemukan keterurutan topologi dari DAG tersebut secara linier.



Gambar 1 Contoh Directed Acyclic Graph (DAG)

Salah satu keterurutan topologi dari DAG di atas adalah: 5, 7, 3, 8, 11, 10, 9, 2. Keterurutan ini dilihat dari sedikitnya sisi yang masuk ke suatu simpul.

Berikut adalah algoritma dari topological sorting secara umum:

```
L ← list kosong penampung simpul terurut
S ← himpunan simpul yang tidak mempunyai sisi yang masuk
while S not-empty do
   ambil simpul n dari S
   masukkan n ke L
   for each simpul m dengan sisi e dari n ke m do
      hilangkan sisi e dari graf
      if m tidak punya sisi masuk lagi then
         masukkan m ke dalam S
if graf memiliki sisi then
   return error (graf memiliki setidaknya satu siklus)
else
   return L (keterurutan secara topologi diperoleh)
```

III. IMPLEMENTASI BFS DAN DFS PADA TOPOLOGICAL SORTING

Dalam melakukan implementasi algoritma Breadth-First Search dan Depth-First Search pada metode Topological Sort pada makalah ini digunakan program dalam bahasa C++. Sebelum melihat hasil implementasi, di bawah ini akan dijabarkan pseudocode untuk mengimplementasikan program tersebut:

A. DFS dalam Topological Sorting

```
procedure topSort(Graph G)
   InitStack(S)
   for each vertex v in G do
      G.setVisited(v,false)
   for each vertex v in G do
      if(not G.visited(v)) then
         dfs(G,v,s)
   printStack(S)

procedure dfs(Graph G, Vertex v, Stack s)
   G.setVisited(v,true)
   for each vertex u bertetangga dengan v do
      if(not G.visited(u)) then
         dfs(G,u,s)
   s.push(u)
```

Algoritma ini akan menelusuri semua anak dari simpul *v* sampai anak paling akhir, kemudian simpul pertama yang ditemukan akan dimasukkan ke stack. Algoritma dilanjutkan dengan proses backtracking sampai semua simpul ditelusuri dan peletakkan simpul ke dalam stack dilakukan secara terbalik sehingga urutan yang didapat menjadi benar. Misal urutan masuk simpul ke dalam stack adalah 1 → 2 → 3 → 4, maka urutan topological sortingnya adalah 4 → 3 → 2 → 1.

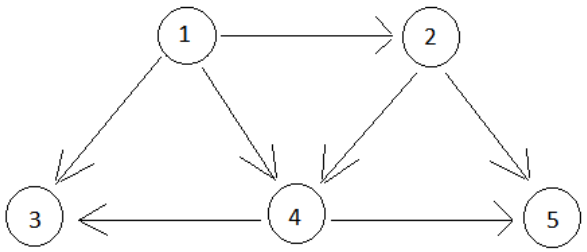
B. BFS dalam Topological Sorting

```
procedure topSortBFS(Graph G)
   InitQueue(Q)
   InitList(L)
   for each vertex v in G do
      G.setVisited(v,false)
   for each vertex v yang tidak punya sisi masuk do
      G.setVisited(v,true)
      Q.add(v)
   while(not Q.empty()) do
      v ← Q.popFirst()
      L.insertFirst(v)
      for each vertex u bertetangga dengan v do
            if(notEmptyInEdge(u)) then
               u.inEdge ← v.inEdge – 1
            else
               Q.add(u)
```

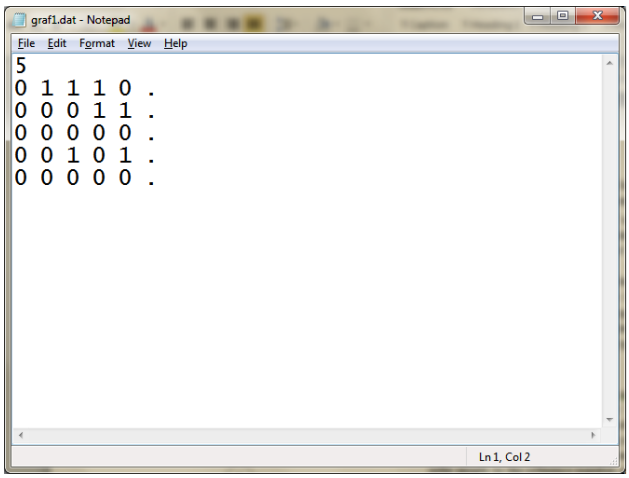
Algoritma BFS pada Topological Sorting ini memanfaatkan queue untuk penampung simpul yang akan dikunjungi dan list untuk penampung simpul yang sudah terurut. Pada pertama kali, semua simpul yang tidak memiliki simpul masuk dimasukkan ke queue dan akan diproses pertama kali (menjadi simpul dengan prioritas urutan tertinggi) serta dimasukkan ke list di awal. Kemudian setiap anak dari simpul-simpul ini akan diproses juga, apabila simpul anak tidak mempunyai sisi masuk, maka simpul itu akan dimasukkan ke akhir antrian, tetapi jika masih memiliki simpul masuk, maka jumlah simpul masuk pada simpul tersebut dikurangi satu. Proses ini diulang terus hingga ke anak dari simpul anak tersebut dan anaknya lagi sampai antrian kosong yang menandakan semua simpul sudah diproses dan list yang berisi simpul yang terurut secara topologi sudah terbentuk.

Kedua pseudocode ini telah penulis implementasi dalam program sederhana berbahasa C++. Program ini akan membaca matriks ketetanggan dari suatu graf dari sebuah file. Dari matriks ketetanggan ini, akan terbentuk objek graf yang memiliki jumlah simpul, dan matriks ketetanggan tiap simpul. Simpul di dalam graf juga didefinisikan sebagai objek. Tiap simpul yang terbentuk di dalam graf memiliki atribut yang menunjukkan jumlah simpul masuk ke dalam simpul tersebut.

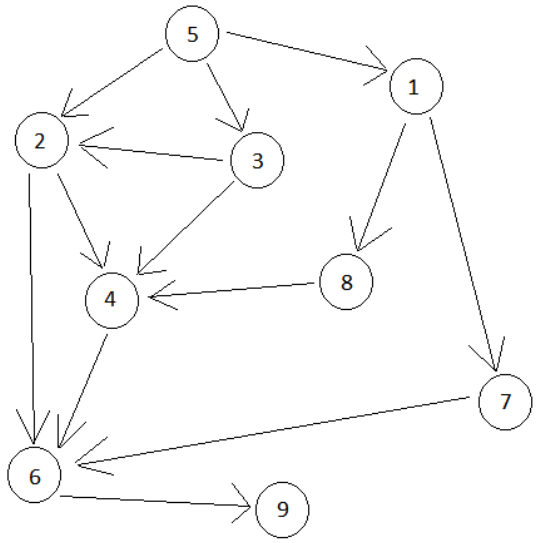
Terdapat beberapa persoalan yang penulis jadikan contoh masukan untuk melakukan pengetesan terhadap implementasi BFS dan DFS pada topological sorting. Berikut adalah graf-graf yang penulis jadikan contoh persoalan beserta masukan matriks ketetanggannya:



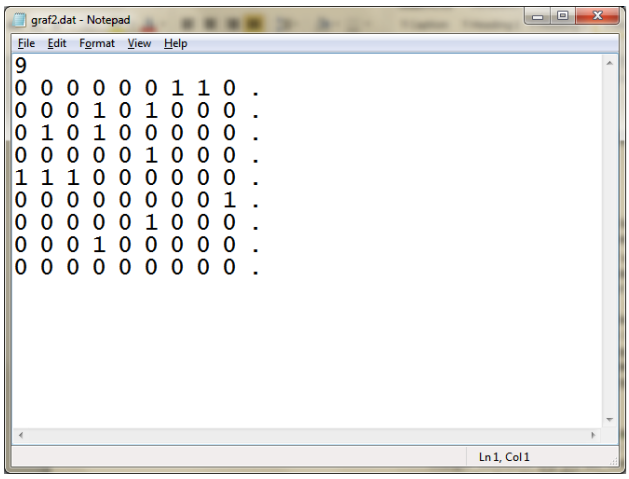
Gambar 2 Graf pertama persoalan Topological Sort



Gambar 3Matriks Ketetanggan untuk persoalan pertama



Gambar 4Graf kedua persoalan Topological Sort



Gambar 5Matriks ketetanggan untuk persoalan kedua

Berikut di bawah ini adalah hasil eksekusi program untuk masing-masing persoalan dengan algoritma BFS dan DFS:

1. Graf pertama

```
D:\ITB\Semester 5\Strategi Algoritma (IF3051)\Makalah>tsortDFS
**************************************************************
************* TOPOLOGICAL SORT USING DEPTH-FIRST SEARCH *************
**************************************************************

Masukkan nama file untuk graf: graf1.dat
Jumlah simpul: 5

Matriks ketetanggaan:
0       1       1       1       0
0       0       0       1       1
0       0       0       0       0
0       0       1       0       1
0       0       0       0       0

Topological order:
1 2 4 5 3

Press any key...
```

Gambar 6 Hasil Topological Sort dengan DFS pada graf 1

```
D:\ITB\Semester 5\Strategi Algoritma (IF3051)\Makalah>tsortBFS
**************************************************************
************* TOPOLOGICAL SORT USING BREADTH-FIRST SEARCH *************
**************************************************************

Masukkan nama file untuk graf: graf1.dat
Jumlah simpul: 5

Matriks ketetanggaan:
0       1       1       1       0
0       0       0       1       1
0       0       0       0       0
0       0       1       0       1
0       0       0       0       0

Topological order:
1 2 4 3 5

Press any key...
```

Gambar 7Hasil Topological Sort dengan BFS pada graf 1

2. Graf kedua

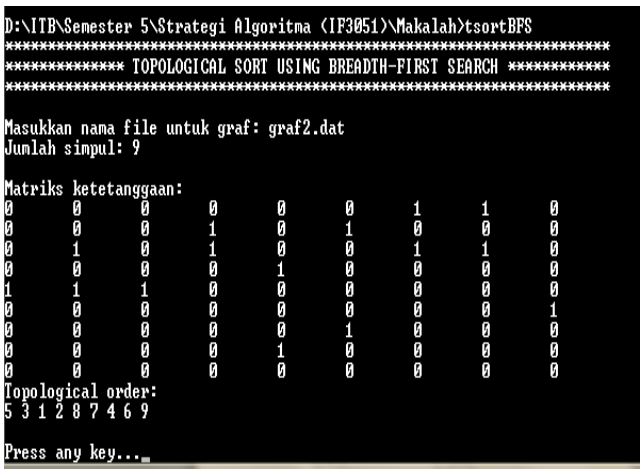
```
D:\ITB\Semester 5\Strategi Algoritma (IF3051)\Makalah>tsortDFS
**************************************************************
************* TOPOLOGICAL SORT USING DEPTH-FIRST SEARCH *************
**************************************************************

Masukkan nama file untuk graf: graf2.dat
Jumlah simpul: 9

Matriks ketetanggaan:
0       0       0       0       0       1       1       0
0       0       0       1       0       1       0       0       0
0       1       0       1       0       0       1       1       0
0       0       0       0       1       0       0       0       0
1       1       1       0       0       0       0       0       0
0       0       0       0       0       0       0       0       1
0       0       0       0       0       1       0       0       0
0       0       0       0       1       0       0       0       0
0       0       0       0       0       0       0       0       0
Topological order:
5 3 2 1 7 8 4 6 9

Press any key...
```

Gambar 8 Hasil Topological Sort dengan DFS pada graf 2



Gambar 9 Hasil Topological Sort dengan BFS pada graf 2

Hasil topological order untuk graf pertama adalah sebagai berikut:
DFS: 1→ 2 → 4 → 5 → 3
BFS: 1→ 2 → 4 → 3 → 5

Hasil topological order untuk graf kedua adalah sebagai berikut:
DFS: 5→ 3 → 2 → 1 → 7 → 8 → 4 → 6 → 9
BFS: 5→ 3 → 1 → 2 → 8 → 7 → 4 → 6 → 9

Dari hasil eksekusi program di atas dapat dilihat bahwa hasil dari algoritma Breadth-First Search dan Depth-First Search terhadap Topological Sort memiliki sedikit perbedaan dalam urutan. Tetapi jika ditinjau kembali ke gambar graf di atas, tidak ada simpul pada topological order pada BFS maupun DFS yang memiliki panah ke simpul di depannya. Hal ini menyatakan bahwa keterurutan setiap simpul sudah benar secara topological walaupun terdapat sedikit perbedaan urutan pada BFS dan DFS.

IV. KESIMPULAN

Topological sorting merupakan suatu algoritma yang digunakan untuk memproses suatu Directed Acyclic Graph (DAG) sehingga mendapatkan keterurutan dari setiap simpul yang ada secara topologi atau sesuai prioritas.

Dalam implementasinya, topological sorting dapat dikembangkan untuk memanfaatkan algoritma Breadth-First Search dan Depth-First Search dalam proses penelusuran struktur pohon atau graf yang ada.

Hasil implementasi dari algoritma BFS serta DFS pada topological sort memiliki sedikit perbedaan. Perbedaan ini tidak menyebabkan hasil eksekusi keduanya salah. Hal ini disebabkan tidak ada simpul yang menunjuk ke simpul di depannya pada urutan yang dihasilkan dari proses topological sort. Ini menyatakan bahwa keterurutan yang dibentuk oleh BFS dan DFS pada topological sort sudah benar secara topologi.

REFERENSI

[1] Munir, Rinaldi, *Diktat Kuliah IF3051Strategi Algoritma*, Penerbit Informatika : Bandung, 2009

[2] Cormen, Thomas H., dkk,. *Introduction to Algorithms Third Edition* . The MIT Press: London, 2009

[3] http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/breadthSearch.htm, diakses pada tanggal 16 Desember 2012, pukul 16.32 WIB

[4] http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/depthSearch.htm, diakses pada tanggal 16 Desember 2012, pukul 16.35 WIB

[5] http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/topoSort.htm, diakses pada tanggal 16 Desember 2012, pukul 17.40 WIB

[6] http://faculty.simpson.edu/lydia.sinapova/www/cmsc250/LN250_Weiss/L20-TopSort.htm, diakses pada tanggal 16 Desember 2012, pukul 17.40 WIB

[7] http://homepages.ius.edu/rwisman/C455/html/notes/Chapter22/TopSort.htm, diakses pada tanggal 16 Desember 2012, pukul 18.03 WIB

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, 16 Desember 2012

ttd

Patrick Lumban Tobing
13510013