

# Pemanfaatan Graf untuk Membuat Tabel Transisi *Deterministic Finite Automaton* Permainan “*The Sims Simulator*”

Moch. Nafkhan Alzamzami - 13518132  
Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia  
13518132@std.stei.itb.ac.id

**Abstrak**—Pembuatan tabel transisi *deterministic finite automaton* dapat dilakukan menggunakan pemanfaatan teori graf, yaitu dengan menggunakan algoritma *breadth-first search*. Makalah ini berisi tentang pembuatan tabel transisi *deterministic finite automaton* menggunakan algoritma *breadth-first search*.

**Kata Kunci**—*Deterministic Finite Automaton*, Pohon, Graf, Algoritma, Struktur Data, Breadth-First Search.

## I. PENDAHULUAN

Pada bulan September 2019 lalu, mahasiswa Teknik Informatika Institut Teknologi Bandung angkatan 2018 mendapatkan tugas besar mata pelajaran Teori Bahasa Formal dan Otomata untuk membuat permainan berjudul “*The Sims Simulator*” beserta tabel transisi *deterministic finite automaton* dari permainan tersebut. Dalam makalah ini, saya memanfaatkan algoritma *breadth-first search* dalam pembuatan tabel *deterministic finite automaton*.

## II. DASAR TEORI

### A. *Deterministic Finite Automaton*

DFA atau *Deterministic Finite Automaton* adalah salah satu cabang dari teori komputasional. Secara definisi, DFA adalah sebuah mesin yang berisi sekumpulan tahapan dan kondisi yang dapat menerima dan menolak *string* ataupun simbol-simbol dan menjalankan proses komputasi untuk setiap elemen dari *string*. Kata *Deterministic* sendiri merujuk kepada keunikan dari setiap proses yang dijalankan pada setiap kondisi berbeda yang dalam kasus DFA sering disebut dengan *finite-state*. DFA ini sering sekali dikaitkan dengan teori komputasional karena konsep ini pertama kali dipakai pada salah satu mesin komputasional paling berpengaruh dalam sejarah, yaitu The Turing Machine. Dalam proses komputasi, mesin komputasi menyortir antara True dan False secara berulang ulang hingga mencapai sebuah kondisi tertentu. Proses ini merupakan salah satu cakupan DFA.

Secara definisi formal, DFA dapat direpresentasikan dengan beberapa unsur, yaitu:

#### 1. State ( $Q$ )

*State* merupakan kondisi-kondisi yang terdefinisi dalam DFA yang biasanya digambarkan dengan *Node* yang berubah-ubah berdasarkan *Alphabet* masukannya.

#### 2. Alphabet ( $\Sigma$ )

*Alphabet* adalah sekumpulan simbol yang didapat dari masukan/*Input*.

#### 3. Initial State ( $q_0$ )

*Initial State* adalah *state* awal mesin sebelum menerima masukan.

#### 4. Final State ( $F$ )

*Final State* adalah *state* akhir yang diterima oleh mesin.

#### 5. Transition Function ( $\delta$ )

*Transition Function* menunjukkan *state* awal yang akan diberi aksi  $a$  sehingga akan mencapai *state* hasil. *Transition Function* dituliskan sebagai berikut :

$$\delta(q, a) = b$$

$\delta$  = tanda *Transition Function*

$q$  = *state* awal sebelum menerima masukan

$a$  = *Alphabet Input*

$b$  = *state* hasil setelah menerima masukan

Selain itu, terdapat representasi lain dan *Transition Function*, yaitu *Extended Transition Function* ( $\delta$ ). Jenis fungsi ini dapat menerima beberapa masukan *alphabet* sekaligus. Contoh sebagai berikut :

$$\delta(q, wa) = \delta(\delta(q, w), a)$$

$\delta$  = tanda *extended transition function*

$q$  = *state* awal sebelum menerima masukan

$w$  = *alphabet* pertama yang dibaca

$a$  = *alphabet* setelahnya yang dibaca

Dalam merepresentasikan DFA, cara yang paling umum digunakan adalah dengan membentuk graf yang disebut *state diagram*. *State diagram* ini merepresentasikan *state-state* tertentu dan menunjukkan proses Bergeraknya string dengan masukan masukan *alphabet*. Pembuatan *state diagram* selalu diikuti dengan *Transition Table*. *Transition Table* ini menunjukkan *state* yang tersedia yang dipadukan dengan aksi. Berikut adalah komposisi dari *State Diagram*:

1. *Nodes*

*Nodes* merupakan representasi *state* pada graf. *Nodes* ditandakan dengan lingkaran. Di dalam node terdapat simbol-simbol yang membedakan antara *state* tertentu. Namun, terdapat *Node* khusus, yaitu :

- *Node Initial State*, yang ditandakan dengan diberi panah yang sebelumnya kosong pada *state* tersebut.
- *Node Final State*, yang ditandakan dengan lingkaran ganda pada *Node* tersebut.

2. Busur

Busur merupakan representasi arah untuk sebuah aksi yang diambil. Aksi merupakan proses pembacaan *alphabet*. Busur menunjukkan arah *state* tujuan dengan aksi tertentu.

Sebagai contoh kasus:

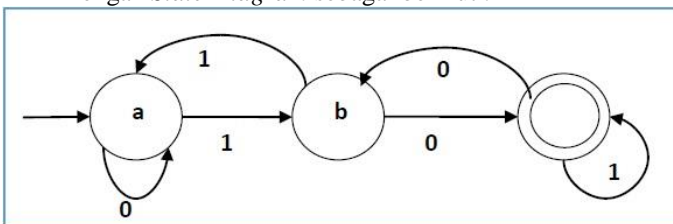
DFA dengan

$Q = \{a, b, c\}$ ,  
 $\Sigma = \{0, 1\}$ ,  
 $q_0 = \{a\}$   
 $F = \{c\}$ , dan

Fungsi Transisi ( $\delta$ ) dengan tabel transisi sebagai berikut:

State Awal	State setelah input 0	State setelah input 1
→ a	a	b
b	c	a
*c	b	c

Dengan *State Diagram* sebagai berikut :

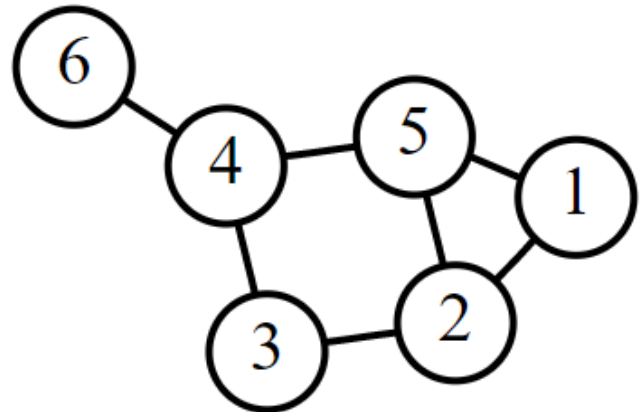


Sumber: [https://irvanfahreza.files.wordpress.com/2019/04/photo\\_2019-04-03\\_21-13-26.jpg](https://irvanfahreza.files.wordpress.com/2019/04/photo_2019-04-03_21-13-26.jpg)

B. Graf

Dalam matematika dan ilmu komputer, sebuah graf adalah objek dasar pelajaran dalam teori graf. Dalam bahasa sehari-hari, sebuah graf adalah himpunan dari objek-objek yang dinamakan titik, simpul, atau sudut dihubungkan oleh penghubung yang dinamakan garis atau sisi. Dalam graf yang memenuhi syarat, di mana biasanya tidak berarah, sebuah garis dari titik A ke titik B dianggap sama dengan garis dari titik B ke titik A. Dalam graf berarah, garis tersebut memiliki arah. Pada dasarnya, sebuah graf digambarkan dengan bentuk diagram sebagai himpunan dari titik-titik (sudut atau simpul) yang digabungkan dengan kurva (garis atau sisi).

Graf adalah sebuah pasangan  $G = (V, E)$ , di mana  $V$  adalah himpunan elemen yang disebut simpul, dan  $E$  adalah himpunan pasangan dari dua simpul, yang tiap elemennya disebut dengan sisi.



Sumber: <https://upload.wikimedia.org/wikipedia/commons/5/5b/6n-graf.svg>

III. DESKRIPSI PERSOALAN

Permainan *The Sims Simulator* dapat direpresentasikan dalam bentuk finite automata yang memiliki state dengan atribut-atribut sebagai berikut:

1. **Hygiene**, gabungan dari atribut *Hygiene* dan *Bladder* dalam *The Sims*.
2. **Energy**, gabungan dari atribut *Energy* dan *Hunger* dalam *The Sims*.
3. **Fun**, gabungan dari atribut *Fun* dan *Social* dalam *The Sims*.

Setiap atribut memiliki nilai maksimum 15 dan nilai minimum 0. Setiap atribut dapat bertambah (sampai nilai maksimum) atau berkurang (sampai nilai minimum) jika pemain melakukan aksi tertentu. Kondisi awal pemain selalu dalam keadaan sudah bangun tidur dengan atribut *Hygiene* bernilai 0, *Energy* bernilai 10, dan *Fun* bernilai 0. Permainan dinyatakan selesai jika semua atribut bernilai 0 atau semua atribut bernilai 15.

Berikut ini adalah tabel aksi yang dapat dilakukan, rentang waktu untuk melakukan aksi tersebut, dan konsekuensinya:

No	Aksi	Jenis	Konsekuensi
1	Tidur <jenis_tidur>	Siang	+10 <i>energy</i>
		Malam	+15 <i>energy</i>
2	Makan <jenis_makanan>	<ul style="list-style-type: none"> <li>• <i>Hamburger</i></li> <li>• <i>Pizza</i></li> <li>• <i>Steak and Beans</i></li> </ul>	<i>Hamburger:</i> +5 <i>energy</i>  <i>Pizza:</i> +10 <i>energy</i>

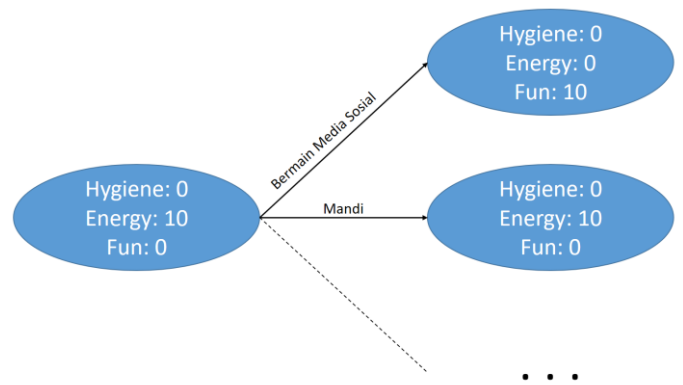
			<i>Steak and Beans:</i> +15 energy
3	Minum <jenis_minuman>	<ul style="list-style-type: none"> <li>• Air</li> <li>• Kopi</li> <li>• Jus</li> </ul>	Air: -5 hygiene  Kopi: +5 energy -10 hygiene  Jus: +10 energy -5 hygiene
4	Buang Air <jenis>	Kecil	+5 hygiene
		Besar	+10 hygiene -5 energy
5	Bersosialisasi ke Kafe	-	+15 fun -10 energy -5 hygiene
6	Bermain Media Sosial	-	+10 fun -10 energy
7	Bermain komputer	-	+15 fun -10 energy
8	Mandi	-	+15 hygiene -5 energy
9	Cuci Tangan	-	+5 hygiene
10	Mendengarkan Musik di Radio	-	+10 fun -5 energy
11	Membaca <jenis_bacaan>	Jenis Bacaan:	Koran: +5 fun -5 energy
		<ul style="list-style-type: none"> <li>• Koran</li> <li>• Novel</li> </ul>	Novel: +10 fun -5 energy

#### IV. PEMBAHASAN

Dalam pengerjaan program *The Sims Simulator*, saya mengimplementasikannya dalam bentuk DFA dengan menggunakan bahasa pemrograman java. Saya memilih bahasa pemrograman java karena java merupakan bahasa pemrograman berorientasi objek, sehingga *state-state* dalam DFA dapat direpresentasikan dalam sebuah objek dan dapat dioperasikan dengan mudah.

Bentuk *state* dalam graf direpresentasikan menjadi untuk setiap state adalah suatu simpul dan tiap *state* yang apabila dapat mencapai *state* lain setelah diberi aksi, merupakan sisi antara kedua *state* tersebut dalam graf.

Sebagian dari graf tersebut dapat direpresentasikan seperti berikut:



Program *The Sims Simulator* mempunyai beberapa kelas sebagai berikut:

#### 1. Kelas State

// Kelas State sebagai objek state yang mempunyai properti hygiene untuk nilai hygiene, energy untuk nilai energy, dan fun untuk nilai fun.

```
public class State {
    // Nilai minimum dan maksimum hygiene, energy, dan fun.
    public static final int MAX_VALUE = 15,
        MIN_VALUE = 0;
    // Error yang akan dihasilkan apabila input tidak valid/tidak ada.
    public static final RuntimeException
    NOT_VALID_EXCEPTION,
    NOT_FOUND_EXCEPTION;
    // Nilai-nilai pada state ini.
    public int hygiene, energy, fun;

    // Konstruktor State dengan nilai hygiene=0, energy=10, fun=0.
    public State() { ... }

    // Konstruktor State.
    public State(int hygiene, int energy, int fun) { ... }

    // Prosedur melakukan aksi dengan input aksi berbentuk string.
    public State act(String input) { ... }

    // Fungsi yang menghasilkan bentuk state dalam format string <hygiene,energy,fun>.
    public String toString() { ... }

    // Fungsi yang menghasilkan string output informasi status hygiene, energy, dan fun.
    public String printString() { ... }

    // Fungsi yang menghasilkan true apabila fungsi isDead atau isFull adalah true.
    public boolean isFinalState() { ... }

    // Fungsi yang menghasilkan true apabila nilai hygiene, energy, dan fun adalah 0.
    public boolean isDead() { ... }
```

```

// Fungsi yang menghasilkan true apabila nilai hygiene,
energy, dan fun adalah 15.
public boolean isFull() { ... }

// Fungsi untuk menambahkan nilai hygiene, energy, dan
fun.
public State add(int h, int e, int f) { ... }

// Menghasilkan sebuah state dari sebuah string yang
sesuai format.
public static State parse(String str) { ... }

// Menghasilkan bentuk hash state ini.
public int hashCode() { ... }

// Menghasilkan true apabila komparasi kedua state sama.
public boolean equals(Object obj) { ... }
}

```

## 2. Kelas ConsoleApp

```

// Kelas ConsoleApp merupakan kelas utama untuk
menjalankan program.
public class ConsoleApp {
    // Prosedur main yang dijalankan program pertama kali.
    // Prosedur ini menjalankan program dengan memulai dari
    // state awal hygiene=0, energy=10, dan fun=0. Kemudian
    // menerima input aksi dari pengguna sampai mencapai final
    // state, kemudian keluar dari program.
    public static void main(String[] args) throws Exception {
        ... }

    // List aksi yang tersedia.
    private static String[] actList = { ... };

    // Prosedur untuk membuat tabel dfa di file csv.
    private static void genscv() throws Exception { ... }
}

```

Dalam proses mengubah persoalan tersebut menjadi tabel transisi DFA, saya menggunakan algoritma *breadth-first search* yang di mana *state-state* dalam DFA tersebut direpresentasikan dalam bentuk graf dahulu. Dimulai dari *state* awal, dilakukan semua macam aksi yang ada, dan hasil *state* selanjutnya disimpan pada tabel. Kemudian untuk tiap *state* hasil, dilakukan semua aksi yang sama lagi seperti sebelumnya, sehingga didapatkan semua kemungkinan *state* yang bisa didapat serta transisi-transisi DFA-nya.

Implementasi algoritma tersebut dilakukan di fungsi `genscv`:

```

/**
 * Prosedur untuk membuat tabel dfa di file csv.
 */
private static void genscv() throws Exception {
    State q0 = new State();
    Queue<State> q = new LinkedList<>();
    List<List<String>> res = new ArrayList<>();
    res.add(new ArrayList<>());
}

```

```

res.get(0).add("State");
for (String s : actList)
    res.get(0).add(s);
HashSet<State> done = new HashSet<>();
q.add(q0);
done.add(q0);
while (!q.isEmpty()) {
    State now = q.poll();
    List<String> list = new ArrayList<>();
    list.add("\\" + (now.equals(q0) ? "→" : now.isFinalS
tate() ? "*" : "")) + now.toString() + "\\");
    for (int i = 0; i < actList.length; i++) {
        State next = now.act(actList[i]);
        list.add("\\" + (next == null ? now.toString() : nex
t.toString()) + "\\");
        if (next == null)
            continue;
        if (!done.contains(next)) {
            done.add(next);
            q.add(next);
        }
    }
    res.add(list);
}
FileWriter writer = new FileWriter(new File("generate
d_dfa.csv"));
for (int i = 0; i < res.size(); i++) {
    if (i > 0)
        writer.append("\n");
    for (int j = 0; j < res.get(i).size(); j++) {
        if (j > 0)
            writer.append(",");
        writer.append(res.get(i).get(j));
    }
    writer.close();
}
}

```

Setelah mengimplementasikan algoritma *breadth-first search* tersebut dan mendapatkan hasil tabel transisi DFA-nya, saya mendapatkan ada 64 *state* yang terdapat 1 *starting state* dan 2 *final state*. Pada *final state* di permainan *The Sims Simulator*, program akan keluar dan tidak akan menerima input kembali.

Dalam implementasi algoritma *breadth-first search*, saya menggunakan data struktur *Queue* dan *Set* yang akan diisi oleh *state* DFA. Langkah algoritmanya adalah sebagai berikut:

1. Membuat 2 objek data struktur *Queue* dan *Set*. *Queue* digunakan untuk urutan simpul yang akan dituju selanjutnya. *Set* digunakan untuk mengetahui simpul mana saja yang sudah dituju.

```
Queue<State> q = new LinkedList<>();
HashSet<State> done = new HashSet<>();
```

2. Mendefinisikan *state* awal, yaitu untuk kasus ini adalah  $\langle 0, 10, 0 \rangle$ .

```
State q0 = new State();
```

3. Menambahkan *state* awal tersebut ke dalam *Queue* dan *Set*.

```
q.add(q0);
done.add(q0);
```

4. Melakukan iterasi untuk semua kemungkinan aksi yang mungkin dilakukan dan menuliskan hasilnya ke dalam tabel transisi DFA.

```
State next = now.act(actList[i]);
list.add("\\" + (next == null ? now.toString() : next.
toString()) + "\\");
```

- a. Apabila aksi tersebut tidak valid, maka akan kembali ke *state* itu sendiri.
- b. Apabila aksi tersebut valid, maka akan didapat *state* selanjutnya.

5. Menambahkan *state* yang didapat ke dalam *Queue* dan *Set* apabila *state* tersebut tidak terdapat dalam *Set*.

```
if (!done.contains(next)) {
    done.add(next);
    q.add(next);
}
```

6. Kemudian:

```
while (!q.isEmpty()) { ... }
```

- a. Apabila data *Queue* masih mempunyai *state*, maka kembali lagi ke langkah no. 4.
- b. Apabila data *Queue* sudah kosong, maka semua kemungkinan *state* beserta transisi-transisinya sudah dilalui.

7. Menuliskan hasilnya ke tabel transisi *Deterministic Finite Automaton*.

Hasil transisi tabel yang dihasilkan adalah sebagai berikut:



Dari analisis tersebut, terdapat 64 states berbeda dalam kasus persoalan ini yang terdefinisi dari tiga aspek berbeda, yaitu Hygiene, Energy, dan Fun yang masing-masing memiliki empat kemungkinan nilai, yaitu 0, 5, 10, dan 15. Maka, terdapat maksimal  $4^3 (= 64)$  kemungkinan kondisi yang terjadi. Serta, terdapat 18 kemungkinan aksi yang dapat dilakukan dari setiap state-nya, namun beberapa fungsi transisi dapat menjadi tidak valid jika salah satu nilai dari ketiga aspek bernilai kurang dari nol atau pun nilainya lebih dari 15.

## V. KESIMPULAN

Teori graf memiliki banyak sekali kegunaan yang dapat diaplikasikan dalam dunia komputasi. Salah satu manfaatnya adalah pembuatan tabel transisi *Deterministic Finite Automaton*. Salah satu algoritma yang memanfaatkan graf adalah *Breadth-first search*. *Breadth-first search* merupakan algoritma yang tepat dalam pembuatan tabel transisi *Deterministic Finite Automaton* karena dapat menghasilkan tabel transisi dari state-state yang pasti akan dicapai, sehingga dapat menghindari kesalahan dalam penulisan state di tabel.

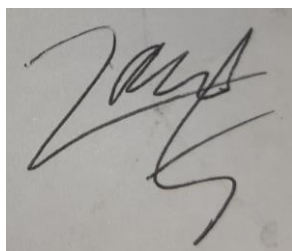
## REFERENSI

- [1] Hopcroft, John E.; Motwani, Rajeev; Ullman, Jeffrey D. (2001). *Introduction to Automata Theory, Languages, and Computation* (2 ed.). Addison Wesley. ISBN 0-201-44124-1. Retrieved 19 November 2012.
- [2] Balakrishnan, V. K. (1997). *Graph Theory* (1st ed.). McGraw-Hill. ISBN 978-0-07-005489-9.

## 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, 4 Desember 2019



Moch. Nafkhan Alzamzami - 13518132