Penggunaan Tarjan's Algorithm untuk Mencari Komponen Terhubung Kuat pada Graf Berarah

Zachary Samuel Tobing – 13522016[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522016@itb.ac.id

***Abstract*—Tarjan's Algorithm adalah sebuah algoritma yang digunakan untuk menentukan komponen-komponen yang terhubung kuat dalam graf berarah. Algoritma sederhana ini dapat digunakan untuk kemudian diaplikasikan dalam beberapa bidang lain seperti *database*, optimisasi *compiler*, rekayasa perangkat lunak, dan analisis jaringan. Makalah ini menganalisis Tarjan's Algorithm dan menjelaskan berbagai penggunaannya dalam masalah dunia nyata.**

***Keywords*—Komponen terhubung kuat, graf berarah, Tarjan, algoritma.**

I. PENDAHULUAN

Tarjan's Algorithm yang dibuat oleh Robert Tarjan, adalah sebuah algoritma graf untuk mencari komponen-komponen terhubung kuat (*Strongly Connected Components*) dalam sebuah graf berarah.

Graf berarah adalah sebuah graf (struktur data linier yang berfokus pada hubungan) yang memiliki arah, yaitu hubungan spesifik antar simpul, sehingga tidak hanya sisi yang menyambungkan kedua simpul sekaligus, tetapi memiliki hubungan yang lebih dalam dan ditekankan.

Graf berarah yang menyatakan hubungan antar simpul dapat digunakan lebih lagi dengan konsep SCC dan algoritma Tarjan untuk memperoleh hubungan-hubungan kuat dari simpul-simpul dalam sebuah graf.

Komponen-komponen terhubung kuat adalah sebuah subgraf dari graf berarah yang terdiri dari kumpulan simpul sedemikian rupa sehingga untuk setiap pasang node pada subgraf terdapat lintasan.

Komponen-komponen terhubung kuat dapat diterapkan prinsipnya dalam berbagai ilmu pengetahuan, seperti *database*, jaringan, dan optimisasi *compiler*.

II. LANDASAN TEORI

A. Graf

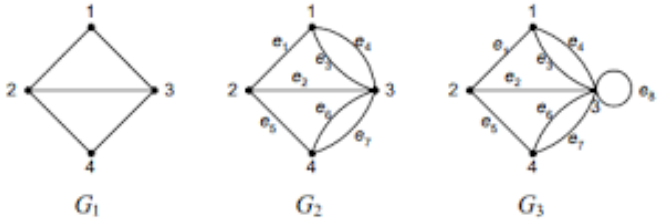
Graf adalah sebuah tipe data yang digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Graf terdiri dari:

V = himpunan tidak-kosong dari simpul (*vertices*)
Contoh: $\{v_1,v_2,\ldots,v_n\}$

E = himpunan sisi (*edges*) yang menghubungkan sepasang simpul
Contoh: $\{e_1,e_2,\ldots,e_n\}$

Graf dapat digolongkan berdasarkan ada tidaknya gelang atau sisi ganda:

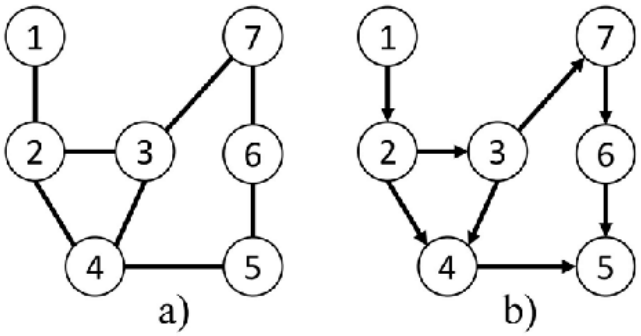
1. Graf sederhana (*simple graph*)
   Graf yang tidak mengandung gelang maupun sisi ganda
2. Graf tak-sederhana
   Graf yang memiliki sisi ganda atau gelang. Jenis ini juga dapat dibagi lagi menjadi graf ganda yaitu sebuah graf yang mengandung sisi ganda (lebih dari satu sisi) antar simpul, dan graf semu yang mengandung simpul dengan sisi gelang dengan dirinya sendiri.



Gambar 2.1. Ilustrasi Graf Sederhana, Ganda, Semu
(Sumber: Discrete Mathematics: Definition and Types of Graphs, Study Statistics)

Graf juga dapat digolongkan berdasarkan orientasi arah pada sisinya menjadi:

1. Graf tak-berarah (*undirected graph*)
   Graf yang sisinya tidak mempunyai orientasi arah (menunjukkan hubungan antar simpul berlaku 2 arah).
2. Graf berarah (*directed graph* atau *digraph*)
   Graf yang sisinya diberikan orientasi arah (menunjukkan hubungan antar simpul yang spesifik pada arah panahnya).



Gambar 2.2 Ilustrasi Graf Tak Berarah dan Graf Berarah
(Sumber: ResearchGate)

Ada juga beberapa terminologi yang sering dipakai dalam graf yaitu:

1. Ketetanggaan (*Adjacent*)
   Dua simpul bertetangga jika terhubung langsung oleh sebuah sisi.
2. Bersisian (*Incidency*)
   Sebuah sisi e = (v_j,v_k) bersisian dengan simpul v_j dan bersisian dengan simpul v_k.
3. Simpul terpencil (*Isolated Vertex*)
   Simpul terpencil adalah simpul yang tidak mempunyai sisi yang bersisian dengannya.
4. Graf kosong (*null graph* atau *empty graph*)
   Graf yang himpunan sisinya merupakan himpunan kosong (N_n), hanya terdiri dari beberapa simpul yang tidak memiliki sisi.
5. Derajat (*Degree*)
   Derajat suatu simpul adalah jumlah sisi yang bersisian dengan simpul tersebut (d(v)), bernilai 0 jika berupa simpul terpencil.
6. Lintasan (*path*)
   Lintasan adalah serangkaian simpul yang berurutan dihubungkan oleh sisi, sebuah jalur atau urutan yang menghubungkan sebuah simpul awal v_0 ke sebuah simpul tujuan v_n dalam graf G dan akan berselang-seling simpul dan sisi. Sebuah lintasan memiliki satuan panjang, yaitu jumlah sisi dalam lintasan tersebut.
7. Siklus (*Cycle*) atau Sirkuit (*Circuit*)
   Sirkuit adalah sebuah lintasan yang berawal dan berakhir pada simpul yang sama. Sebuah sirkuit memiliki satuan panjang yaitu jumlah sisi dalam sirkuit, serupa dengan panjang lintasan.
8. Keterhubungan (*Connected*)
   Dua buah simpul terhubung jika terdapat lintasan dari simpul pertama ke simpul kedua. Sebuah graf dikatakan graf terhubung (*connected graph*) jika setiap pasang simpul dalam himpunan V memiliki lintasan yang menghubungkannya.
9. Graf Terhubung Kuat (*Strongly Connected Graph*)
   Graf disebut terhubung kuat (*strongly connected*) jika terdapat lintasan berarah dari sebuah simpul u ke simpul v dan lintasan berarah dari v ke u, sehingga khusus hanya untuk graf berarah.
10. Graf Terhubung Lemah (*Weakly Connected Graph*)
    Graf disebut terhubung lemah (*weakly connected*) apabila tidak semua pasang simpul sembarang u dan v di G memiliki lintasan dari u ke v dan dari v ke u.
11. Komponen Graf (*connected component*)
    Komponen graf adalah himpunan simpul yang dapat mencapai satu sama lain melalui tepian, khusus pada graf tak terhubung.
12. Graf Berbobot (*Weighted Graph*)
    Graf berbobot adalah graf yang setiap sisinya diberi sebuah harga (bobot).
13. Directed Acyclic Graph (DAG)
    DAG adalah sebuah graf berarah yang sesuai namanya, tidak memiliki siklus. Graf ini unik karena akan menampilkan urutan dari graf berarah sehingga akan lebih mudah membaca dan memahami alur dalam graf.

Dalam graf juga ada algoritma pencarian yang biasa digunakan untuk mencari simpul tertentu yaitu:

1. BFS (*Breadth First Search*)

   BFS adalah algoritma yang melakukan pencarian secara melebar, mengunjungi simpul secara *preorder* (akar, anak kiri, anak kanan) kemudian mengunjungi semua simpul yang bertetangga dengan simpul tersebut terlebih dahulu, dilakukan hingga ditemukan simpul yang sesuai.

   Algoritma ini memerlukan sebuah antrian (*queue*) untuk menyimpan data simpul yang telah dikunjungi, mengetahui data simpul yang bertetanggaan, dan menandakan selesainya pencarian, yaitu ketika simpul ditemukan atau semua simpul sudah dikunjungi.

   Cara kerjanya adalah:
   - Masukkan simpul dalam *queue*.
   - Bandingkan nilai simpul dengan nilai simpul yang dicari.
   - Jika sesuai, pencarian akan selesai dan hasilnya dikembalikan.
   - Jika belum sesuai, masukkan semua simpul bertetangga dengan simpul yang sedang kita periksa dan masukkan simpul anak dalam antrian.
   - Pencarian juga dapat selesai apabila antrian kosong dan setiap simpul sudah dicek.
   - Ulangi pencarian dari langkah kedua.

2. DFS (*Depth First Search*)

   DFS adalah algoritma penelusuran dalam graf berdasarkan kedalamannya. Simpul akan ditelusuri dari akarnya ke salah satu simpul anaknya seterusnya sesuai prioritas dari simpul anak pertama hingga mencapai simpul terdalam (tidak memiliki anak).

   DFS dilakukan secara rekursif atau dengan struktur data *stack* untuk mencatat simpul-simpul sebelumnya yang memiliki anak selain lintasan yang telah dipilih dalam algoritma.
   Cara kerja DFS menggunakan struktur data *stack* adalah:

- Masukkan simpul akar dalam *stack.*
- Simpan isi elemen pada tumpukan teratas dengan *push* sesuai struktur *stack.*
- Hapus isi *stack* paling atas dengan *pop* sesuai ciri-ciri struktur data *stack*.
- Periksa simpul pohon yang disimpan untuk anak simpul selain lintasan yang telah dipilih.
- Jika memiliki anak simpul lain, *push* semua anak simpul dalam *stack*.
- Jika *stack* kosong maka proses akan berhenti, dan kembali ke langkah kedua jika nilai belum sesuai atau *stack* belum kosong.

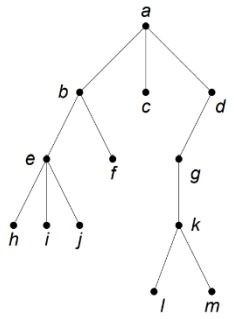
BFS akan mencari jalur terpendek dari simpul awal ke simpul dengan nilai yang sesuai jika bukan graf berbobot. Kerugiannya memerlukan lebih banyak memori karena harus menyimpan informasi lapisan pertama sebelum melanjutkan lapisan berikutnya.

DFS tidak menjamin bahwa akan menemukan jalur terpendek pada simpul dengan nilai yang sesuai, tetapi lebih menguntungkan karena akan memerlukan memori yang lebih sedikit, hanya menyimpan informasi tentang jalur yang sedang diikuti.

B. Struktur Data

Pohon adalah graf tak-berarah terhubung yang tidak mengandung sirkuit. Pohon berakar adalah pohon yang satu buah simpulnya diperlakukan sebagai akar dan sisi-sisinya diberi arah sehingga menjadi graf berarah.
Beberapa terminology yang digunakan dalam pohon berakar adalah:

- Anak (*child*) dan Orangtua (*parent*)
  e dan f adalah anak-anak simpul b, b adalah orangtua dari anak-anak itu.



Gambar 2.3 Ilustrasi Pohon
(Sumber: [4])

- Lintasan (*path*)
  Lintasan dari a ke f adalah a, b, f. Panjang lintasan dari a ke f adalah 2.
- Saudara kandung (*sibling*) h adalah saudara kandung i, tetapi k bukan saudara kandung h, karena orangtua mereka berbeda.

Stack adalah struktur data linier yang elemennya dimasukkan dan dihilangkan dari satu sisi, yaitu *top.* Stack mengikuti prinsip LIFO (*Last In First Out*) seperti tumpukan sehingga elemen yang terakhir dimasukkan adalah elemen yang pertama dapat dihilangkan atau diakses.

Dalam *stack* ada 2 operasi utama yang sering digunakan, yaitu *push* dan *pop.* Push adalah operasi dalam *stack* untuk memasukkan data di *top* dari *stack*, dan *pop* adalah operasi dalam *stack* untuk mengeluarkan atau menghilangkan data dari *top* stack. Ketika dilakukan *push,* elemen baru yang dimasukkan menjadi *top* dari *stack* dan ketika melakukan *pop,* elemen yang dikeluarkan tidak lagi menjadi *top, topnya* akan berpindah pada elemen di bawahnya.

Queue adalah struktur data linier juga yang serupa dengan *stack* memiliki pengecualian yaitu mengikuti prinsip FIFO (*First In First* Out), hanya dapat memasukkan elemen pada ujung daftar elemen (*tail*) dan menghilangkan atau mengakses elemen pada awal daftar elemen (*head*), seperti antrian.

Queue juga memiliki 2 operasi utama, yaitu *enqueue* dan *dequeue*. Proses *enqueue* akan memasukkan elemen pada ujung daftar sehingga menjadi *tail* baru, dan proses *dequeue* akan menghapus elemen dari awal daftar elemen dan *head* akan berpindah pada elemen setelahnya.

C. Algoritma Tarjan

Algoritma Tarjan adalah sebuah algoritma dalam teori graf yang digunakan untuk menemukan komponen terhubung kuat (*strongly connected components*) dalam sebuah graf terarah (*Directed Graph*). Algoritma ini dinamai dari Robert Tarjan, seorang ahli teori graf yang mengembangkannya pada tahun 1972.

Fokus utama dari algoritma Tarjan adalah untuk mengidentifikasi himpunan simpul-simpul yang membentuk komponen terhubung kuat dalam graf terarah. Komponen terhubung kuat adalah himpunan simpul-simpul yang saling terhubung di antara satu sama lain dan tidak dapat dijangkau dari simpul-simpul di luar himpunan tersebut.

Berikut adalah program singkat untuk algoritma Tarjan dalam Bahasa Python:

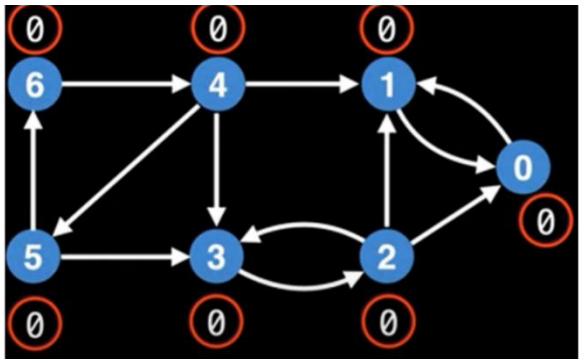
```python
class TarjanSCC:
    def __init__(self, graph):
        self.graph = graph
        self.visited = set()
        self.stack = []
        self.low = {}
        self.ids = {}
        self.scc_components = []
        self.current_id = 0
```

Gambar 2.4 Ilustrasi Struktur Data dalam Python
(Sumber: Dokumentasi Pribadi)

Dalam kebutuhan algoritma Tarjan, ada beberapa komponen dan variabel baru yang digunakan. Awalnya, ada *graph* yaitu graf yang akan ditelusuri.

Kemudian, ada sebuah *set* yang akan digunakan untuk memeriksa node yang sudah ditelusuri telah dikunjungi yang penting bagi algoritmanya.

Kemudian, ada juga sebuah *stack* yang digunakan untuk prosesnya dengan tujuan memastikan SCC nya tercatat dengan benar. Ini dikarenakan pada algoritma Tarjan, pemilihan node awal bersifat acak sehingga memungkinkan terjadinya salah pencatatan seperti di bawah ini.



Gambar 2.5 Ilustrasi Kasus Tanpa Stack
(Sumber: Tarjan's Algorithm For Strongly Connected Components, TopCoder)

Seharusnya dapat dibentuk beberapa SSC, tetapi karena dimulai dari yang salah, algoritmanya tidak akan berjalan dengan benar. Oleh karena itu, perlu adanya *stack* untuk memastikan semua SCC tercatat dan terpisah dengan benar.

Kemudian, ada sebuah nilai *low* pada setiap simpul. Low, disebut juga *low-link value* atau "nilai tarjan", adalah sebuah nilai yang dibuat pada algoritma Tarjan untuk mencatat sebuah nilai yang nantinya akan dibandingkan untuk menentukan awal dari sebuah SCC.

Kemudian, ada juga nilai *id* yang berupa nilai indeks yang *increment* dari 0 setiap kali node baru dikunjungi yang fungsinya untuk memberikan label urutan setiap simpul untuk dibandingkan dalam algoritma.

Terakhir, ada sebuah *array* yaitu *ssc_components*, yaitu sebuah *array* untuk daftar semua SSC yang ditemukan, yang pada algoritma di bawah ini akan disimpan dulu sebagai *array* lagi bernama *component* dan akan ditambahkan dalam *array ssc_components*.

Langkah-langkah utama algoritma Tarjan adalah sebagai berikut:

1. Inisialisasi
- Setiap simpul diberi nomor indeks (*id*) yang awalnya belum dikunjungi dan dimasukkan ke dalam tumpukan (stack).
- Setiap simpul juga diberi nilai yang disebut "nomor tarjan" untuk membantu mengidentifikasi komponen-komponen terhubung kuat berdasarkan urutannya.
2. Traversal DFS (Depth-First Search)
- Algoritma melakukan penelusuran graf menggunakan DFS.
- Saat DFS menemukan simpul, simpul tersebut diberi "nomor tarjan" yang sesuai dan dimasukkan ke dalam stack.
3. Menyusun Komponen Terhubung Kuat
- Selama traversal DFS, algoritma memeriksa apakah ada jalur mundur (*backtrack*) yang dapat ditemukan dari simpul yang sedang diproses ke simpul yang lebih awal.
- Jika jalur mundur ditemukan, simpul-simpul dalam jalur tersebut membentuk komponen terhubung kuat.
- Algoritma kemudian menandai simpul-simpul tersebut dengan mencari nilai *low-link* (nilai tarjan) terendah yang dapat dikunjungi dari simpul yang saat ini ditelusuri.
- Jika dilanjutkan ke belakang (*backtrack*) dan akhirnya menemukan simpul dengan *id* yang sama dengan *low-link value-nya*, maka akan terbentuk sebuah SCC (*Strongly Connected Component*) dengan semua simpul yang sebelumnya di-*pop* dari *stack*, dan simpul tadi menjadi awal dari SCC tersebut.
4. Pengeluaran Komponen Terhubung Kuat
- Setelah selesai traversal, komponen terhubung kuat akan terletak dalam *array scc_components*.
- Elemen-elemen dalam *array scc_components* berupa *array* lagi yang berisi urutan simpul-simpul dari setiap SCC yang ditemukan.

Dengan langkah-langkah tersebut, algoritma Tarjan dapat efisien mengidentifikasi dan menemukan semua komponen terhubung kuat dalam sebuah graf berarah.

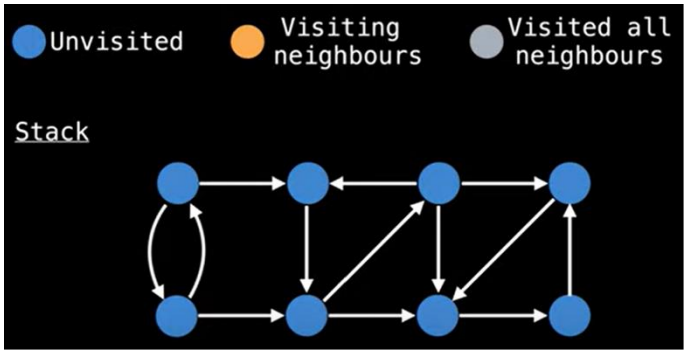
```python
def dfs(self, node):
    self.visited.add(node)
    self.low[node] = self.current_id
    self.ids[node] = self.current_id
    self.current_id += 1
    self.stack.append(node)

    for neighbor in self.graph[node]:
        if neighbor not in self.visited:
            self.dfs(neighbor)
            self.low[node] = min(self.low[node], self.low[neighbor])
        elif neighbor in self.stack:
            self.low[node] = min(self.low[node], self.ids[neighbor])

    if self.low[node] == self.ids[node]:
        component = []
        while True:
            top = self.stack.pop()
            component.append(top)
            if top == node:
                break
        self.scc_components.append(component)
```

Gambar 2.6 Ilustrasi Algoritma Tarjan dalam Bahasa Python
(Sumber: Dokumentasi Pribadi)

Dalam fungsinya itu sendiri, akan dilakukan prosesnya seperti langkah-langkah yang telah dijelaskan sebelumnya.

III. EKSPERIMEN

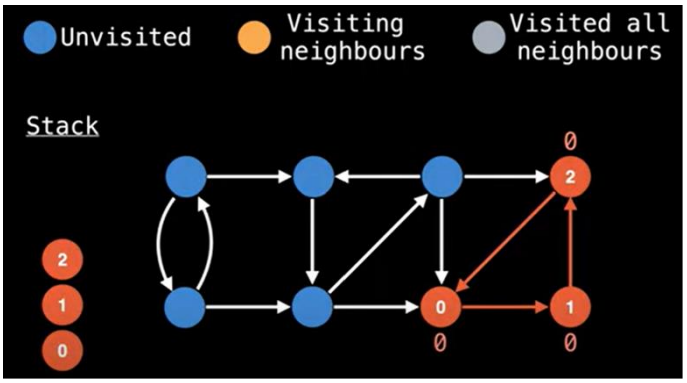
Diberikan contoh graf sebagai berikut:



Gambar 3.1 Ilustrasi Contoh Soal
(Sumber: WilliamFiset)

Awalnya, graf akan kosong dan tidak akan memiliki label atau komponen seperti yang sudah dijelaskan sebelumnya.

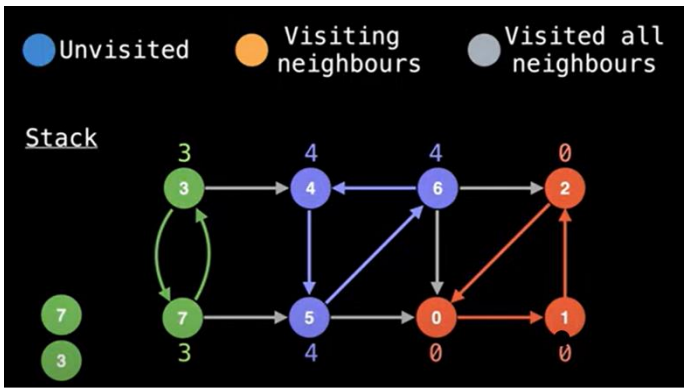
Kemudian, kita akan memulai inisialisasi seperti pada langkah di atas, yaitu memilih salah satu node dan melakukan DFS. DFS akan dilakukan untuk semua tetangga yang mungkin bagi setiap simpul yang dilalui, dengan juga memasukkan simpul-simpul pada *stack*.



Gambar 3.2 Ilustrasi Langkah DFS
(Sumber: WilliamFiset)

Setelah misalnya mendapatkan DFS seperti gambar di atas, nilai nol menunjukkan *low-link value* dari setiap node, yang seperti kode Python di atas, merupakan hasil fungsi minimum antara nilai awalnya ketika inisialisasi dengan *low-link value* simpul yang dapat dijangkaunya.

Dapat dilihat juga bahwa simpul dengan *id* 0 sama dengan *low-link value* nya, sehingga simpul itu adalah simpul awal dari SCC dan semua simpul berwarna merah adalah komponen-komponennya. Setelah mendapatkan SCC lengkap, masukkan nilai-nilai simpulnya pada *array* untuk menyimpan datanya dan kemudian *pop* semua nilai SCC. Setelah itu, pilih simpul lain lagi selain yang sudah dijumpai dan lakukan proses yang sama lagi.



Gambar 3.3 Ilustrasi Penyelesaian
(Sumber: WilliamFiset)

Proses ini terus dilakukan dengan memilih salah satu simpul lain setelah SCC pertama (warna merah) selesai dibentuk. Dengan terus melakukan proses DFS dan *pop* serta *push stack* akan diperoleh 3 SCC seperti gambar di atas.

IV. Analisis

Algoritma Tarjan sebenarnya adalah sebuah modifikasi dari algoritma DFS yang sebelumnya telah dijelaskan. Perbedaannya terletak pada pencatatan indeks dan komponen-komponen saja.

Kompleksitas algoritma Tarjan berupa O(V+E) yaitu kompleksitas linier dalam graf yang menunjukkan V sebagai jumlah simpul dan E sebagai jumlah sisi. Hal ini dapat dengan mudah diamati dengan melihat algoritma DFS yang berjalan dengan melihat semua node pada *worst case* sekali.

Selain algoritma Tarjan, ada juga algoritma pencarian SCC (*strongly connected components*) yang cukup terkenal juga, yaitu algoritma Kosaraju.

Kedua algoritma ini digunakan untuk mencari SSC, tetapi berbeda dalam pendekatan dan implementasinya.

Berbeda dengan algoritma Tarjan, algoritma Kosaraju memiliki 2 traversal DFS, yang pertama untuk menentukan saat penyelesaian dari simpul, dan yang kedua untuk graph yang sudah di *transpose* dari waktu yang tertinggi yang didapatkan.

Secara kompleksitas waktu dan kompleksitas ruang keduanya hampir sama karena menggunakan DFS dan *stack*, tetapi algoritma Tarjan akan jauh lebih mudah untuk dipahami karena hanya menggunakan 1 traversal untuk semua node, sedangkan algoritma Kosaraju perlu menggunakan 2 traversal DFS.

Algoritma Kosaraju memiliki kelebihan yaitu akan mengeluarkan urutan SSC sesuai SSC yang telah selesai ditemukan semua komponennya sehingga akan lebih baik digunakan pada penerapan yang membutuhkan urutan SSC dengan benar, seperti *topological sort* pada DAG (*Directed Acyclic Graph*), sementara algoritma Tarjan mengeluarkan SSC secara acak, tergantung dari simpul pertama yang dikunjunginya, sehingga tidak akan terurut.

V. Penerapan

Algoritma Tarjan digunakan untuk berbagai kegunaan, khususnya terkait penggunaan SCC (*Strongly Connected Components*) yaitu:

- Ilmu Komputer
  Algoritma Tarjan dapat digunakan untuk menyelesaikan masalah *Strongly Connected Components* dan menyelesaikan dekomposisi Dulmage-Mendelsohn, sebuah metode untuk mengklasifikasi sisi pada graf bipartit.
- Analisis Situs dan Sosial Media
  Algoritma Tarjan digunakan pada jaringan sosial media dan berbagai *web* untuk menemukan komunitas dan membuat rekomendasi bagi pengguna dengan minat yang sama.
- Sistem Komputer
  Algoritma Tarjan juga dapat digunakan untuk mengoptimisasi *compiler*. Dengan adanya konsep SCC, *compiler* dapat mengoptimisasi penggabungan dan identifikasi *loop* sejajar, mengidentifikasi *dead code* (bagian kode yang tidak dapat dijangkau bagian lain) dan analisis *critical path* (informasi untuk optimisasi alur dan pengelolaan *compiler*), dan melakukan *alias analysis*, yaitu analisis alur data

untuk menentukan kemungkinan terjadinya memori yang saling menimpa.

- Desain Sirkuit dan Perangkat Keras
  Algoritma Tarjan dapat digunakan untuk menentukan alur dan arah sirkuit digital untuk memahami cara kerja sirkuit listrik dan pembuatan perangkat keras agar dapat berjalan tanpa adanya peringatan atau bahaya.
- Database
  Karena sifat SCC yang tetap berupa graf yang memiliki data simpul dan hubungan antar simpul, maka tidak akan lepas dari pengelolaan data. Dengan adanya SCC yang disediakan algoritma Tarjan, *query* dapat dioptimisasi untuk hubungan rekursif dan dapat digunakan untuk teknik optimisasi *database* seperti pada data hierarki yang tidak perlu melakukan traversal terus-menerus setiap kali ada *query*.

VI. Conclusion

Algoritma Tarjan adalah sebuah proses untuk menelusuri sebuah graf dan menghasilkan SSC (*Strongly Connected Components*), yaitu komponen dalam graf berarah yang setiap simpul pada komponen dapat saling terhubung (ada lintasan).

Algoritma ini adalah algoritma sederhana dengan kompleksitas linier, menggunakan konsep DFS (*Depth First Search*) dan struktur data *stack* untuk mencatat keberadaan semua simpul dan menentukan komponen yang terhubung.

Algoritma ini memiliki banyak kegunaan, terutama dalam konsep SCC yang dapat berperan banyak dalam optimisasi data yang berbentuk graf atau serupa memiliki suatu nilai dan hubungan antar nilai yang harus ditelusuri. Beberapa contohnya adalah dalam optimisasi *compiler*, *database*, dan jaringan situs serta sosial media.

VI. Ucapan Terima Kasih

Segala puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa oleh karena hikmat dan anugerahnya penulis dapat menyelesaikan makalah ini. Penulis menyampaikan terima kasih kepada Dr. Nur Ulfa Maulidevi selaku dosen pengampu IF2120 Matematika Diskrit kelas K01 atas bimbingan dan pemberian ilmu yang menjadi landasan penulisan makalah ini.

Penulis juga berterimakasih kepada teman-teman penulis serta pihak-pihak lain yang telah mendorong dan menolong penulis dalam penyelesaian makalah ini.

Referensi

[1] "Strongly Connected Components." GeeksforGeeks, December 11, 2023. https://www.geeksforgeeks.org/strongly-connected-components/

[2] "Tarjan's Algorithm to find Strongly Connected Components." OpenGenus. December 4, 2023. https://iq.opengenus.org/tarjans-algorithm/

[3] "Unlocking the Mystery of Algorithms: Exploring Strongly Connected Components in Depth." Localhost. December 4, 2023. https://locall.host/what-is-algorithm-for-strongly-connected-components/

[4] Munir, Rinaldi. "Homepage Rinaldi Munir". https://informatika.stei.itb.ac.id/~rinaldi.munir/. (diakses pada 4 Desember 2023).

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, 11 Desember 2023

Zachary Samuel Tobing 13522016