Mendeteksi Siklus dari *Dots* pada Game *Dots: A Game About Connecting* dengan Algoritma DFS

Ivan Jonathan / 13516059
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516059@stei.itb.ac.id

***Abstrak* — Algoritma DFS merupakan salah satu algoritma penelusuran struktur data graf atau pohon secara sistematis. Pencarian solusi dengan menggunakan DFS tidak diberikan informasi tambahan (un*informed* / *blindsearch*) layaknya algoritma BFS. Algoritma ini banyak sekali dipakai diberbagai cabang *computer science* contohnya pemetaan rute, *scheduling*, analisis jaringan, dan menemukan pohon merentang.**

Permainan *Dots: Game About Connecting* adalah permainan jenis *puzzle* yang sederhana sehingga pengguna tidak perlu banyak waktu untuk mempelajarinya, aturannya adalah pengguna harus menemukan cara untuk koneksikan *dot-dot* yang berwarna sama yang bertetangga. Skor yang didapat akan semakin besar apabila *dot* yang dikoneksikan pengguna mempunyai jarak koneksi yang besar, lalu jika pengguna berhasil membentuk *dot-dot* menjadi bentuk tertutup (*closed shapes*) misalnya persegi, maka jumlah skor yang didapat lebih banyak ketimbang membentuk garis dari sebuah *dot*.

Makalah ini akan membahas penggunaan algoritma DFS untuk mencari kemungkinan *closed shape* yang dapat dibentuk oleh *dot-dot* yang berwarna sama dan bertetangga.

***Kata kunci* — DFS, *dot*, warna, bentuk tertutup, siklus.**

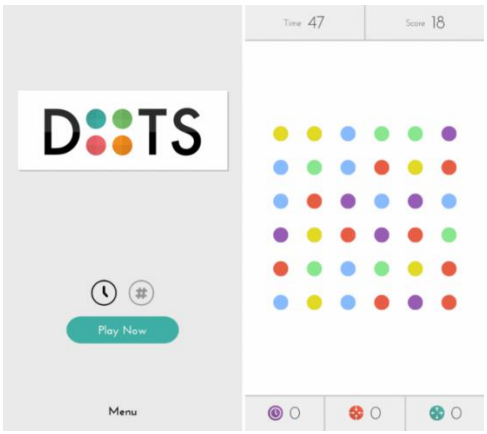
I. PENDAHULUAN

Permainan *Dots: Game About Connecting* adalah permainan jenis *puzzle* sederhana yang dikembangkan oleh PlayDots. Permainan ini menuntut pemain untuk membuat strategi yang cepat dan efisien agar mendapatkan skor yang maksimal. Ruang permainan yang berukuran 6 x 6 *dot* yang berisi lima warna yang berbeda. *Dot* yang bertetangga hanya dengan sekitar *dot* tersebut dan diagonal dari suatu *dot* tidak dianggap tetangganya.

Secara singkat, aturan pada game ini adalah pengguna harus mengkoneksikan dua atau lebih *dot* yang berwarna dan bertetangga untuk membentuk garis, garis dapat ditarik ke segala arah tetapi tidak diagonal. Skor yang didapat lebih tinggi saat pengguna berhasil mengkoneksikan *dot* membentuk bentuk tertutup seperti persegi. Titik-titik yang dihubungkan pengguna akan menghilang dan segera digantikan dengan titik-titik yang baru. Goalnya adalah pengguna harus menghilangkan banyak titik dan menciptakan skor setinggi mungkin dalam batas waktu 60 detik.

Game ini menyediakan 3 mode permainan yang berbeda, tidak hanya pada *time-based* dalam mengumpulkan skor tetapi ada mode yang dinamakan *moves mode*. Pada mode ini, pemain diberikan batasan pergerakan untuk menciptakan *dot* yang terkoneksi. Setiap kali *dot* yang terkoneksi maka pergerakan pemain dikurangi satu. Permainan berakhir setelah pergerakan pemain tersisa nol. Lalu, mode terakhir adalah *Endless mode* yaitu bermain tanpa ada batasan.

Game ini secara tidak langsung memberikan dampak positif bagi penggunanya yaitu melatih pengguna agar fokus, strategis dan gesit. Berikut tampilan game di ponsel pintar.



Gambar 1. Tampilan antarmuka permainan *Dots: A Game About Connecting*.
Sumber: https://www.droid-life.com/wp-content/uploads/2013/08/Dots-650x575.png.

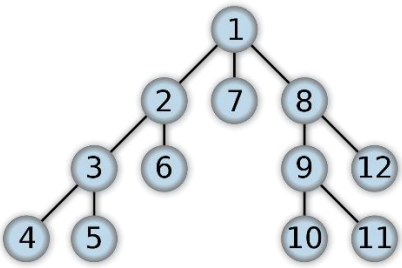
Pada makalah ini, penulis ingin membuat program sederhana yang mengilustrasikan game ini dan membuat bot yang menggunakan algoritma DFS untuk menemukan bentuk tertutup yang mungkin dapat diciptakan *dot-dot* yang ada di papan pada waktu t, sehingga dapat memberikan *hint* kepada pengguna atau dapat diimplementasikan pada program lain.

II. LANDASAN TEORI

Algoritma DFS akan menjadi algoritma yang akan digunakan untuk menemukan siklus / bentuk tertutup yang dapat dibentuk oleh *dots* yang ada.

Berdasarkan definisi, Algoritma DFS menelusuri graf atau pohon yang dimulai dari *root* untuk mencapai ke sebuah solusi secara mendalam. Pencarian mendalamnya yaitu dengan mengunjungi anak yang diekspan dari *parent* dan menelusuri anak tersebut sampai ditemukannya solusi atau sampai semua simpul ditelusuri. Apabila simpul v adalah simpul daun, maka

dilakukan *backtrack* ke simpul sebelumnya dan menelusuri mendalam kembali pada simpul yang belum ditelusuri. Pencarian berhenti sampai semua simpul ditelusuri (tidak ada solusi) atau ditemukan solusi.



Gambar 2. Contoh Graf.

Sumber: https://upload.wikimedia.org/wikipedia/commons/thumb/1/1f/Depth-first-tree.svg/300px-Depth-first-tree.svg.png

Berikut algoritma DFS secara umum dijabarkan dalam bentuk poin :

1. Traversal dimulai dari simpul 1 (*root)*.
2. Ekspan anak yang dapat diekspan simpul 1.
3. Telusuri simpul anak, penelusuran bisa dilakukan berdasarkan aturan *alphabet* atau *numeric* tergantung programmer. Sehingga, apabila berdasarkan aturan angka maka simpul yang akan ditelusuri adalah simpul 2.
4. Ulangi DFS mulai dari simpul yang dipilih dari langkah 3.
5. Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, maka lakukan pencarian runut-balik (*backtrack*) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
6. Pencarian berakhir apabila tidak ada lagi simpul yang dapat dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Dalam bentuk *pseudocode*, algoritma DFS ditulis 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 w←1 to n do
      if A[v,w]=1 then {simpul v dan simpul w bertetangga }
         if not dikunjungi[w] then
            DFS(w)
         endif
      endif
   endfor
```

Langkah-langkah penelusurannya menjadi :

1. DFS(1): v = 1; dikunjungi[1] = true; DFS(2).
2. DFS(2): v = 2; dikunjungi[2] = true; DFS(3).
3. DFS(3): v = 3; dikunjungi[2] = true; DFS(4).
4. DFS(4): v = 4; dikunjungi[4] = true; DFS(5).
5. DFS(5): v = 5; dikunjungi[5] = true; DFS(6).
6. DFS(6): v = 6; dikunjungi[6] = true; DFS(7).
7. DFS(7): v = 7; dikunjungi[7] = true; DFS(8).
8. DFS(8): v = 8; dikunjungi[8] = true; DFS(9).
9. Dan seterusnya.

III. IMPLEMENTASI

Ada beberapa hal yang perlu diasumsikan sebelum implementasi program yang akan dibuat. Asumsi-asumsi tersebut disebutkan dibawah ini.

1. Developer game *Dots: A Game About Connecting* tidak membuka source code mereka sehingga penulis mengasumsikan papan permainan sebagai struktur data graf tidak berarah.
2. Simpul-simpul pada graf adalah *dots* yang ada pada papan permainan.
3. Setiap simpul hanya bisa bertetangga kanan, kiri, atas , dan bawahnya sehingga tidak dapat bertetangga dengan diagonalnya.
4. Beberapa tambahan kondisi dan modifikasi algoritma DFS agar dapat menemukan siklus *dots* yaitu koneksi hanya dapat dilakukan dengan warna yang sama dan untuk setiap simpul v yang punya tetangga dengan u, dimana u telah dikunjungi dan u bukan *parent* dari v maka terdapat siklus disana.
5. Menggunakan stack untuk menyimpan lokasi siklus *dot* yang punya warna yang sama.

Sebelumnya, penulis ingin membuat program berbasis Android tetapi karena terbatasnya waktu akhirnya hanya membuat program berbasis *command-line interface* (CLI) dengan bahasa Java. Struktur data graf tidak berarah yang dibuat menggunakan *List of Set of Node* yang masing-masing simpul punya ketetanggan dengan simpul lain yang ditaruh di *Set Of Node* dan semua itu di generasi otomatis saat penciptaan object Graf. Penomoran simpul mulai dari 0 s/d n^2-1, dimana n adalah ukuran papan permainan (saat penciptaan objek).

```
public class Graph{
  public static final String ANSI_RESET = "\u001B[0m";
  public static final String ANSI_BLACK = "\u001B[30m";
  public static final String ANSI_RED = "\u001B[31m";
  public static final String ANSI_GREEN = "\u001B[32m";
  public static final String ANSI_YELLOW = "\u001B[33m";
  public static final String ANSI_BLUE = "\u001B[34m";
  public static final String ANSI_PURPLE = "\u001B[35m";
  public static final String ANSI_CYAN = "\u001B[36m";
  public static final String ANSI_WHITE = "\u001B[37m";
  private List<Set<Node>> listAdjacency;
  private final String[] colors;
  private int nbNodes;
```

```
class Node{
  private int id;
  private String color;

  public Node(int id){
    this.id = id;
    color = null;
  }

  public Node(int id, String color){
    this.id = id;
    this.color = color;
  }
```

Gambar 3. Struktur data graf tidak berarah (Untuk papan permainan) dan Node (simpul). Sumber: Dokumentasi pribadi.

Sehingga, ambil contoh misalkan nilai n adalah 4, maka graf akan membuat 4x4 simpul yang koneksinya dapat dilihat pada tabel dibawah ini.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Tabel 1. Representasi dots dengan pendekatan struktur data graf tidak berarah. Sumber: Dokumentasi pribadi.

Nomor pada tabel adalah simpul-simpul yang telah dibuat dan merupakan *id* dari sebuah simpul dan garis putih menandakan

edge antar simpul, warna pada elemen tabel menandakan warna simpul tersebut. Setelah itu, untuk menemukan *cycle* pada graf dibuatlah perubahan pada algoritma DFS untuk menyesuaikan dengan persoalan. Langkah algoritmanya akan menjadi seperti dibawah ini:

1. Menginisialisasi semua tetangga pada simpul 0 s/d simpul ke n^2-1 dan inisialisasi semua warna saat pemanggilan konstruktor.
2. Buat sebuah senarai bertipe boolean yang menyimpan *state* pada sebuah simpul apakah simpul telah dikunjungi atau tidak, misalnya namanya adalah visited.
3. Inisialisasi elemen dari senarai visited dengan *false* semua, menandakan belum ada simpul yang dikunjungi.
4. Deklarasi dan inisialisasi variabel *parent* dengan nilai 1, peubah *parent* ditunjukkan untuk mengetahui *parent* dari simpul yang sedang dikunjungi (untuk menentukan siklus pada graf).
5. Deklarasi stack.
6. Mulai DFS, untuk simpul v yang belum dikunjungi, cek semua simpul yang bertetangga dengan v, ganti nilai visited[v] jadi true. Ini akan ada 3 kondisi *disjoint* yang dapat terjadi:
    a. Jika ada yang belum dikunjungi katakan simpul u, lalu jika warna dari simpul u sama dengan warna simpul v maka kunjungi simpul u dan ganti nilai *parent* menjadi v (karena v adalah parent dari u), ulangi langkah 6.
    b. Jika ada simpul (katakan simpul u_ yang telah dikunjungi dan berwarna sama dengan v, lalu nilai *parent* tidak sama dengan simpul u maka terdapat siklus pada graf, lalu push simpul u ke stack. Ini akan dilakukan secara rekursif sehingga stack akan berisi simpul u dan parent-parentnya.
    c. Apabila warna dari simpul u tidak sama dengan v dan nilai *parent* tidak sama dengan simpul u dan warna u tidak sama dengan v maka lewati simpul u, cari simpul tetangga dari v yang lain.

    Lakukan, sampai semua simpul dikunjungi.
7. Hasil keluaran akan menunjukkan apakah graf tersebut punya siklus atau tidak. Jika Ya, maka program akan menunjukkan letak siklus, Jika Tidak, maka hanya akan memberikan nilai *false*.

Penulis akan melakukan tiga percobaan untuk mencoba program ini, percobaan pertama dilakukan dengan mengikuti contoh graf sebelumnya yaitu pada tabel 1.

A. Percobaan I

Dengan mengikuti contoh tabel 1, maka tampilan aplikasi akan menjadi seperti dibawah ini.

```
PS E:\Documents\Ivan\Java Project\Cycle-Detection> javac Main.java
PS E:\Documents\Ivan\Java Project\Cycle-Detection> java Main
Y---P---G---R
|   |   |   |
P---Y---G---P
|   |   |   |
R---B---R---R
|   |   |   |
G---Y---R---R
```

Gambar 4. Simpul digenerate saat penciptaan objek. Dengan ukuran papan game = 4, jumlah total simpul sebanyak 16. Sumber: Dokumentasi pribadi.

Dengan hasil keluaran sebagai berikut:

```
Y---P---G---R
|   |   |   |
P---Y---G---P
|   |   |   |
R---B---R---R
|   |   |   |
G---Y---R---R
Apakah anda ingin mencari siklus yang mungkin dipapan permainan? = Ya

Cycle at = [10, 11, 15, 14]
Papan permainan punya siklus dengan dot berwarna sama
```

Gambar 5. Hasil keluaran dari input pada Gambar 4. Sumber: Dokumentasi pribadi.

Penjelasan dan langkah-langkah pencapaian solusi dari program dijelaskan sebagai berikut:

Iterasi ke-	Rekursif ke-	v	u	Parent	Visited[]	Apakah warna sama ?
1	1	0 (Y)	4 (P)	-1	[1,…,0]	Tidak
		0 (Y)	1 (P)	-1	[1,…,0]	Tidak
2	1	1 (P)	5 (Y)	-1	[1,1,…,0]	Tidak
		1 (P)	2 (G)	-1	[1,1,…,0]	Tidak
3	1	2 (G)	3 (R)	-1	[1,1,1,…,0]	Tidak
		2 (G)	6 (G)	-1	[1,1,1,…,0]	Sama
	2	6 (G)	7 (R)	2	[1,1,1,0,0,0,1,…,0]	Tidak
		6 (G)	5 (Y)	2	[1,1,1,0,0,0,1,…,0]	Tidak
		6 (G)	10 (P)	2	[1,1,1,0,0,0,1,…,0]	Tidak

4	1	3 (R)	7 (P)	-1	[1,1,1,0,0,0,1, …,0]	Tidak
dst.						

Tabel 2. Penjelasan langkah-langkah pencapaian solusi. Tanda titik tiga kali menandakan nilai false (nol).

Penjelasan langkah-langkah tidak sepenuhnya ditampilkan karena keterbatasan waktu pengerjaan dan ruang.

B. Percobaan II

Pada percobaan kedua ini, penulis masih menggunakan papan game 4x4 simpul, misalnya input warnanya diganti menjadi seperti dibawah ini

```
String[] colors = {"Yellow", "Yellow", "Blue", "Green",
                   "Blue", "Green", "Blue", "Red",
                   "Blue", "Red", "Purple", "Blue",
                   "Purple", "Yellow", "Red", "Purple"};
```

Gambar 6. Input warna dari 16 simpul. Sumber: Dokumentasi pribadi.

```
PS E:\Documents\Ivan\Java Project\Cycle-Detection> java Main
Y---Y---B---G
|   |   |   |
B---G---B---R
|   |   |   |
B---R---P---B
|   |   |   |
P---Y---R---P
```

Gambar 7. Tampilan papan. Sumber: Dokumentasi pribadi.

Dengan hasil keluaran program sebagai berikut:

```
PS E:\Documents\Ivan\Java Project\Cycle-Detection> java Main
Y---Y---B---G
|   |   |   |
B---G---B---R
|   |   |   |
B---R---P---B
|   |   |   |
P---Y---R---P
Apakah anda ingin mencari siklus yang mungkin dipapan permainan? = y
a


Papan permainan tidak punya siklus dengan dot berwarna sama
```

Gambar 8. Keluaran program dari input pada gambar 7. Sumber: Dokumentasi pribadi.

Penjelasan dan langkah-langkah pencapaian solusi dari program dijelaskan sebagai berikut:

Iterasi ke-	Rekursif ke-	v	u	Parent	Visited[]	Apakah warna sama?
1	1	0 (Y)	4 (P)	-1	[0,…,0]	Tidak
		0 (Y)	1 (Y)	-1	[1,…,0]	Sama
	2	1	5	0	[1,1,…,0]	Tidak
		(P)	(G)			
		1 (P)	2 (B)	0	[1,1,…,0]	Tidak
2	1	2 (B)	3 (G)	-1	[1,1,1,…,0]	Tidak
		2 (B)	6 (B)	-1	[1,1,1,…,0]	Sama
	2	6 (B)	7 (R)	2	[1,1,1,0,0,0,1, …,0]	Tidak
		6 (B)	5 (G)	2	[1,1,1,0,0,0,1, …,0]	Tidak
		6 (B)	10 (P)	2	[1,1,1,0,0,0,1, …,0]	Tidak
dst.						

Bila dilihat memang tidak ada *dots* yang dapat membentuk siklus.

C. Percobaan III

Pada percobaan terakhir, penulis menggunakan papan game 3x3 simpul, dengan konfigurasi warna sebagai berikut:

```
PS E:\Documents\Ivan\Java Project\Cycle-Detection> java Main
G---G---R
|   |   |
G---G---G
|   |   |
R---P---G
```

Gambar 9. Tampilan papan 3x3. Sumber: Dokumentasi pribadi.

Dengan hasil keluaran program sebagai berikut:

```
PS E:\Documents\Ivan\Java Project\Cycle-Detection> javac Main.java
PS E:\Documents\Ivan\Java Project\Cycle-Detection> java Main
G---G---R
|   |   |
G---G---G
|   |   |
R---P---G
Apakah anda ingin mencari siklus yang mungkin dipapan permainan? = ya

Cycle at = [0, 1, 4, 3]
Papan permainan punya siklus dengan dot berwarna sama
```

Gambar 10. Keluaran program dari input pada gambar 9. Sumber: Dokumentasi pribadi.

Penjelasan dan langkah-langkah pencapaian solusi dari program dijelaskan sebagai berikut:

Iterasi ke-	Rekursif ke-	v	u	Parent	Visited[]	Apakah warna

						sama?
1	1	0 (G)	3 (G)	-1	[0,…,0]	Sama
	2	3 (G)	4 (G)	0	[1,…,0]	Sama
	3	4 (G)	7 (P)	3	[1,0,0,1…,0]	Tidak sama
		4 (G)	1 (G)	3	[1,0,0,1…,0]	Sama
	4	1 (G)	2 (R)	4	[1,0,0,1,1…, 0]	Tidak sama
		1 (G)	0 (G)	4	[1,0,0,1,1…, 0]	Sama

Karena nilai parent tidak sama dengan nilai simpul u dan simpul u telah dikunjungi maka memenuhi syarat sebuah graf tak berarah yang punya siklus, sehingga papan game memang terdapat siklus *dots* yang warnanya sama. Stack akan mulai push nilai simpul u dan graf akan melakukan *backtracking*. Sehingga, siklus bisa dideteksi di daerah [0,1,4,3].

D. Kompleksitas Algoritma

Cukup sulit menghitung kompleksitas algoritmanya, dikarenakan mungkin saja ada *path* dimana simpul v melakukan rekursif terus-menerus dan menemukan simpul u yang warnanya sama tetapi pada akhirnya tidak membentuk siklus, mengingat kembali senarai visited akan selalu di-set jadi true, sehingga tidak bisa dikatakan kompleksitasnya adalah O(n) atau lebih dimana n adalah jumlah simpul.

Maka dari itu, penulis membagi kasus-kasus yang kira-kira akan terjadi:

1. *Worst case :* Dari penjelasan diatas, asumsi setiap simpul v punya rata-rata tetangga berjumlah 3, sedangkan jumlah simpul sejumlah n. Misalkan, bila jumlah tetangga dipukul rata sejumlah 3, karena jumlah simpul sebanyak n maka setiap v iterasi 3 kali. Sehingga, kompleksitasnya O(2n(V+E)), dimana V adalah jumlah simpul dan E adalah jumlah edge.

2. *Best case:* Kasus dimana ditemukan siklus di kotak [0,1, $\sqrt{n}$, $\sqrt{n}+1$], sehingga kompleksitasnya linear yaitu hanya O(4(V+E)) = O(V+E).

3. *Average case:* Sulit dan tidak terbayang kasus rata-ratanya. Tetapi, penulis menduga bahwa *average case* tidak akan jauh berbeda dengan *worst case*.

IV. KESIMPULAN

Algoritma DFS adalah salah satu materi penulusuran graf dari mata kuliah Strategi Algoritma yang telah dipelajari penulis ternyata memberikan banyak manfaat yang tidak disadari. Eksplorasi terhadap algoritma DFS dalam mendeteksi siklus memberikan ide bagi penulis untuk membuat program mengecek *dot* berwarna sama yang dapat membentuk bentuk tertutup. Sebenarnya, penulis ingin membuat program ini lebih efektif lagi yaitu mencari siklus terpanjang yang dapat dibuat dari papan permainan *dots* game ini. Tetapi, melihat beberapa sumber menyebutkan bahwa ternyata persoalan tersebut bukanlah persoalan yang mudah. Persoalan tersebut masuk ke dalam kategori *NP-Complete problem* sehingga itu diluar batasan kuliah dan diluar batasan kemampuan penulis lalu persoalan tersebut otomatis tidak bisa diselesaikan dengan Algoritma DFS karena DFS sendiri masuk ke kategori *P problem*.

Algoritma ini mungkin akan lebih baik didampingi dengan algoritma pencarian jalur terpanjang yang dapat dibentuk *dots* secara paralel (Misalkan: *thread 0* menjalankan algoritma pertama, *thread 1* menjalankan algoritma kedua) sehingga akan sangat *powerful* apabila diimplementasikan di A.I bot, bot bisa memilih *dots* mana yang memberikan skor tertinggi.

Mohon maaf apabila ada kesalahan dalam penulisan makalah ini, Terimakasih.

V. SARAN

Algoritma pencarian siklus *dots* diatas mungkin tidak cocok diterapkan pada tombol "*hint"* di game tersebut maka dari itu programmer PlayDots tidak menyertakan *hint* pada game mereka, ini dikarenakan mata manusia jauh lebih cepat menemukan *dots* yang bertetangga yang mayoritas warnanya sama. Tetapi bayangkan, apabila algoritma ini dikembangkan lebih lanjut dan ditambah algoritma pencarian membentuk garis terpanjang dari *dots* dan diterapkan ke program bot A.I, maka akan sangat berpotensi bot tersebut mendapatkan skor tertinggi. Sehingga, pemain bisa lebih tertantang untuk memecah skor tertinggi.

VI. UCAPAN TERIMAKASIH

Penulis ingin mengucapkan terima kasih yang sebesar-besarnya kepada Tuhan Yang Maha Esa, berkat rahmat-Nya penulis dapat menyelesaikan makalah ini tepat waktu. Penulsi juga ingin mengucapkan terimakasih kepada Bapak-Ibu Dosen Strategi Algoritma ITB yang telah membimbing penulis selama satu semester sehingga penulis mendapatkan banyak sekali manfaat dan pengetahuan dari mata kuliah ini, yaitu kepada Dr. Nur Ulfa Maulidevi, ST, M.Sc. Juga semua orang dan sumber yang telah membantu proses penyelesaian makalah ini sehingga penulis berhasil menyelesaikan makalah ini dengan tepat waktu.

LAMPIRAN

Dalam membuat makalah ini, penulis telah berhasil membuat program yang telah disinggung. Berkas-berkas yang telah dibuat telah ditaruh sepenuhnya di github penulis yaitu di halaman website https://github.com/ivanj09/Dots-A-Game-About-Connecting .

REFERENCES

[1] Munir, Rinaldi. Diktat Kuliah Strategi Algoritma.
[2] http://learningworksforkids.com/playbooks/dots-a-game-about-connecting/ diakses pada tanggal 10 Mei 2018 pukul 09:19 WIB.
[3] https://brilliant.org/wiki/depth-first-search-dfs/ diakses pada tanggal 10 Mei 2018 pukul 09:15 WIB.
[4] https://en.wikipedia.org/wiki/Depth-first_search diakses pada tanggal 10 Mei 2018 pukul 09:30 WIB.
[5] https://play.google.com/store/apps/details?id=com.nerdyoctopus.gamedots&hl=en diakses pada tanggal 10 Mei pukul 9:35 WIB.
[6] https://www.geeksforgeeks.org/detect-cycle-undirected-graph/ diakses pada tanggal 09 Mei 2018 pukul 19:34 WIB.

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, 10 Mei 2017

Ttd (scan atau foto ttd)

Ivan Jonathan
13516059