W

;/--n

**TUGAS AKHIR – KI1502**

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

**MUHAMMAD YUNUS BAHARI**

**NRP 5111100079**

**Dosen Pembimbing I**

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

**Dosen Pembimbing II**

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

**JURUSAN TEKNIK INFORMATIKA**

**Fakultas Teknologi Informasi**

**Institut Teknologi Sepuluh Nopember**

**Surabaya 2015**

***[Halaman ini sengaja dikosongkan]***

**TUGAS AKHIR – KI1502**

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

**MUHAMMAD YUNUS BAHARI**

**NRP 5111100079**

**Dosen Pembimbing I**

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

**Dosen Pembimbing II**

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

**JURUSAN TEKNIK INFORMATIKA**

**Fakultas Teknologi Informasi**

**Institut Teknologi Sepuluh Nopember**

**Surabaya 2015**

****

***[Halaman ini sengaja dikosongkan]***

**UNDERGRADUATE THESES – KI1502**

**IMPLEMENTATION OF DETERMINISTIC FINITE AUTOMATON MODEL FOR REGULAR EXPRESSION INTERPRETATION A CASE STUDY AT SPOJ CLASSIC PROBLEM 10354**

**MUHAMMAD YUNUS BAHARI**

**NRP 5111100079**

**Supervisor I**

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

**Supervisor II**

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

**DEPARTMENT OF INFORMATICS**

**FACULTY OF INFORMATION TECHNOLOGY**

**INSTITUT TEKNOLOGI SEPULUH NOPEMBER**

**SURABAYA 2015**

# LEMBAR PENGESAHAN

**IMPLEMENTASI MODEL *DETERMINISTIC FINITE AUTOMATON* UNTUK INTERPRETASI *REGULAR EXPRESSION* PADA STUDI KASUS PERMASALAHAN SPOJ KLASIK 10354**

**TUGAS AKHIR**

Diajukan Untuk Memenuhi Salah Satu Syarat

Memperoleh Gelar Sarjana Komputer

pada

Bidang Studi Desain dan Terapan Komputasi

Program Studi S-1 Jurusan Teknik Informatika

Fakultas Teknologi Informasi

Institut Teknologi Sepuluh Nopember

Oleh

**MUHAMMAD YUNUS BAHARI**

**NRP : 5111 100 079**

Disetujui oleh Dosen Pembimbing Tugas Akhir:

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

NIP: 198409042010121002 (Pembimbing 1)

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

NIP: 197002131994021001 (Pembimbing 2)

**SURABAYA**

**MEI, 2015**

***[Halaman ini sengaja dikosongkan]***

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

|  |  |  |
| --- | --- | --- |
| **Nama Mahasiswa** | **:** | **MUHAMMAD YUNUS BAHARI** |
| **NRP** | **:** | **5111100079** |
| **Jurusan** | **:** | **Teknik Informatika FTIF-ITS** |
| **Dosen Pembimbing 1** | **:** | **Arya Yudhi Wijaya, S.Kom., M.Kom.** |
| **Dosen Pembimbing 2** | **:** | **Rully Soelaiman, S.Kom., M.Kom.** |

# *Abstrak*

*Dewasa ini, …*

***Kata kunci: …***

**IMPLEMENTATION OF DETERMINISTIC FINITE AUTOMATON MODEL FOR REGULAR EXPRESSION INTERPRETATION A CASE STUDY AT SPOJ CLASSIC PROBLEM 10354**

|  |  |  |
| --- | --- | --- |
| **Student’s Name** | **:** | **MUHAMMAD YUNUS BAHARI** |
| **Student’s ID** | **:** | **5111100079** |
| **Department** | **:** | **Teknik Informatika FTIF-ITS** |
| **First Advisor** | **:** | **Arya Yudhi Wijaya, S.Kom., M.Kom.** |
| **Second Advisor** | **:** | **Rully Soelaiman, S.Kom., M.Kom.** |

# *Abstract*

*Nowadays , …*

***Keywords: …***

# KATA PENGANTAR



Surabaya, Mei 2015

# DAFTAR ISI

[LEMBAR PENGESAHAN v](#_Toc421020251)

[*Abstrak* vii](#_Toc421020252)

[*Abstract* viii](#_Toc421020253)

[KATA PENGANTAR ix](#_Toc421020254)

[DAFTAR ISI x](#_Toc421020255)

[DAFTAR GAMBAR xiii](#_Toc421020256)

[DAFTAR TABEL xiv](#_Toc421020257)

[DAFTAR KODE SUMBER xvi](#_Toc421020258)

[BAB I PENDAHULUAN 1](#_Toc421020259)

[1.1 Latar Belakang 1](#_Toc421020260)

[1.2 Rumusan Masalah 1](#_Toc421020261)

[1.3 Batasan Masalah 2](#_Toc421020262)

[1.4 Tujuan 2](#_Toc421020263)

[1.5 Manfaat 2](#_Toc421020264)

[1.6 Metodologi 3](#_Toc421020265)

[1.7 Sistematika Penulisan Laporan Tugas Akhir 4](#_Toc421020266)

[BAB II TINJAUAN PUSTAKA 6](#_Toc421020267)

[2.1 Regular Expression 6](#_Toc421020269)

[2.2 String Matching Menggunakan Regular Expression 7](#_Toc421020270)

[2.3 Nondeterministic Finite Automaton 8](#_Toc421020271)

[2.4 Metode Thompson 10](#_Toc421020272)

[2.5 Deterministic Finite Automaton 12](#_Toc421020273)

[2.6 Metode Subset Construction 13](#_Toc421020274)

[2.7 Perpangkatan Matriks 14](#_Toc421020275)

[2.8 Permasalahan CTSTRING di SPOJ 15](#_Toc421020276)

[BAB III DESAIN PERANGKAT LUNAK 17](#_Toc421020277)

[3.1 Desain Sistem Secara Umum 17](#_Toc421020279)

[3.2 Desain Algoritma 18](#_Toc421020280)

[3.2.1 Desain fungsi Preprocess 18](#_Toc421020281)

[3.2.2 Desain fungsi ConvertREtoNFA 19](#_Toc421020282)

[3.2.3 Desain fungsi ConvertNFAtoDFA 21](#_Toc421020283)

[3.2.4 Desain fungsi MatrixPower 25](#_Toc421020284)

[BAB IV IMPLEMENTASI 26](#_Toc421020285)

[4.1 Lingkungan Implementasi 26](#_Toc421020287)

[4.1.1 Parameter yang Digunakan 26](#_Toc421020288)

[BAB V UJI COBA DAN EVALUASI 27](#_Toc421020289)

[5.1 Lingkungan Uji Coba 27](#_Toc421020291)

[5.2 Data *Training* dan Data *Testing* 27](#_Toc421020292)

[BAB VI KESIMPULAN DAN SARAN 29](#_Toc421020293)

[6.1 Kesimpulan 29](#_Toc421020295)

[6.2 Saran 29](#_Toc421020296)

[DAFTAR PUSTAKA 30](#_Toc421020297)

[LAMPIRAN 32](#_Toc421020298)

***[Halaman ini sengaja dikosongkan]***

# DAFTAR GAMBAR

Gambar 2.3.1 Representasi NFA dalam *directed* *graph* 9

Gambar 2.4.1 *Automaton* transisi epsilon 10

Gambar 2.4.2 *Automaton* dengan sebuah input karakter 10

Gambar 2.4.3 *Automaton* operator *concatenate* 11

Gambar 2.4.4 *Automaton* operator Union 11

Gambar 2.4.5 *Automaton* operator Klenee star 11

Gambar 2.4.6 *Automaton* awal yang terbentuk 12

Gambar 2.4.7 *Automaton* setelah operator *Klenee star* 12

Gambar 2.4.8 *Automaton* setelah operator *Union* 12

Gambar 2.4.9 NFA representasi dari *regular expression*  12

Gambar 2.7.1 Deskripsi permasalahan CTSTRING 15

Gambar 2.7.2 Contoh masukan dan keluaran permasalahan 16

Gambar 3.1.1 *Pseudocode* fungsi main(1) 18

Gambar 3.1.2 *Pseudocode* fungsi main(2) 18

Gambar 3.2.1 *Pseudocode* fungsi preprocess 19

Gambar 3.2.2 *Pseudocode* fungsi ConvertREtoNFA 21

Gambar 3.2.3 *Pseudocode* fungsi ConvertNFAtoDFA 22

Gambar 3.2.4 *Pseudocode* fungsi EpsilonClosure 23

Gambar 3.2.5 *Pseudocode* fungsi MoveClosure 24

Gambar 3.2.6 *Pseudocode* fungsi CreateDFAState 25

Gambar 3.2.7 *Pseudocode* fungsi MatrixPower 25

***[Halaman ini sengaja dikosongkan]***

# DAFTAR TABEL

Tabel 2.3.1 Representasi NFA dalam tabel 9

***[Halaman ini sengaja dikosongkan]***

# DAFTAR KODE SUMBER

***[Halaman ini sengaja dikosongkan]***

# PENDAHULUAN

## Latar Belakang

Topik Tugas Akhir ini diangkat dari permasalahan yang terdapat pada *Online Judge* SPOJ dengan nomor soal 10354 dengan kode CTSTRING. Pada permasalahan ini diberikan dua buah masukan yaitu sebuah *string* yang merupakan *Regular Expression* dan sebuah bilangan N. Dari dua masukan tersebut kita diminta untuk menentukan berapa banyak *string* dengan panjang N yang memenuhi *Regular Expression* yang diberikan.

*Regular Expression* merupakan metode yang umum digunakan dalam pemrosesan *string*. Sebagai contoh untuk pengecekan alamat e-mail yang sah dan pencarian kata dalam dokumen. Untuk menyelesaikan permasalahan tersebut diperlukan cara untuk menginterpretasikan *Regular Expression* ke suatu model yang dapat diterjemahkan kedalam logika pemrograman. *Deterministic Finite Automaton* merupakan model dengan keadaan tertentu yang dapat berubah menjadi keadaan yang lain mengikuti masukan tertentu yang dapat diterjemahkan kedalam logika pemrograman.

Oleh karena itu, dalam penyelesaian studi kasus SPOJ klasik 10354 teknik pemodelan yang digunakan untuk menginterpretasikan *Regular Expression* adalah *Deterministic Finite Automaton*.

## Rumusan Masalah

Rumusan masalah yang diangkat dalam Tugas Akhir ini sebagai berikut:

1. Bagaimana memodelkan *Regular Expression* ke dalam bentuk *Deterministic Finite Automaton* ?
2. Bagaimana mengimplementasikan model *Deterministic Finite Automaton* untuk menginterpretasikan *Regular Expression*?

## Batasan Masalah

Permasalahan yang dibahas pada Tugas Akhir ini memiliki beberapa batasan, yaitu sebagai berikut:

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 10­9.

## Tujuan

Tujuan dari Tugas Akhir ini adalah sebagai berikut:

1. Mengetahui bagaimana cara memodelkan *Regular Expression* kedalam bentuk *Deterministic Finite Automaton*.
2. Mengetahui pengaplikasian model *Deterministic Finite Automaton* untuk interpretasi *Regular Expression*.

## Manfaat

Tugas Akhir ini diharapkan dapat membantu memahami penggunaan *Deterministic Finite Automaton* sebagai salah satu teknik pemodelan untuk menginterpretasikan *Regular Expression*.

## Metodologi

Tahapan-tahapan yang dilakukan dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir.

Tahap awal untuk memulai pengerjaan Tugas Akhir ada penyusunan proposal Tugas Akhir. Pada proposal ini, penulis mengajukan Implementasi Model *Deterministic Finite Automaton* untuk Interpretasi *Regular Expression* pada Studi Kasus Permasalahan SPOJ Klasik 10354.

1. Studi literatur

Pada tahap studi literatur penulis melakukan pencarian informasi yang diperlukan untuk penyelesaian permasalahan yang akan dikerjakan. Informasi untuk penyelesaian permasalahan didapatkan dari buku acuan yang berhubungan dengan algoritma penyelesaian permasalahan yang diangkat pada Tugas Akhir ini.

1. Implementasi perangkat lunak

Tahap implementasi merupakan tahap untuk merubah rancangan algoritma yang diperoleh dan disusun dari berbagai literatur yang kemudian disesuaikan dengan permasalahan terkait. Implementasi program ini ditulis dengan menggunakan bahasa pemrograman C++.

1. Pengujian dan evaluasi

Pada tahap ini penulis melakukan uji coba dengan melakukan pengiriman kode program pada *Online Judge* SPOJsesuai dengan permasalahan yang terkait apakah solusi yang diusulkan sudah sesuai dengan kriteria permasalahan yang ada.

1. Penyusunan buku Tugas Akhir.

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat.

## Sistematika Penulisan Laporan Tugas Akhir

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini:

**Bab I Pendahuluan**

Bab yang berisi mengenai latar belakang, tujuan, dan manfaat dari pembuatan Tugas Akhir. Selain itu permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penulisan juga merupakan bagian dari bab ini.

**Bab II Dasar Teori**

Bab ini berisi penjelasan mengenai dasar-dasar implementasi dan teori-teori yang digunakan untuk mendukung pembuatan program Tugas Akhir ini.

**Bab III Desain Perangkat Lunak**

Bab ini berisi tentang desain sistem yang disajikan dalam bentuk *pseudocode*.

**Bab IV Implementasi**

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa kode sumber yang digunakan untuk proses implementasi.

**Bab V Uji Coba Dan Evaluasi**

Bab ini menjelaskan kemampuan perangkat lunak dengan melakukan pengujian kebenaran dan pengujian kinerja dari sistem yang telah dibuat.

**Bab VI Kesimpulan Dan Saran**

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

# 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.



## Regular Expression

Sebuah *string* didefinisikan sebagai rangkaian karakter berhingga yang ada dalam semesta simbol . Sebagai contoh FAZZIBASHASOL adalah *string*  dalam semesta simbol [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*  adalah sebuah *string* yang memiliki simbol dalam set, yang didefinisikan secara rekursifsebagai *string* kosong *,* simboldimana *,* dan *,, ,* dan *,* dimanadanadalah *regular expression* [1].

Salah satu contoh *regular expression* adalah . Untuk kemudahan membaca *regular expression* sering kali ditulis menjadi . Sehingga contoh *regular expression* tersebut dapat disederhanakan menjadi . 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 dimana yang memiliki struktur *RE* sebagai berikut [1]:

* Jika adalah , maka = {}, *string* kosong.
* Jika adalah , maka = , sebuah *string* dengan panjang satu karakter.
* Jika adalah *regular expression* dengan bentuk , maka .
* Jika adalah *regular expression* dengan bentuk , maka . Dimana adalah kumpulan string w sedemikian hingga , dimana dan . Operator “” adalah operator *concatenate*.
* Jika adalah *regular expression* dengan bentuk , maka . Operator “|” adalah operator *Union*.
* Jika adalah *regular expression* dengan bentuk , maka . Dimana dan untuk semua . Operator “\*” adalah operator *Klenee Star*.

Sebagai contoh *regular expression* akan merepresentasikan bahasa

.

## String Matching Menggunakan Regular Expression

Mencari *string* dalam sebuah kumpulan *string* dengan menggunakan *regular expression* diperlukan untuk merubah 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 sub-bab 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 sah untuk menuju *state* berikutnya dari simbol yang diberikansecara 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 aka nada satu *state* yang aktif, konversi dari NFA ke DFA akan dibahas pada sub-bab 2.6.

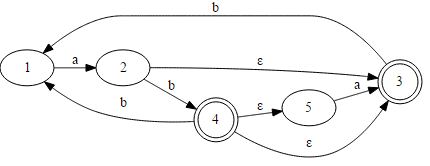
## Nondeterministic Finite Automaton

*Nondeterministic Finite Automaton* adalah sebuah *state machine* yang memiliki sekumpulan *state* berhingga *Q*, sekumpulan simbol sebagai masukan transisi *state*, fungsi transisi *F* dari *state* dengan masukan simbol menuju *state* , sekumpulan state sebagai *state* NFA selesai, dan sebuah *state* 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 symbol masukan . *state* awal NFA dimulai dari *state*  dan memiliki dua *state* selesai yakni . 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 sebagai simbol masukan dalam **Tabel 2.3.1**.

Tabel 2.3.1 Representasi NFA dalam tabel

|  |  |  |  |
| --- | --- | --- | --- |
|  | a | b |  |
|  | 2 | - | - |
|  | - | 4 | 3 |
|  | - | 1 | - |
|  | - | 1 | 3,5 |
|  | 3 | - | - |

Perpindahan dengan tanpa masukan simbol disebut dengan perpindahan/transisi epsilon. Dengan adanya transisi epsilon dalam NFA, suatu 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 . 

Gambar 2.3.1 Representasi NFA dalam *directed* *graph*

Hal ini membuat NFA menjadi sangat lambat ketika digunakan untuk melakukan pencarian *string*, terlebihjika 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.

## Metode Thompson

Metode Thompson adalah sebuah metode yang digunakan untuk melakukan konversi dari *regular expression* kedalam bentuk *Nondeterministic Finite Automaton* (NFA). Metode ini merupakan metode yang sederhana dan akan menghasilkan maksimal  *states* dan maksimal transisi dimana 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 ke bisa terjadi tanpa masukan simbol apapun.



Gambar 2.4.1 *Automaton* transisi epsilon

* *Automaton* untuk sebuah input karakter ditunjukkan pada **Gambar 2.4.2**. *Automaton* ini berarti perpindahan state dari ke terjadi apabila karakter dimasukkan sebagai simbol masukan ke NFA saat sebagai *state* aktif.



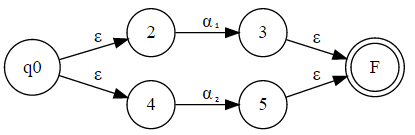
Gambar 2.4.2 *Automaton* dengan sebuah input karakter

* Automaton untuk operator “” (*Concatenate*) ditunjukkan pada **Gambar 2.4.3**. *Automaton* ini berarti perpindahan *state* dari ke terjadi apabila NFA diberikan dua kali masukan simbol secara berurutan. untuk berpindah dari *state* q0 ke *state* 2 dengan masukan , dan dari *state* 3 ke *state* F dengan masukan



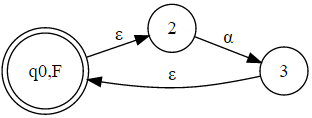
Gambar 2.4.3 *Automaton* operator *concatenate*

* *Automaton* untuk operator “|” (*Union*) ditunjukkan pada **Gambar 2.4.4**. *Automaton* ini berarti perpindahan *state* dari ke dapat terjadi apabila masukan yang diberikan merupakan salah satu dari simbol atau.



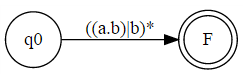
Gambar 2.4.4 *Automaton* operator Union

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

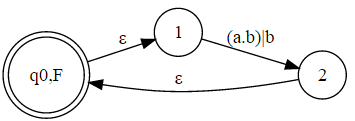


Gambar 2.4.5 *Automaton* operator Klenee star

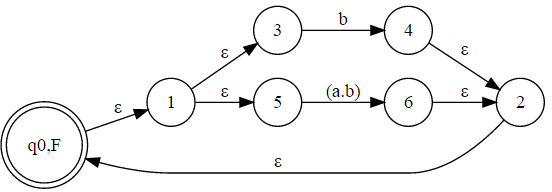
Dengan metode diatas contoh *regular expression*  akan terbentuk NFA dengan melakukan operasi yang sudah didefinisikan oleh metode tersebut dengan proses yang ditunjukan secara berurutan pada **Gambar 2.4.6, Gambar 2.4.7, Gambar 2.4.8,** dan **Gambar 2.4.9**.



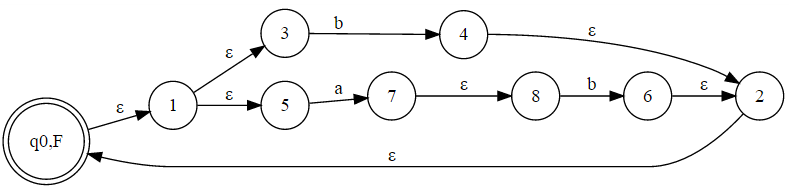
Gambar 2.4.6 *Automaton* awal yang terbentuk



Gambar 2.4.7 *Automaton* setelah operator *Klenee star*



Gambar 2.4.8 *Automaton* setelah operator *Union*



Gambar 2.4.9 NFA representasi dari *regular expression*

## Deterministic Finite Automaton

*Deterministic Finite Automaton* (DFA) adalah sebuah *state machine* spesialisasi dari NFA dimana 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 artian untuk masukan simbol yang berbeda maka *state* selanjutnya akan berbeda pula.

## 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 .
   1. 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.
   2. 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].

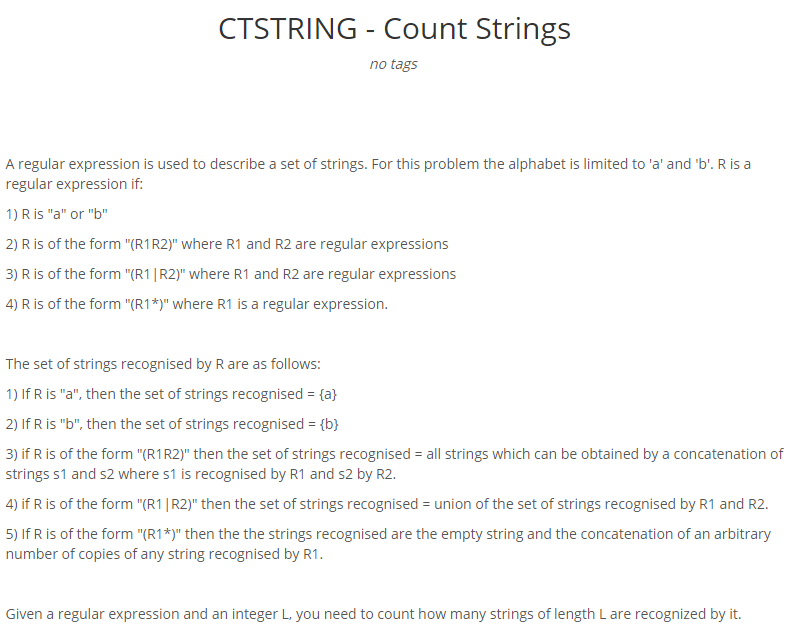
## Perpangkatan Matriks

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 . Kompleksitas big O dari proes 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 . Teknik ini secara rekursif didefinisikan pada persamaan 1 [2].

## Permasalahan CTSTRING di SPOJ

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** [3].



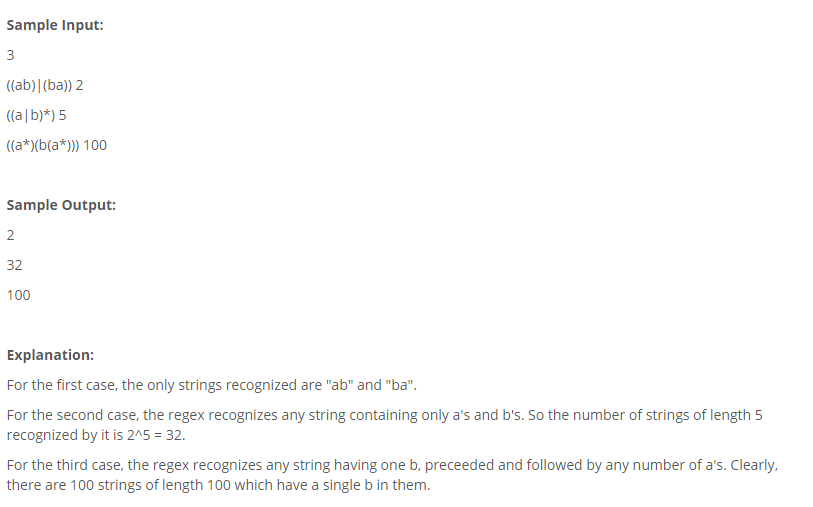
Gambar 2.8.1 Deskripsi permasalahan CTSTRING

Pada **Gambar 2.8.1** dijelaskan bahwa diberikan sebuah *regular expression* REdan sebuah bilangan bulat L,dengan batasan *regular expression* memiliki tiga operator yakni *concatenate*, *union,* dan *klenee star* dan simbol . 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 dimana 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 ). Contoh masukan dan keluaran juga dapat dilihat pada **Gambar 2.8.2**.



Gambar 2.8.2 Contoh masukan dan keluaran permasalahan

Adapun batasan dari permasalahan tersebut adalah,

1. .
2. .
3. .
4. Dinilai dalam lingkungan penilaian Intel G860 3Ghz
5. Batasan waktu 0.1 detik – 0.150 detik.
6. Batasan memori 1536 MB.

# DESAIN PERANGKAT LUNAK

Pada bab ini penulis menjelaskan tentang Desain metode yang digunakan untuk menyelesaikan permasalahan pada Tugas Akhir.



## Desain Sistem Secara Umum

Sistem yang dibuat akan menerima masukan berupa sebuah bilangan N pada awal sistem bekerja kemudian N masukan berikutnya berisikan *regular expression* RE dan bilangan L. Keluaran yang diharapkan dari sistem adalah bilangan T yang merupakan banyaknya *string* dengan panjang L yang dapat diterima oleh RE.

Pada setiap kasus uji, sistem akan melakukan *preprocess* pada RE untuk memberikan penanda operator *concatenate* dan kemudian sistem akan memulai untuk membaca RE. Saat pembacaan RE sistem akan membentuk NFA berdasar alfabet dan operator yang dibaca. Usai NFA terbentuk akan dilakukan konversi dari NFA ke DFA. DFA merupakan sebuah *graph* berarah yang dapat dimodelkan menjadi *adjcacency matrix* untuk melakukan menghitung T, akan dilakukan pepangkatan *matrix* sehingga hasil T merupakan jumlah bilangan pada matrix yang ada pada kolom *state* selesai pada sebuah NFA. *Pseudocode* fungsi utama program dapat dilihat pada **Gambar 3.1.1** dan **Gambar 3.1.2**.

|  |  |
| --- | --- |
| main() | |
| 1.  2.  3.  4.  5.  6.  7. | input N  for i=1 to N  clear stack states  clear stack operator  input RE, L  RE := Preprocess(RE)  NFA := ConvertREtoNFA(RE) |

Gambar 3.1.1 *Pseudocode* fungsi main(1)

|  |  |
| --- | --- |
| 8.  9.  10.  11.  12. | DFA := ConvertNFAtoDFA(NFA)  adjM := Create adjacency matrix from DFA  MatrixPower(adjM, L);  sum cell of the matrix which have finish state of NFA  Output the sum |

Gambar 3.1.2 *Pseudocode* fungsi main(2)

## Desain Algoritma

Sistem terdiri dari beberapa fungsi utama yaitu Preprocess, ConvertREtoNFA, ConvertNFAtoDFA, dan MatrixPower. Pada sub-bab ini akan dijelaskan tentang masing-masing fungsi utama dan *pseudocode* dari fungsi-fungsi tersebut.

### Desain fungsi Preprocess

Fungsi Preprocess digunakan untuk memberikan operator *concatenate* pada *string* *regular expression*. Fungsi Preprocess dibutuhkan karena pada format masukan, operator *concatenate* tidak memiliki simbol sehingga dapat menimbulkan ambiguitas. *Pseudocode* fungsi Preprocess dapat dilihat pada **Gambar 3.2.1**.

|  |  |
| --- | --- |
| /\*  \* RE unpreprocessed regular expression  \*  \* return value  \* retRE preprocessed regular expression  \*/  retRE := Preprocess(RE) | |
| 1.  2.  3.  4, | for i=0 to length(RE)  if (RE[i] is alphabet or ‘(’) and (RE[i-1] is alphabet or ‘)’)  insert operator “.” to RE[i+1]  return RE |

Gambar 3.2.1 *Pseudocode* fungsi preprocess

Fungsi Preprocess dilakukan dengan menambah operator “.” pada indeks dimana terdapat kemungkinan kasus seperti berikut

* “aa”, “ab”, “ba”, “bb”.
* “)a”, “)b”.
* “a(“, “b(“.
* dan “)(“.

Kasus-kasus tersebut diimplementasikan dalam fungsi preprocess pada baris ke 2.

### Desain fungsi ConvertREtoNFA

Fungsi ConvertREtoNFA digunakan untuk melakukan konversi *regular expression* ke NFA. Fungsi ini mengimplementasikan model *Automaton* pada metode Thompson. *Pseudocode* fungsi ini dapat dilihat pada **Gambar 3.2.2**. Fungsi ini berjalan dengan membaca *regular expression* per karakter (baris 1) dan akan mengevaluasi sebuah *regular expression* untuk setiap pasang karakter “(” dan “)” (baris 6). Untuk melakukan evaluasi tersebut dibutuhkan dua buah *stack*, yakni *stack* yang menyimpan *automaton* NFA sementara, dan *stack* operator yang menyimpan operator yang harus dievaluasi saat karakter “)” ditemui. Untuk setiap alfabet yang ditemui dalam pembacaan *regular expression* akan dibuat *automaton* seperti pada **Gambar 2.4.2** (baris 4). Saat karakter “)” ditemui dalam pembacaan *regular expression*, akan dilakukan evaluasi terhadap operator yang digunakan dalam operasi selanjutnya dengan mengambil operator teratas pada *stack* operator (baris 7 dan 8).

*Automaton* baru akan dibuat setiap operator dievaluasi, apabila operator *concatenate* ditemui, maka akan diambil dua *automaton* sementara dari *stack* *states* yang kemudian digunakan untuk melakukan operasi penggabungan sehingga dihasilkan *automaton* baru untuk disimpan ke *stack states* (baris 9 s.d. 11). Proses yang sama juga berlaku untuk operator *union* dengan melakukan operasi *union* untuk menghasilkan *automaton* baru (baris 12 s.d. 14). Pada saat melakukan operasi *klenee star* hanya ada satu *automaton* yang diambil dari *stack states* yang kemudian di mofifikasi sehingga membentuk *automaton* *klenee star* (baris 15 s.d. 17).

|  |  |
| --- | --- |
| /\*  \* RE preprocessed regular expression  \*  \* return value  \* NFA finite state machine model that has  \* epsilon transition  \*/  NFA := ConvertREtoNFA(RE) | |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18.  19.  20 | for i=1 to length(RE)  if RE[i] is ‘(‘ continue loop  else if RE[i] is alphabet  create new NFA automaton with alphabet as transition symbol  push newly created NFA automaton to states  else if RE[i] is ‘)’  if stack operator is not empty  pop 1 from stack operator to op  if op is concatenate operator  pop 2 automaton from stack states to st1 and st2 then create new automaton by st1 concat st2  push newly created automaton to states  else if op is union operator  pop 2 automaton from stack states to st1 and st2 then create new automaton by st1 union st2  push newly created automaton to states  else if op is klenee star operator  pop 1 automaton from stack to st1 and create new automaton by modifying st1 to macth klenee star Thompson model  push newly created automaton to states  else if RE[i] is operator  push operator to operator stack  return NFA = pop last automaton from states stack after for loop finished |

Gambar 3.2.2 *Pseudocode* fungsi ConvertREtoNFA

Apabila karakter yang ditemui dalam *regular expression* adalah operator yakni karakter “|”, “.”, dan ”\*”, maka karakter tersebut akan disimpan ke *stack* operator (baris 18 dan 19). Setelah seluruh karakter *regular expression* selesai dibaca dan dievaluasi akan tersisa sebuah *automaton* yang ada pada *stack states*. *Automaton* ini adalah *automaton* NFA hasil konversi dari RE (baris 20).

### Desain fungsi ConvertNFAtoDFA

|  |  |
| --- | --- |
| /\*  \* NFA finite state machine model that has  \* epsilon transition  \*  \* return value  \* DFA finite state machine model that has no  \* epsilon transition  \*/  DFA = ConvertNFAtoDFA(NFA) | |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11.  12.  13.  14.  15.  16.  17.  18. | DFA := create empty state machine  NFAinitialState = NFA.getInitialstate()  St := EpsilonClosure(NFAinitialState)  DFASt := CreateDFAState(St, &DFA)  push DFASt to stack tmp  while tmp is not empty  pop tmp and assign to dv  for i=0 to sizeof(alphabet)  St := MoveClosure(dv,alphabet[i])  St := EpsilonClosure(St)  if St is empty set  continue loop  else  DFASt := CreateDFAState(St, &DFA)  Assign alphabet[i] as transition symbol from dv to DFASt  If DFAst not already created before  push DFAst to stack tmp  return DFA |

Gambar 3.2.3 *Pseudocode* fungsi ConvertNFAtoDFA

Fungsi ConvertNFAtoDFA merupakan fungsi yang digunakan untuk melakukan konversi NFA menjadi DFA. Fungsi ini memiliki beberapa fungsi pendukung diantaranya adalah fungsi EpsilonClosure, MoveClosure, dan CreateDFAState. *Pseudocode* fungsi ConvertNFAtoDFA dapat dilihat pada **Gambar 3.2.3**. Fungsi ini dimulai dengan membuat *state* awal DFA dimana *state* tersebut merupakan *state* awal NFA dan kumpulan *state* NFA yang dapat dicapai dari *state* awal NFA melalui transisi epsilon (baris 3 dan 4). Setelah *state* awal DFA terbentuk tahapan yang dilakukan selanjutnya adalah melakukan metode *Subset construction* yang telah dijelaskan pada sub-bab **2.6** halaman 13, pada *pseudocode* **Gambar 3.2.3** hal tersebut dilakukan pada baris 6 s.d 17.

Fungsi EpsilonClosure merupakan salah satu fungsi pendukung dalam proses pembuatan DFA. *Pseudocode* fungsi ini dapat dilihat pada **Gambar 3.2.4**. Fungsi ini memiliki sebuah parameter, yakni kumpulan *state* NFA yang saat ini sedang aktif. Fungsi tersebut diawali dengan membuat daftar pengecekan *state* NFA yang saat ini sedang aktif (baris 1 s.d. 3). Kemudian untuk setiap *state* NFA yang sedang aktif tersebut akan dilakukan pengecekan terhadap *state* aktif selanjutnya yang dapat dikunjungi dengan transisi epsilon (baris 4 s.d 6). Apabila dari *state* yang baru dikunjungi setelah dilakukan transisi epsilon masih memiliki *state* lain yang dapat dikunjungi dengan transisi epsilon maka *state* baru tersebut akan dimasukkan ke daftar pengecekan dan daftar *state* yang dapat dikunjungi dengan transisi epsilon untuk kemudian dilakukan pengecekan (baris 8 s.d 10). Pengecekan tersebut dilakukan selama daftar pengecekan masih belum seluruhnya dilakukan pengecekan (baris 4). Apabila seluruh daftar telah selesai dilakukan pengecekan maka daftar *state* yang dapat dikunjungi dengan transisi epsilon tersebut dikembalikan ke pemanggil fungsi EpsilonClosure.

|  |  |
| --- | --- |
| /\*  \* St set of states to transite from  \*  \*return value  \* retSt set of states that can be reached  \* from St with epsilon transition  \*/  retSt = EpsilonClosure(St) | |
| 1.  2.  3.  4.  5.  6.  7.  8.  9.  10.  11. | for i=0 to sizeof(St)  add st to retSt  push st to stack tmp  while stack tmp is not empty  pop tmp and assign to t  r:=next state that can be reached from t with epsilon transition  for i=0 to r  if t.next[i] not exist in retSt  add t.next[i] to retSt  push t.next[i] to stack tmp  return retSt |

Gambar 3.2.4 *Pseudocode* fungsi EpsilonClosure

|  |  |
| --- | --- |
| /\*  \* St set of states to transite from  \* alphabet symbol input to state so the  \* trasition can happen from a state  \* to another set of states  \*return value  \* retSt set of states that can be reached  \* from St with input alphabet  \*/  retSt = MoveClosure(St, alphabet) | |
| 1.  2.  3.  4. | for i=0 to sizeof(St)  if St.next(alphabet) is not empty  add St.next(alphabet) to retSt  return retSt |

Gambar 3.2.5 *Pseudocode* fungsi MoveClosure

Fungsi pendukung lain dalam konversi NFA ke DFA adalah fungsi MoveClosure. *Pseudocode* fungsi ini dapat dilihat pada **Gambar 3.2.5**. Fungsi ini menerima kumpulan *state* aktif sebagai parameter dan sebuah alfabet. Untuk setiap *state* aktif akan dilakukan pengecekan apakah ada *state* lain yang dapat dikunjungi dari state yang sedang aktif dengan masukan alfabet dari parameter fungsi (baris 1 dan 2). Apabila ada *state* yang dapat dikunjungi dengan masukan alphabet tersebut, maka *state* tersebut dimasukan ke daftar state yang dapat dikunjungi (baris 3). Setelah seluruh *state* aktif selesai dilakukan pengecekan maka daftar *state* yang dapat dikunjungi dikembalikan ke pemanggil fungsi MoveClosure (baris 4).

Fungsi pendukung terakhir dalam konversi NFA ke DFA adalah CreateDFAState. *Pseudocode* fungsi ini dapat dilihat pada **Gambar 3.2.6**. Fungsi ini digunakan untuk membuat dan menambahkan *state* yang baru terbentuk dalam proses pembuatan DFA ke dalam *state machine* DFA. Dalam fungsi ini juga dilakukan pengecekan apakah state DFA yang akan ditambahkan sudah pernah dibuat sebelumnya atau belum (baris 1). Apabila sudah pernah dibuat, maka fungsi ini hanya akan mengembalikan *state* yang sudah ada (baris 2). Apabila belum, maka fungsi ini akan membuat state DFA kemudian menambahkannya sebelum dikembalikan ke pemanggil fungsi CreateDFAState (baris 4).

|  |  |
| --- | --- |
| /\*  \* St set of states to join as one DFA  \* state  \* DFA DFA state machine  \*  \*return value  \* DFASt DFA state  \*/  DFASt = CreateDFAState(St, &DFA) | |
| 1.  2.  3.  4. | if St already created and exists in DFA  return St  else add St to DFA  return St |

Gambar 3.2.6 *Pseudocode* fungsi CreateDFAState

### Desain fungsi MatrixPower

|  |  |
| --- | --- |
| /\*  \* AdjM Matrix that represent DFA as  \* adjacency matrix  \*  \* L an integer to power the matrix to  \* Lth power  \*  \*return value  \* retAdjM adjacency matrix that already  \* powered to Lth power  \*/  retAdjM = MatrixPOwer(AdjM, L) | |
| 1.  3.  4.  5.  6.  7.  8.  9. | Create identity matrix I that has same dimension as adjM  while L>0  if L is odd  I := AdjM x I  AdjM := AdjM x AdjM  L := L/2  return I |

Gambar 3.2.7 *Pseudocode* fungsi MatrixPower

Fungsi MatrixPower memiliki dua parameter yakni matriks AdjM dan bilangan L. Fungsi ini digunakan untuk memangkatkan matriks AdjM ke pangkat L. *pseudocode* fungsi ini dapat dilihat pada **Gambar 3.2.7**. Fungsi ini menggunakan teknik *Exponentiation by Squaring* seperti yang dijelaskan pada sub-bab **2.7** halaman 14.

## Desain Struktur Data

Sistem yang dibuat membutuhkan beberapa struktur data

# IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Sebelum penjelasan implementasi akan ditunjukkan terlebih dahulu lingkungan untuk melakukan implementasi.



## Lingkungan Implementasi

Lingkungan implementasi yang akan digunakan adalah,

* 1. Perangkat Keras

Processor Intel® Core™ i3-2310M CPU @ 2.10 GHz

RAM 4 GB

Sistem Operasi 64-bit

* 1. Perangkat Lunak

Sistem Operasi Windows 8.0

Integrated Development Environment Code::Blocks 13.12

## Parameter yang Digunakan

# UJI COBA DAN EVALUASI

Pada bab ini akan dijelaskan uji coba yang dilakukan pada aplikasi yang telah dikerjakan serta analisa dari uji coba yang telah dilakukan. Pembahasan pengujian meliputi lingkungan uji coba, skenario uji coba yang meliputi uji kebenaran dan uji kinerja serta analisa setiap pengujian.



## Lingkungan Uji Coba

## Data *Training* dan Data *Testing*

***[Halaman ini sengaja dikosongkan]***

# KESIMPULAN DAN SARAN



## Kesimpulan

## Saran

# DAFTAR ACUAN

|  |  |
| --- | --- |
| [1] | G. Navaro and M. Rafinot, Flexible Pattern Matching in Strings, Cambridge: Cambridge University Press, 2002. |
| [2] | kuruma, "A tutorial on Fast Modulo Multiplication (Exponential Squaring)," 12 Agustus 2013. [Online]. Available: http://discuss.codechef.com/questions/20451/a-tutorial-on-fast-modulo-multiplication-exponential-squaring. [Accessed 25 April 2015]. |
| [3] | V. Jalan, "Count Strings," 9 September 2012. [Online]. Available: http://www.spoj.com/problems/CTSTRING/. [Accessed 10 Desember 2014]. |
| [4] | J. Power, "Constructing a DFA from an NFA (Subset Construction)," National University of Ireland, 29 November 2002. [Online]. Available: http://www.cs.nuim.ie/~jpower/Courses/Previous/parsing/node9.html. [Accessed 25 April 2015]. |

***[Halaman ini sengaja dikosongkan]***

# LAMPIRAN