TOPOLOGICAL SORT
dengan DFS dan METODE LAIN

Ferry Pangaribuan **13505080**

Teknik Informatika Institut Teknologi Bandung
Alamat Jl. Kyai Luhur No 10
e-mail: if15080@students.if.itb.ac.id

ABSTRAK

Di dalam teori graf, sebuah *topological sort* dari sebuah graf tidak sekuler yang terhubung (*directed acyclic graph* (DAG)) adalah sebuah urutan linier dari simpul-simpulnya yang cocok dengan urutan parsial R yang disebabkan pada simpul-simpulnya di mana x mendahului y (xRy) jika ada sebuah jalan terhubung dari x ke y di dalam DAG. Sebuah definisi yang sama bahwa setiap simpul mendahului semua simpul yang mempunyai sisi-sisi. Setiap DAG paling sedikit memiliki satu *topological sort*, dan mungkin banyak

Kata kunci: DAG, simpul, sisi..

1. PENDAHULUAN

Sebuah *topological sort* dari sebuah graf tidak serkuler yang terhubung adalah sebuah pengurutan pada simpul-simpulnya seperti semua sisi datang dari kiri ke kanan. Hanya sebuah graf tidak sirkuler dapat memiliki sebuah *topological sort,* karena sebuah graf sirkuler yang terhubung pada akhirnya harus kembali ke awal ke simpul pertama sirkuler tersebut. Namun, semua DAG memiliki paling sedikit sebuah topological sort dan kita dapat menggunakan DFS untuk menemukan sebuah pengurutan.

DFS dapat digunkan untuk menguji apakah sebuah graf adalah DAG, dan jika ya maka ditemukan sebuah *topological sort* untuk graf tersebut.

Sebuah graf terhubung adalah sebuah DAG jika dan hanya jika tidak ada sisi untuk kembali ditemui selama DFS.

Pelabelan setiap simpul di dalam urutan yang terbalik bahwa mereka ditandai *completely-explored* menemukan sebuah topological sort dari sebuah DAG. Pertimbangkan apa yang terjadi pada setiap sisi yang terhubung {u,→v} selama proses eksplorasi simpul *u*

- Jika v sekarang ini tidak ditemukan, kemudian kita memulai suatu DFS dari v sebelum kita dapat melanjutkan dengan u. Dengan begitu v ditandai *completely-explored* sebelum u, dan v terlihat sebelum u di dalam urutan topological.
- Jika v ditemukan tetai tidak *completely-explored,* kemudian {u,→v} adalah sisi kembali, yang dilarang di dalam DAG.
- Jika v adalah *completely-explored,* kemudian ini akan dilabelkan sebelum u. Oleh karana itu, u kelihatan sebelum v dalam sebuah urutan topological.

Tiga fakta penting tentang *topological sorting :*

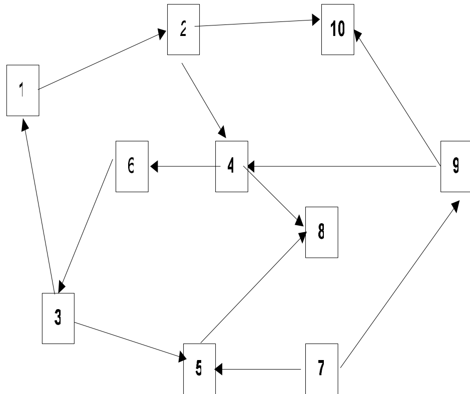
- Hanya graf tidak sirkuler yang terhubung dapat memiliki perluasan linier, karena beberapa sirkuler terhubung adalah sebuah kontradiksi yang melekat pada sebuah urutan linier dari tugas-tugas.
- Setiap DAG dapat menjadi topologically sorted, jadi harus selalu paling sedikit satu jadwal untuk untuk beberapa hal yang beralasan membatasi seluruh pekerjaan.
- DAG khusus memeberikan beberapa jadwal, khususnya ketika ada beberapa kendala. Pertimbangkan n buah pekerjaan tanpa kendala. N! permutasi dari pekerjaan tersebut merupakan sebuah perluasan linier yang valid.

Contoh kererurutan parsial banyak dijumpai dalam kehidupan sehari-hari, misalnya:

1. dalam suatu kurikulum, suatu mata pelajaran mempunyai prerequisit mata pelajaran lain. Urutan linier adalah urutan untuk seluruh mata pelajaran dalam kurikulum.
2. dalam suatu proyek, suatu pekerjaan harus dikerjakan lebih dalu dari pekerjaan lain (misalnya membuat pondasi harus sebelum dinding, membuat dinding harus sebelum pintu. Namun pintu dapat dikerjakan bersamaan dengan jendela)
3. dalam sebuah program Pascal, pemanggilan prosedur harus sedemikian rupa, sehingga pelerakkan prosedur pada teks program harus sesuai dengan urutan (parsial) pemanggilan.

Dalam pembuatan tabel pada basis data, tabel yang di-refer oleh tabel lain harus dideklarasikan terlebih dahulu. Jika suatu aplikasi terdiri dari banyak tabel, maka urutan pembuatan tabel harus sesuai dengan definisinya.

Jika X<Y adalah simbol untuk X "sebelum" Y, dan keterurutan parsial digembarkan sebgai graf, maka graf senagai berikut :

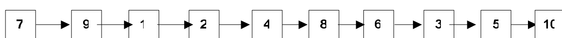


Gambar 1. Contoh topological sort

akan dikatakan mempunyai keterurutan parsial

1<2	2<4	4<6	2<10	4<8	6<3
1<3	3<5	5<8	7<5	9<4	9<10

salah satu urutan linier adalah graf sebagai berikut:



Gambar 2. Hasil keterurutan gambar di atas

Proses yang dilakukan untuk mendapatkan urutan linier:

1. Andaikan item yang mempunyai keterurutan parsial adalah anggota himpunan S.
2. Plih salah satu item yang tidak mempunyai predesesor, misalnya X. Minimal ada satu elemen seperti ini. Jika tidak maka akan looping.
3. Hapus X dari himpunan S, dan inser ke dalam list.
4. Sisa himpunan S masih merupakan himpunan terurut parsial, maka proses 2-3 dapat dilakukan lagi terhadap sisa dari S.
5. Lakukan sampai s menjadi kosong, dan list. Hasil mempunyai elemen dengan keterurutan linier.

2. METODE

Metode yang akan dibandingkan pada saat ini untuk memecahkan masalah *topological sort* ada tiga jenis: pendekatan secara struktur data penyelesaian, pendekatan fungsional dengan list linier sederhana, dan dengan DFS.

Pseudo code secara umum dalam menyelesai kan pesoalan topological sort

```
Algoritma TopologicalSort(G)
      H ← G// copy G ke H
      n ← G.numVertices()
      while H tidak kosong  do
      v ← temukan simpul dengan
tidak ada sisi yang outgoing
      Label v ← n
      n ← n - 1
      Hapus v dari H
```

2.1 Pendekatan Struktur Data

Untuk melakukan hal ini, perlu ditentukan suatu representasi internal. Operasi yang penting adalah memillih elemen tanpa predesesor (yaitu junlah predesesot elemen sama dengan nol). Maka setiap elemen memiliki 3 karakteristik: identifikasu, list suksesornya, dan banyaknya predesesor. Karena jumlah elemen bervariasi, maka representasi yang paling cocok adalah list berkait dengan representasi dinamis (pointer). List dari suksesor direpresentasi secara berkait. Ilustrasi struktur data dan program dalam bahasa Pascal diberikan sebgai berikut,

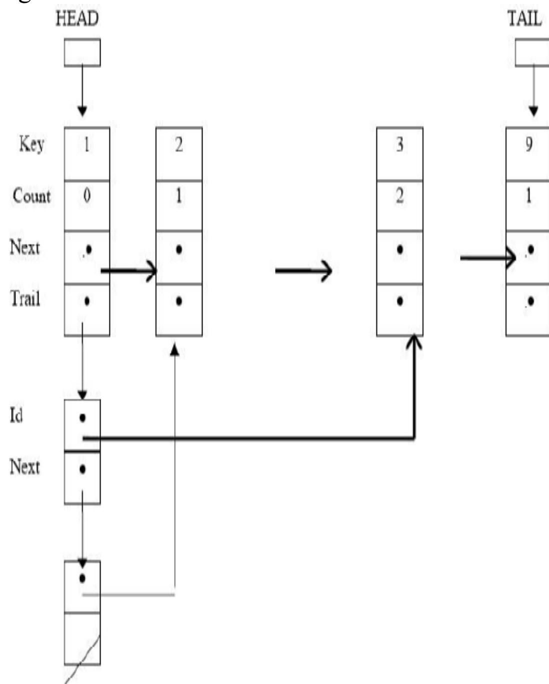
Representasi yang dipilih untuk persoalan ini adalah multilist sebagai berikut:

1. List yang digambarkan horisantal adalah list dari banyaknya predesesor setiap item, disebut list *"Leader",* yang direpresentasi sebagai list yang dicatat alamat elemen oertama dan terakhir *(Head-Tail)* serta elemen terurut menurut key, List ini dibentuk dari pembacaan data. Untuk stiap data keterurutan parsial X<Y. Jika X dan/atau Y belum ada pada list *leader, insert* pada Tail dengan metoda *search* dengan *sentinel*.
2. List yang digambarkan vertikal (ke bawah) adalah list yang merupakan indirect addressing ke stiap predesesor, disebut sebagai *"Trailer*". Untuk setiap elemen list elemennya berisi alamat dari suksesor. Penyisipan data suatu suksesor (X<Y), dengan diketahui X, maka akan dilakukan dengan InsertFirst alamat Y sebagai elemen list Trailer dangna key X.

Algoritma secara keseluruhan terdiri dari dua pass:

1. Bentuk list *leader* dan *Trailer* dari data keterurutan parsial: baca pasangan nilai (X<Y). Tentukan alamat X dan Y (jika belum ada sisipkan), kemudian dengan mengetahui alamat X dan Y pada list *Leader, InsertFirst* alamat Y sebagai *trailer* X.
2. Lakukan *topological sort* dengan melakukan search list *leader* dengan jumlah predesesor=0, kemudian insert sebagai elemen list linier hasil pengurutan.

Ilustrasi umum dari list *Leader* dan *Trailer* untuk representasi internal persoalan topological sorting adalah sebagai berikut:



Gambar 3. Struktur data internal topological sort

2.2 Pendekatan "Fungsional" dengan List Linier Sederhana

Pada solusi ini, proses untuk mendapatkan urutan linier diterjemahkan secara fungsional, dengan representasi sederhana. Graf parsial dinyatakan sebagai list linier dengan representasi fisik. *First-Last* dengan dummy seperti representasi pada solusi sebelumnya, dengan eleneb yang terdiei dari <*Precc,Succ*>. Contoh sebuah elemen bernilai <1,2> artinya predesesor dari 2.

Langkah penyelesaian:

1. Fase input: Bentuk list linier yang merepresentasi graf
2. Fase output: Ulangi langkah berikut sampai list "habis", artinya semua elemen list selesai ditulis sesuai dengan urutan total.
   - P adalah elemen pertama *(First(L))*
   - Search pada sisa list, apakah *X=Precc(P)* mempunyai predesesor.
     - Jika ya, maka elemen in harus dipertahankan sampai saatnya dapat dihapus dari list untuk di-*output*-kan
       a. *Delete* P, tapi jangan didealokasi
       b. *Insert* P sebagai *Last(L)* yang baru
     - Jika tidak mempunyai predesesor, maka X siap untuk dioutputkan, tetapi Y masih harus dipertanyakan. Maka langkah yang harus dilakukan :
       a. Outputkan X
       b. Search apakah Y masih ada sisa list, baik sebagai Precc maupun sebagai Succ
          i. Jika ya, maka Y akan dioutputkan nanti. Hapus eleman pertama yang sedang diproses dari list.
          ii. Jika ridak muncul sama sekali berarti Y tidak mempunyai predesesor. Maka outputkan Y, baru hapus elemen pertama dari list.

2.3 DFS

Algoritma DFS secara umum:

1. Kunjungi simpul v,
2. Kunjungi simpul w yang bertetangga dengan simpul v.
3. Ulangi DFS mulai dari simpul w.
4. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
5. Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Skema algoritma DFS adalah algoritma rekursif sebagai berikut

```
procedure DFS(input v:integer)
{Mengunjungi seluruh simpul graf dengan
algoritma pencarian DFS

Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpulyang dikunjungi ditulis
ke layar
}
Deklarasi
  w : integer
Algoritma:
  write(v)
  dikunjungi[v]←true
  for tiap simpul w yang bertetangga dengan
simpul v do
    if not dikunjungi[w] then
       DFS(w)
    endif
  endfor
```

Algoritma DFS selengkapnya adalah:

```
procedure DFS(input v:integer)
{Mengunjungi seluruh simpul graf dengan
algoritma pencarian DFS

Masukan: v adalah simpul awal kunjungan
Keluaran: semua simpulyang dikunjungi ditulis
ke layar
}
Deklarasi
  w : integer

Algoritma:
  write(v)
  dikunjungi[v]←true
  for w←l to n do
    if A[v,w]=l then {simpul v dan simpul w
bertetangga }
        if not dikunjungi[w] then
          DFS(w)
        endif
    endif
  endfor
```

Pencarian solusi dengan metode DFS pada pohon ruang status digambarkan dengan algoritma ini:

1. Masukkan simpul akar ke dalam antrian Q. Jika simpul akar = simpul solusi, maka Stop.
2. Jika Q kosong, tidak ada solusi. Stop.
3. Ambil simpul v dari kepala (head) antrian. Jika kedalaman simpul v sama dengan batas kedalaman maksimum, kembali ke langkah
4. Bangkitkan semua anak dari simpul v. Jika v tidak mempunyai anak lagi, kembali ke langkah 2. Tempatkan semua anak dari v di awal antrian Q. Jika anak dari simpul v adalah simpul tujuan, berarti solusi telah ditemukan, kalau tidak, kembali lagi ke langkah 2.

Kompleksitas waktu algoritma DFS pada kasus terburuk adalah O(bm). Kompleksitas ruang algoritma DFS adalah O(bm), karena kita hanya hanya perlu menyimpan satu buah lintasan tunggal dari akar sampai daun, ditambah dengan simpul-simpul saudara kandungnya yang belum dikembangkan.

Kesulitan utama pada metode DFS adalah menentukan batas maksimum kedalaman pohon ruang status.Strategi yang digunakan untuk memecahkan masalah kedalaman terbaik ini adalah dengan mencoba semua kedalaman yang mungkin, mula-mula kedalaman 0, kedalaman 1, kedalaman 2, dan seterusnya.

Sebuah topological sort dapat diimplementasikan dalam O(n + m). Ini berlaku untuk beberapa varian dari DFS.

```
Algoritma topologicalDFS(G)
      Input dag G
      Output topological ordering of G
    n ← G.numVertices()
for all  u ∈ G.vertices()
        setLabel(u, UNVISITED)
for all  e ∈ G.edges()
        setLabel(e, UNEXPLORED)
for all  v ∈ G.vertices()
       if  getLabel(v) = UNEXPLORED
           topologicalDFS(G, v)
```

```
Algoritma topologicalDFS(G, v)
Input graph G and a start vertex v of G
      Output labeling of the vertices
of G
            in the connected component
of v
      setLabel(v, VISITED)
for all  e ∈ G.incidentEdges(v)
      if  getLabel(e) = UNEXPLORED
            w ← opposite(v,e)
            if getLabel(w) = UNVISITED
                  setLabel(e,
DISCOVERY)
                  topologicalDFS(G, w)
            else
                  {e is a forward or
cross edge}
    Set the topological order of v to n
    n ← n - 1          //  akhir  dari
loop
```

3. KESIMPULAN

Hasil analisis dari ketiga metode di atas:

1. Kompleksitas algoritma DFS untuk menyelesaikan masalah topological sort
$$\sum_{i=1}^{n} m^i n = O(\deg(v_i)) = O(n+m)$$
2. Kompleksitas algoritma yang diperlukan untuk menyelesaikan pendekatan secara struktur data memerlukan dua kali pass sehingga waktu yang diperlukan kuadrat dari jumlah simpul yang ada $O(n^2)$
3. Kompleksitas algoritma dengan pendekatan fungsional hampir sama dengan DFS yaitu $O(n+m)$

Kesimpulan dari ketiga metode di atas adalah bahwa algoritma penyelesaian topological sort dengan DFS memiliki kompleksitas waktu yang lebih baik dari kedua jenis pendekatan atau algoritma yang lain.

REFERENSI

[1] Munir, Rinaldi. 2006. *Diktat Kuliah IF2251 Strategi Algoritmik*. Bandung : Penerbit ITB.

[2] Skiena, Steven S. *The Algorithm Design Manual*. New York: Department of Computer Science State University of New York

[3] Liem, Inggriani. *Struktur Data*. Bandung : Penerbit ITB.