

Aplikasi Komponen Terhubung Kuat untuk Memeriksa Aksesibilitas Jalan pada Permainan Cities: Skylines

Michael Utama - 13521137¹
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
¹13521137@std.stei.itb.ac.id

Abstrak—*Cities: Skylines* merupakan sebuah permainan simulasi pengembangan kota yang dikembangkan oleh Colossal Order Ltd. dan diterbitkan oleh Paradox Interactive. Jalan pada permainan *Cities: Skylines* rawan terjadi kemacetan, sehingga diperlukan pengaturan jalan yang rumit. Pengaturan jalan tersebut berpotensi memunculkan jalan yang tidak terhubung. Makalah ini membahas cara memeriksa aksesibilitas jalan pada permainan *Cities: Skylines*.

Keywords—jalan, satu arah, *Cities: Skylines*, Kosaraju's algorithm, strongly connected component.

I. PENDAHULUAN

Cities: Skylines merupakan permainan simulasi pengembangan kota yang dibuat oleh Colossal Order Ltd. Permainan yang dirilis pada 2015 ini memiliki fitur layaknya permainan sejenis, seperti sistem jalan, penataan kota, kebijakan publik, keuangan, dan sistem kebutuhan dasar [1]. Selain permainan dasar, terdapat *downloadable content* yang berisi fitur-fitur tambahan seperti modifikasi bandara dan universitas.

Sistem transportasi pada *Cities: Skylines* merupakan salah satu keunikan gim ini. Transportasi dari luar kota ke dalam kota umumnya berupa jalan tol, kereta, kapal, dan bandara, sedangkan transportasi di dalam kota ditangani berbagai jenis jalan, bus, dan metro. Sistem transportasi ini terhubung dengan fitur lain pada permainan. Beberapa bangunan dapat menjadi terbengkalai jika tidak mendapat pasokan barang, yang dapat menurunkan pendapatan pajak kota.

Salah satu cara memastikan pasokan barang lancar adalah membuat sistem jalan yang optimal. *Cities: Skylines* memiliki fitur untuk memperbaiki arus kendaraan, seperti hierarki jalan yaitu *freeways*, *arterials*, sampai *local roads*. Selain itu, pemain juga dapat memanfaatkan *in-game tools* untuk membuat ruas jalan menjadi satu arah. Akan tetapi, perubahan arah jalan dapat mengakibatkan lokasi tertentu tidak dapat diakses.

Pada makalah ini, penulis akan membahas cara menentukan aksesibilitas jalan pada *Cities: Skylines* menggunakan *Strongly Connected Graph*.

II. LANDASAN TEORI

A. Graf

Graf merupakan himpunan tidak kosong dari simpul yang dihubungkan oleh sisi. Secara formal, graf $G = (V, E)$ sehingga V adalah himpunan simpul-simpul dan E adalah himpunan sisi [2].

Berdasarkan orientasi arah sisi, graf dapat dibagi menjadi dua jenis, yaitu

1. *Undirected graph*

Merupakan graf yang sisinya tidak memiliki orientasi arah.

2. *Directed graph*

Merupakan graf yang sisinya memiliki orientasi arah.

Berdasarkan keterhubungan simpul, graf dapat dibedakan menjadi empat jenis [2], yaitu

1. Graf terhubung

Merupakan graf dengan tiap pasang simpulnya terhubung.

2. Graf terhubung kuat

Merupakan *directed graph* dengan tiap pasang simpul u dan v terdapat lintasan dari u ke v dan dari v ke u .

3. Graf terhubung lemah

Merupakan *directed graph* yang tidak terhubung kuat namun terhubung pada *undirected graph*.

4. Graf tak-terhubung

Merupakan graf yang tidak terhubung. Pada *undirected graph* terdapat pasangan simpul u dan v tetapi tidak terdapat lintasan dari u ke v .

Beberapa istilah pada graf, diantaranya

1. Upagraf

Graf $G = (V, E)$ memiliki upagraf $G_1 = (V_1, E_1)$, sehingga $V_1 \subseteq V$ dan $E_1 \subseteq E$.

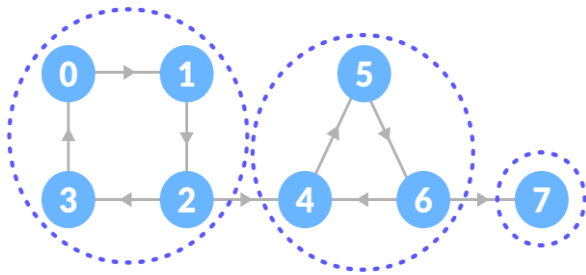
2. Terhubung

Simpul u dan v terhubung jika terdapat lintasan dari u ke v .

B. Komponen Terhubung Kuat

Komponen terhubung kuat merupakan upagraf maksimum pada *directed graph* yang terhubung kuat. Gambar 1 di bawah menunjukkan graf yang mempunyai tiga buah komponen

terhubung kuat.



Gambar 1. Komponen terhubung kuat pada graf berarah
 Sumber: <https://www.programiz.com/dsa/strongly-connected-components>

C. Depth First Search

Depth First Search (DFS) adalah algoritma *traversal* pada sebuah graf. Menurut [3], ide dasar algoritma DFS adalah

1. Pilih satu simpul sebagai *source* atau *root*, DFS akan mulai dari simpul ini.
2. Tandai simpul sekarang.
3. Cari simpul yang belum ditandai, pindah ke simpul tersebut.
4. Ulangi langkah 2-3 sampai tidak ditemukan simpul yang belum ditandai.

5. Lakukan *backtrack*, kembali ke simpul sebelumnya.
6. Kembali ke langkah 3.

Berdasarkan ide di atas, algoritma DFS dapat diimplementasikan menggunakan rekursi [3]. Kompleksitas DFS dengan rekursi adalah $O(V + E)$, dengan V adalah jumlah simpul dan E adalah jumlah sisi.

D. Algoritma Kosaraju

Algoritma Kosaraju merupakan cara untuk menemukan komponen terhubung kuat pada sebuah *directed graph*. Algoritma Kosaraju terdiri atas tiga bagian [4], yaitu

1. Lakukan DFS pada graf, catat urutan *post-order* dari DFS ke dalam stack S , sehingga simpul awal berada di posisi paling atas stack.
2. Balik graf dengan membuat graf G_1 sehingga untuk setiap sisi yang menghubungkan simpul u dan v pada G , terdapat sisi yang menghubungkan simpul v dan u pada G_1 .
3. Lakukan *traversal* pada G_1 dari nilai teratas pada stack S , lalu *pop* S . Seluruh simpul yang tercapai dari *traversal* tersebut merupakan sebuah komponen terhubung kuat. Ulangi sampai S kosong.

Graf berarah pada gambar 1 memiliki urutan *post-order* seperti pada tabel I.

Tabel I. Proses pencarian urutan *post-order* graf pada Gambar 1

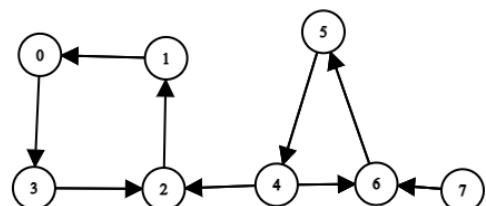
No	Posisi	Array visited	Urutan <i>post-order</i>
1	0	[1, 0, 0, 0, 0, 0, 0, 0]	[]
2	1	[1, 1, 0, 0, 0, 0, 0, 0]	[]
3	2	[1, 1, 1, 0, 0, 0, 0, 0]	[]
4	3	[1, 1, 1, 1, 0, 0, 0, 0]	[3]
5	2	[1, 1, 1, 1, 0, 0, 0, 0]	[3]
6	4	[1, 1, 1, 1, 1, 0, 0, 0]	[3]
7	5	[1, 1, 1, 1, 1, 1, 0, 0]	[3]
8	6	[1, 1, 1, 1, 1, 1, 1, 0]	[3]
9	7	[1, 1, 1, 1, 1, 1, 1, 1]	[3, 7]
10	6	[1, 1, 1, 1, 1, 1, 1, 1]	[3, 7, 6]
11	5	[1, 1, 1, 1, 1, 1, 1, 1]	[3, 7, 6, 5]
12	4	[1, 1, 1, 1, 1, 1, 1, 1]	[3, 7, 6, 5, 4]
13	2	[1, 1, 1, 1, 1, 1, 1, 1]	[3, 7, 6, 5, 4, 2]
14	1	[1, 1, 1, 1, 1, 1, 1, 1]	[3, 7, 6, 5, 4, 2, 1]
15	0	[1, 1, 1, 1, 1, 1, 1, 1]	[3, 7, 6, 5, 4, 2, 1, 0]

Algoritma ini dapat bekerja karena pada urutan *post-order* DFS, seluruh simpul yang dapat dicapai suatu simpul v terletak sebelum v [4]. Artinya, jika urutan dibalik, seluruh simpul u setelah v adalah salah satu dari dua kemungkinan berikut.

1. Simpul u dapat diakses dari v , sehingga jika terdapat lintasan dari v ke u pada G_1 , kedua komponen dipastikan terhubung kuat.
2. Simpul u tidak dapat diakses dari v . Dapat dipastikan simpul u dan v tidak terhubung dan sebaliknya, sehingga tidak mungkin ada lintasan dari v ke u pada G_1 .







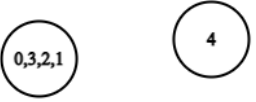
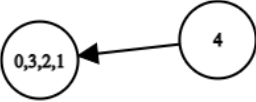
Penggunaan stack berisi DFS *post-order* pada graf di gambar 1 dijelaskan pada tabel II. Graf komponen terhubung kuat pada

tabel II masih dalam keadaan terbalik akibat pembalikan graf awal. Pencarian komponen terhubung kuat memerlukan hasil pembalikan graf yang terdapat pada gambar 2.



Gambar 2. Hasil pembalikan graf pada gambar 1
 Sumber: dokumentasi pribadi

Tabel II. Proses pencarian komponen terhubung kuat graf pada Gambar 1

No	Posisi	Array visited	Stack urutan <i>post-order</i>	Graf komponen terhubung kuat
1	-	[0, 0, 0, 0, 0, 0, 0, 0]	[0, 1, 2, 4, 5, 6, 7, 3]	-
2	0	[1, 0, 0, 0, 0, 0, 0, 0]	[1, 2, 4, 5, 6, 7, 3]	
3	3	[1, 0, 0, 1, 0, 0, 0, 0]	[1, 2, 4, 5, 6, 7, 3]	
4	2	[1, 0, 1, 1, 0, 0, 0, 0]	[1, 2, 4, 5, 6, 7, 3]	
5	1	[1, 1, 1, 1, 0, 0, 0, 0]	[1, 2, 4, 5, 6, 7, 3]	
6	-	[1, 1, 1, 1, 0, 0, 0, 0]	[2, 4, 5, 6, 7, 3]	
7	-	[1, 1, 1, 1, 0, 0, 0, 0]	[4, 5, 6, 7, 3]	
8	4	[1, 1, 1, 1, 1, 0, 0, 0]	[5, 6, 7, 3]	
9	2	[1, 1, 1, 1, 1, 0, 0, 0]	[5, 6, 7, 3]	

10	6	[1, 1, 1, 1, 1, 0, 1, 0]	[5, 6, 7, 3]	
11	5	[1, 1, 1, 1, 1, 1, 1, 0]	[5, 6, 7, 3]	
12	-	[1, 1, 1, 1, 1, 1, 1, 0]	[6, 7, 3]	
13	-	[1, 1, 1, 1, 1, 1, 1, 0]	[7, 3]	
14	7	[1, 1, 1, 1, 1, 1, 1, 1]	[3]	
15	6	[1, 1, 1, 1, 1, 1, 1, 1]	[3]	
16	-	[1, 1, 1, 1, 1, 1, 1, 1]	[]	

III. APLIKASI KOMPONEN TERHUBUNG KUAT UNTUK MEMERIKSA AKSESIBILITAS JALAN

A. Implementasi Program

Program pembuat graf komponen terhubung kuat dibuat dengan algoritma Kosaraju menggunakan bahasa C++. Karena ukuran kode cukup kecil, penulis memuat seluruh kodenya dalam satu file cpp dengan bagian preproses dan deklarasi seperti pada gambar 3.

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 // Representasi graf berupa adjacency list
6 typedef vector<vector<int>> graph;
7
8 // Variabel Global
9 graph Graph, GraphReverse, GraphSCC;
10 vector<vector<int>> scc;
11 stack<int> order;
12 vector<bool> vis;
13 vector<int> newVertex;
```

Gambar 3. Bagian preproses dan deklarasi file SCC.cpp
Sumber: dokumentasi pribadi

Implementasi program pembuat graf komponen terhubung kuat dibagi menjadi beberapa fungsi, yaitu pengisian stack DFS *post-order* pada gambar 4, pembalikan graf pada gambar 5, dan DFS untuk menemukan komponen terhubung kuat pada gambar 6.

```
1 void getDfsOrder(int v)
2 /* I.S. v anggota Graph, ukuran vis >= ukuran Graph */
3 /* F.S. mengisi stack order dengan urutan dfs */
4 {
5     // Jika simpul sudah pernah dikunjungi, tidak melakukan traversal
6     if(vis[v]) return;
7     vis[v] = true;
8
9     // Kunjungi seluruh simpul yang terhubung
10    for(auto x : Graph[v]) {
11        getDfsOrder(x);
12    }
13
14    // Masukkan simpul ke stack (post-order)
15    order.push(v);
16 }
```

Gambar 4. Prosedur *getDfsOrder*
Sumber: dokumentasi pribadi

```
1 graph Reverse(const graph &g)
2 /* Mengembalikan graf yang memiliki seluruh simpul graf g tetapi arahnya terbalik */
3 {
4     // Membuat graf dengan jumlah node = jumlah node awal
5     graph ret(g.size());
6
7     // Memasukkan seluruh simpul graf g ke ret
8     for(int i = 0; i < g.size(); i++) {
9         for(auto &x: g[i]) {
10            ret[x].push_back(i);
11        }
12    }
13
14    return ret;
15 }
```

Gambar 5. Fungsi *Reverse*
Sumber: dokumentasi pribadi

```
1 void DfsUtil(int v)
2 {
3     // Jika simpul sudah pernah dikunjungi, tidak melakukan traversal
4     if(vis[v]) return;
5     vis[v] = true;
6     scc.back().push_back(v);
7     newVertex[v] = scc.size() - 1;
8
9     // Kunjungi seluruh simpul yang terhubung
10    for(auto x : GraphReverse[v]) {
11        DfsUtil(x);
12    }
13 }
```

Gambar 6. Prosedur *DfsUtil*
Sumber: dokumentasi pribadi

Ketiga fungsi/prosedur di atas disatukan dengan sebuah prosedur *SCC()* yang menjalankan seluruh langkah algoritma Kosaraju, konversi sisi dari graf awal ke graf komponen terhubung kuat, dan memanggil prosedur *outputHasil()*.

```
1 void SCC()
2 {
3     // Inisialisasi nilai variabel
4     int n = Graph.size();
5     GraphReverse = Reverse(Graph);
6     vis = vector<bool>(n, 0);
7     newVertex = vector<int>(n);
8
9     // Cari dfs-order graph
10    for(int i = 0; i < n; i++) {
11        getDfsOrder(i);
12    }
13
14    // Pakai hasil order untuk mencari connected component
15    vis = vector<bool>(n, 0);
16    while(!order.empty()) {
17        int root = order.top();
18        order.pop();
19        if(vis[root]) continue;
20
21        scc.push_back(vector<int>());
22        GraphSCC.push_back(vector<int>());
23        DfsUtil(root);
24    }
25
26    // Tambahkan edge ke graf SCC
27    for(int i = 0; i < n; i++) {
28        for(auto &dest: Graph[i]) {
29            if(newVertex[i] != newVertex[dest]) {
30                bool found = false;
31                for(auto &newDest: GraphSCC[newVertex[i]]) {
32                    if(newDest == newVertex[dest])
33                        found = true;
34                }
35                if(!found) GraphSCC[newVertex[i]].push_back(newVertex[dest]);
36            }
37        }
38    }
39
40    outputHasil();
41
42 }
```

Gambar 7. Prosedur *SCC*
Sumber: dokumentasi pribadi

```

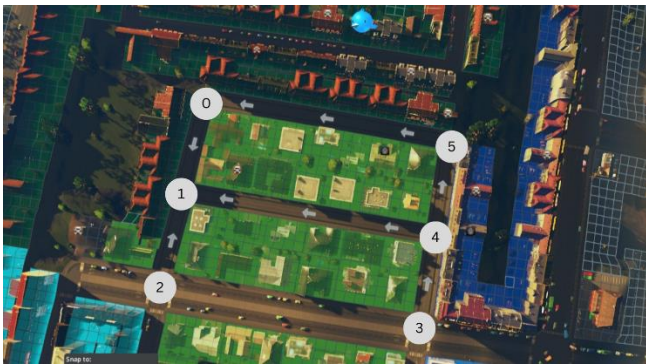
1 void outputHasil()
2 {
3     for(int i = 0; i < scc.size(); i++) {
4         cout << "Simpul " << i << ": ";
5         for(int j = 0; j < scc[i].size(); j++) {
6             if(j) cout << ", ";
7             cout << scc[i][j];
8         }
9         cout << '\n';
10    }
11    cout << "\nDaftar sisi:\n";
12    for(int i = 0; i < GraphSCC.size(); i++) {
13        for(int &dest: GraphSCC[i]) {
14            cout << i << "->" << dest << '\n';
15        }
16    }
17 }

```

Gambar 8. Prosedur outputHasil
 Sumber: dokumentasi pribadi

B. Uji Kasus

Proses uji kasus dilakukan terhadap gambar peta dari permainan *Cities: Skylines*. Pertama-tama, dilakukan pemodelan dari bentuk gambar menjadi graf seperti pada gambar 9.



Gambar 9. Jalan satu arah di *Cities: Skylines* dengan modifikasi

Sumber: dokumentasi pribadi

Terdapat enam simpul pada gambar 9 yang masing-masing melambangkan persimpangan jalan, dinomori 0 sampai 5. Simpul-simpul tersebut dihubungkan dengan delapan sisi, yang ditunjukkan oleh tabel III.

Tabel III. Daftar sisi pada graf dari gambar 9

No	Source	Destination
1	0	1
2	2	3
3	3	2
4	2	1
5	3	4
6	4	5
7	5	0
8	4	1

Jumlah simpul, jumlah sisi, dan daftar sisi kemudian dimasukkan ke dalam program, sehingga didapat hasil pada gambar 10. Graf komponen terhubung kuat pada keluaran program memiliki lebih dari satu simpul, sehingga graf asal

tidak terhubung kuat. Dengan kata lain, desain jalan pada gambar 9 menyebabkan beberapa ruas jalan tidak dapat diakses.

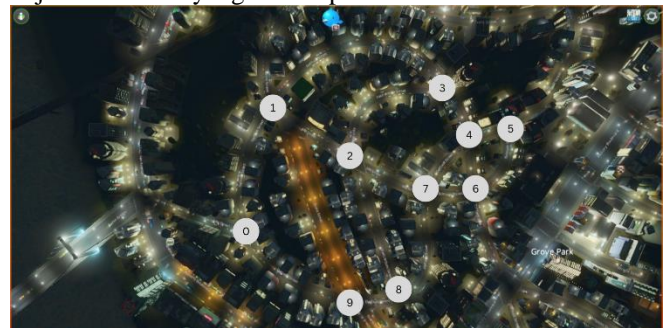
```

6 8
0 1
2 3
3 2
2 1
3 4
4 5
5 0
4 1
Simpul 0: 2, 3
Simpul 1: 4
Simpul 2: 5
Simpul 3: 0
Simpul 4: 1
Daftar sisi:
0->4
0->1
1->2
1->4
2->3
3->4

```

Gambar 10. Masukan dan keluaran kasus uji 1
 Sumber: dokumentasi pribadi

Pada kasus uji kedua, peta yang digunakan hanya memiliki jalan dua arah. Pada gambar 11 terlihat ada sepuluh persimpangan jalan pada daerah yang diperhatikan, disambung oleh jalan dua arah yang dicatat pada tabel IV.



Gambar 11. Jalan dua arah di *Cities: Skylines* dengan modifikasi

Sumber: dokumentasi pribadi

Tabel IV. Daftar sisi pada graf dari gambar 11

No	Source	Destination
1	0	1
2	1	0
3	1	2
4	2	1
5	1	3
6	3	1
7	2	3
8	3	2
9	3	4
10	4	3
11	4	5
12	5	4
13	4	6
14	6	4
15	4	7
16	7	4

17	5	6
18	6	5
19	6	7
20	7	6
21	2	7
22	7	2
23	2	8
24	8	2
25	7	8
26	8	7
27	8	9
28	9	8
29	0	9
30	9	0
31	1	9
32	9	1

Data jumlah dan daftar sisi dimasukkan ke dalam program, sehingga didapat keluaran pada gambar 12. Graf hanya memiliki satu komponen terhubung kuat, sehingga setiap ruas jalan dapat diakses dari posisi manapun.

```
Simpul 0: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Daftar sisi:
```

Gambar 12. Keluaran kasus uji 2
Sumber: dokumentasi pribadi

IV. KESIMPULAN

Komponen terhubung kuat merupakan upagraf maksimum dari suatu graf berarah yang tiap simpulnya saling terhubung kuat. Sebuah graf terhubung kuat jika graf hanya memiliki satu komponen terhubung kuat. Graf terhubung kuat berarti seluruh simpul dapat diakses dari simpul lain, secara tidak langsung berarti seluruh sisi dapat diakses dari simpul manapun.

Konsep *strongly connected component* (SCC) berguna untuk pemodelan jalan sebab setiap jalan harus dapat diakses dari lokasi manapun. Dengan melambangkan persimpangan jalan menjadi simpul dan jalan menjadi sisi berarah, aksesibilitas jalan dapat diketahui dengan menghitung jumlah SCC pada graf. Jika graf memiliki lebih dari satu SCC, ada setidaknya satu jalan yang tidak dapat diakses dari suatu lokasi.

Pada kasus uji pertama, terdapat lima komponen terhubung kuat. Artinya, ada ruas jalan yang tidak dapat diakses dari lokasi tertentu. Hal ini dapat terlihat pada gambar 9, kendaraan yang berada di posisi simpul 1 tidak dapat bergerak.

Pada kasus uji kedua, terdapat satu komponen terhubung kuat pada graf. Artinya, seluruh ruas jalan dapat diakses dari lokasi manapun.

V. UCAPAN TERIMA KASIH

Penulis ingin mengucapkan puji dan syukur ke hadirat Tuhan Yang Maha Esa karena atas rahmat dan berkat-Nya penulis dapat menyelesaikan makalah “Aplikasi Komponen Terhubung Kuat untuk Memeriksa Aksesibilitas Jalan pada Permainan

Cities: Skylines”. Penulis ingin menyampaikan terima kasih kepada Dr. Fariska Zakhralativa Ruskanda, Dr. Nur Ulfa Maulidevi, dan Dr. Rinaldi Munir selaku dosen pengampu mata kuliah IF2120 Matematika Diskrit atas bimbingan dan pengajarannya sehingga penulis dapat menyelesaikan makalah. Penulis juga ingin mengucapkan terima kasih kepada orang tua dan rekan penulis yang telah memberikan berbagai dukungan kepada penulis.

REFERENSI

- [1] Cities: Skylines. Paradox Interactive. (n.d.). <https://www.paradoxinteractive.com/games/cities-skylines/about>. [diakses 10 Desember 2022]
- [2] R. Munir. “Graf (Bag.1)”. <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf>. [diakses 11 Desember 2022]
- [3] geeksforgeeks.com, “Depth First Search or DFS for a Graph”. <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>. [diakses 11 Desember 2022]
- [4] geeksforgeeks.com, “Strongly Connected Components”. <https://www.geeksforgeeks.org/strongly-connected-components/>. [diakses 11 Desember 2022]

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, 12 Desember 2022



Michael Utama (13521137)