Aplikasi *Directed Acyclic Graph* untuk Manajemen Proyek *Multiparty*

Hanif Arroisi Mukhlis (13519072)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*author@itb.ac.id*

***Abstract*—Proyek yang dilaksanakan secara multiparty memerlukan manajemen yang baik agar dapat berjalan secara efektif dan efisien. Algoritma menggunakan Direct Acyclic Graph untuk membagi tugas-tugas ke partisipan-partisipan. Algoritma ini masih belum teruji efisiensinya.**

***Keywords*—Directed Acyclic Graph, Algoritma, Manajemen Proyek.**

I. Latar Belakang

Sebuah proyek biasanya tidak dapat dilakukan oleh seorang saja. Karena itu tidak jarang proyek-proyek besar memerlukan beberapa tim dan sistem manajemen. Kesalahan manajemen mengakibatkan proyek melewati batas waktu dan mengalami kegagalan. Diperlukan sebuah sistem manajemen proyek yang dapat menjadwalkan tugas-tugas kepada setiap partisipan sehingga waktu yang ada dapat digunakan seefisien mungkin.

Directed Acyclic Graph (DAG) adalah graf berarah yang tidak memiliki siklus. Salah satu sifat penting DAG adalah *Topological Sorting*, yaitu mengurutkan semua *vertex* sehingga semua *edge* berarah ke *vertex* setelahnya. Semua DAG memiliki setidaknya satu *topological sorting*.

Proyek dapat direpresentasikan sebagai rangkaian tugas yang bergantung pada tugas lainnya. Tugas tidak dapat bergantung pada hasil tugas itu sendiri, baik secara langsung maupun tidak langsung. Hubungan-hubungan antar tugas tersebut menghasilkan sebuah DAG. Tugas-tugas direpresentasikan sebagai *vertex* graf, dan hubungan dependensi tugas direpresentasikan sebagai *edge* dari presenden ke dependen.

Makalah ini membahas algoritma penjadwalan/*scheduling* proyek *multiparty* dengan memanfaatkan DAG dan *topological sorting*.

II. Landasan Teori

Acyclic graph adalah graph tanpa siklus yang ketika diikuti dari node ke node, maka tidak akan melewati node yang sama dua kali [1]. Sedangkan Directed Acyclic Graph merupakan acyclic graph yang mempunyai arah [1,2].

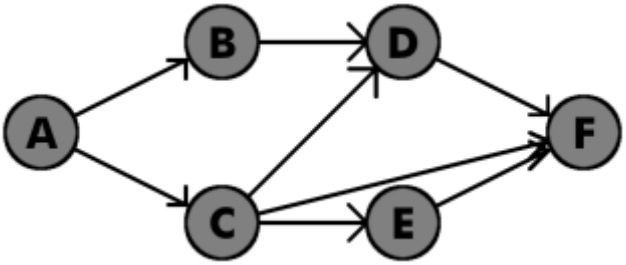
Sebuah *Directed Graph* $G = (V, E)$ terdiri dari sebuah *set* tidak kosong dari node-node V dan sebuah *set edge* berarah E. Setiap *edge e* dari E dispesifikan oleh pasangan terurut *vertices* $u, v \in V$. Sebuah *directed graph* adalah *simple* jika ia tidak memiliki *loop* (yaitu *edge* berbentuk $u \rightarrow u$) dan tidak memiliki *edge* ganda. Sebuah *directed graph* disebut *directed acyclic graph* (atau *DAG*) jika ia tidak memiliki siklus berarah. [3]

III. Pembahasan

A. Representasi DAG

Proyek direpresentasikan sebagai DAG, dengan tugas-tugas sebagai *vertex*, dan dependensi sebagai *edge*. Tidak diperbolehkan adanya *circular dependency* di dalam graf, karena hal ini akan merusak sifat DAG.

Contoh DAG dapat dilihat pada gambar berikut:



Gambar 1: Sebuah DAG.

Terlihat tugas A tidak membutuhkan hasil dari tugas manapun. Tugas B dan C membutuhkan tugas A. Tugas D membutuhkan tugas B dan C, sedangkan tugas E hanya membutuhkan tugas C. Tugas F membutuhkan tugas C, D, dan E.

B. Simbol

Algoritma yang akan disusun membutuhkan simbol-simbol berikut:

- $G = (V, E)$: Sebuah DAG yang mempunyai *set vertex* V dan *edge* E.
- $v \in V$: Sebuah *vertex* di dalam graf yang merepresentasikan sebuah tugas. v memiliki atribut lama tugas, berupa sebuah bilangan riil positif.
- $(v_1, v_2) \in E$: Sebuah *edge* di dalam graf yang merepresentasikan dependensi. Panah bergerak dari v_1 menuju v_2. v_1 disebut presenden v_2, dan v_2 disebut desenden v_1.

- H : *Heap* maksimum terurut lama tugas bertipe *vertex*. *Heap* mempunyai dua operasi, *push* dan *pop*. *Push* menambahkan *vertex-vertex* baru. *Pop* menghapus *vertex* yang memiliki lama tugas terlama dari *heap*.
- P : *List* partisipan yang ada. Setiap partisipan p_i mempunyai *timeline* tugas-tugas yang akan dikerjakan. *Timeline* tidak harus kontinyu, dan dapat mempunyai jeda waktu dimana tidak ada tugas di selang tersebut.

C. Algoritma

Langkah-langkah algoritma diberikan sebagai berikut:

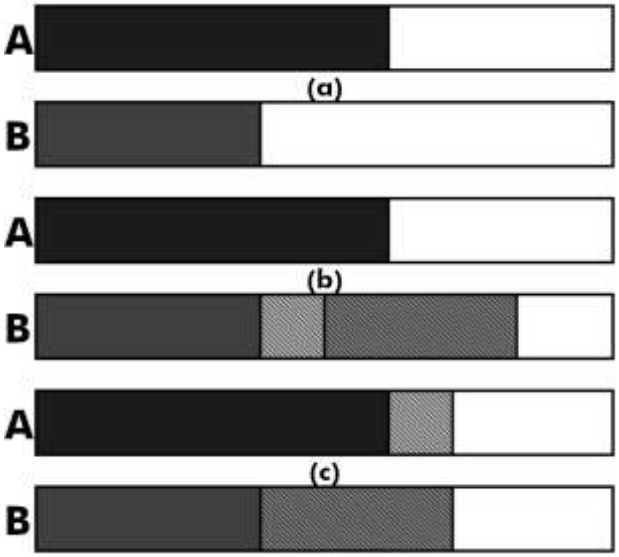
1. Cari semua *vertex* $v \in V$ yang tidak mempunyai presenden ($\forall v_i[(v_i, v) \notin E]$). *Push vertex-vertex* tersebut ke *heap* H.
2. *Pop vertex* v dari H. Dari definisi H diketahui v membutuhkan waktu terlama.
3. Cari semua desenden v_2 yang hanya bergantung pada v ($\forall v_i[(v_i, v_2) \in E \leftrightarrow v_i = v]$). *Push* v_2 ke H.
4. Hapus v dari G.
5. Cari satu partisipan p_i yang selesai paling awal atau tidak memiliki tugas sama sekali.
6. Tambahkan tugas didalam v ke akhir *timeline* p_i. Tambahkan jeda waktu bila diperlukan.
7. Ulangi langkah 2-6 hingga semua tugas berhasil dijadwalkan.

Contoh *pseudocode* algoritma adalah sebagai berikut:

```
procedure Schedule (
  input G : Graph, { DAG yang berisi
hubungan tugas-tugas }
  input/output P : list of
Participant, { Daftar partisipan }
)

KAMUS LOKAL
  H : Heap { Heap/priority queue yang
berisi tugas-tugas }

  V : Vertex { Sebuah tugas }

  Pi : Participant { Sebuah partisipan
}

ALGORTIMA
  CreateEmpty(H) { Inisialisasi heap }

  { Selama ada vertex di G }
  while not IsEmpty(G) do
    { Cari semua vertex di G }
    V traversal G
      if HasNoPrecendent(G,V) then
        { Jika vertex tidak memiliki
presenden, tambahkan ke heap }
       Push(H,V)

    Pop(H,V) { Ambil satu vertex }

    { Hapus dari graph }
    DeleteNode(G,V)

    { Cari partisipan yang selesai
paling awal }
    FindEarliestParticipant(P,Pi)
    AddTask(P,V) { Tambahkan tugas ke
partisispan }
```

Ketika mengimplementasikan algoritma, perlu diperhatikan kesalahan-kesalahan yang dapat terjadi:

- *Heap* bukan *heap* maksimum.
  Urutan penambahan tugas dimulai dari tugas terlama. Efek urutan penambahan dapat dilihat di gambar 2 berikut:

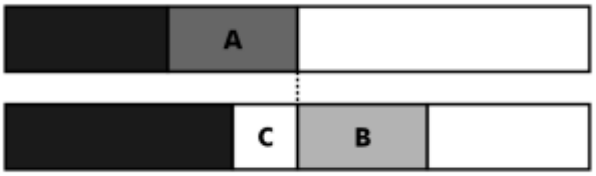


Gambar 2: (a) Sebelum penambahan. (b) Tugas yang lebih singkat ditambahkan lebih dahulu. (c) Tugas yang lebih lama ditambahkan lebih dahulu.

Perhatikan pada bagian (b) kedua tugas yang diarsir ditambahkan ke peserta B, sedangkan pada bagian (c) dibagi merata antara peserta A dan B. Jelas terlihat bahwa (c) selesai lebih cepat dibandingkan (b).

- Tidak memperhatikan *Most Recent Precendent Finished* (MRPF).

  Ketika penambahan tugas ke peserta, perlu diperhatikan kapan semua presenden selesai. Untuk lebih jelasnya dapat dilihat di gambar 3 berikut:



Gambar 3

Karena tugas B dependen tugas A, tugas B perlu digeser dan ditambahkan jeda waktu C. MRPF tugas B adalah ketika tugas A selesai dikerjakan. Perhatikan tugas B dapat ditambahkan setelah tugas A. Tentunya hal ini lebih efisien, karena tidak diperlukan jeda C lagi.

D. Efisiensi

Algoritma diatas dapat ditingkatkan efisiensinya dengan cara menyelipkan tugas pada jeda waktu yang pas atau lebih longgar sebelum menambahkan di akhir. Tentu saja penyelipan tidak boleh berada dibelakang MRPF.

Jika hanya ada satu peserta, algoritma ini ekivalen dengan *topological sorting*. Tentunya *topological sorting* adalah metode *scheduling* yang paling efisien. *Topological sorting* tidak dapat digeneralisasi untuk dua peserta atau lebih.

Tidak diketahui apakah terdapat algoritma yang lebih efisien. Untuk kasus semua tugas membutuhkan waktu yang sama, algoritma ini (setelah dimodifikasi) adalah algoritma yang paling efisien.

E. Kelebihan dan Kekurangan

Kelebihan algoritma ini adalah langkah-langkahnya sangat sederhana, dengan *library* yang cukup, dapat diimplementasikan dengan cepat. Algoritma ini dapat diimplementasikan di *hardware*, dengan sedikit perubahan, seperti mengganti *heap* dengan *list*, simplifikasi representasi graf, dll.

Algoritma ini dapat dimodifikasi menjadi mode *online/streaming*, memberikan tugas kepada partisipan setiap kali ada tugas baru yang muncul. Hal ini sangat berguna untuk kegunaan tertentu.

Kekurangan algoritma ini adalah tidak memperhatikan hubungan antar tugas yang lebih kompleks. Misalnya jika serangkaian tugas lebih cepat dikerjakan oleh partisipan yang sama atau beberapa partisipan sekaligus.

Algoritma ini mengasumsikan semua partisipan dapat mengerjakan semua tugas. Kadangkala ada partisipan yang tidak mampu mengerjakan tugas tertentu, atau membutuhkan waktu lebih lama. Algoritma dapat dimodifikasi untuk memperhitungkan perbedaan kemampuan partisipan.

E. Kegunaan

Algoritma ini dapat digunakan untuk membagi tugas-tugas dalam sebuah proyek kepada beberapa partisipan.

Algoritma ini juga dapat digunakan oleh *compiler/interpreter* untuk menata ulang kode dan membagi-bagi tugas ke beberapa *threads*. Kode akan dipotong menjadi sekuen-sekuen subkode dan dependensi data direpresentasikan sebagai *edge* yang menghubungkan potongan-potongan tersebut. DAG hasilnya kemudian dapat dimasukkan ke algoritma ini, dengan partisipan adalah *worker threads*.

Server dapat menggunakan algoritma ini untuk membagi-bagi *request* ke semua mesin yang tersedia. Lama tugas tidak perlu eksak, cukup diestimasi.

Operating System dapat menggunakan algoritma ini untuk membagi *processes* ke *core* komputer. Sedikit modifikasi memungkinkan algoritma ini menangani prioritas dan kemampuan/*performance* setiap *core*.

IV. KESIMPULAN

Algoritma ini mampu menyelesaikan permasalahan *scheduling* tugas dengan beberapa partisipan/*multiparty*. Algoritma ini efisien untuk kasus satu peserta dan waktu pengerjaan yang sama. Modifikasi dapat membuat algoritma ini mampu menangani tugas-tugas yang berdatangan satu persatu. Modifikasi juga dapat menangani perbedaan kemampuan partisipan.

VII. ACKNOWLEDGMENT

Penulis ingin berterimakasih kepada ibu dan ayah, yang telah memberikan motivasi dan *support* sehingga penulis dapat menyelesaikan makalah ini. Penulis juga ingin berterimakasih kepada Bu Ulfa, yang telah mengajarkan penulis sebagian teori graf dan menjawab berbagai pertanyaan dari penulis.

REFERENCES

[1] https://www.statisticshowto.com/directed-acyclic-graph/ diakses pada 07 Desember 2020

[2] https://www.techopedia.com/definition/5739/directed-acyclic-graph-dag diakses pada 07 Desember 2020

[3] https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-042j-mathematics-for-computer-science-fall-2010/readings/MIT6_042JF10_chap06.pdf diakses pada 09 Desember 2020

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, 3 Desember 2020

Hanif Arroisi Mukhlis (13519072)