Penerapan Topological Sorting pada Penjadwalan Proses

Arno Alexander - 13515141
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515141@std.stei.itb.ac.id

***Abstrak*—Sebagian besar perangkat lunak yang ada saat ini memiliki struktur yang kompleks. Salah satu komponen perangkat lunak adalah program, yang jika dieksekusi disebut proses. Proses dalam sebuah perangkat lunak memiliki ketergantungan satu sama lain sedemikian sehingga mungkin terdapat proses yang tidak boleh dijalankan kecuali proses lain yang menjadi prasyaratnya telah selesai dijalankan. Hubungan antar proses tersebut dapat dinyatakan secara matematis dalam bentuk graf. Agar proses dapat dijalankan oleh prosesor dengan urutan yang tepat, struktur graf tersebut harus diterjemahkan menjadi jadwal proses yang terurut secara linear. Algoritma topological sorting dapat digunakan untuk keperluan tersebut.**

Kata kunci—depth first search, graf, penjadwalan, proses, topological sorting.

I. Pendahuluan

Teknologi dan perkembangannya yang pesat dapat dilihat dalam kehidupan sehari-hari. Contoh yang paling fenomenal adalah komputer. Komputer telah menunjukkan perannya dalam mempermudah pemecahan masalah di berbagai bidang dan memanjakan orang dari berbagai kalangan.

Pada mulanya, komputer hanya memiliki satu fungsi yang sekaligus mencerminkan namanya, yaitu sebagai mesin hitung. Kemampuan perhitungan yang dilakukan masih terbatas pada operasi dasar seperti penjumlahan dan pengurangan. Komputer-komputer pertama menggunakan prinsip mekanik atau elektrik yang bersifat kontinu untuk merepresentasikan permasalahan sehingga dapat juga disebut komputer analog. Jenis komputer yang menjadi penerus komputer analog adalah komputer yang menggunakan sinyal diskrit untuk menyelesaikan permasalahan, yaitu komputer digital. Perkembangan perangkat keras komputer digital meningkat secara eksponensial seiring waktu. Menurut Gordon Moore, co-founder dari Intel Corporation, kepadatan transistor akan bertambah dua kali lipat setiap dua tahun[4]. Transistor merupakan komponen penyusun prosesor, menggantikan tabung hampa yang dianggap terlalu besar dan panas dari komputer digital generasi pertama. Sebaliknya, ukuran dan harga komputer justru terus mengalami penurunan[4].

Bagian lain dari komputer digital selain perangkat keras adalah perangkat lunak, yaitu sekumpulan instruksi program beserta data-data yang terkait. Komponen perangkat lunak tersebut dimuat ke memori dan dieksekusi oleh prosesor[5]. Sebelum perangkat lunak diciptakan, instruksi dan data dimasukkan secara manual oleh pemrogram. Jika terjadi kesalahan, proses harus diulangi. Adanya perangkat lunak jelas mempermudah penggunaan komputer karena bentuk abstrak permasalahan telah tersedia dan pengguna tinggal memasukkan data spesifiknya.

Semakin mudah, murah, dan kecilnya komputer membuat cakupan pengguna komputer semakin luas. Hal tersebut diikuti dengan tumbuhnya kebutuhan pengguna, mendorong industri perangkat lunak untuk berkembang agar dapat memenuhi kebutuhan-kebutuhan tersebut. Kemampuan perangkat lunak menjadi bervariasi, mulai dari yang kecil dan sederhana seperti sebatas menghitung operasi aritmatika hingga yang besar dan kompleks seperti melakukan simulasi. Perangkat lunak berskala besar memiliki data yang lebih besar, program yang lebih banyak, dan hubungan antar proses yang lebih kompleks.

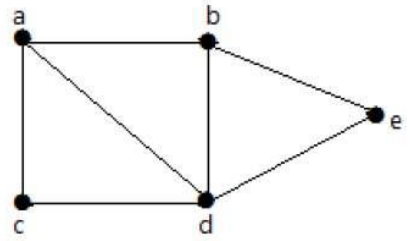
Salah satu bentuk hubungan antar proses adalah dependensi, yaitu keadaan di mana sebuah proses tidak dapat dijalankan sebelum proses yang lain selesai. Karena itu, urutan eksekusi harus diatur sedemikian sehingga masing-masing proses dijalankan setelah proses lain yang menjadi prasyaratnya selesai dieksekusi. Makalah ini akan membahas penyusunan jadwal eksekusi proses berdasarkan hubungan dependensi antar proses menggunakan algoritma topological sorting.

II. Dasar Teori

A. Graf

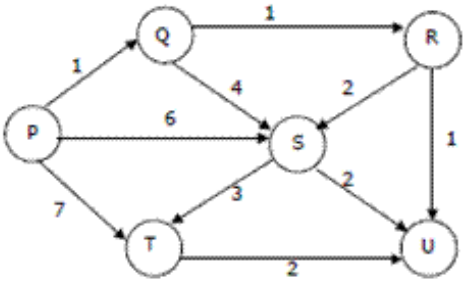
Graf adalah struktur diskrit yang tersusun oleh simpul (*vertex*) dan sisi (*edge*). Setiap sisi merupakan penghubung antara dua simpul dan dapat dituliskan sebagai pasangan (v_1, v_2), dengan v_1 dan v_2 masing-masing adalah simpul yang dihubungkannya. Kedua simpul yang berhubungan tersebut dikatakan bertetangga, sedangkan sisi tersebut dikatakan bersisian dengan kedua simpul. Masing-masing simpul terhubung dengan beberapa sisi atau tidak terhubung sama sekali. Sebuah graf harus memiliki setidaknya sebuah simpul, namun boleh tidak memiliki sisi. Secara formal, definisi dari

sebuah graf *G* adalah *G*=(*V*,*E*) dengan *V* adalah himpunan tak kosong dari simpul dan *E* adalah himpunan dari sisi[2]. Graf digunakan untuk merepresentasikan hubungan antara entitas diskrit. Entitas dilambangkan dengan simpul, sedangkan hubungan dinyatakan sebagai sisi.



Gambar 1. *Contoh graf dengan 5 simpul dan 7 sisi*
(Sumber: https://www.tutorialspoint.com/graph_theory/graph_theory_connectivity.htm diakses pada 15 Mei 2017)

Baik simpul maupun sisi graf dapat memiliki atribut. Contoh atribut untuk simpul adalah simbol, sedangkan atribut untuk sisi dapat berupa bobot atau arah. Graf yang memiliki atribut arah pada sisi-sisinya disebut graf berarah, sebaliknya disebut graf tak berarah. Graf yang setiap sisinya memiliki nilai disebut graf berbobot, sebaliknya disebut graf tak berbobot.



Gambar 2. *Contoh graf berbobot sekaligus berarah*
(Sumber: http://www.geeksforgeeks.org/algorithms-gq/graph-shortest-paths-gq/ diakses pada 15 Mei 2017)

Dalam graf berarah, sisi disebut juga sebagai busur (*arc*). Setiap busur menghubungkan dua simpul dengan arah tertentu. Kedua simpul ini masing-masing disebut simpul asal dan simpul terminal. Simpul asal adalah pangkal atau titik mulai dari sebuah busur, sedangkan simpul terminal adalah tujuan atau ujung depan dari busur yang bersangkutan. Dalam penulisan sisi, simpul asal ditulis terlebih dahulu sebelum simpul terminal. Jadi, untuk setiap busur (v_1, v_2), arah busur adalah dari v_1 ke v_2. Busur (v_1, v_2) tidak sama dengan (v_2, v_1) karena memiliki arah yang berbeda.

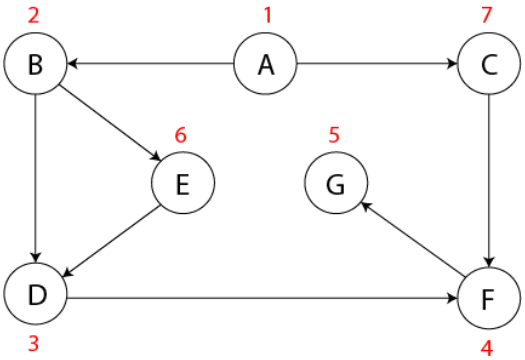
Sebuah graf tersusun dari graf-graf lain yang lebih sederhana yang disebut upagraf (*subgraph*). *G'*(*V'*,*E'*) adalah upagraf dari *G*(*V*,*E*) jika *V'* adalah himpunan bagian dari *V* dan *E'* adalah himpunan bagian dari *E*.

Setiap simpul pada graf memiliki derajat (*degree*). Pada graf tak berarah, derajat adalah banyaknya sisi yang bersisian dengan suatu simpul[2]. Sedangkan, pada graf tak berarah, derajat digolongkan menjadi dua, yaitu derajat-masuk (*in-degree*) dan derajat-keluar (*out-degree*). Pada sebuah simpul *v*, derajat-masuk adalah banyaknya busur yang bersisian dengan *v* dan masuk ke *v*, sedangkan derajat-keluar adalah banyaknya busur yang bersisian dengan *v* dan keluar dari *v*.

B. Depth First Search (DFS)

Algoritma *Depth First Search* merupakan salah satu algoritma pencarian dalam struktur graf dan pohon Pencarian dilakukan dengan cara mengunjungi simpul tetangga satu demi satu secara mendalam. Arti kata mendalam di sini adalah menelusuri sebuah simpul beserta simpul-simpul lain yang terletak setelahnya. Setelah tidak ada simpul tetangga yang bisa dikunjungi, dilakukan runut balik ke simpul sebelumnya, lalu algoritma dilanjutkan[3]. Algoritma akan dihentikan jika seluruh simpul dalam graf telah dikunjungi. Pencarian juga dapat dihentikan segera setelah menemukan simpul yang dicari jika tidak ingin menemukan seluruh solusi yang mungkin. Secara umum, langkah-langkah DFS dengan simpul awal *v* dapat dijabarkan secara rekursif sebagai berikut[1]:

1) Tandai *v* sebagai simpul yang telah dikunjungi.
2) Misalkan himpunan simpul tetangga dari *v* adalah V_t = {v_1, v_2, v_3, ... , v_n}, *n* adalah banyaknya simpul tetangga. Untuk v_k yang merepresentasikan setiap elemen V_t yang belum dikunjungi, lakukan DFS dengan simpul awal v_k.



Gambar 3. *Urutan penelusuran simpul pada graf dengan DFS, dimulai dari simpul A*
(Sumber: Koleksi penulis)

DFS juga dapat dilakukan secara iteratif tanpa menggunakan fungsi rekursif. Struktur data tumpukan (*stack*) digunakan untuk menampung simpul yang akan dikunjungi selanjutnya. *Stack* adalah struktur data *Last In First Out* (LIFO) karena data yang baru dimasukkan diletakkan di atas data terakhir yang dimasukkan dan pengambilan data dari *stack* hanya dapat dilakukan pada data yang terletak paling atas. Operasi memasukkan data disebut *push*, sedangkan operasi mengambil data disebut *pop*. Data yang terletak paling atas disebut *top*. Misalkan DFS dimulai dari simpul *v*:

1) Misalkan *S* adalah tumpukan. Pada kondisi awal, *S* kosong (tidak memiliki elemen anggota).
2) Masukkan simpul *v* ke *S*.
3) Jika *S* tidak kosong, lanjut ke langkah selanjutnya. Jika *S* kosong, algoritma selesai.

4) Lakukan *pop* pada *S*. Misalkan hasilnya adalah simpul *u*.

5) Jika *u* belum dikunjungi, lanjut ke langkah selanjutnya. Jika *u* telah dikunjungi, kembali ke langkah 3.

6) Tandai *u* sebagai simpul yang telah dikunjungi.

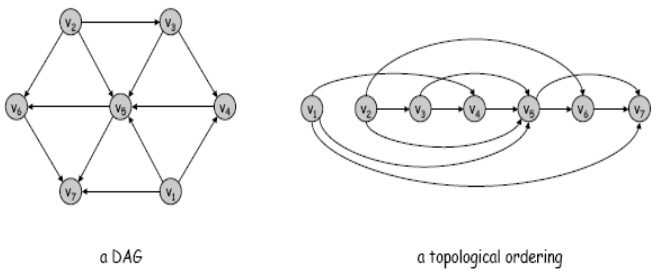
7) Misalkan himpunan simpul tetangga dari *u* adalah $U_t = \{u_1, u_2, u_3, \ldots, u_n\}$, *n* adalah banyaknya simpul tetangga. Masukkan setiap elemen U_t ke *S*.

8) Kembali ke langkah 3.

DFS rekursif dan DFS iteratif memiliki perilaku yang sama. Secara umum, masing-masing dapat dibagi menjadi dua bagian. Bagian pertama adalah penelusuran simpul yang belum dikunjungi dan mengubah statusnya menjadi telah dikunjungi. Bagian ini dilakukan di setiap simpul tepat satu kali, sehingga kompleksitas bagian ini adalah $O(|V|)$, dengan $|V|$ adalah banyaknya simpul. Bagian kedua adalah penelusuran simpul tetangga dari masing-masing simpul, dilakukan satu kali setiap sebuah operasi dari bagian pertama selesai. Pada akhirnya, bagian ini akan menelusuri seluruh hubungan ketetanggaan. Karena setiap hubungan ketetanggaan dihubungkan oleh sebuah sisi, kompleksitas bagian ini adalah $O(|E|)$, dengan $|E|$ adalah banyaknya sisi. Jadi, kompleksitas waktu total dari DFS adalah $O(|V| + |E|)$.

C. *Topological Sorting*

Pada graf berarah, simpul-simpul penyusunnya dapat disusun secara linier pada sebuah senarai sedemikian sehingga untuk setiap busur (v_1, v_2), simpul asal v_1 selalu terletak sebelum simpul tujuan v_2. Permasalahan ini disebut *topological sorting*[6]. Prekondisi *topological sorting* agar solusi ditemukan adalah graf tidak boleh memiliki upagraf melingkar searah. Sebuah graf *G*(*V*,*E*) memiliki upagraf melingkar searah jika terdapat himpunan simpul $V' = \{v_1, v_2, v_3, \ldots, v_k\}$, himpunan busur $E' = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), \ldots, (v_{k-1}, v_k), (v_k, v_1)\}$, *V'* adalah himpunan bagian dari *V*, dan *E'* adalah himpunan bagian dari *E*. Graf berarah yang tidak memiliki upagraf melingkar searah disebut *Directed Acyclic Graph* (DAG).



Gambar 4. *Sebuah DAG (kiri) dan urutan simpul-simpulnya secara topologis (kanan)*
(Sumber: http://www.cs.cornell.edu/courses/cs3110/2011sp/lectures/lec21-graphs/graphs.htm diakses pada 17 Mei 2017)

Algoritma ini dapat diimplementasikan dengan bantuan struktur data antrean (*queue*), yaitu struktur data dengan data yang baru dimasukkan akan diletakkan di belakang data terakhir sebelumnya dan pengambilan data hanya dapat dilakukan pada data yang terletak paling depan. Oleh karena itu, *queue* tergolong struktur data *First In First Out* (FIFO) karena data yang dimasukkan lebih awal akan selalu diambil terlebih dahulu. Operasi memasukkan elemen disebut *enqueue*, sedangkan operasi mengambil elemen disebut *dequeue*. Data yang terletak paling depan disebut kepala (head), sedangkan data yang terletak paling belakang disebut ekor (tail). Struktur data ini digunakan untuk menampung simpul yang akan diperiksa selanjutnya dan menampung hasil pengurutan simpul. Urutan solusi yang dihasilkan adalah dari *head* ke *tail*, didapatkan dengan cara melakukan *dequeue* antrean hasil hingga kosong. Langkah-langkah yang dilakukan adalah sebagai berikut:

1) Misalkan *Q* dan *R* adalah sebuah struktur data antrean yang awalnya kosong. *Q* digunakan untuk menyimpan simpul yang akan diperiksa selanjutnya, sedangkan *R* digunakan untuk menyimpan hasil.

2) Hitung derajat-masuk dari setiap simpul.

3) Inisialisasi variabel yang menyatakan banyaknya simpul yang telah dikunjungi dengan 0.

4) Pilih seluruh simpul dengan derajat-masuk 0. Masukkan semuanya ke *Q*. Urutan proses *enqueue* bebas.

5) Jika *Q* kosong, lanjut ke langkah 11. Jika *Q* tidak kosong, lanjut ke langkah selanjutnya.

6) Lakukan *dequeue* dari *Q*. Misalkan hasil operasi tersebut adalah *v*. Simpul *v* disebut sebagai simpul yang sedang dikunjungi.

7) Masukkan *v* ke *R*.

8) Tambahkan banyaknya simpul yang telah dikunjungi dengan 1.

9) Untuk *u* adalah setiap simpul yang bertetangga dengan *v*, kurangi derajat-masuk dengan 1. Jika ada sebuah simpul *u* yang derajat-masuknya menjadi 0, tambahkan *u* ke *Q*.

10) Kembali ke langkah 5.

11) Jika banyaknya simpul yang telah dikunjungi tidak sama dengan banyaknya simpul yang dimiliki graf, solusi tidak ada. Sebaliknya, solusi telah didapatkan.

Algoritma di atas disebut algoritma Kahn. Misalkan $|V|$ adalah banyaknya simpul pada graf dan $|E|$ adalah banyaknya sisi pada graf. Penghitungan derajat simpul memiliki kompleksitas waktu $O(|V| + |E|)$ karena melibatkan penelusuran setiap simpul sebanyak satu kali untuk menginisialisasi derajat-masuk dengan 0 (dilakukan sebanyak $|V|$ kali) dan untuk setiap busur, dilakukan penambahan derajat-masuk dari simpul terminalnya (dilakukan sebanyak

$|E|$ kali). Tampak juga bahwa *enqueue* dan *dequeue* dilakukan masing-masing dua kali dan satu kali untuk setiap simpul sehingga kompleksitas proses ini adalah $O(|V|)$. Proses lain yang dilakukan adalah mengurangi derajat-masuk simpul tetangga dengan 1. Untuk setiap hubungan ketetanggaan, proses tersebut dilakukan sekali. Karena hubungan ketetanggaan ekuivalen dengan busur, kompleksitas proses ini adalah $O(|E|)$. Maka, kompleksitas waktu dari algoritma *topological sort* Kahn adalah $O(|V| + |E|)$.

DFS juga dapat digunakan untuk *topological sorting*. Berbeda dengan Kahn, struktur data yang digunakan untuk penampung data adalah tumpukan (*stack*). Urutan solusi yang dihasilkan adalah dari *top* menuju ke bawah, dapat diperoleh dengan cara melakukan *pop* hingga tumpukan kosong. Ada dua penanda kunjungan, yaitu penanda tetap dan penanda sementara. Penanda tetap menyimpan status kunjungan dari jalur pemeriksaan lain yang telah dilakukan, sedangkan penanda sementara menyimpan status kunjungan dari jalur pemeriksaan yang sedang dilalui. Berikut adalah langkah yang harus dilakukan:

1) Misalkan *S* adalah sebuah struktur data tumpukan yang awalnya kosong. Buat *S* menjadi variabel global agar dapat diakses oleh fungsi antara.
2) Selama masih terdapat simpul yang belum dikunjungi (tanda kunjungan tetap bernilai *false*), kunjungi salah satunya dari simpul yang belum dikunjungi tersebut.

Fungsi mengunjungi adalah sebuah fungsi antara. Langkah untuk mengunjungi sebuah simpul, misalkan simpul *v*, adalah:

1) Jika tanda kunjungan sementara *v* bernilai *true*, hentikan seluruh algoritma dan nyatakan bahwa tidak ada solusi yang memenuhi karena graf pasti bukan merupakan DAG. Jika tanda kunjungan sementara *v* bernilai *false*, lanjut ke langkah selanjutnya.
2) Set tanda kunjungan sementara *v* menjadi *true*. Simpul *v* telah menjadi bagian dari jalur pemeriksaan yang sedang dilalui.
3) Untuk *u* adalah setiap simpul yang bertetangga dengan *v*, jika tanda kunjungan tetap *u* bernilai *false*, kunjungi simpul *u*.
4) Set tanda kunjungan sementara *v* menjadi *false*. Seluruh jalur pemeriksaan yang melalui *v* telah dilalui.
5) Set tanda kunjungan tetap *v* menjadi *true*. Tidak mungkin ada upagraf melingkar searah yang melalui simpul *v*.
6) Masukkan simpul *v* ke *S*.

Setiap simpul akan mengalami pengubahan tanda kunjungan sebanyak empat kali. Kompleksitas proses ini adalah $O(|V|)$. Proses lain yang dilakukan adalah perpindahan dari sebuah simpul yang sedang dikunjungi ke setiap tetangganya. Karena hubungan ketetanggaan dinyatakan oleh busur, bagian ini akan dilakukan sebanyak jumlah busur sehingga kompleksitasnya adalah $O(|E|)$. Maka, kompleksitas waktu total dari *topological sorting* dengan DFS adalah $O(|V| + |E|)$, sama seperti algoritma Kahn meskipun keduanya memiliki perilaku yang berbeda.

III. Analisis Masalah Penjadwalan Proses

Proses adalah program yang sedang dijalankan[7]. Agar sebuah program dapat menjadi proses, program tersebut harus dimuat ke dalam memori terlebih dahulu dan dijalankan oleh prosesor. Banyaknya proses dapat lebih dari satu, atau bahkan dapat melebihi kapasitas prosesor untuk menjalankan proses. Oleh karena itu, setiap proses akan diletakkan dalam sebuah antrean secara berurutan dengan aturan tertentu dan menunggu gilirannya untuk dijalankan. Kegiatan mengurutkan proses disebut penjadwalan.

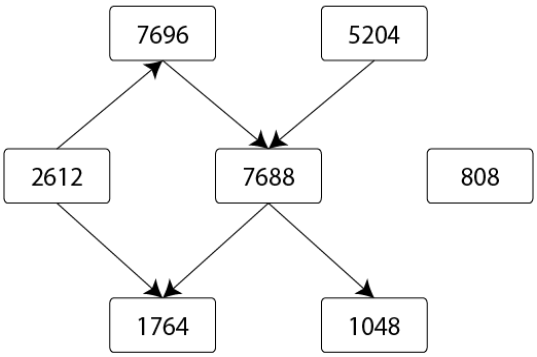
Terdapat berbagai algoritma untuk mengurutkan proses. Contoh yang sederhana antara lain pengurutan berdasarkan waktu kedatangan proses atau lamanya proses. Namun, algoritma tersebut hanya dapat diterapkan jika proses-proses tidak ada yang memiliki hubungan ketergantungan atau dependensi. Sebuah proses *A* disebut dependen terhadap proses *B* jika proses *B* wajib dijalankan terlebih dahulu sampai selesai sebelum proses *A* mulai dijalankan. Jika sebuah proses tidak dependen terhadap proses manapun, proses tersebut dapat dijalankan kapanpun.

Setiap proses memiliki sebuah penanda unik yang membedakannya dari proses lain, yaitu *Process ID* (PID)[7]. Tabel 1 menunjukkan contoh sekumpulan proses (dituliskan PID-nya) dan dependensinya. Dari tabel tersebut akan dicari urutan proses yang harus dijalankan menggunakan *topological sorting*.

Proses	Dependen Terhadap
7696	2612
7688	7696, 5204
5204	-
2612	-
1764	7688, 2612
1048	7688
808	-

Tabel 1. *Contoh proses dan dependensinya*
(Sumber: Koleksi penulis)

Hubungan dependensi antar proses dapat digambarkan sebagai graf berarah. Proses dilambangkan oleh simpul, sedangkan hubungan dependensi dilambangkan oleh busur. Misalkan terdapat proses *A* yang dilambangkan dengan simpul v_1 dan proses tersebut dependen terhadap proses *B* yang dilambangkan dengan simpul v_2, arah busur adalah dari v_2 ke v_1. Busur tidak mengarah ke arah sebaliknya karena yang ingin dicari adalah urutan proses. Dalam kasus ini, proses *B* harus dijalankan sebelum proses *A*, sehingga busur sebaiknya menunjukkan hubungan dari *B* ke *A*. Dari tabel 1, dapat dibuat graf berarah seperti yang ditunjukkan oleh gambar 5 di bawah ini.

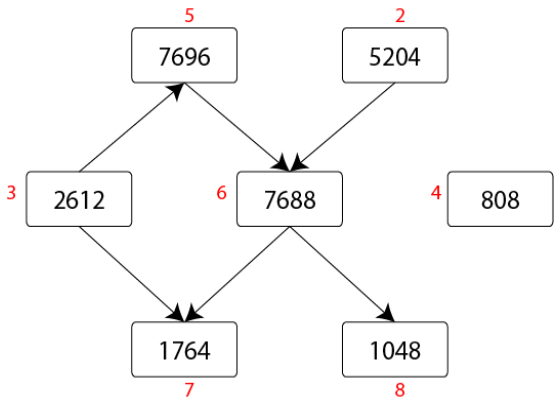


Gambar 5. *Representasi tabel 1 dalam graf berarah*
(Sumber: Koleksi penulis)

Permasalahan tersebut dapat diselesaikan menggunakan *topological sorting* Kahn dengan langkah-langkah seperti pada tabel 2. Langkah pertama menunjukkan tahap penghitungan derajat masuk setiap simpul. Pada langkah tersebut, belum ada simpul yang dikunjungi. Langkah sisanya menunjukkan setiap iterasi dari pengunjungan simpul. Kolom no menunjukkan nomor urut setiap proses. Kolom derajat masuk memiliki beberapa nilai dengan format *v*:*d*, dengan *v* adalah PID dari suatu proses dan *d* adalah banyaknya proses yang harus dilakukan sebelum proses *v* dilakukan. Kolom queue Q dan R masing-masing berisi struktur data antrean seperti yang telah dijelaskan sebelumnya pada langkah-langkah algoritma Kahn. Data dituliskan secara terurut dari atas ke bawah. Data paling atas menunjukkan kepala, sedangkan data paling bawah menunjukkan ekor. Kolom banyak kunjungan menunjukkan banyaknya simpul yang telah dikunjungi. Jika terdapat lebih dari satu pilihan pada suatu langkah, proses dengan PID paling besar akan diprioritaskan.

No	Derajat-masuk	Queue Q	Queue R	Banyak kunjungan
1	7696 : 1 7688 : 2 5204 : 0 2612 : 0 1764 : 2 1048 : 1 808 : 0	5204 2612 808	-	0
2	7696 : 1 7688 : 1 5204 : 0 2612 : 0 1764 : 2 1048 : 1 808 : 0	2612 808	5204	1
3	7696 : 0 7688 : 1 5204 : 0 2612 : 0 1764 : 1 1048 : 1 808 : 0	808 7696	5204 2612	2
4	7696 : 0 7688 : 1 5204 : 0 2612 : 0 1764 : 1 1048 : 1 808 : 0	7696	5204 2612	3
5	7696 : 0 7688 : 0 5204 : 0 2612 : 0 1764 : 1 1048 : 1 808 : 0	7688	5204 2612 808 7696	4
6	7696 : 0 7688 : 0 5204 : 0 2612 : 0 1764 : 0 1048 : 0 808 : 0	1764 1048	5204 2612 808 7696 7688	5
7	7696 : 0 7688 : 0 5204 : 0 2612 : 0 1764 : 0 1048 : 0 808 : 0	1048	5204 2612 808 7696 7688 1764	6
8	7696 : 0 7688 : 0 5204 : 0 2612 : 0 1764 : 0 1048 : 0 808 : 0	-	5204 2612 808 7696 7688 1764 1048	7

Tabel 2. *Penyelesaian masalah pada tabel 1 dengan algoritma Kahn*
(Sumber: Koleksi penulis)



Gambar 6. *Simpul yang dikunjungi untuk setiap nomor tahap selain tahap 1 di tabel 2*
(Sumber: Koleksi penulis)

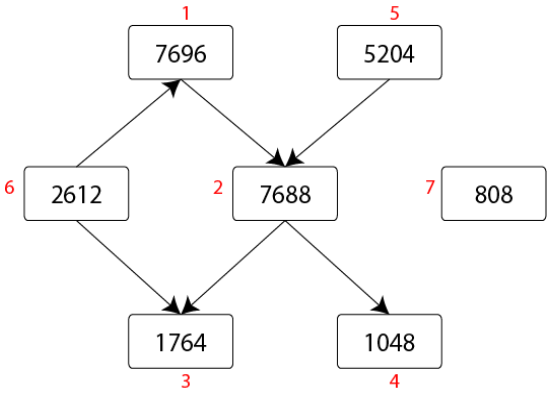
Di akhir algoritma, tampak bahwa banyaknya simpul yang telah dikunjungi adalah 7, sama dengan banyaknya simpul pada graf. Hal tersebut menunjukkan bahwa graf tersebut merupakan DAG. Maka, solusi yang ditampung oleh *queue* R sahih. Urutan proses yang harus dijalankan adalah:

5204, 2612, 808, 7696, 7688, 1764, 1048

Cara lain untuk menyelesaikan permasalahan di atas adalah menggunakan *topological sorting* DFS. Tabel 3 memperlihatkan langkah-langkah yang dilakukan. Kolom no menunjukkan nomor urut setiap proses. Kolom simpul yang diamati menunjukkan simpul yang sedang dilewati dalam proses penelusuran. Kolom status simpul memberikan informasi tentang tetangga dari simpul yang diamati. Kolom stack S berisi struktur data tumpukan penampung hasil seperti yang telah dijelaskan pada langkah-langkah *topological sorting* dengan DFS. Data dituliskan secara vertikal mengikuti morfologi struktur data tumpukan. Data teratas merupakan *top*. Prioritas tertinggi diberikan pada proses dengan PID terbesar seandainya terdapat lebih dari satu pilihan pada suatu langkah.

No	Simpul yang diamati	Status simpul	Stack S
1	7696	Memiliki tetangga yang belum dikunjungi: 7688	-
2	7688	Memiliki tetangga yang belum dikunjungi: 1764 dan 1048	-
3	1764	Tidak memiliki tetangga	1764
4	1048	Tidak memiliki tetangga	1048 1764
5	7688	Sudah tidak memiliki tetangga yang belum dikunjungi	7688 1048 1764
6	7696	Sudah tidak memiliki tetangga yang belum dikunjungi	7696 7688 1048 1764
7	5204	Sudah tidak memiliki tetangga yang belum dikunjungi	5204 7696 7688 1048 1764
8	2612	Sudah tidak memiliki tetangga yang belum dikunjungi	2612 5204 7696 7688 1048 1764
9	808	Tidak memiliki tetangga	808 2612 5204 7696 7688 1048 1764

Tabel 3. *Penyelesaian masalah pada tabel 1 dengan topological sort DFS (Sumber: Koleksi penulis)*



Gambar 7. *Urutan simpul yang dikunjungi menurut tabel 3 (Sumber: Koleksi penulis)*

Program mampu berjalan hingga selesai tanpa ada pernyataan bahwa tidak ada solusi. Maka, solusi yang ditampung *stack* S sahih. Urutan proses yang harus dijalankan adalah:

808, 2612, 5204, 7696, 7688, 1048, 1764

Perhatikan bahwa hasil dari kedua jenis algoritma di atas sama-sama benar dan memiliki skala prioritas pemilihan simpul yang sama, namun menghasilkan jawaban yang berbeda. Penyelesaian untuk *topological sorting* memang mungkin ada sebanyak lebih dari satu. Sebuah algoritma *topological sorting* dengan prioritas pemilihan simpul tertentu hanya mampu menemukan salah satu solusinya. Masih ada solusi lain, salah satunya:

2612, 7696, 5204, 7688, 1048, 808, 1764

IV. Kesimpulan

Topological sorting dapat diterapkan untuk mencari urutan dari proses-proses yang memiliki dependensi satu sama lain. Penyelesaian tidak mungkin ditemukan jika dependensi beberapa proses membentuk struktur siklik searah. Permasalahan ini mungkin memiliki banyak solusi, namun sebuah algoritma *topological sorting* hanya mampu menemukan salah satu solusinya, tergantung perilaku penelusuran graf dan prioritas pengambilan simpul. Selain masalah penjadwalan, ada banyak kasus lain yang dapat direpresentasikan dengan graf berarah dan oleh karena itu, dapat diselesaikan dengan *topological sorting*. Contohnya adalah penjadwalan aktivitas sehari-hari dengan beberapa kegiatan tidak boleh dilakukan sebelum kegiatan lain selesai.

Ucapan Terima Kasih

Penulis mengucapkan syukur kepada Tuhan Yang Maha Esa karena atas berkat-Nya, pembuatan makalah ini berjalan dengan lancar dan dapat terselesaikan. Ucapan terima kasih juga penulis sampaikan kepada Bapak Dr. Ir. Rinaldi Munir, M.T. selaku dosen IF2211 Strategi Algoritma karena atas bimbingan beliau, penulis dapat memahami materi-materi yang terkandung dalam mata kuliah ini. Penulis juga berterima kasih kepada orang tua, keluarga, dan rekan-rekan yang telah

memberikan dukungan selama penulis menimba ilmu. Harapan penulis, semoga makalah ini bermanfaat.

REFERENSI

[1] R. Munir, Diktat Kuliah IF2211 Strategi Algoritma. Bandung: Program Studi Teknik Informatika Institut Teknologi Bandung, 2009.

[2] R. Munir, Diktat Kuliah IF2120 Matematika Diskrit. Bandung: Program Studi Teknik Informatika Institut Teknologi Bandung, 2006.

[3] A. Levitin, Introduction to the Design and Analysis of Algorithms, 3rd ed., New Jersey: Pearson, 2012.

[4] http://dujs.dartmouth.edu/2013/05/keeping-up-with-moores-law/ (diakses pada 15 Mei 2017).

[5] https://www.utdallas.edu/~ivor/cs1315/history.html (diakses pada 15 Mei 2017).

[6] http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/topoSort.htm (diakses pada 16 Mei 2017).

[7] A. Silberschatz, PB. Galvin, dan G. Gagne, Operating System Concepts, 9th ed., New Jersey: Wiley, 2013.

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, 19 Mei 2017

Arno Alexander
13515141