Implementasi Model Deterministic Finite Automaton Untuk Interpretasi Regular Expression pada Studi Kasus Permasalahan SPOJ Klasik 10354

Penyusun Tugas Akhir

Muhammad Yunus Bahari

NRP. 5111100079

Dosen Pembimbing

Arya Yudhi Wijaya, S.Kom., M.Kom.

Rully Soelaiman, S.Kom., M.Kom.

ITS
Institut
Teknologi
Sepuluh Nopember

Agenda

Pendahuluan

Latar Belakang

1. Permasalahan yang diangkat pada tugas ahir ini bersumber dari permasalahan dari SPOJ yang memiliki code permasalahan CTSTRING
   - *http://www.spoj.com/problems/CTSTRING*
2. Untuk menyelesaikan permasalahan tersebut diperlukan adanya model interpretasi yang tepat, yakni dengan model Deterministic Finite Automaton
3. Dibutuhkan juga metode perhitungan yang efisien untuk perpangkatan matriks

Deskripsi Studi Kasus

1. Diberikan sebuah *string regular expression* RE dan sebuah bilangan L untuk dihitung berapa banyak *string* dengan panjang L yang dapat diterima oleh pola *regular expression* RE.

2. *Regular expression* yang diberikan memenuhi ketentuan berikut
   a) Terdapat 3 operator yang digunakan yakni *star*, *union*, dan *concatenate.*
   b) Karakter alfabet yang digunakan hanya “a” dan “b”.

Rumusan Masalah

1. Bagaimana memodelkan *regular expression* ke dalam bentuk *Deterministic Finite Automaton*?
2. Bagaimana mengimplementasikan model *Deterministic Finite Automaton* untuk menginterpretasikan *regular expression*?
3. Bagaimana uji coba untuk mengetahui kebenaran dan kinerja dari implementasi program yang dilakukan?

Batasan Permasalahan

1. Implementasi dilakukan dengan bahasa pemrograman C++.
2. Terdapat dua masukan program yakni *regular expression* dan sebuah bilangan L.
3. Batas maksimum jumlah karakter *regular expression* adalah 100 karakter.
4. Alfabet yang digunakan dalam *regular expression* ada dua yakni ‘a’ dan ‘b’.
5. Operasi yang dapat dilakukan dalam *regular expression* ada tiga yakni *concatenation* (ab), *union* (a|b) dan *star quantifier/Klenee star* (a*).
6. Batas maksimum Nilai L adalah 10^9.

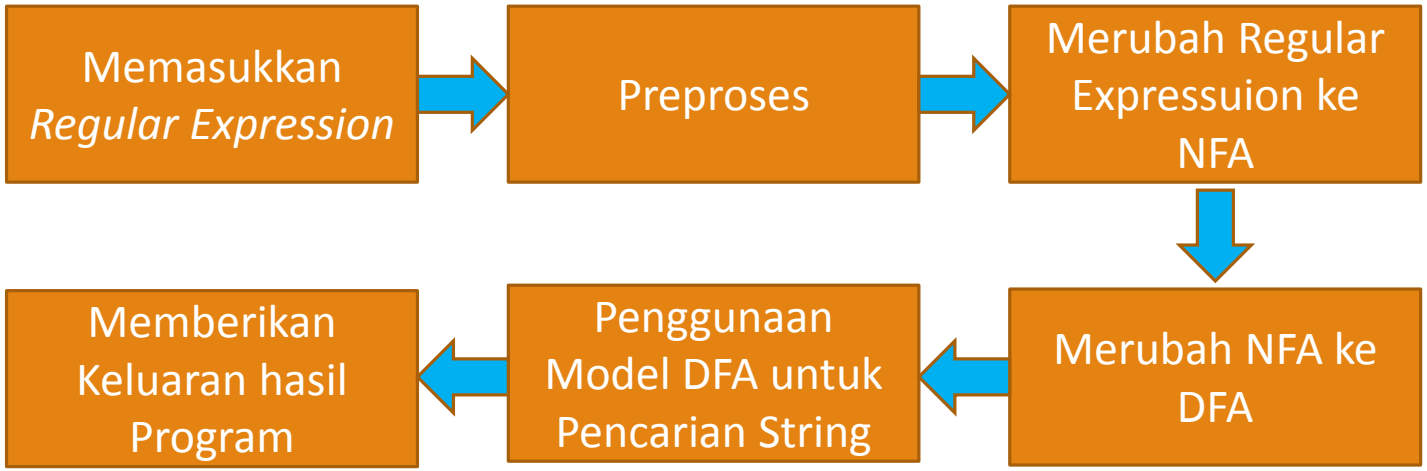
ITS
Institut
Teknologi
Sepuluh Nopember

Tujuan

1. Mengetahui bagaimana cara memodelkan *regular expression* kedalam bentuk *Deterministic Finite Automaton*.
2. Mengetahui pengaplikasian model *Deterministic Finite Automaton* untuk interpretasi *regular expression*.
3. Menguji kebenaran dan kinerja model yang telah diimplementasi

Desain & Ilustrasi

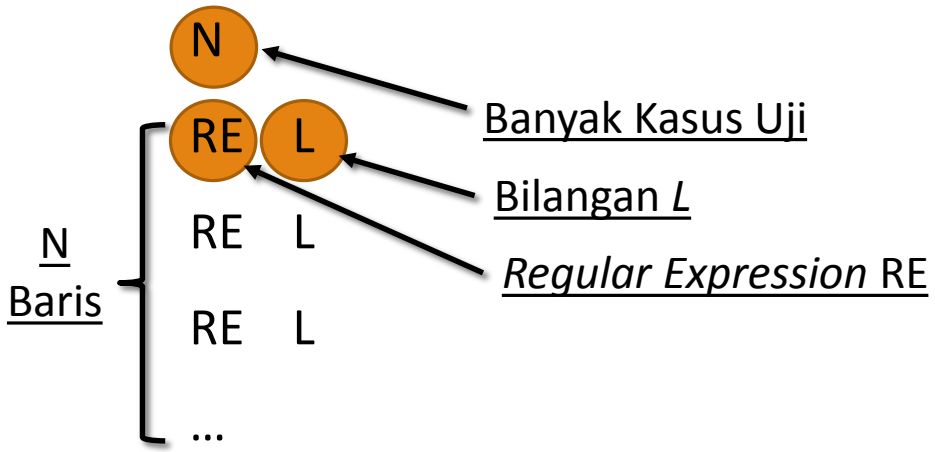
Gambaran Umum



Memasukkan Regular Expression

Masukan data yang diharapkan pada studi kasus yang digunakan adalah sebuah bilangan N dan N baris selanjutnya berisi *regular expression* RE dan bilangan *L*.

Berikut adalah format masukan dari studi kasus yang digunakan.



Contoh:

2
((ab)*) 10
((ab)|((ba)(b*))) 2

Preproses

Contoh masukan

- RE = ((a*)(b(a*)))
- L = 100

Preproses dilakukan untuk memberikan penanda operator concatenate yang tidak diberikan secara eksplisit pada masukan

Hasil Preproses

- RE = ((a*)+(b+(a*)))
- L = 100

Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = “(“

Lanjutkan ke iterasi berikutnya

Struktur Data					
Stack OP	-				
Stack State	-				

Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = “(“

Lanjutkan ke iterasi berikutnya

Struktur Data					
Stack OP	-				
Stack State	-				

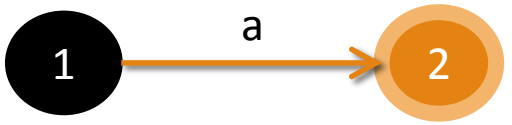
Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = “a“

Membuat Automaton untuk a



Auto_1

Struktur Data					
Stack OP	-				
Stack State	-				

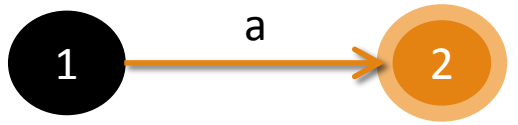
Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = “a“

Memasukkan Auto_1 ke Stack State



Auto_1

Struktur Data					
Stack OP	-				
Stack State	Auto_1				

Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = “*“

Memasukkan operator ke Stack OP

Struktur Data					
Stack OP	*				
Stack State	Auto_1				

Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Evaluasi Operator

Iterasi saat ini:

RE[i] = ")"

Struktur Data					
Stack OP	*				
Stack State	Auto_1				

Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Ambil Operator pada stack teratas dan cek

Iterasi saat ini:

RE[i] = ")"

Struktur Data					
Stack OP	-				
Stack State	Auto_1				

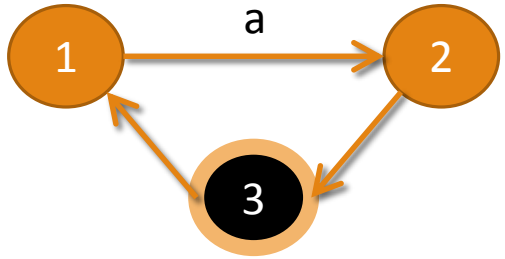
Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = ")"

Lakukan Operasi Pada Operator



Auto_1

Struktur Data					
Stack OP	-				
Stack State	-				

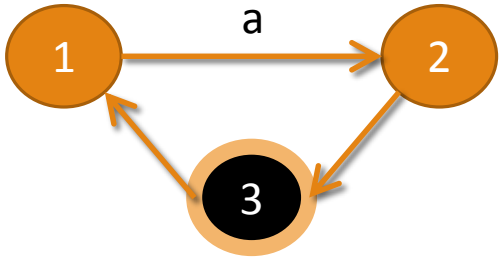
Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = ")"

Masukkan automaton baru ke stack state



Auto_1

Struktur Data					
Stack OP	-				
Stack State	Auto_1				

Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = “+“

Masukkan Operator ke Stack OP

Struktur Data					
Stack OP	+				
Stack State	Auto_1				

Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = “(“

Lanjutkan ke iterasi berikutnya

Struktur Data					
Stack OP	+				
Stack State	Auto_1				

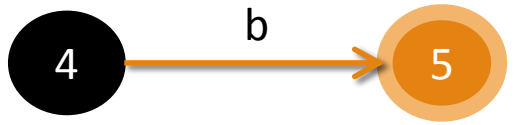
Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = “b“

Membuat Automaton untuk b



Auto_2

Struktur Data					
Stack OP	+				
Stack State	Auto_1				

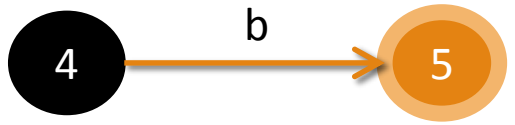
Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = “b“

Masukkan automaton baru ke stack state



Auto_2

Struktur Data					
Stack OP	+				
Stack State	Auto_1	Auto_2			

Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Masukkan Operator ke Stack OP

Iterasi saat ini:

RE[i] = “+“

Struktur Data					
Stack OP	+	+			
Stack State	Auto_1	Auto_2			

Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = “(“

Lanjutkan ke iterasi berikutnya

Struktur Data					
Stack OP	+	+			
Stack State	Auto_1	Auto_2			

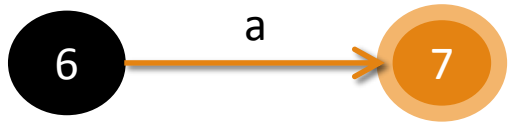
Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = “a“

Membuat Automaton untuk a



Auto_3

Struktur Data					
Stack OP	+	+			
Stack State	Auto_1	Auto_2			

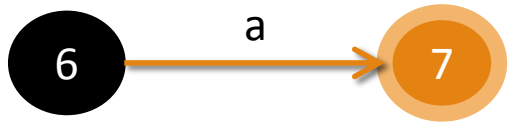
Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = “a“

Masukkan automaton baru ke stack state



Auto_3

Struktur Data					
Stack OP	+	+			
Stack State	Auto_1	Auto_2	Auto_3		

Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Masukkan Operator ke Stack OP

Iterasi saat ini:

RE[i] = "*"

Struktur Data					
Stack OP	+	+	*		
Stack State	Auto_1	Auto_2	Auto_3		

Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Evaluasi Operator

Iterasi saat ini:

RE[i] = ")"

Struktur Data					
Stack OP	+	+	*		
Stack State	Auto_1	Auto_2	Auto_3		

Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Ambil Operator pada stack teratas dan cek

Iterasi saat ini:

RE[i] = ")"

Struktur Data					
Stack OP	+	+			
Stack State	Auto_1	Auto_2	Auto_3		

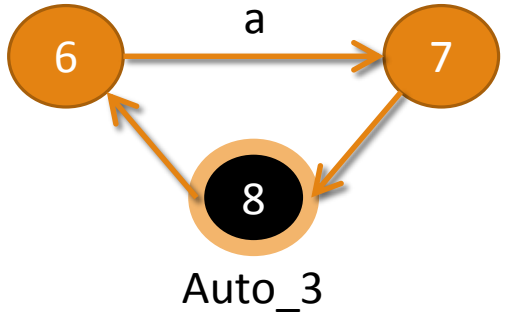
Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = ")"

Lakukan Operasi Pada Operator



Struktur Data					
Stack OP	+	+			
Stack State	Auto_1	Auto_2			

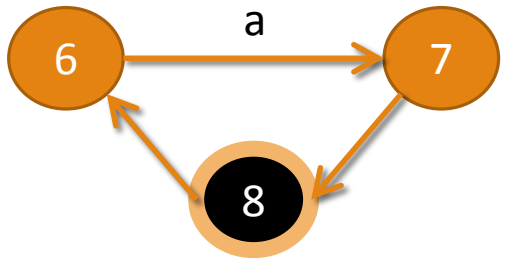
Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = ")"

Masukkan automaton baru ke stack state



Struktur Data					
Stack OP	+	+			
Stack State	Auto_1	Auto_2	Auto_3		

Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Evaluasi Operator

Iterasi saat ini:

RE[i] = ")"

Struktur Data					
Stack OP	+	+			
Stack State	Auto_1	Auto_2	Auto_3		

Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Ambil Operator pada stack teratas dan cek

Iterasi saat ini:

RE[i] = ")"

Struktur Data					
Stack OP	+				
Stack State	Auto_1	Auto_2	Auto_3		

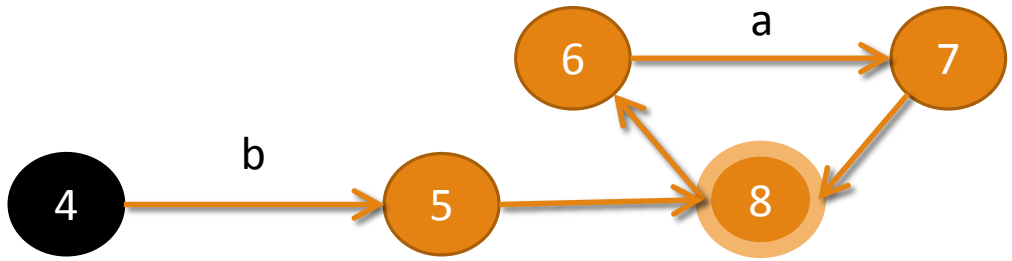
Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = ")"

Lakukan Operasi Pada Operator



Auto_2

Struktur Data					
Stack OP	+				
Stack State	Auto_1				

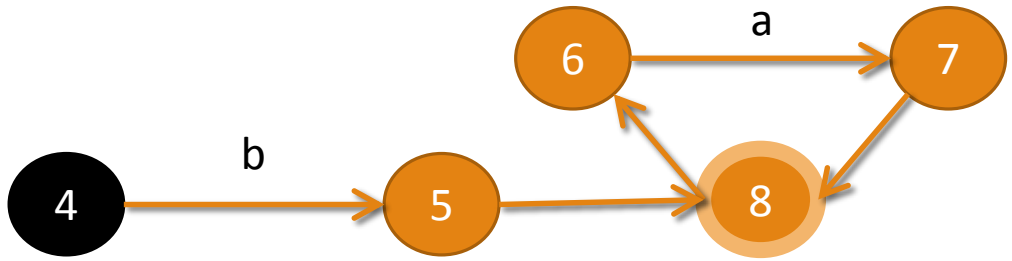
Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = ")"

Masukkan automaton baru ke stack state



Auto_2

Struktur Data					
Stack OP	+				
Stack State	Auto_1	Auto_2			

Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Evaluasi Operator

Iterasi saat ini:

RE[i] = ")"

Struktur Data					
Stack OP	+				
Stack State	Auto_1	Auto_2			

Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Ambil Operator pada stack teratas dan cek

Iterasi saat ini:

RE[i] = “)“

Struktur Data					
Stack OP	-				
Stack State	Auto_1	Auto_2			

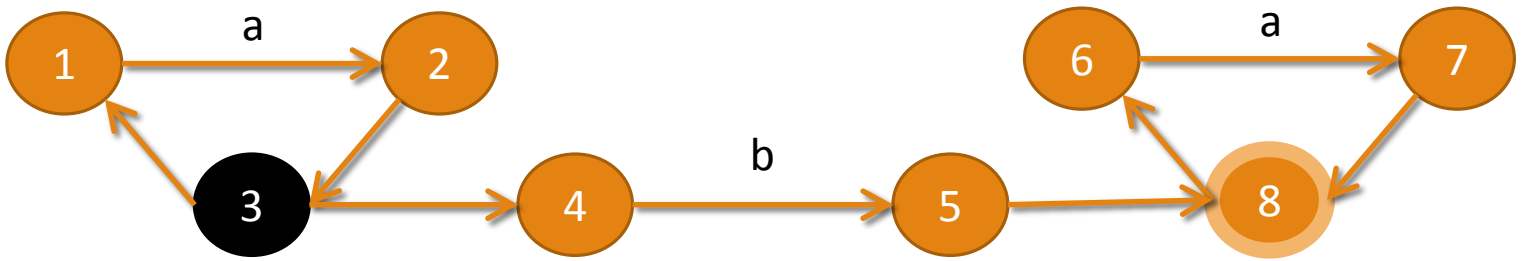
Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = ")"

Lakukan Operasi Pada Operator



Auto_1

Struktur Data					
Stack OP	-				
Stack State	-				

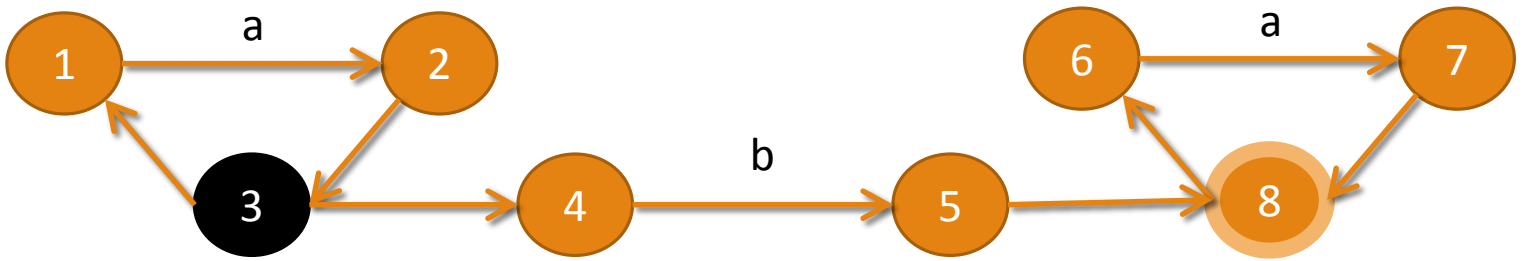
Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = ")"

Masukkan automaton baru ke stack state



Auto_1

Struktur Data					
Stack OP	-				
Stack State	Auto_1				

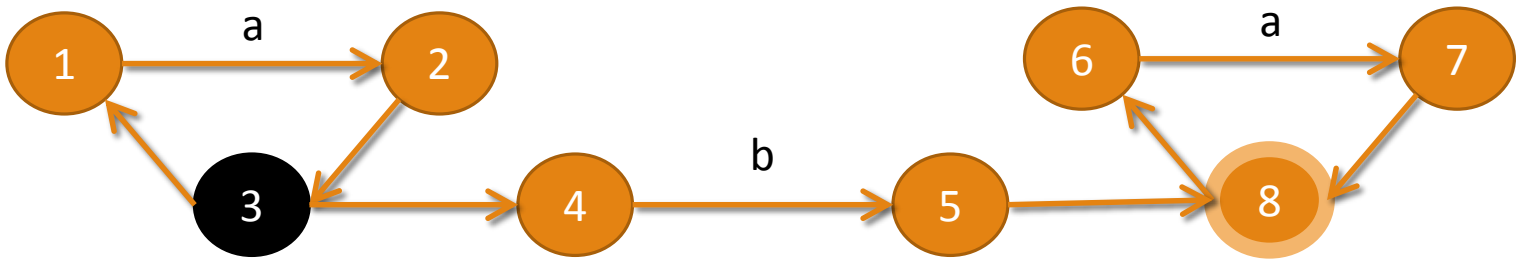
Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = ")"

Iterasi selesai, automaton yang tersisa pada stack state adalah NFA



Auto_1

Struktur Data					
Stack OP	-				
Stack State	Auto_1				

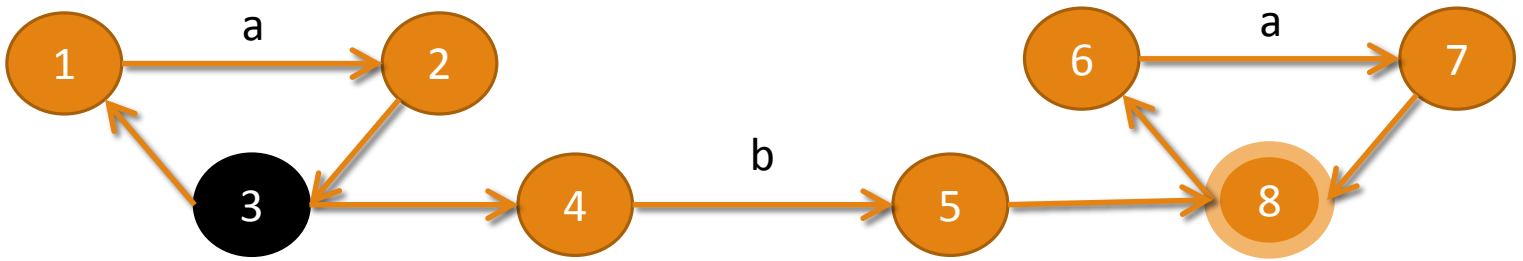
Merubah Regular Expression Ke NFA

RE = ((a*)+(b+(a*)))

Iterasi saat ini:

RE[i] = ")"

Iterasi selesai, automaton yang tersisa pada stack state adalah NFA

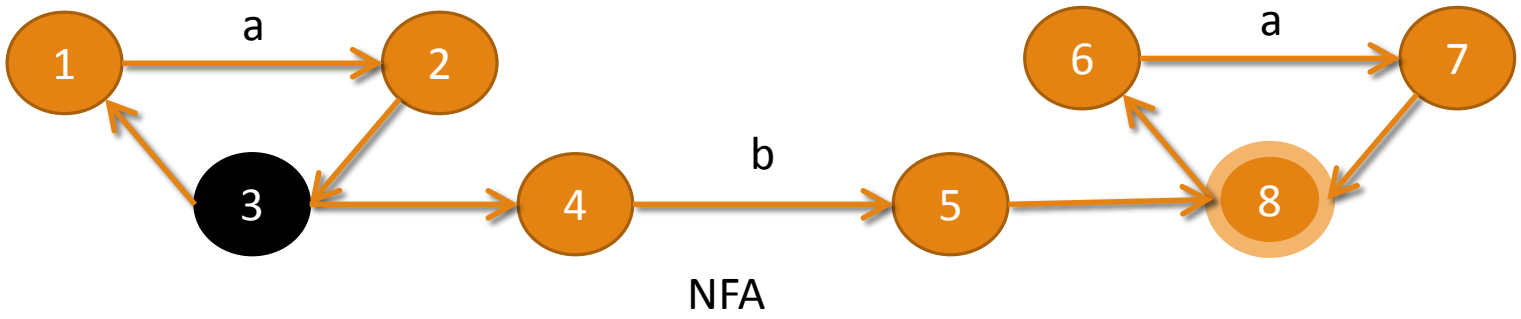


Auto_1

Struktur Data					
Stack OP	-				
Stack State	Auto_1				

Merubah NFA ke DFA

Iterasi selesai, automaton yang tersisa pada stack state adalah NFA

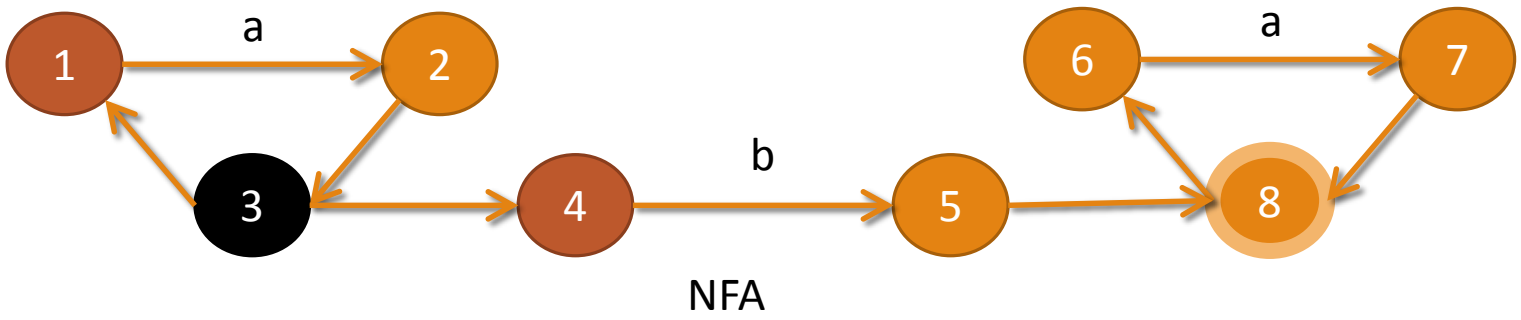


NFA

Untuk merubah NFA ke DFA, diperlukan inisialisasi state awal DFA dengan melakukan *Epsilon Closure* pada state awal NFA yakni state 3.

Merubah NFA ke DFA

Iterasi selesai, automaton yang tersisa pada stack state adalah NFA



NFA

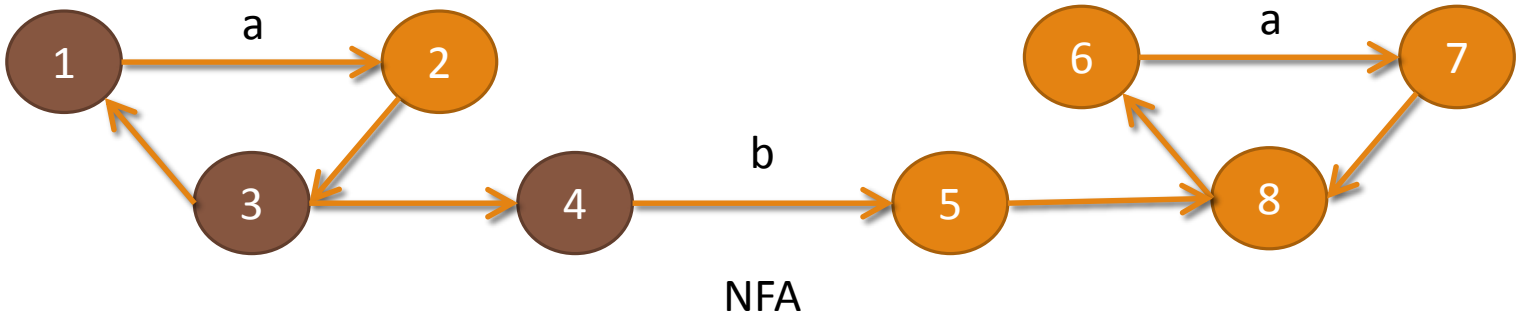
Untuk merubah NFA ke DFA, diperlukan inisialisasi state awal DFA dengan melakukan *Epsilon Closure* pada state awal NFA yakni state 3.

Hasil epsilon closure dari state 3 adalah state_1 = {1,3,4}.

Merubah NFA ke DFA

	aε	bε
1,3,4		

EpsilonClosure(Move('1,3,4', 'a'))

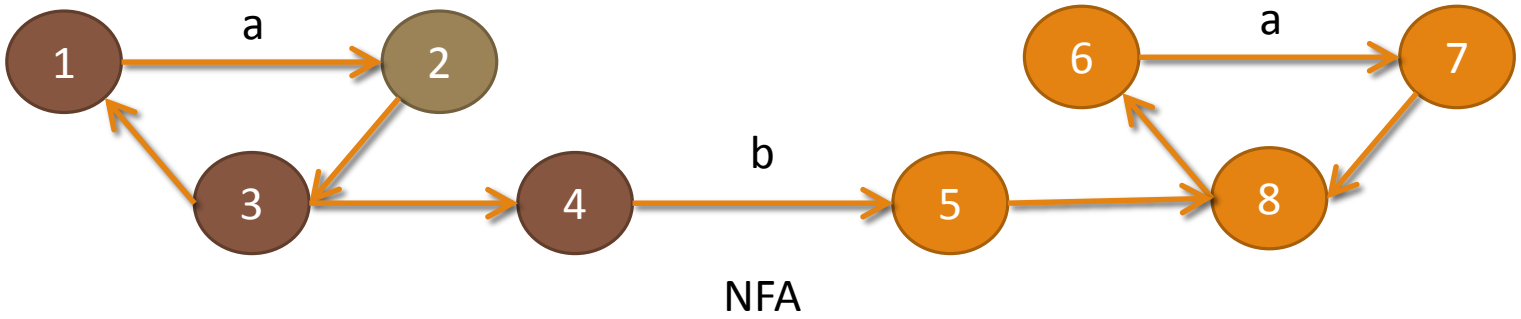


Struktur Data				
Proses	1,3,4			

Merubah NFA ke DFA

	aε	bε
1,3,4		

EpsilonClosure(Move('1,3,4', 'a'))

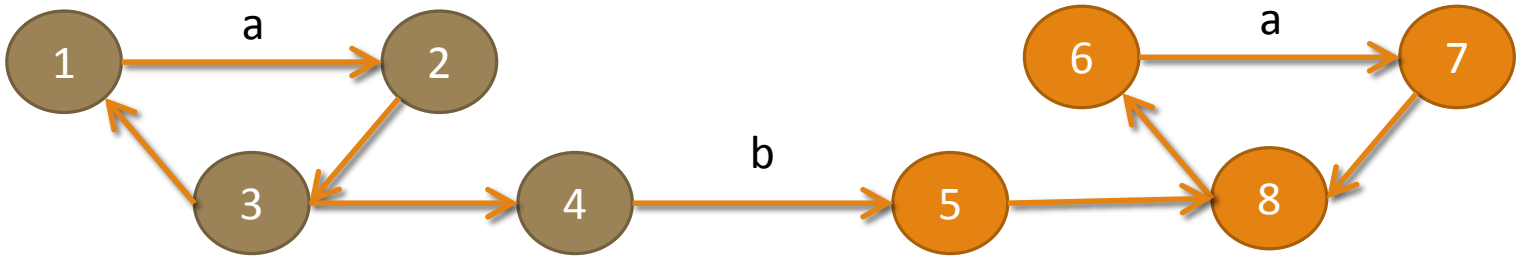


Struktur Data				
Proses	1,3,4			

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	
1,2,3,4		

EpsilonClosure(Move('1,3,4', 'a'))



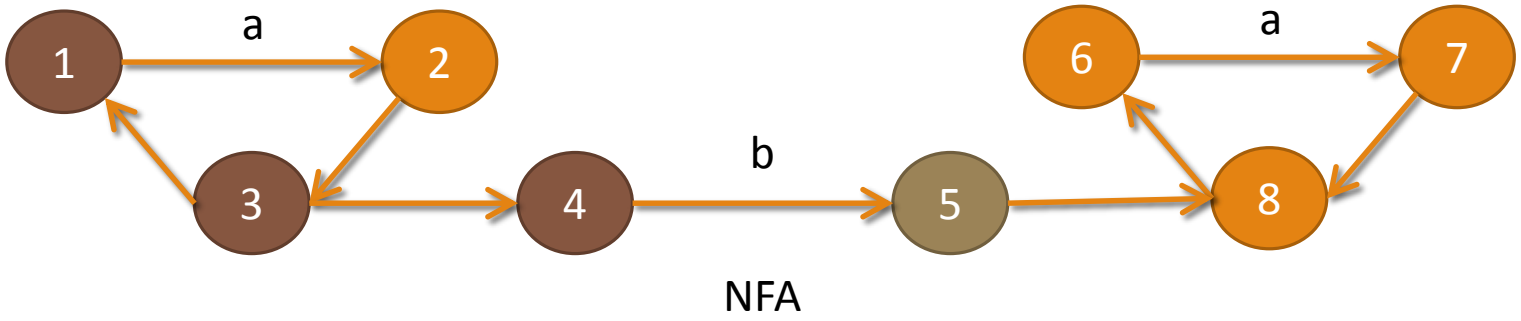
NFA

Struktur Data				
Proses	1,3,4	1,2,3,4		

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	
1,2,3,4		

EpsilonClosure(Move('1,3,4', 'b'))

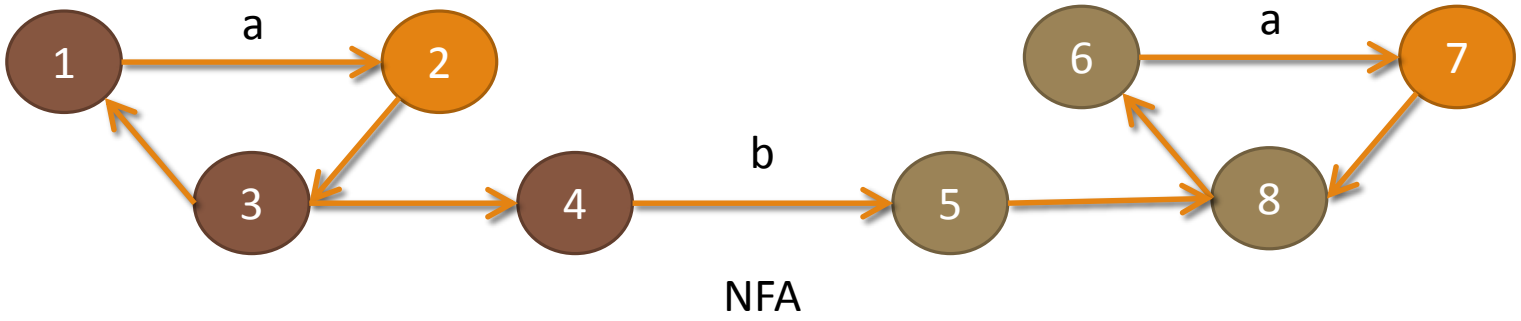


Struktur Data				
Proses	1,3,4	1,2,3,4		

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	5,6,8
1,2,3,4		
5,6,8		

EpsilonClosure(Move('1,3,4', 'b'))



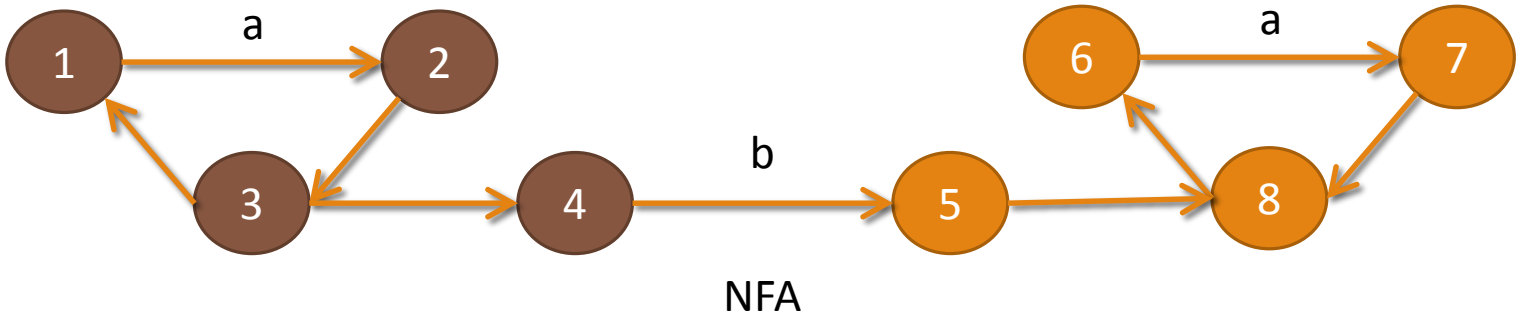
NFA

Struktur Data				
Proses	1,3,4	1,2,3,4	5,6,8	

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	5,6,8
1,2,3,4		
5,6,8		

EpsilonClosure(Move('1,2,3,4', 'a'))



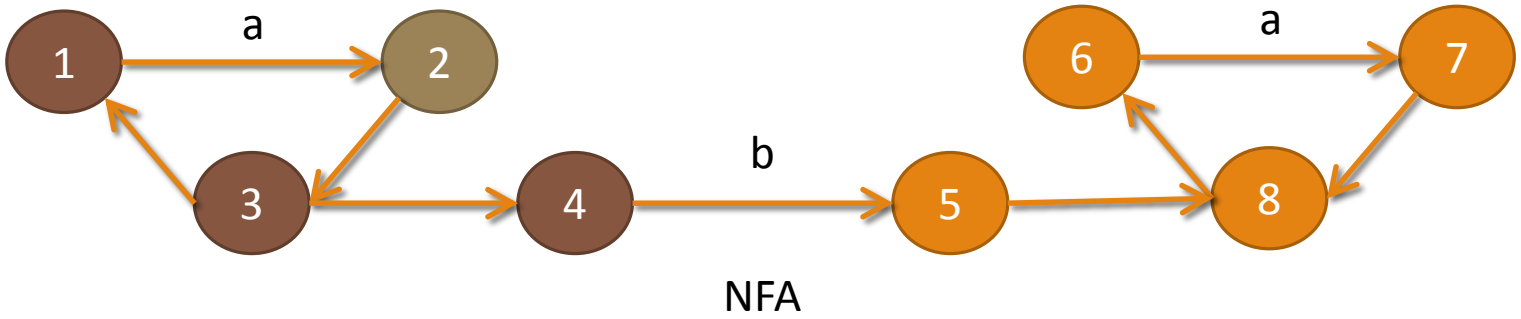
NFA

Struktur Data				
Proses	1,2,3,4	5,6,8		

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	5,6,8
1,2,3,4		
5,6,8		

EpsilonClosure(Move('1,2,3,4', 'a'))

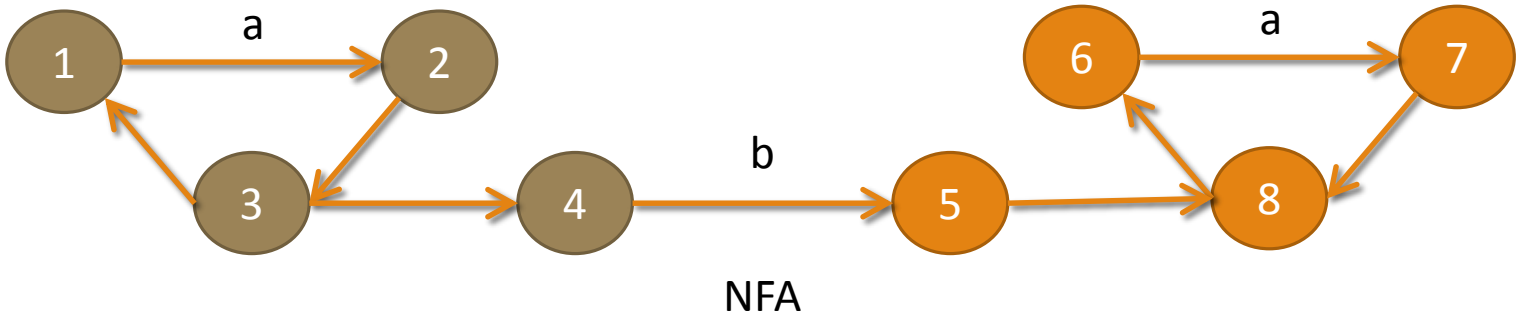


Struktur Data				
Proses	1,2,3,4	5,6,8		

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	5,6,8
1,2,3,4	1,2,3,4	
5,6,8		

EpsilonClosure(Move('1,2,3,4', 'a'))



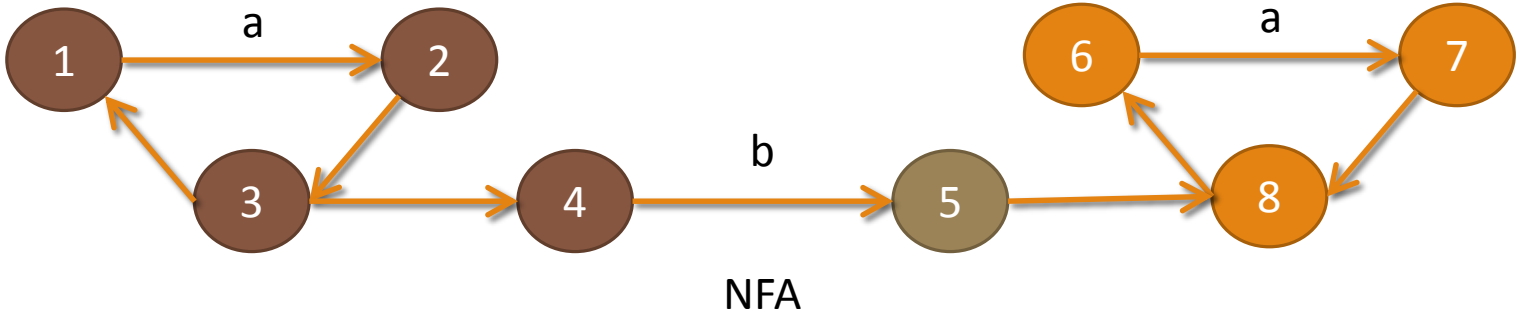
NFA

Struktur Data				
Proses	1,2,3,4	5,6,8		

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	5,6,8
1,2,3,4	1,2,3,4	
5,6,8		

EpsilonClosure(Move('1,2,3,4', 'b'))



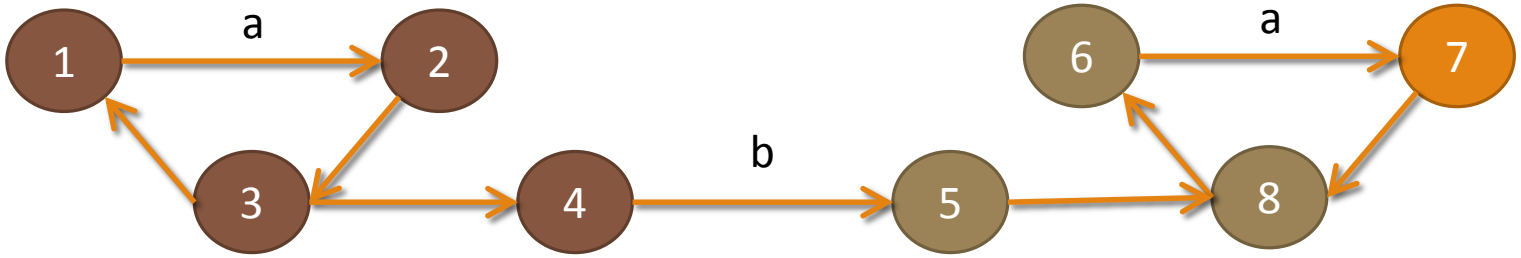
NFA

Struktur Data				
Proses	1,2,3,4	5,6,8		

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	5,6,8
1,2,3,4	1,2,3,4	5,6,8
5,6,8		

EpsilonClosure(Move('1,2,3,4', 'b'))



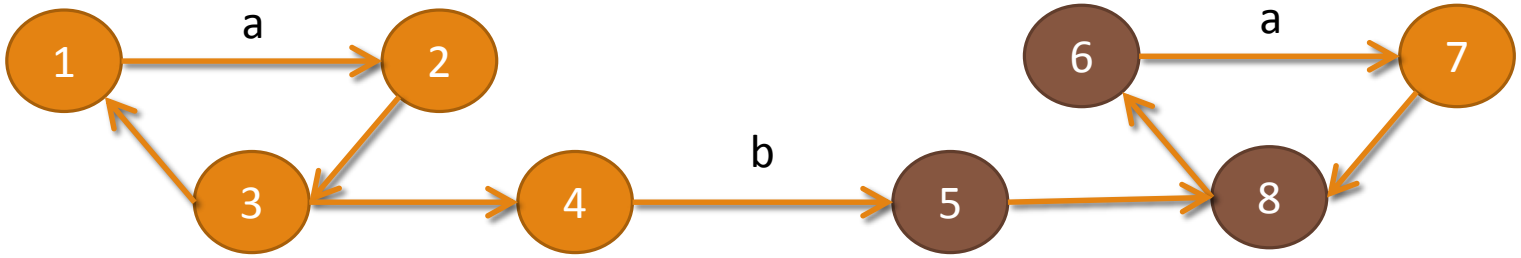
NFA

Struktur Data				
Proses	1,2,3,4	5,6,8		

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	5,6,8
1,2,3,4	1,2,3,4	5,6,8
5,6,8		

EpsilonClosure(Move(‘5,6,8’, ‘a’))



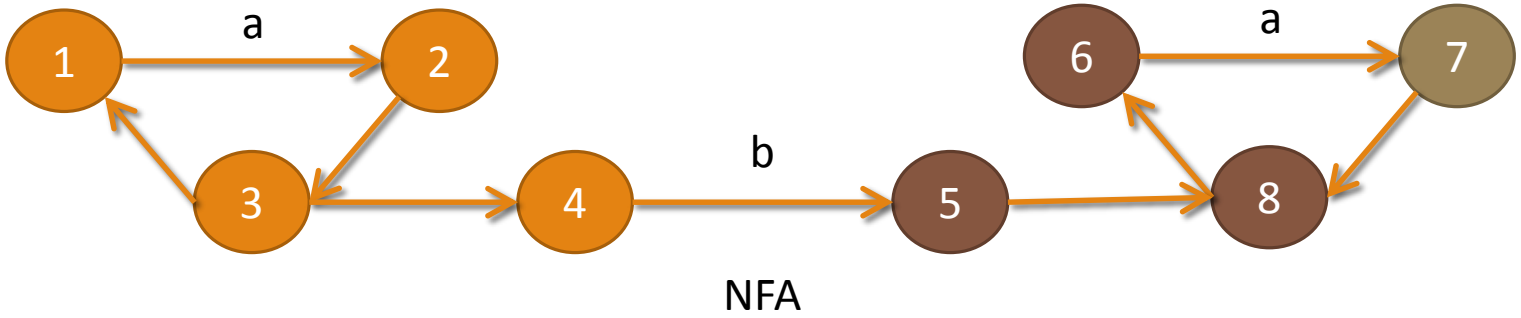
NFA

Struktur Data				
Proses	5,6,8			

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	5,6,8
1,2,3,4	1,2,3,4	5,6,8
5,6,8		

EpsilonClosure(Move(‘5,6,8’, ‘a’))



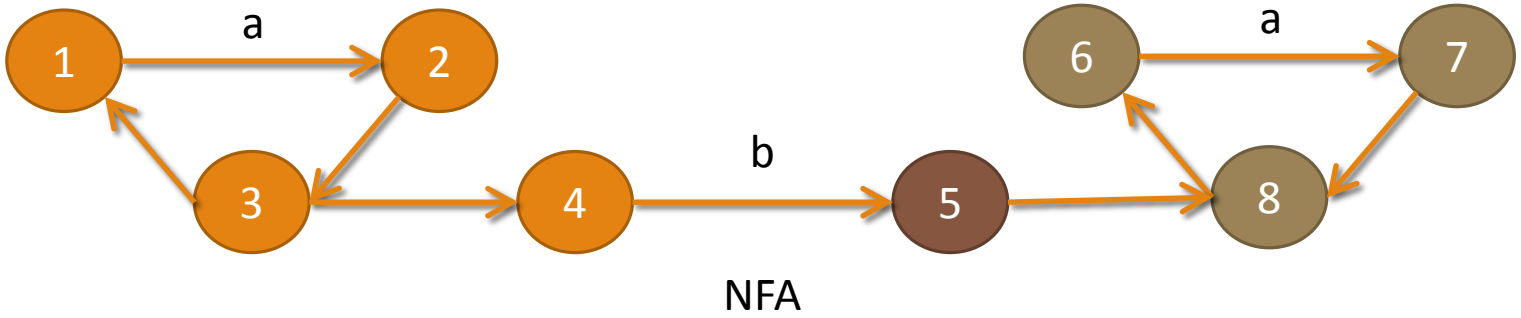
NFA

Struktur Data				
Proses	5,6,8			

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	5,6,8
1,2,3,4	1,2,3,4	5,6,8
5,6,8	6,7,8	
6,7,8		

EpsilonClosure(Move('5,6,8', 'a'))



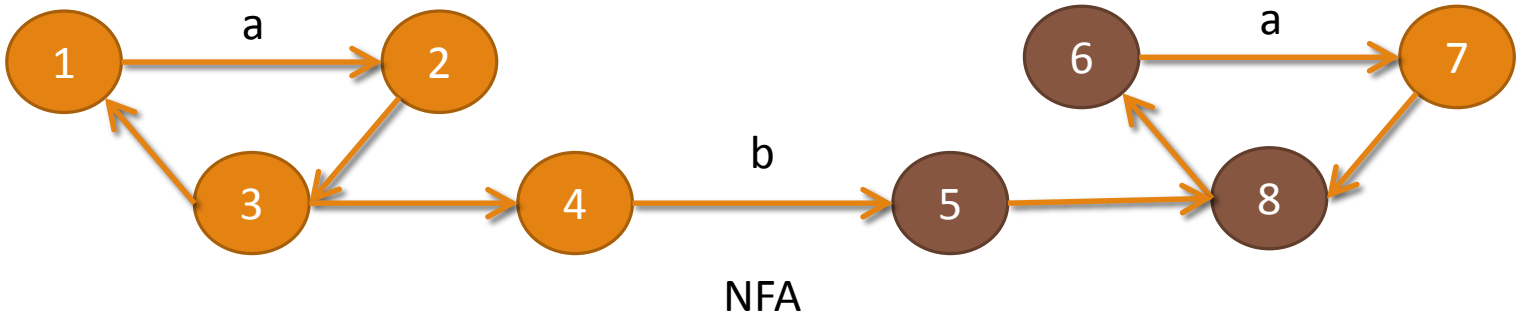
NFA

Struktur Data				
Proses	5,6,8	6,7,8		

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	5,6,8
1,2,3,4	1,2,3,4	5,6,8
5,6,8	6,7,8	-
6,7,8		

EpsilonClosure(Move('5,6,8', 'b'))



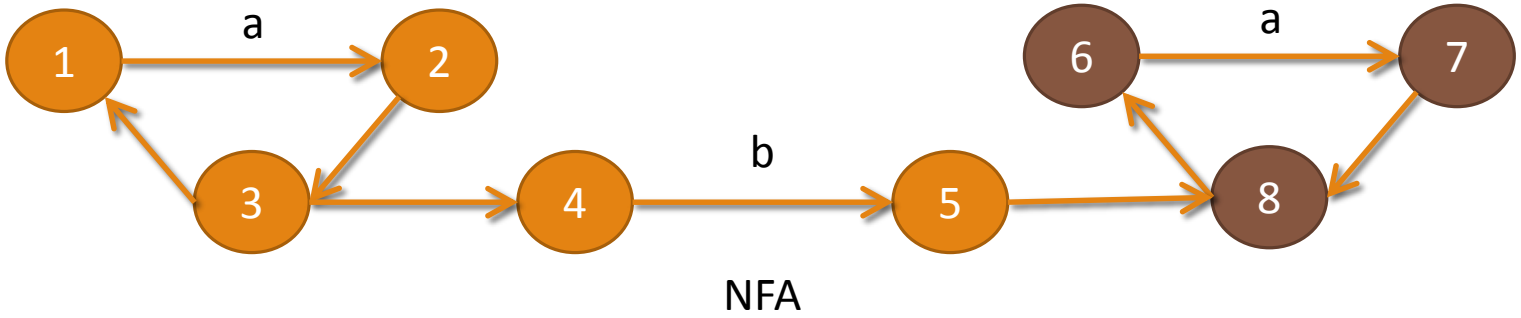
NFA

Struktur Data				
Proses	5,6,8	6,7,8		

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	5,6,8
1,2,3,4	1,2,3,4	5,6,8
5,6,8	6,7,8	-
6,7,8		

EpsilonClosure(Move('6,7,8', 'a'))



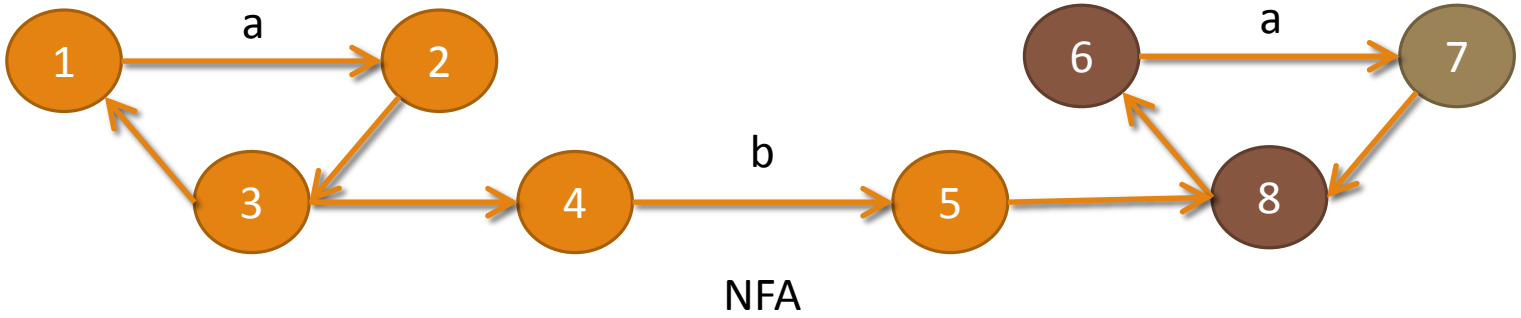
NFA

Struktur Data				
Proses	6,7,8			

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	5,6,8
1,2,3,4	1,2,3,4	5,6,8
5,6,8	6,7,8	-
6,7,8		

EpsilonClosure(Move('6,7,8', 'a'))



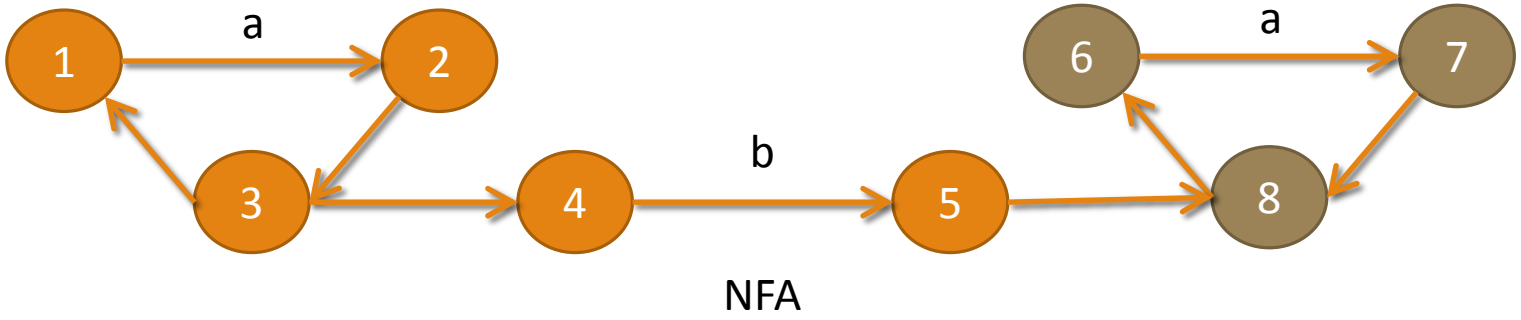
NFA

Struktur Data				
Proses	6,7,8			

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	5,6,8
1,2,3,4	1,2,3,4	5,6,8
5,6,8	6,7,8	-
6,7,8	6,7,8	

EpsilonClosure(Move('6,7,8', 'a'))



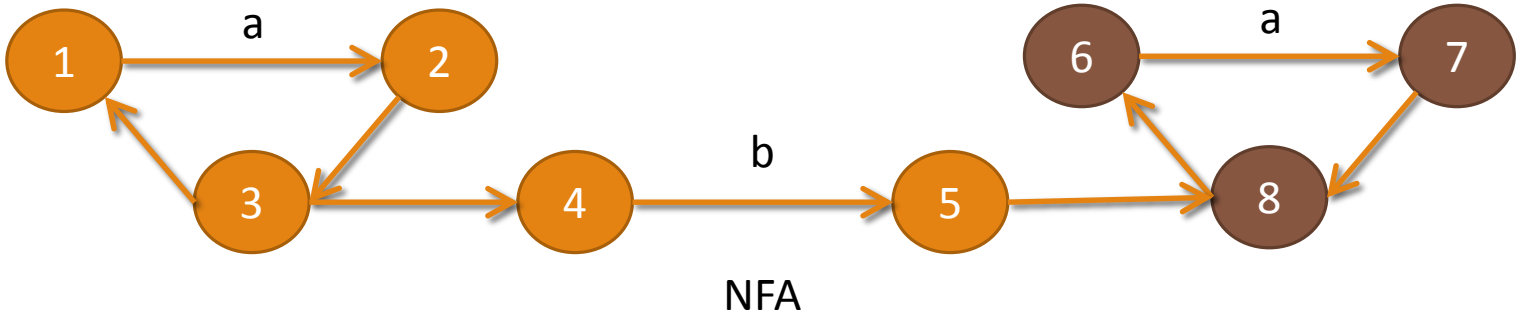
NFA

Struktur Data				
Proses	6,7,8			

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	5,6,8
1,2,3,4	1,2,3,4	5,6,8
5,6,8	6,7,8	-
6,7,8	6,7,8	-

EpsilonClosure(Move('6,7,8', 'b'))



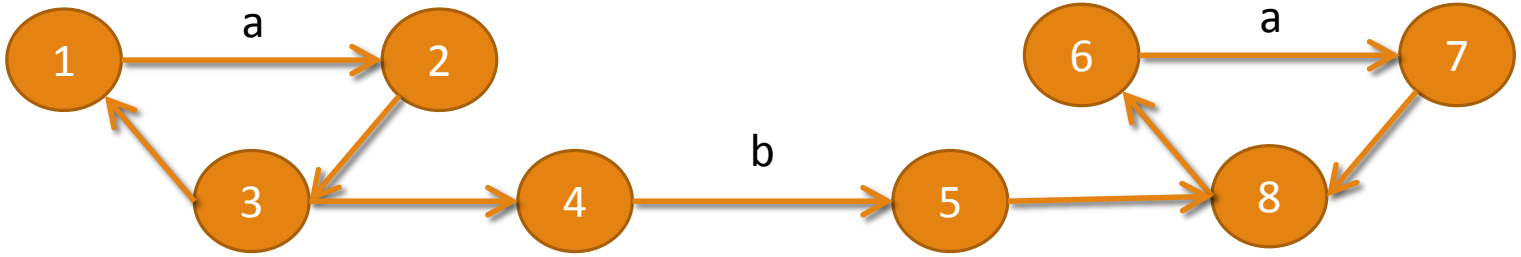
NFA

Struktur Data				
Proses	6,7,8			

Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	5,6,8
1,2,3,4	1,2,3,4	5,6,8
5,6,8	6,7,8	-
6,7,8	6,7,8	-

Karena queue proses telah kosong, maka proses konversi selesai



NFA

Struktur Data				
Proses	-			

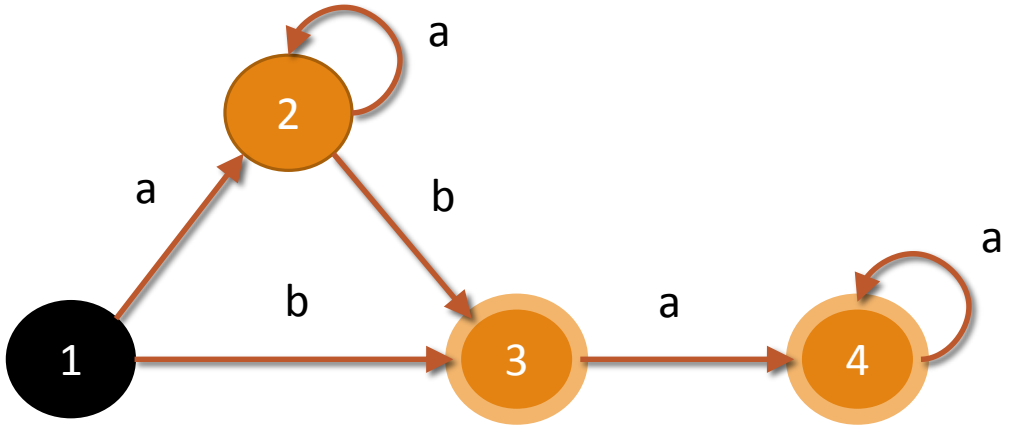
Merubah NFA ke DFA

	aε	bε
1,3,4	1,2,3,4	5,6,8
1,2,3,4	1,2,3,4	5,6,8
5,6,8	6,7,8	-
6,7,8	6,7,8	-

Tabel diatas adalah representasi DFA setelah dikonversi dari NFA atau dapat disederhanakan menjadi

	aε	bε
1	2	3
2	2	3
3	4	-
4	4	-

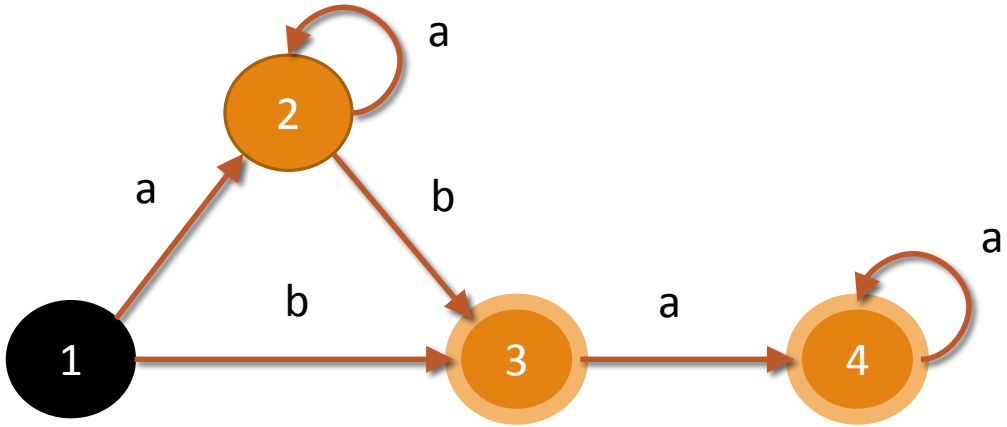
Merubah NFA ke DFA



Graph diatas adalah DFA state machine yang terbentuk setelah dikonversi dari NFA.

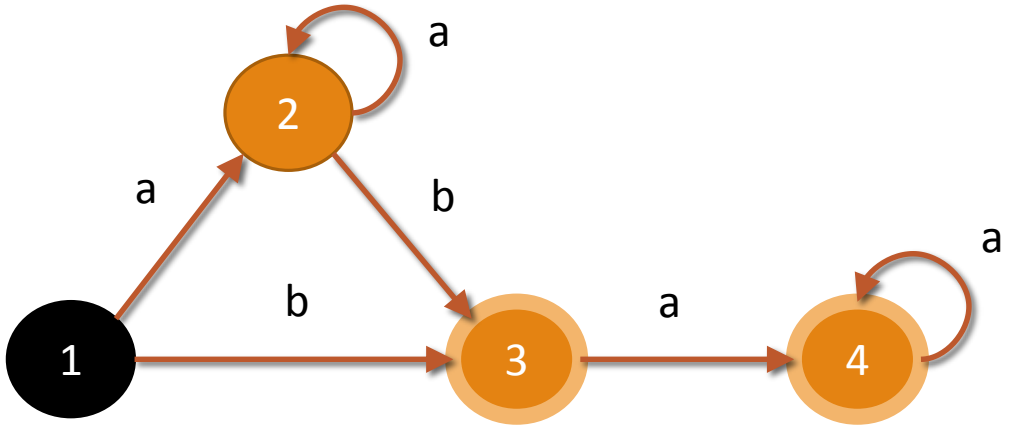
	aε	bε
1	2	3
2	2	3
3	4	-
4	4	-

Penggunaan DFA untuk Pencarian String



L yang diberikan saat masukan adalah 100.

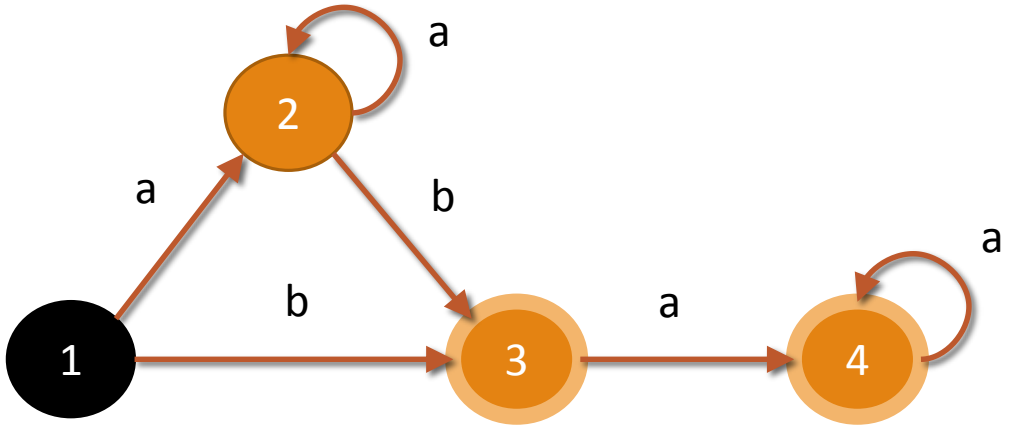
Penggunaan DFA untuk Pencarian String



$$M = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Ajacency matrix M untuk DFA yang terbentuk

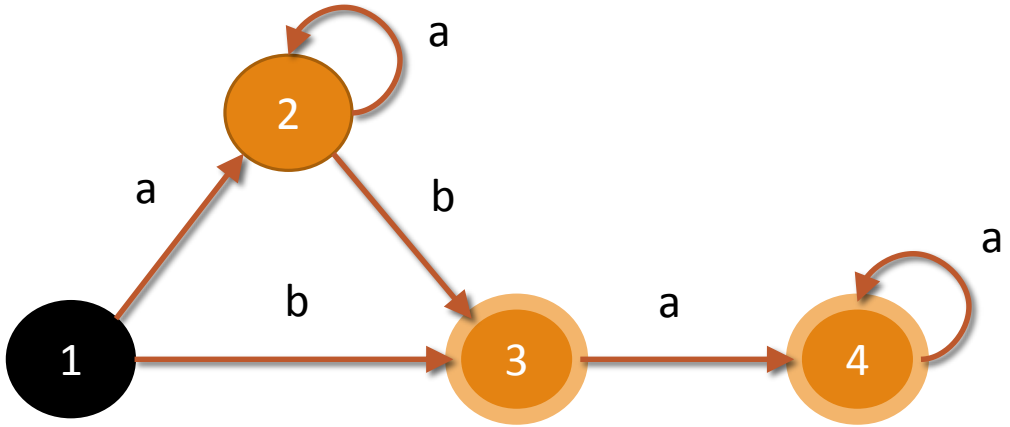
Penggunaan DFA untuk Pencarian String



$$M^{100} = \begin{bmatrix} 0 & 1 & 1 & 99 \\ 0 & 1 & 1 & 99 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Hasil M^{100} ditunjukkan seperti pada matriks diatas

Penggunaan DFA untuk Pencarian String & Keluaran program



$$M^{100} = \begin{bmatrix} 0 & 1 & 1 & 99 \\ 0 & 1 & 1 & 99 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Banyak string yang dapat di terima adalah jumlah seluruh matrix [i][j] dimana i adalah state mulai dan j adalah state finish.

Maka output yang dikeluarkan pada program adalah $hasil = M_{1,3} + M_{1,4}$

$= 1 + 99 = 100$ strings

Uji Coba & Evaluasi

ITS
Institut
Teknologi
Sepuluh Nopember

Lingkungan Uji Coba

Lingkungan uji coba yang akan digunakan adalah,

Perangkat Keras

- Processor Intel® Core™ i3-2310M CPU @ 2.10 GHz
- RAM 4 GB
- Sistem Operasi 64-bit

Perangkat Lunak

- Sistem Operasi Windows 8.0
- Integrated Development Environment Code::Blocks 13.12

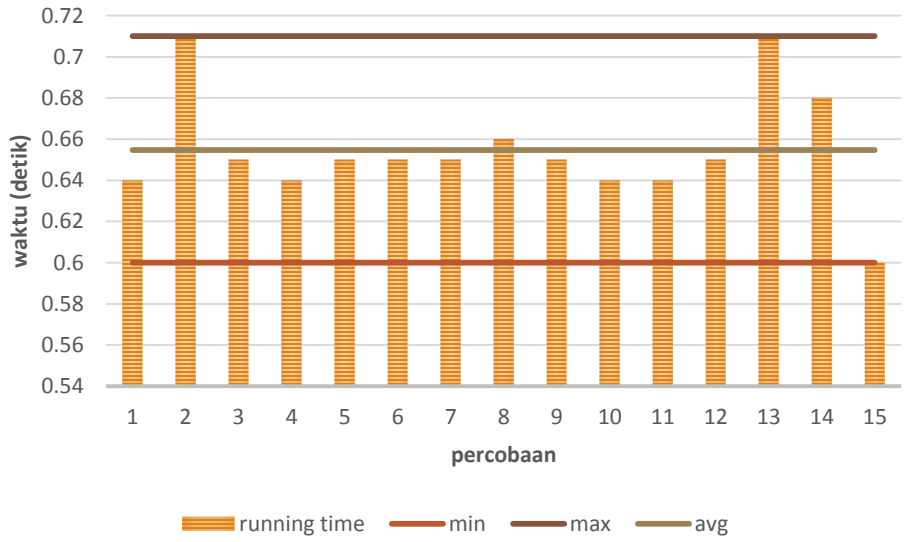
Uji Coba Kebenaran

14373986	☐	2015-06-02 06:46:05	Count Strings	**accepted** edit ideone it	0.60	3.0M	C++ 4.3.2

Uji coba kebenaran dilakukan pada SPOJ dan mendapatkan umpan balik Accepted yang menandakan program yang dibuat dapat menyelesaikan permasalahan yang diberikan

Untuk beberapakali uji coba berikut grafik waktu dari program untuk menyelesaikan permasalahan dengan statistik sebagai berikut,

- Rata-rata waktu eksekusi 0.654 detik
- Minimum waktu eksekusi 0.600 detik
- Maksimum waktu eksekusi 0.710 detik



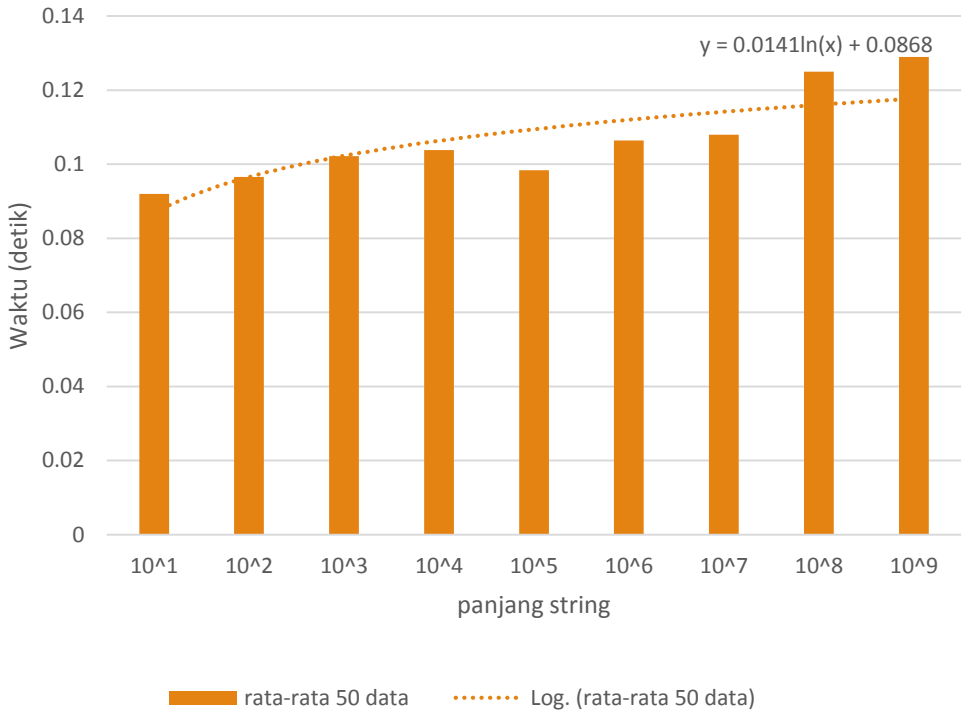
Uji Coba Kinerja

Pengaruh Panjang L terhadap waktu eksekusi program

Parameter uji yang digunakan adalah,

- Bilangan L yang berbeda yakni $10^1, 10^2, \ldots, 10^9$
- *Regular expression* yang sama sepanjang 100 karakter

Hasil yang didapatkan adalah dengan parameter tersebut waktu yang diperlukan untuk eksekusi program mendekati kurva logaritmik



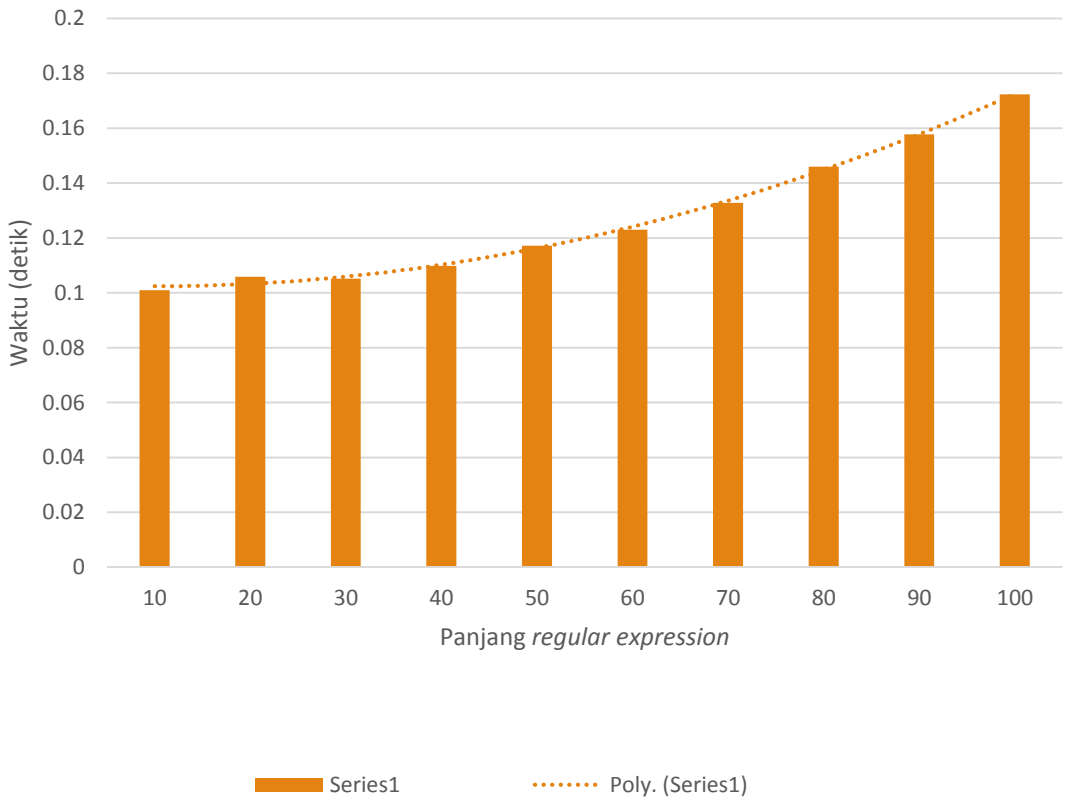
Uji Coba Kinerja

Pengaruh Panjang L terhadap waktu eksekusi program

Parameter uji yang digunakan adalah,

- Bilangan L yang sama yakni 10^9
- *Regular expression* yang berbeda dengan panjang $10,20,30, \ldots, 100$ karakter

Hasil yang didapatkan adalah dengan parameter tersebut waktu yang diperlukan untuk eksekusi program mendekati kurva polynomial berderajat 2



Kesimpulan & Saran

Kesimpulan

- Implementasi yang dilakukan berhasil menyelesaikan permasalahan studi kasus yang digunakan dalam penyusunan tugas akhir, yakni studi kasus SPOJ klasik 10354.
- Untuk evaluasi *regular expression*, ditandai dengan besarnya L yang berbeda memiliki pengaruh terhadap waktu yang dibutuhkan program secara logaritmik.
- Untuk konversi *regular expression* menjadi *state machine* NFA dan DFA, panjang *regular expression* memiliki pengaruh terhadap waktu yang dibutuhkan secara polynomial berderajat 2, dimana semakin panjang *regular expression* maka semakin besar pula waktu yang dibutuhkan.

Saran

1. Mencoba menggunakan metode Glushkov untuk interpretasi *regular expression*, metode Glushkov memiliki properti khusus pada NFA yang terbentuk, yakni NFA tersebut tidak memiliki transisi epsilon. NFA hasil dari metode Glushkov sudah dapat digunakan untuk melakukan pencarian *string* sehingga secara teori metode Glushkov akan memiliki waktu yang lebih cepat untuk menyelesaikan permasalahan disbanding metode Thompson.

ITS
Institut
Teknologi
Sepuluh Nopember

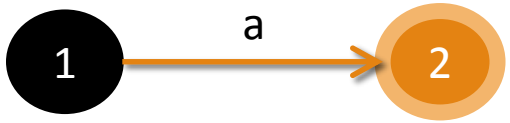
Terimakasih

Merubah Regular Expression Ke NFA

RE = ((a|b)+a)

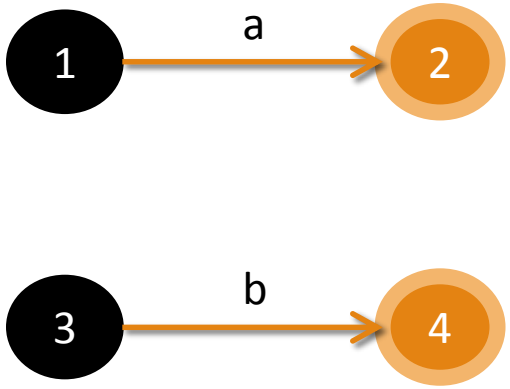
Merubah Regular Expression Ke NFA

RE = ((a|b)+a)



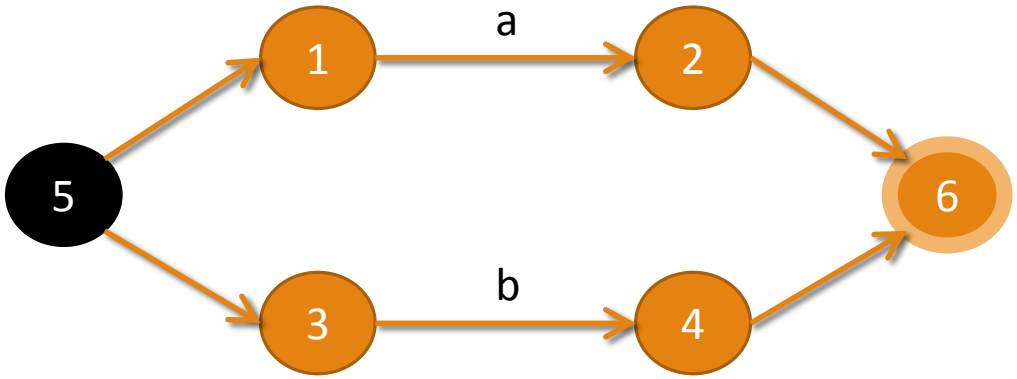
Merubah Regular Expression Ke NFA

RE = ((a|b)+a)



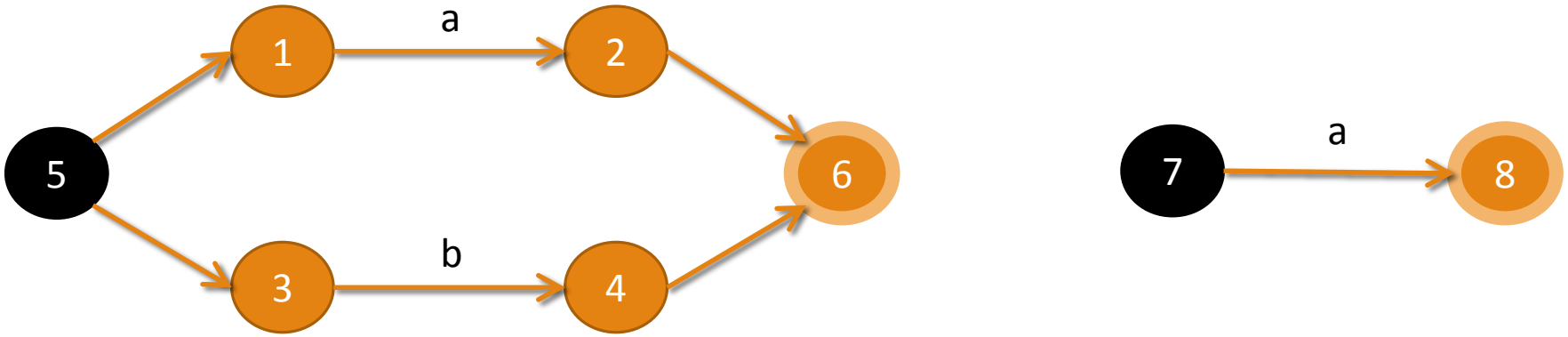
Merubah Regular Expression Ke NFA

RE = ((a|b)+a)



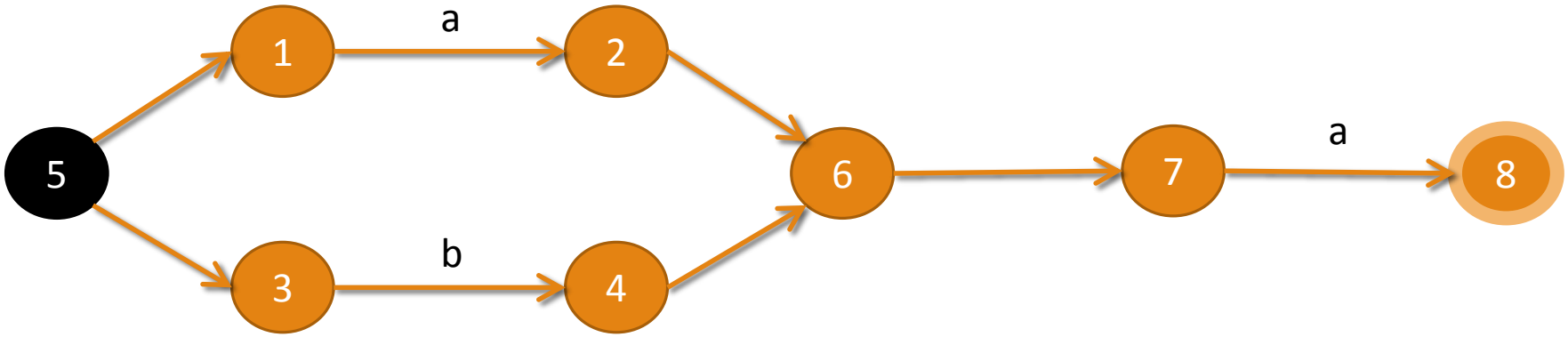
Merubah Regular Expression Ke NFA

RE = ((a|b)+a)



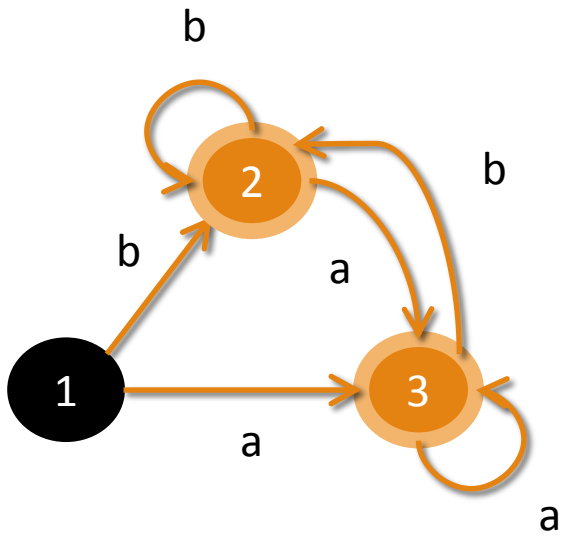
Merubah Regular Expression Ke NFA

RE = ((a|b)+a)



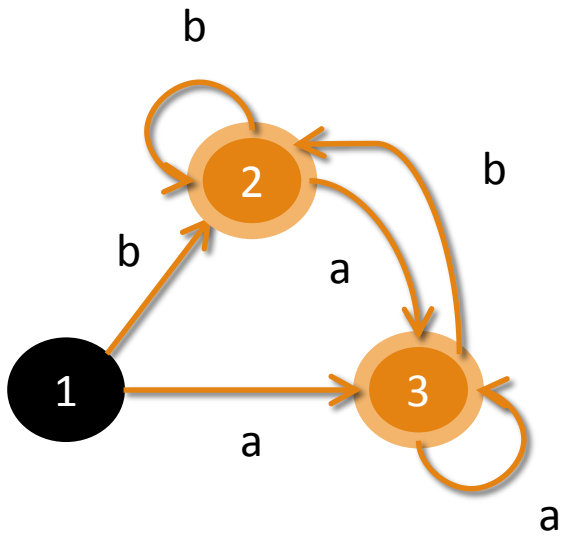
Adjacency Matrix

((a|b)*)

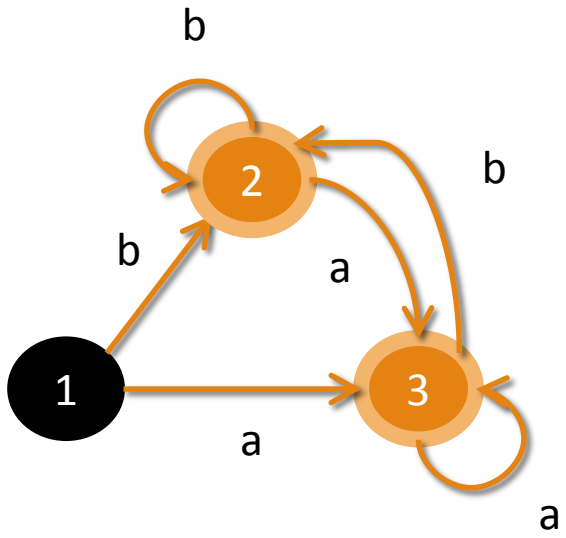


Adjacency Matrix

$$A = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$



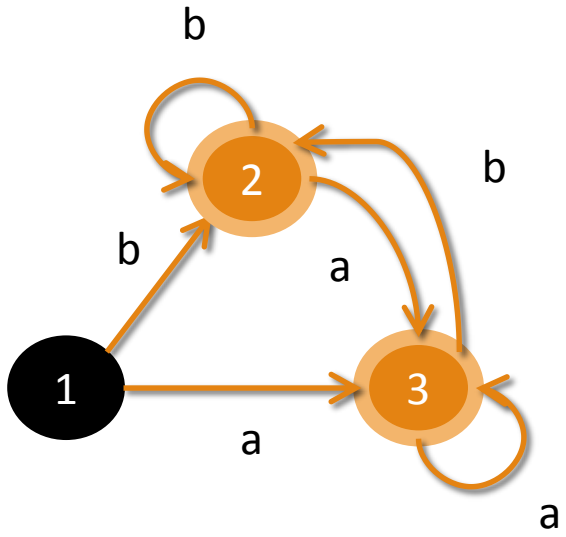
Adjacency Matrix



$$A = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

$$A^2 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 2 \\ 0 & 2 & 2 \\ 0 & 2 & 2 \end{bmatrix}$$

Adjacency Matrix



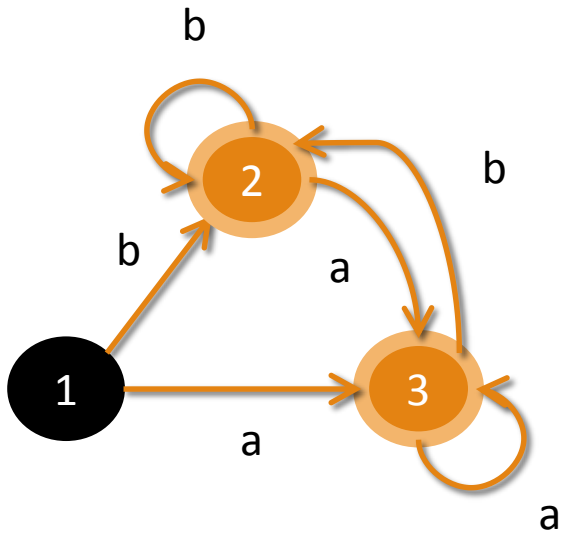
$$A = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

$$A^2 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 2 \\ 0 & 2 & 2 \\ 0 & 2 & 2 \end{bmatrix}$$

$$A^3 = \begin{bmatrix} 0 & 2 & 2 \\ 0 & 2 & 2 \\ 0 & 2 & 2 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 4 & 4 \\ 0 & 4 & 4 \\ 0 & 4 & 4 \end{bmatrix}$$

Start pada 1 dan berakhir pada 2

$$A^2 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 2 \\ 0 & 2 & 2 \\ 0 & 2 & 2 \end{bmatrix}$$



Start pada 1 dan berakhir pada 2

$$A^3 = \begin{bmatrix} 0 & 2 & 2 \\ 0 & 2 & 2 \\ 0 & 2 & 2 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 4 & 4 \\ 0 & 4 & 4 \\ 0 & 4 & 4 \end{bmatrix}$$