Penerapan Pewarnaan Graf pada Algoritma DFS untuk Menguji Sifat Bipartit Suatu Graf

Daniel Ryan Levyson - 13516132
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516132@std.stei.itb.ac.id

***Abstract*—Graf bipartit merupakan salah satu jenis graf di dalam teori graf yang memiliki banyak penerapan di dalam kehidupan. Semakin kompleks suatu graf, semakin sulit untuk menentukan apakah suatu graf merupakan graf bipartit atau bukan. Oleh karena itu, diperlukan suatu metode untuk menguji sifat bipartit suatu graf. Metode pewarnaan graf dapat dimanfaatkan untuk menentukan apakah sutatu graf merupakan graf bipartit atau bukan. Namun, ketika suatu program komputer dituntut untuk dapat mengenali graf yang bipartit, diperlukan suatu algoritma tertentu. Konsep pewarnaan graf dapat diterapkan pada algoritma DFS untuk membuat program komputer mengenali graf yang bipartit.**

***Kata Kunci*—pewarnaan, dfs, bipartit.**

I. PENDAHULUAN

Teori graf merupakan salah ilmu dalam Matematika Diskrit. Teori ini membahas berbagai macam graf dan sifat-sifatnya. Salah satu jenis graf yang menarik untuk dibahas dan cukup penting dampaknya di dalam kehidupan adalah graf bipartit.

Graf bipartit digunakan dalam banyak hal, beberapa diantaranya adalah *bipartite matching*, *penjadwalan tugas*, *hall's marriage theorem*, konkurensi sistem, dan lain-lain. Untuk memanfaatkan graf bipartit pada program komputer, komputer perlu mengenali bagaimana suatu graf dapat dikatakan bipartit. Oleh karena itu, perlu dirancang suatu algoritma yang sesuai dengan keperluan tersebut.

Pada dasarnya, metode pewarnaan graf cukup untuk mengenali suatu graf yang bipartit. Dengan mewarnai setiap simpul pada graf dengan dua warna sedemikian rupa dapat diketahui apakah suatu graf bipartit atau bukan. Di dalam makalah ini, metode pewarnaan graf akan diimplementasikan menggunakan algoritma penelusuran graf, yaitu DFS (*Depth-First-Search*). Proses penelusuran oleh DFS dan pewarnaan graf akan dijabarkan secara lebih rinci hingga dapat disimpulkan suatu graf uji memiliki sifat bipartit atau tidak.

II. LANDASAN TEORI

A. Teori Graf

Sebuah Graf G merupakan sekumpulan simpul V yang dihubungkan dengan sekumpulan sisi E, sehingga G = (V, E). Secara visual, simpul digambarkan sebagai titik, dan sisi digambarkan sebagai garis yang ditarik dari suatu simpul ke simpul lainnya atau ke simpul itu sendiri. Apabila sebuah simpul v_1 dihubungkan dengan simpul v_2 oleh sisi e, maka dapat dinotasikan sebagai e = (v_1, v_2).

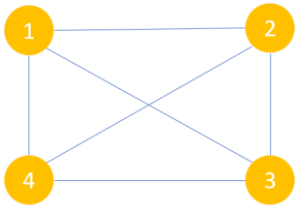
Graf dapat digolongkan ke dalam dua jenis berdasarkan ada dan tidaknya sisi ganda. Sisi ganda merupakan sisi-sisi yang menghubungkan dua simpul yang sama. Graf yang memiliki sisi ganda disebut graf tak-sederhana, sedangkan graf yang tidak memiliki sisi ganda disebut graf sederhana. Apabila sisi ganda pada graf tak-sederhana menghubungkan suatu simpul dengan dirinya sendiri, maka graf tersebut dapat dinamakan graf semu.

Terdapat beberapa terminologi dasar dalam teori graf, yaitu:

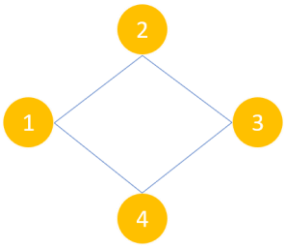
1. Ketetanggaan
   Simpul v_1 disebut bertetangga dengan simpul v_2 jika dan hanya jika ada sisi e yang menghubungkannya.
2. Kebersisian
   Sisi e disebut bersisian dengan simpul v_1 dan v_2 jika dan hanya jika e menghubungkan v_1 dengan v_2.
3. Simpul Terpencil
   Simpul v dikatakan terpencil jika dan hanya jika tidak ada sisi e yang bersisian dengan simpul v.
4. Graf Kosong
   Suatu graf G dikatakan graf kosong jika dan hanya jika himpunan sisi E merupakan himpunan kosong.
5. Derajat
   Jumlah sisi yang bersisian dengan suatu simpul disebut derajat

Graf-graf sederana dengan sifat-sifat tertentu digolongkan sebagai graf sederhana khusus. Berikut ini adalah graf sederhana khusus:

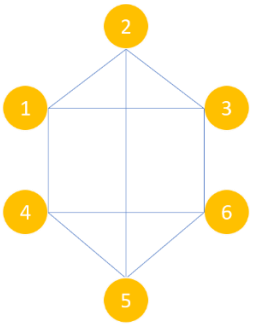
1. Graf Lengkap
   Suatu graf disebut graf lengkap jika dan hanya jika setiap simpul pada graf bertetangga dengan setiap simpul lainnya, sehingga setiap simpul pada graf lengkap dengan n buah simpul akan memiliki derajat n-1.



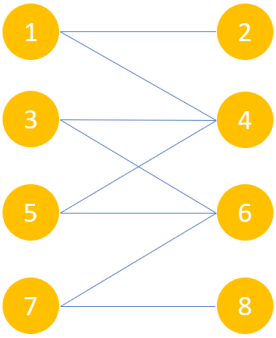
2. Graf Lingkaran
   Suatu graf sederhana dikatakan graf lingkaran jika dan hanya jika setiap simpul pada graf tersebut hanya memiliki derajat sebesar dua.



3. Graf Teratur
   Suatu graf sederhana dikatakan graf teratur jika dan hanya jika setiap simpulnya memiliki derajat yang sama.



4. Graf Bipartit
   Graf bipartit memiliki dua himpunan simpul V_1 dan V_2. Setiap simpul pada V_1 tidak bertetangga dengan simpul manapun pada V_1 dan setiap simpul pada V_2 tidak bertetangga dengan simpul mananpun pada V_2. Simpul-simpul pada V_1 hanya dapat bertetangga dengan simpul-simpul pada V_2, begitu juga sebaliknya. Apabila setiap simpul pada V_1 bertetangga dengan setiap simpul pada V_2, maka graf tersebut dikatakan graf bipartit lengkap.



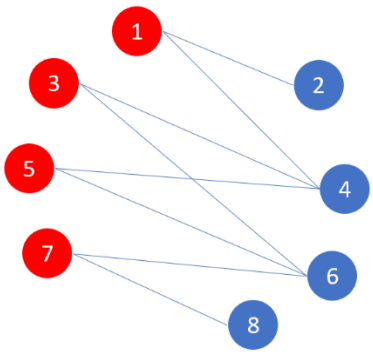
Untuk memroses graf di dalam program komputer, diperlukan sebuah struktur data untuk merepresentasikan graf. Berikut ini adalah beberapa alternatif representasai graf:

1. Matriks Ketetanggaan
   Sebuah matriks A berukuran n x n dapat merepresentasikan graf G dengan n buah simpul. Apabila simpul v_i bertetangga dengan simpul v_j pada graf G, maka $A_{ij} = 1$, jika tidak maka $A_{ij} = 0$.
2. Matriks Bersisian
   Sebuah matriks A berukuran n x m dapat merepresentasikan graf G dengan n buah simpul dan m buah sisi. Apabila simpul v_i bersisian dengan sisi e_j, maka $A_{ij} = 1$, jika tidak maka $A_{ij} = 0$.
3. Senarai Ketetanggan
   Graf G dengan n buah simpul dapat direpresentasikan ke dalam struktur data *Linear List* dengan n buah elemen. Apabila simpul v_i bertetangga dengan simpul v_j, maka elemen-i akan menyimpan *pointer* ke elemen-j dan juga sebaliknya. Selain itu senarai ketetangaan dapat juga direpresentasikan dalam struktur data *array* eksplisit dua dimensi.

B. Pewarnaan Graf

Pewarnaan graf dilakukan untuk tujuan tertentu. Baik simpul, sisi, maupun wilayah pada graf dapat diwarnai tergantung tujuannya. Salah satu tujuannya adalah untuk menggolongkan simpul ke dalam beberapa kelompok dengan banyak kelompok tergantung pada jumlah warna yang digunakan.

Dalam konteks pengujian sifat bipartit suatu graf, pewarnaan dilakukan untuk mewarnai semua simpul pada satu himpunan dengan warna yang sama, kemudian mengambil warna yang berbeda untuk mewarnai semua simpul pada himpunan lainnya. Pewarnaan tersebut akan dilakukan dengan langkah-langkah yang akan ditentukan. Apabila pewarnaan tersebut dapat dilakukan, berarti graf tersebut memiliki dua himpunan simpul yang tiap anggotanya tidak bertetangga dengan anggota di himpunan yang sama, sehingga graf tersebut dapat dikatakan bipartit.



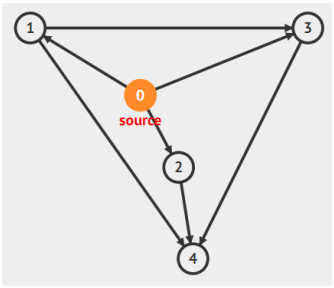
C. Algoritma DFS

Algoritma DFS (Depth-First-Search) merupakan algoritma untuk menelusuri data yang berstruktur graf atau pohon. Algoritma ini memulai penelusuran dari simpul yang merupakan akar penelusuran dan melanjutkan penelusuran ke salah satu simpul yang bertetangga, lalu kemudian menelusuri simpul tetangga pada tingkat berikutnya (semakin banyak simpul yang harus ditelusuri dari suatu simpul untuk sampai ke akar, semakin tinggi tingkatnya). Algoritma ini menelusuri sampai simpul yang terjauh dari simpul akar terlebih dahulu, baru kemudian menelusuri tetangga selanjutnya.

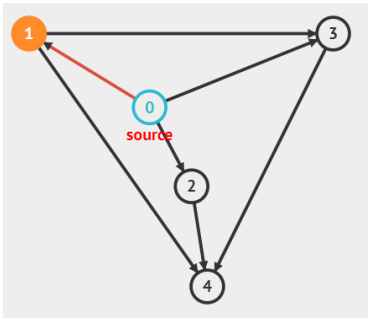
Apabila diimplementasikan secara non-rekursif, algoritma ini memanfaatkan struktur data *stack* dalam melakukan penelusuran simpul. Simpul akar mula-mula akan dimasukan ke dalam *stack*, lalu kemudian akan dikeluarkan diikuti dengan memasukan tetangganya ke dalam *stack*. Sesuai prinsip *stack*, *first in last out*, simpul tetangga akar akan diproses diikuti dengan memasukan tetangganya ke dalam *stack*. Proses tersebut

akan diiterasi sampai isi *stack* kosong.

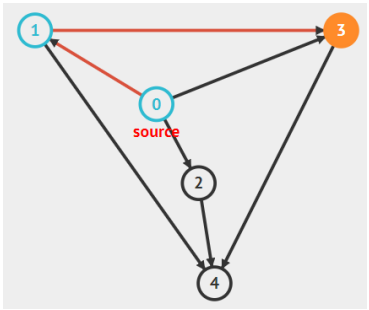
Kompleksitas algoritma ini tergantung pada representasi graf yang digunakan. Apabila direpresentasikan sebagai matriks ketetanggaan, algoritma DSF memiliki kompleksitas $O(|V|^2)$ karena algoritma ini perlu menelusuri setiap kolom pada suatu baris untuk memeriksa ketetanggaan. Apabila direpresentasikan sebagai senarai ketetanggaan, algorima DFS memiliki kompleksitas $O(|V| + |E|)$, sehingga semakin banyak simpul dan sisi yang ada pada graf, semakin lama algoritma ini akan bekerja.



Visualisasi DFS 1: Simpul nol adalah simpul akar
Sumber: https://visualgo.net/en/dfsbfs



Visualisasi DFS 2: Salah satu simpul tetangga simpul nol ditelusuri, yaitu simpul satu.
Sumber: https://visualgo.net/en/dfsbfs



Visualisasi DFS 3: Penelusuran dilanjutkan ke tetangga simpul satu terlebih dahulu, bukan tetangga simpul nol yang lainnya.
Sumber: https://visualgo.net/en/dfsbfs

III. PEMBAHASAN

A. Pewarnaan Graf dengan Algoritma DFS

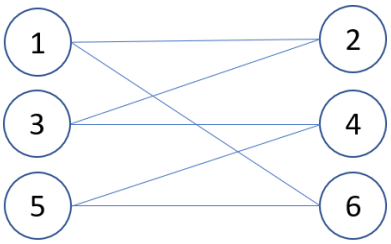
Graf G yang akan diuji akan direpresentasikan menggunakan senarai ketetanggan, sehingga kompleksitas algoritma ini adalah $O(|V| + |E|)$. Algoritma DFS akan bekerja sebagai berikut:

1. Inisialisasi graf G dengan graf yang akan diuji dan *array* warna W_i dengan nilai 0.
2. Ambil sembarang simpul pada graf G, misal simpul v_1, masukan ke dalam *stack*. Ambil warna W_1 sebagai Wc.
3. Lakukan iterasi sampai isi *stack* kosong.
4. Ambil simpul teratas v_i pada *stack* (sambil mengeluarkan v_i dari *stack*). Apabila simpul vi belum ditandai sudah ditelusuri, tandai sebagai simpul yang pernah ditelusuri. Tandai $W_i = \sim Wc$ (negasi dari Wc, untuk memberi warna yang berbeda). Ambil $Wc = W_i$. Masukan semua tetangga v_i ke dalam *stack*.
5. Apabila v_i sudah ditandai pernah ditelusuri, periksa apakah W_i sama dengan Wc, jika ya, maka keluar dari iterasi. Graf G bukan merupakan graf bipartit.
6. Jika W_i tidak sama dengan Wc, lanjutkan iterasi (kembali ke nomor 3).
7. Apabila *stack* telah kosong, graf G merupakan graf bipartit karena semua simpul pada graf G berhasil diberi warna sedemikian rupa sehingga setiap simpul yang bertetangga memiliki warna yang berbeda dengan tetangganya.

B. Analisa Kasus

Akan ditinjau algoritma DFS yang sudah dijelaskan untuk memeriksa sifat bipartit dua graf yang berbeda.

1. Tinjau Graf G1 dalam bentuk senarai ketetanggaan berikut: (bilangan merupakan nomor simpul)



1 → 2, 6
2 → 1, 3
3 → 2, 4
4 → 3, 5
5 → 4, 6
6 → 5, 1

Berikut ini kondisi *stack* dan *array* warna selama algoritma DFS bekerja:

1) Simpul pertama dimasukan ke dalam *stack*, kemudian masuk ke dalam iterasi yang berhenti apabila *stack* kosong.

Stack	1

Simpul	1	2	3	4	5	6
Warna	0	0	0	0	0	0

2) Simpul pertama diambil, ditandai pernah ditelusuri, dan warnanya diberi nilai satu. *Stack* diisi dengan tetangga simpul satu (dua dan enam).

Stack	2	6				

Simpul	1	2	3	4	5	6
Warna	1	0	0	0	0	0

3) Simpul enam diambil, ditandai pernah ditelusuri, dan warnanya diberi nilai nol (negasi dari satu). *Stack* diisi dengan tetangga simpul enam (satu dan lima).

Stack	2	1	5			

Simpul	1	2	3	4	5	6
Warna	1	0	0	0	0	0

4) Simpul lima diambil, ditandai pernah ditelusuri, dan warnanya diberi nilai satu (negasi dari nol). *Stack* diisi dengan tetangga simpul lima (empat dan enam).

Stack	2	1	4	6		

Simpul	1	2	3	4	5	6
Warna	1	0	0	0	1	0

5) Simpul enam diambil, karena pernah ditelusuri, namun warnanya berbeda dengan warna sebelumnya (warna sebelumnya satu, warna enam adalah nol), maka simpul enam dikeluarkan dan tidak diproses.

Stack	2	1	4			

Simpul	1	2	3	4	5	6
Warna	1	0	0	0	1	0

6) Simpul empat diambil, ditandai pernah ditelusuri, dan warnanya diberi nilai nol (negasi dari satu). *Stack* diisi dengan tetangga simpul empat (tiga dan lima).

Stack	2	1	3	5		

Simpul	1	2	3	4	5	6
Warna	1	0	0	0	1	0

7) Simpul lima diambil, karena pernah ditelusuri, namun warnanya berbeda dengan warna sebelumnya (warna sebelumnya nol, warna lima adalah satu), maka simpul lima dikeluarkan dan tidak diproses.

Stack	2	1	3			

Simpul	1	2	3	4	5	6
Warna	1	0	0	0	1	0

8) Simpul tiga diambil, ditandai pernah ditelusuri, dan warnanya diberi nilai satu (negasi dari nol). *Stack* diisi dengan tetanga simpul tiga (dua dan empat).

Stack	2	1	2	4		

Simpul	1	2	3	4	5	6
Warna	1	0	1	0	1	0

9) Simpul empat diambil, karena pernah ditelusuri, namun warnanya berbeda dengan warna sebelumnya (warna sebelumnya satu, warna empat adalah nol), maka simpul empat dikeluarkan dan tidak diproses.

Stack	2	1	2			

Simpul	1	2	3	4	5	6
Warna	1	0	1	0	1	0

10) Simpul dua diambil, ditandai pernah ditelusuri, dan warnanya diberi nilai nol (negasi dari satu). *Stack* diisi tetangga simpul dua (satu dan tiga), namun simpul satu dan tiga yang sudah pernah ditelusuri, dikeluarkan tanpa diproses lagi.

Stack	2	1				

Simpul	1	2	3	4	5	6
Warna	1	0	1	0	1	0

11) Simpul satu dikeluarkan tanpa diproses, karena sudah pernah ditelusuri.

Stack	2					

Simpul	1	2	3	4	5	6
Warna	1	0	1	0	1	0

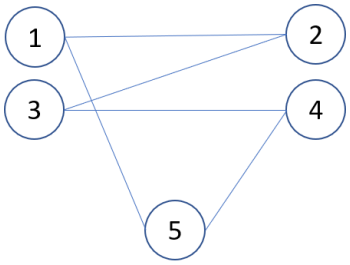
12) Simpul dua dikeluarkan tanpa diproses, karena sudah pernah ditelusuri. *Stack* kosong membuat iterasi berhenti.

Stack						

Simpul	1	2	3	4	5	6
Warna	1	0	1	0	1	0

Dapat dilihat pada *array* warna bahwa warna-warna simpul yang bertetangga berbeda satu sama lain, sehingga graf G1 merupakan graf bipartit.

2. Tinjau Graf G2 dalam bentuk senarai ketetanggaan berikut: (bilangan merupakan nomor simpul)



1 → 2, 5
2 → 1, 3
3 → 2, 4
4 → 3, 5
5 → 1, 4

Berikut ini kondisi *stack* dan *array* warna selama algoritma DFS bekerja:

1) Simpul pertama dimasukan ke dalam *stack*, kemudian masuk ke dalam iterasi yang berhenti apabila *stack* kosong.

Stack	1

Simpul	1	2	3	4	5
Warna	0	0	0	0	0

2) Simpul pertama diambil, ditandai pernah ditelusuri, dan warnanya diberi nilai satu. *Stack* diisi dengan tetangga simpul satu (dua dan lima).

Stack	2	5

Simpul	1	2	3	4	5
Warna	1	0	0	0	0

3) Simpul lima diambil, ditandai pernah ditelusuri, dan warnanya diberi nilai nol. *Stack* diisi dengan tetangga simpul lima (satu dan empat).

Stack	2	1	4

Simpul	1	2	3	4	5
Warna	1	0	0	0	0

4) Simpul empat diambil, ditandai pernah ditelusuri, dan warnanya diberi nilai satu. *Stack* diisi dengan tetangga simpul empat (tiga dan lima).

Stack	2	1	3	5

Simpul	1	2	3	4	5
Warna	1	0	0	1	0

5) Simpul lima diambil, karena pernah diproses dan warnanya berbeda dengan warna sebelumnya (satu), simpul lima dikeluarkan dari *stack* dan tidak diproses.

Stack	2	1	3

Simpul	1	2	3	4	5
Warna	1	0	0	1	0

6) Simpul tiga diambil, ditandai pernah ditelusuri dan warnanya diberi nilai nol. *Stack* diisi dengan tetangga simpul tiga (dua dan empat).

Stack	2	1	2	4

Simpul	1	2	3	4	5
Warna	1	0	0	1	0

7) Simpul empat diambil, namun diabaikan karena pernah ditelusuri

Stack	2	1	2

Simpul	1	2	3	4	5
Warna	1	0	0	1	0

8) Simpul dua diambil dua diambil, ditandai pernah ditelusuri dan warnanya diberi nilai satu. *Stack* diisi dengan tetangga simpul dua (satu dan tiga).

Stack	2	1	1	3

Simpul	1	2	3	4	5
Warna	1	1	0	1	0

9) Simpul tiga diambil dan diabaikan. Simpul satu diambil. Simpul satu pernah ditelusuri, nilai warnanya satu dan nilai warna sebelumnya (simpul empat) adalah satu. Oleh karena itu, iterasi dihentikan.

Stack	2	1	1

Simpul	1	2	3	4	5
Warna	1	1	0	1	0

Graf G2 bukan merupakan graf bipartit. Pada saat pewarnaan, simpul satu dan simpul dua yang saling bertetangga memiliki warna yang sama.

IV. KESIMPULAN

Dengan menggunakan metode pewarnaan graf yang diimplementasikan pada algoritma DFS, sifat bipartit suatu graf dapat ditentukan. Algoritma ini memungkinkan komputer untuk mengenali suatu graf yang bipartit. Dengan begitu, graf yang bipartit dapat dimanfaatkan oleh program komputer untuk keperluan tertentu.

V. LAMPIRAN

Pseudocode mirip C++ dari algoritma pewarnaan graf dengan DFS.

```
boolean isBipartite(integer G[V][], integer root)
{
    // Inisialisasi
    integer arrayWarna[V];
    boolean discovered[V];
    for (integer i = 0; i < V; ++i)
        arrayWarna[i] = 0;
        discovered[i] = false;
    // Variable untuk mengingat "warna sebelumnya"
    integer wc = 0;

    // Stack
    Stack<integer> stack;
    stack.push(root);

    // Iterasi selama stack tidak kosong
    while (!stack.empty())
    {
        // Ambil elemen stack teratas
        integer u = stack.top();
        stack.pop();

        if(!discovered[u]) { // Jika belum pernah ditelusuri
            // Tandai pernah ditelusuri
            discovered[u] = true;
            // Berikan nilai warna yang berbeda dari wc
            arrayWarna[u] = !wc;

            // Masukan semua simpul bertetangga ke dalam stack
            for (integer v = 0; v < G[u].size(); ++v)
            {
                stack.push(G[u][v]);
            }
        } else if(discovered[u] && arrayWarna[u] == wc) {
            // Jika pernah ditelusuri,
            // dan memiliki warna sama dengan wc,
            // artinya bukan graf bipartit
            return false;
        }
    }

    // Jika program sampai ke titik ini,
    // artinya stack telah kosong
    // Jika stack telah kosong,
    // artinya pewarnaan berhasil dilakukan,
    // sehingga graf G adalah graf bipartit
    return true;
}
```

VI. UCAPAN TERIMA KASIH

Dengan tersusunnya makalah ini, penulis mengucapkan puji dan syukur kepada Tuhan Yang Maha Esa karena berkat kekuatan dan kesempatan yang diberikan, penulis mampu menyelesaikan makalah ini. Penulis berterima kasih kepada bapak Dr. Judhi Santoso, M.Sc. yang telah membimbing penulis selama perkuliahan Matematika Diskrit. Penulis berterima kasih kepada orang tua yang selalu mendukung dan memberikan motivasi. Penulis juga berterima kasih kepada teman-teman yang telah mendukung penulis dalam pengerjaan makalah ini.

VII. REFERENSI

[1] Munir, Rinaldi. (2016). Matematika Diskrit. Bandung: Penerbit Informatika.

[2] Cormen, Thomas H., et al. Introduction to Algorithms 3rd Edition, Massachusetts Institute of Technology, 2009.

[3] http://www.geeksforgeeks.org/depth-first-traversal-for-a-graph/, diakses pada tanggal 3 Desember 2017 pukul 1.00

[4] http://www.techiedelight.com/determine-given-graph-bipartite-graph-using-dfs/, diakses pada tanggal 3 Desember 2017 pukul 1.35

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 2017

Daniel Ryan Levyson
13516132