

## BAB II

### TINJAUAN PUSTAKA

Bab ini berisi penjabaran teori yang berkaitan dengan algoritma yang digunakan dalam implementasi program. Penjelasan dalam bab ini berguna untuk memberikan gambaran bagaimana program yang dibuat bekerja dan diharapkan dengan adanya penjelasan berikut pengembangan metode yang diimplementasikan akan lebih mudah.

#### 2.1 Regular Expression

Sebuah *string* didefinisikan sebagai rangkaian karakter berhingga yang ada dalam semesta simbol  $\Sigma$ . Sebagai contoh FAZZIBASHASOL adalah *string* dalam semesta simbol  $\Sigma = \{A, B, F, H, I, L, O, S, Z\}$  [1]. *Regular expression* adalah salah satu cara yang sering digunakan untuk merepresentasikan pola pencarian yang lebih kompleks dibandingkan dengan sebuah *string* ataupun sekumpulan *string*. *Regular expression* didefinisikan sebagai berikut.

*regular expression*  $RE$  adalah sebuah *string* yang memiliki simbol dalam set  $\Sigma \cup \{\epsilon, |, *, \cdot, (\ ),\}$ , yang didefinisikan secara rekursif sebagai *string* kosong  $\epsilon$ , simbol  $\alpha$  di mana  $\alpha \in \Sigma$ , dan  $(RE_1)$ ,  $(RE_1 *)$ ,  $(RE_1 | RE_2)$ , dan  $(RE_1 \cdot RE_2)$ , di mana  $RE_1$  dan  $RE_2$  adalah *regular expression* [1].

Salah satu contoh *regular expression* adalah  $((A \cdot B) * | (B \cdot B))$ . Untuk kemudahan membaca *regular expression* sering kali  $(RE_1 \cdot RE_2)$  ditulis menjadi  $(RE_1 RE_2)$ . Sehingga contoh *regular expression* tersebut dapat disederhanakan menjadi  $((AB) *) | (BB)$ . Simbol “A”, “B” adalah alfabet dari *regular expression* sedangkan “\*”, “|”, “.” adalah operator *regular expression*.

Sebuah bahasa (Kumpulan dari *string*)  $L$  yang direpresentasikan oleh *regular expression*  $RE$  adalah bahasa

dengan simbol  $\alpha$  dimana  $\alpha \in \Sigma$  yang memiliki struktur  $RE$  sebagai berikut [1],

- Jika  $RE$  adalah  $\epsilon$ , maka  $L(RE) = \{\epsilon\}$ , string kosong.
- Jika  $RE$  adalah  $\alpha \in \Sigma$ , maka  $L(RE) = \{\alpha\}$ , sebuah string dengan panjang satu karakter.
- Jika  $RE$  adalah regular expression dengan bentuk  $RE_1$ , maka  $L(RE) = L(RE_1)$ .
- Jika  $RE$  adalah regular expression dengan bentuk  $(RE_1 \cdot RE_2)$ , maka  $L(RE) = L(RE_1) \cdot L(RE_2)$ . di mana  $W_1 \cdot W_2$  adalah kumpulan string w sedemikian hingga  $w = w_1w_2$ , dimana  $w_1 \in W_1$  dan  $w_2 \in W_2$ . Operator “.” adalah operator concatenate.
- Jika  $RE$  adalah regular expression dengan bentuk  $(RE_1 | RE_2)$ , maka  $L(RE) = L(RE_1) \cup L(RE_2)$ . Operator “|” adalah operator Union.
- Jika  $RE$  adalah regular expression dengan bentuk  $(RE_1 *)$ , maka  $L(RE) = L(RE_1)^* = \bigcup_{i \geq 0} L(RE_1)^i$ . di mana  $L^0 = \{\epsilon\}$  dan  $L^i = L \cdot L^{i-1}$  untuk semua  $L$ . Operator “\*” adalah operator Kleene Star.

Sebagai contoh regular expression  $((AB)^*)|(BB)$  akan merepresentasikan bahasa  $L((AB)^*)|(BB) = \{\epsilon, AB, BB, ABAB, ABABAB, ABABAB, \dots\}$ .

## 2.2 String Matching Menggunakan Regular Expression

Mencari string dalam sebuah kumpulan string dengan menggunakan regular expression diperlukan untuk mengubah bentuk regular expression menjadi sebuah state machine yang disebut Nondeterministic Finite Automaton (NFA). Ada dua metode yang umum digunakan dalam melakukan konversi regular expression menjadi NFA, yakni dengan menggunakan metode Thompson dan menggunakan metode Glushkov. Di buku ini penulis akan menjelaskan metode Thompson pada subbab 2.4.

Hasil konversi *regular expression* ke NFA akan menghasilkan sebuah *state machine* yang bisa digunakan untuk melakukan pencarian *string* dalam kumpulan *string*. Teknik menggunakan NFA sebagai pencari *string* yakni dengan memasukkan setiap simbol dalam *string* dari awal *string* hingga akhir *string* pada *state* awal sebuah NFA, atau hingga sebuah NFA tidak menemui masukan simbol yang valid untuk menuju *state* berikutnya dari simbol yang diberikan secara berurutan sehingga masukan simbol berikutnya akan kembali dimulai dari *state* awal NFA. Apabila suatu simbol dari *string* mencapai *state* selesai dari NFA maka sebuah *string* dinyatakan cocok dengan pola *regular expression* yang diberikan.

Dalam penggunaannya NFA memiliki kelemahan yakni waktu yang digunakan untuk melakukan pencarian lambat apabila terdapat banyak *state* dalam NFA. Untuk meningkatkan performa pencarian *string*, NFA dapat dikonversikan lagi menjadi *Deterministic Finite Automaton* (DFA) dengan menghilangkan *state* transisi kosong sehingga untuk setiap simbol yang dimasukkan hanya akan ada satu *state* yang aktif, konversi dari NFA ke DFA akan dibahas pada subbab 2.6.

### 2.3 Nondeterministic Finite Automaton

*Nondeterministic Finite Automaton* adalah sebuah *state machine* yang memiliki sekumpulan *state* berhingga  $Q$ , sekumpulan simbol  $\Sigma$  sebagai masukan transisi *state*, fungsi transisi  $F$  dari *state*  $q \in Q$  dengan masukan simbol  $\alpha \in \Sigma$  menuju *state*  $Q' \subseteq Q$ , sekumpulan state  $A \subseteq Q$  sebagai *state* NFA selesai, dan sebuah *state*  $q_0$  sebagai *state* NFA dimulai [1].

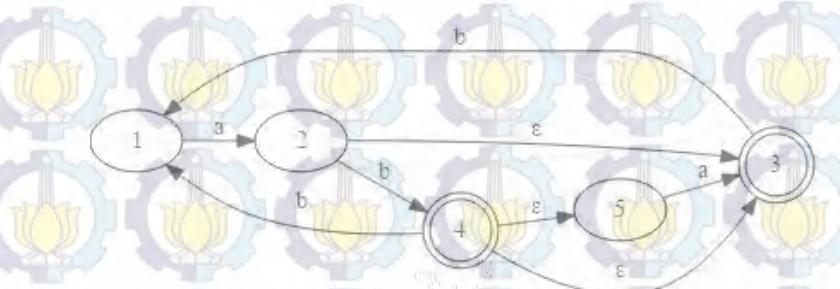
NFA dapat direpresentasikan dengan sebuah tabel atau sebuah *graph*. Tabel 2.3.1 merepresentasikan sebuah NFA dengan 5 *state* dengan kumpulan simbol masukan  $\Sigma = \{a, b\}$ . *state* awal NFA dimulai dari *state*  $q_0$  dan memiliki dua *state* selesai yakni  $A = \{3,4\}$ . NFA tersebut juga dapat direpresentasikan dalam bentuk *graph* seperti pada Gambar 2.3.1.

NFA memiliki karakteristik khusus yakni sebuah *state* dapat berpindah ke *state* lain tanpa masukan simbol apapun yang dinotasikan dengan  $\epsilon$  sebagai simbol masukan dalam Tabel 2.3.1.

Tabel 2.3.1 Representasi NFA dalam tabel

	a	b	$\epsilon$
$q_0$	2	-	-
2	-	4	3
{3}	-	1	-
{4}	-	1	3,5
5	3	-	-

Perpindahan dengan tanpa masukan simbol disebut dengan perpindahan/transisi epsilon. Dengan adanya transisi epsilon dalam NFA, satu masukan simbol dapat mengaktifkan lebih dari satu *state*. Sebagai contoh, apabila *state* aktif saat ini adalah state 2, dengan masukan simbol “b” maka beberapa *state* yang aktif selanjutnya adalah  $Q' = \{4,1\}$ .



Gambar 2.3.1 Representasi NFA dalam *directed graph*

Hal ini membuat NFA menjadi sangat lambat ketika digunakan untuk melakukan pencarian *string*, terlebih jika NFA memiliki banyak *state*. Lambatnya NFA dalam melakukan pencarian *string* ini dikarenakan diperlukan penyimpanan setiap jalur *state* yang mungkin dikunjungi untuk bisa menyatakan suatu rangkaian simbol masukan dapat mencapai *state* selesai atau tidak.

## 2.4 Metode Thompson

Metode Thompson adalah sebuah metode yang digunakan untuk melakukan konversi dari *regular expression* ke dalam bentuk *Nondeterministic Finite Automaton* (NFA). Metode ini merupakan metode yang sederhana dan akan menghasilkan maksimal  $2m$  states dan maksimal  $4m$  transisi dimana  $m$  adalah banyak karakter alfabet dalam *regular expression*.

Metode ini menggunakan *automaton* yang spesifik untuk setiap operator yang digunakan dalam *regular expression*. Beberapa *automaton* tersebut adalah [1]

- *Automaton* untuk transisi epsilon ditunjukkan pada **Gambar 2.4.1**. *Automaton* ini berarti perpindahan state dari  $q_0$  ke  $F$  bisa terjadi tanpa masukan simbol apapun.

**Gambar 2.4.1 Automaton transisi epsilon**

- *Automaton* untuk sebuah masukan karakter ditunjukkan pada **Gambar 2.4.2**. *Automaton* ini berarti perpindahan state dari  $q_0$  ke  $F$  terjadi apabila karakter  $\alpha$  dimasukkan sebagai simbol masukan ke NFA saat  $q_0$  sebagai *state* aktif.

**Gambar 2.4.2 Automaton dengan sebuah masukan karakter**

- Automaton untuk operator “.” (*Concatenate*) ditunjukkan pada **Gambar 2.4.3**. *Automaton* ini berarti perpindahan *state* dari  $q_0$  ke  $F$  terjadi apabila NFA diberikan dua kali masukan simbol secara berurutan. Untuk berpindah dari *state*  $q_0$  ke *state* 2 dengan masukan  $\alpha_1$ , dan dari *state* 3 ke *state* F dengan masukan  $\alpha_2$

Gambar 2.4.3 Automaton operator concatenate

- Automaton untuk operator “|” (Union) ditunjukkan pada Gambar 2.4.4. Automaton ini berarti perpindahan state dari  $q_0$  ke  $F$  dapat terjadi apabila masukan yang diberikan merupakan salah satu dari simbol  $\alpha_1$  atau  $\alpha_2$ .

Gambar 2.4.4 Automaton operator Union

- Automaton untuk operator “ $^*$ ” (Klenee star) ditunjukkan pada Gambar 2.4.5. Automaton berarti perpindahan state dari  $q_0$  ke  $F$  dapat terjadi dengan tanpa masukan karena  $q_0$  adalah state yang sama dengan  $F$  atau dengan masukan simbol  $\alpha$ .

Gambar 2.4.5 Automaton operator Kleene star

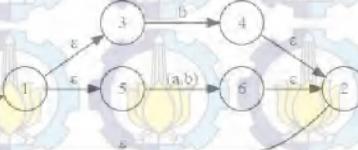
Dengan metode di atas contoh regular expression  $((a.b)|b)^*$  akan terbentuk NFA dengan melakukan operasi yang sudah didefinisikan oleh metode tersebut dengan proses yang ditunjukkan secara berurutan pada Gambar 2.4.6, Gambar 2.4.7, Gambar 2.4.8, dan Gambar 2.4.9.



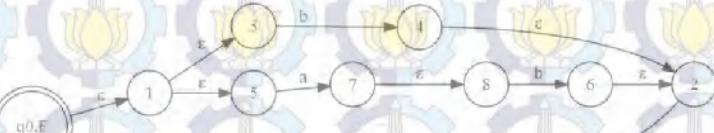
Gambar 2.4.6 Automaton awal yang terbentuk



Gambar 2.4.7 Automaton setelah operator Kleene star



Gambar 2.4.8 Automaton setelah operator Union



Gambar 2.4.9 NFA representasi dari regular expression  $((a.b)|b)^*$

## 2.5 Deterministic Finite Automaton

*Deterministic Finite Automaton (DFA)* adalah sebuah *state machine* spesialisasi dari NFA di mana DFA memiliki properti khusus yang harus dipenuhi, yaitu,

- Tidak ada transisi epsilon pada *state machine*.
- Setiap masukan simbol pada suatu *state* akan menghasilkan perpindahan ke *state* unik yang lain, dengan arti untuk masukan simbol yang berbeda maka *state* selanjutnya akan berbeda pula.

## 2.6 Metode Subset Construction

Karena setiap DFA merupakan NFA, maka setiap NFA yang terbentuk dari hasil konversi *regular expression* akan bisa

dikonversi ke DFA yang sepadan. Konversi NFA ke DFA dilakukan dengan menggabungkan beberapa *states* pada NFA menjadi sebuah *states* pada DFA. Penggabungan ini dapat dilakukan karena dua alasan yaitu,

- Pada NFA apabila ada dua atau lebih *state* yang terhubung dengan transisi epsilon, maka beberapa *state* tersebut dapat dianggap satu *state* pada DFA karena untuk berpindah dari satu *state* ke *state* lain dengan transisi epsilon tidak diperlukan adanya masukan simbol.
- Pada NFA apabila terdapat sebuah *state* yang apabila diberi sebuah masukan dapat berpindah ke beberapa *state* yang lain maka kumpulan *state* tersebut dapat dianggap sebuah *state* pada DFA.

Dari dua alasan tersebut maka didefinisikanlah dua fungsi yang akan digunakan sebagai operasi penggabungan beberapa *state* NFA menjadi sebuah *state* DFA. Dua fungsi tersebut adalah,

- Move Closure

*Move closure* adalah fungsi yang digunakan untuk menemukan sekumpulan *state* pada NFA yang bisa dikunjungi dari sebuah state NFA dengan suatu masukan simbol tertentu.

- Epsilon Closure

*Epsilon Closure* adalah fungsi yang digunakan untuk menemukan sekumpulan *state* pada NFA yang bisa dikunjungi dari sebuah state NFA tanpa diperlukan adanya masukan simbol tertentu.

Untuk melakukan konversi DFA dari NFA ada beberapa langkah yang perlu dilakukan dengan menggunakan fungsi *move closure* dan *epsilon closure* yang telah didefinisikan. Langkah-langkah tersebut adalah,

1. Membuat *state* awal DFA dengan melakukan *epsilon closure* pada *state* awal NFA. *state* awal DFA merupakan kumpulan *state* NFA yang dapat dikunjungi dengan transisi epsilon dari *state* awal NFA.

2. Untuk setiap *state* DFA yang baru terbentuk dilakukan langkah-langkah berikut untuk semua  $\alpha \in \Sigma$ .
  - a. Melakukan *move closure* dengan parameter *state* DFA yang berupa kumpulan satu atau lebih *state* NFA dan masukan simbol tertentu untuk mendapatkan kumpulan *state* NFA yang dapat dikunjungi dari *state* DFA dengan masukan simbol tertentu.
  - b. Melakukan *epsilon closure* pada kumpulan *state* NFA yang didapatkan pada langkah 2a untuk mendapatkan *state* baru DFA (*state* baru ini merupakan satu atau lebih *state* NFA yang dikombinasikan menjadi satu *state* DFA). Apabila pada langkah 2a tidak ada *state* NFA yang dihasilkan tidak ada *state* DFA baru yang dihasilkan.
3. Apabila masih ada *state* DFA baru yang terbentuk pada langkah 2, maka langkah 2 dilakukan hingga tidak ada *state* baru yang terbentuk. Apabila telah tidak ada *state* baru yang terbentuk maka dilanjutkan ke langkah 4.
4. DFA selesai dibentuk, dan *state* selesai DFA merupakan *state* DFA yang terbentuk dari *state* selesai NFA [2].

## 2.7 Perpangkatan Matriks

Perpangkatan matriks atau *Matrix Exponentiation* digunakan untuk mencari jumlah kemungkinan *string* yang dapat diterima oleh *regular expression* dengan memanfaatkan properti adjacency matriks yang memberikan jumlah *path* dari i ke j dengan tepat k langkah tertentu, dimana k adalah pangkat dari perpangkatan matriks. Adjacency matriks dibangun dari DFA yang dihasilkan pada proses *subset construction*.

Perpangkatan matriks dengan dimensi D dengan pangkat P dapat dilakukan dengan melakukan perkalian matriks sebanyak N kali. Pada proses perpangkatan tersebut akan terjadi

operasi perkalian sebanyak  $D^2 \times N$ . Kompleksitas big O dari proses tersebut akan memiliki orde 3.

Teknik mempercepat perpangkatan matriks diantaranya adalah melakukan *Exponentiation by Squaring* atau melakukan perpangkatan dengan mengkuadratkan. Teknik ini mengurangi proses perkalian menjadi  $D^2 \times \log_2 N$ . Teknik ini secara rekursif didefinisikan pada persamaan 1 [3].

$$x^n = \begin{cases} x(x^2)^{\frac{n-1}{2}}, & \text{apabila } n \text{ ganjil} \\ (x^2)^{\frac{n}{2}}, & \text{apabila } n \text{ genap} \end{cases} \quad (1)$$

## 2.8 Permasalahan CTSTRING di SPOJ

A regular expression is used to describe a set of strings. For this problem the alphabet is limited to 'a' and 'b'. R is a regular expression if:

- 1) R is "a<sup>n</sup>" or "b<sup>n</sup>"
- 2) R is of the form "(R1R2)" where R1 and R2 are regular expressions
- 3) R is of the form "(R1 | R2)" where R1 and R2 are regular expressions
- 4) R is of the form "(R1<sup>\*</sup>)" where R1 is a regular expression.

The set of strings recognised by R are as follows:

- 1) If R is "a<sup>n</sup>", then the set of strings recognised = {a<sup>n</sup>}
- 2) If R is "b<sup>n</sup>", then the set of strings recognised = {b<sup>n</sup>}
- 3) If R is of the form "(R1R2)" then the set of strings recognised = all strings which can be obtained by a concatenation of strings s1 and s2 where s1 is recognised by R1 and s2 by R2.
- 4) If R is of the form "(R1 | R2)" then the set of strings recognised = union of the set of strings recognised by R1 and R2.
- 5) If R is of the form "(R1<sup>\*</sup>)" then the the strings recognised are the empty string and the concatenation of an arbitrary number of copies of any string recognised by R1.

Given a regular expression and an integer L, you need to count how many strings of length L are recognized by it.

**Gambar 2.8.1** Deskripsi permasalahan CTSTRING

Permasalahan representasi *regular expression* menjadi ke NFA atau DFA juga terdapat pada situs *Online Judge* SPOJ yang mempunyai kode permasalahan CTSTRING. Deskripsi dari permasalahan ditunjukkan pada **Gambar 2.8.1** [4]. Pada **Gambar 2.8.1** dijelaskan bahwa diberikan sebuah *regular expression* RE dan sebuah bilangan bulat L, dengan batasan *regular expression* memiliki tiga operator yakni *concatenate*, *union*, dan *klenee star* dan simbol  $\Sigma = \{a, b\}$ . Dari masukan

yang diberikan, hitung banyaknya *string* dengan panjang L yang dapat di terima oleh *regular expression* RE.

Berikut merupakan Format masukan pada permasalahan tersebut,

1. Baris pertama dari masukan adalah sebuah bilangan bulat N yang menyatakan berapa banyak kasus uji yang diberikan.
2. N Baris berikutnya merupakan masukan kasus uji dengan format *string* RE dan bilangan bulat L.

Format keluaran pada permasalahan tersebut adalah N buah bilangan T di mana T adalah banyaknya *string* dengan panjang L yang dapat diterima oleh *regular expression* RE. Karena bilangan T bisa sangat besar, maka T ditampilkan setelah dilakukan operasi modulus terhadap T yakni  $T = T \bmod (10^9 + 7)$ . Contoh masukan dan keluaran juga dapat dilihat pada **Gambar 2.8.2**.

Sample Input:

3  
((ab)|(ba)) 2  
((a|b)\* 5  
((a\*)(b(a\*)) 100

Sample Output:

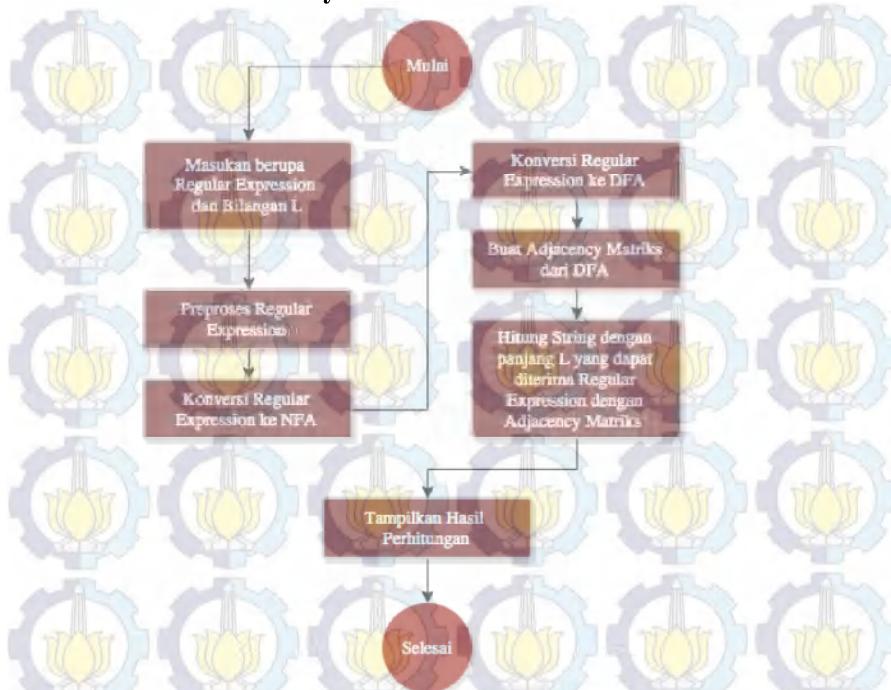
2  
32  
100

**Gambar 2.8.2 Contoh masukan dan keluaran permasalahan**

Adapun batasan dari permasalahan tersebut adalah,

1.  $1 \leq N \leq 50$ .
2.  $1 \leq \text{panjang}(RE) \leq 100$ .
3.  $1 \leq L \leq 10^9$ .
4. Dinilai dalam lingkungan penilaian Intel G860 3Ghz
5. Batasan waktu 0.1 detik – 0.150 detik.
6. Batasan memori 1536 MB.

## 2.9 Desain Umum Penyelesaian Permasalahan CTSTRING



Gambar 2.9.1 Diagram Alur penyelesaian permasalahan

Untuk menyelesaikan permasalahan studi kasus, tahap-tahap yang ditempuh dapat dilihat pada Gambar 2.9.1. Pada tahap awal setelah masukan diperoleh diperlukan preproses untuk memberikan operator concatenate yang tidak diberikan pada masukan studi kasus. Kemudian hasil preproses tersebut diproses untuk dikonversi menjadi NFA dengan model NFA Thompson. Usai NFA terbentuk, dilakukan konversi NFA menjadi DFA dengan metode *Subset Construction*, dan pada tahap akhir dibentuk sebuah *Adjacency Matrix* dari DFA untuk kemudian digunakan sebagai sarana perhitungan jumlah *string* yang dapat diterima oleh *regular expression* yang diberikan.

## 2.10 Ilustrasi Penyelesaian permasalahan CTSTRING

Pada permasalahan CTSTRING diberikan dua masukan yaitu *regular expression RE* dan bilangan  $L$ . Contoh yang akan dibahas diasumsikan dengan masukan sebagai berikut,  $RE = ((a^*)(b(a^*)))$  dan  $L = 100$ .

Dari kedua masukan tersebut, hal yang pertama dilakukan adalah melakukan preproses terhadap  $RE$  dikarenakan tidak terdapat operator *concatenate* yang diberikan secara eksplisit dalam format masukan dari Sphere Online Judge. Usai preproses dilakukan maka *regular expression* yang terbentuk menjadi  $RE = ((a^*) + (b + (a^*)))$ .

Proses konversi *regular expression* ke NFA dilakukan setelah preproses. Proses ini dilakukan dengan melakukan iterasi setiap karakter  $RE$  hasil preproses. Pada proses iterasi akan dibentuk kombinasi *automaton* untuk setiap evaluasi operator. Pada contoh di atas akan ada 4 evaluasi operator yang akan dilakukan, yakni dua kali “ $*$ ”, dan dua kali “ $+$ ”.

Evaluasi pertama pertama dan kedua adalah  $(a^*)$ , ketiga adalah  $((b + (a^*)))$ , dan terakhir  $((a^*) + (b + (a^*)))$ . Ilustrasi **Gambar 2.10.1**, **Gambar 2.10.2**, dan **Gambar 2.10.3** adalah tahap-tahap pembentukan NFA untuk setiap operasi yang dilakukan. Operasi tersebut dilakukan dengan mengikuti metode Thompson seperti dijelaskan pada subbab 2.4.

Hasil akhir dari tahap tersebut (**Gambar 2.10.3**) merupakan NFA yang kemudian digunakan untuk konversi menjadi DFA. Proses konversi DFA dilakukan dengan proses yang telah dijelaskan pada subbab 2.6. Inisialisasi awal DFA dengan melakukan *EpsilonClosure* dari awal DFA yakni *state 9*.



Gambar 2.10.1 Evaluasi operasi  $(a^*)$  pertama dan kedua



Gambar 2.10.2 Evaluasi operasi  $(b+(a^*))$



Gambar 2.10.3 Evaluasi operasi  $((a^*)+(b+(a^*)))$

State awal DFA merupakan kombinasi dari beberapa state NFA yang dapat dicapai dengan transisi epsilon, yakni state 1,3, dan 4. Dari state awal yang didapatkan, kemudian dilakukan iterasi untuk memperoleh DFA seutuhnya, iterasi yang dilakukan dapat dilihat pada Tabel 2.10.1.

Tabel 2.10.1 Iterasi dalam konversi NFA menjadi DFA

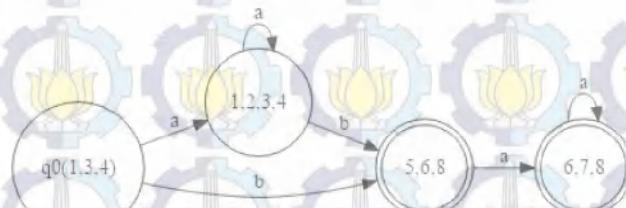
Iterasi	DFA state	Masukan Alfabet	
		$a\epsilon$	$b\epsilon$
1	1,3,4	1,2,3,4	5,6,8
2	1,2,3,4	1,2,3,4	5,6,8
3	5,6,8	6,7,8	-
4	6,7,8	6,7,8	-

Selesai iterasi untuk membangkitkan DFA dari NFA selesai maka DFA yang terbentuk dapat dilihat pada Gambar 2.10.4, dan Gambar 2.10.5.

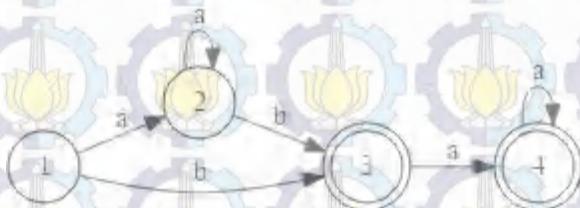
Apabila telah terbentuk DFA, maka tahap selanjutnya yang dilakukan adalah menghitung banyaknya string dengan

panjang  $L$  yang dapat diterima pada model tersebut. *String* diterima jika dari state awal DFA diberikan masukan urutan karakter alfabet pada *string* hingga seluruh karakter masukan telah berjumlah  $L$  dan state terakhir yang aktif adalah state akhir DFA yakni state 3 dan 4.

Sebagai contoh *string*  $s = "aaaa"$  akan melalui urutan state  $1 \rightarrow 2 \rightarrow 2 \rightarrow 2 \rightarrow 2$  dan tidak berakhir pada state akhir DFA, sehingga *string* aaaa tidak diterima.



Gambar 2.10.4 hasil konversi NFA ke DFA



Gambar 2.10.5 DFA yang telah disederhanakan

Pada contoh sebelumnya *string*  $s$  memiliki panjang 4 karakter, tentu saja sangat mudah untuk mencoba seluruh kemungkinan, dengan alfabet= $\{"a", "b"\}$  dan panjang 4 hanya diperlukan  $2^4$  percobaan untuk menguji semua *string* dengan panjang 4. Namun pada permasalahan studi kasus yang digunakan, batas atas  $L$  adalah  $10^9$  dan tentu saja untuk mencoba seluruh kemungkinan *string* dengan panjang tersebut sangat tidak mungkin karena akan diperlukan  $2^{10^9}$  operasi.

Untuk itu perlu dibuat *adjacency matrix* untuk DFA yang notabene juga merupakan *directed graph*. *Adjacency matrix* untuk sebuah *graph* merepresentasikan relasi tiap *vertex/states*. Dari DFA pada Gambar 2.10.5 *adjacency matrix* yang dihasilkan dapat dilihat pada Gambar 2.10.6.

0	1	1	0
0	1	1	0
0	1	1	0
0	1	1	0
0	1	1	0

Gambar 2.10.6 *Adjacency matrix* dari DFA yang dibuat

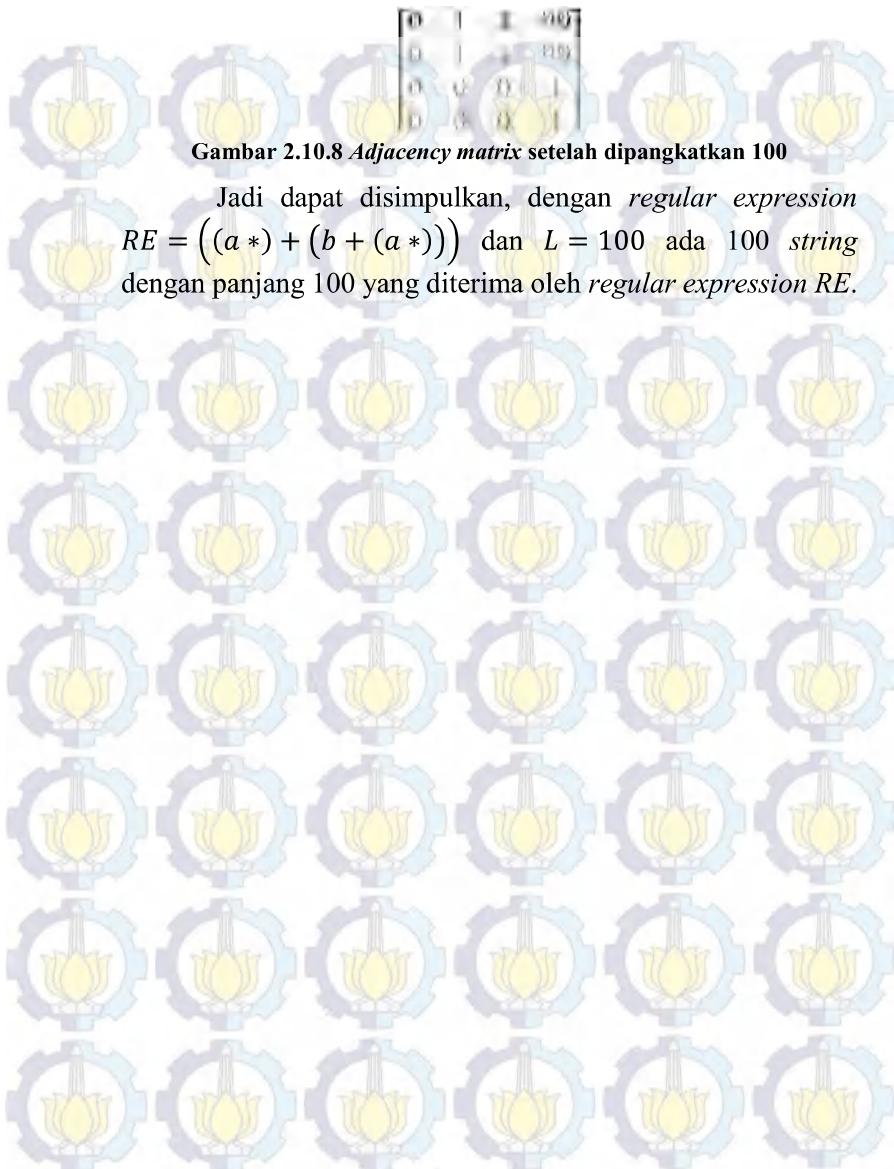
Dari matriks tersebut dapat dilihat relasi antar *state* sebagai contoh pada baris pertama  $\{0,1,1,0\}$  yang memiliki arti *state 1* memiliki hubungan ke *state 2*, dan *3* dengan satu langkah perpindahan, untuk dua langkah perpindahan maka *adjacency matrix* yang dihasilkan dapat dilihat pada Gambar 2.10.7.

0	1	1	0
0	1	1	0
0	1	1	0
0	1	1	0
0	1	1	0

Gambar 2.10.7 *Adjacency matrix* untuk perpindahan sebanyak 2 langkah

Matriks pada gambar Gambar 2.10.7 diperoleh dengan memangkatkan matriks pada Gambar 2.10.6 dengan pangkat 2. Untuk memperoleh *string* yang dapat diterima oleh *regular expression* dengan panjang  $L = 100$  hal yang perlu dilakukan adalah memangkatkan matriks pada Gambar 2.10.6 dengan 100. Hasil yang diperoleh dapat dilihat pada Gambar 2.10.8.

Dari Gambar 2.10.8 untuk memperoleh berapa banyak *string* dengan panjang 100 yang diterima, maka dilakukan penjumlahan untuk setiap *cell* yang merepresentasikan relasi *state* awal DFA dan *state* akhir DFA, yakni baris pertama kolom ketiga, dan baris pertama kolom keempat. Hasil yang didapatkan adalah 100.



Gambar 2.10.8 *Adjacency matrix* setelah dipangkatkan 100

Jadi dapat disimpulkan, dengan *regular expression*  $RE = ((a^*) + (b + (a^*)))$  dan  $L = 100$  ada 100 string dengan panjang 100 yang diterima oleh *regular expression*  $RE$ .