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

Muhammad Yunus Bahari, Arya Yudhi Wijaya, dan Rully Soelaiman
Teknik Informatika, Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember
Jl. Arief Rahman Hakim, Surabaya 60111 Indonesia
e-mail: yunus11@mhs.if.its.ac.id

***Abstrak*—Regular expression merupakan salah satu bentuk pola yang banyak digunakan untuk melakukan pencarian dan validasi string. Namun pada implementasinya sering kali regular expression hanya ditransformasi menjadi sebuah model Nondeterministic Finite Automaton (NFA) yang masih memiliki kekurangan apabila digunakan untuk melakukan validasi dan pencarian string. Kekurangan tersebut adalah lamanya waktu yang diperlukan untuk melakukan proses pencarian dan validasi. Tugas Akhir ini mengimplementasikan interpretasi sebuah regular expression menjadi sebuah model Deterministic Finite Automata (DFA) dengan menggunakan model Thompson NFA dan kemudian membangun DFA dengan menggunakan metode *Subset Construction*. Hasil uji coba kebenaran yang dilakukan membuktikan bahwa metode yang diimplementasi dapat memberikan hasil yang valid. Hasil uji coba kinerja memberikan bukti bahwa implementasi yang dilakukan telah sesuai dan memiliki kompleksitas yang benar yakni O(n) untuk konstruksi NFA dan $O(n^2)$ untuk konstruksi DFA.**

***Kata Kunci*— Deterministic Finite Automaton, Nondeterministic Finite Atomaton, Regular expression, SPOJ.**

I. PENDAHULUAN

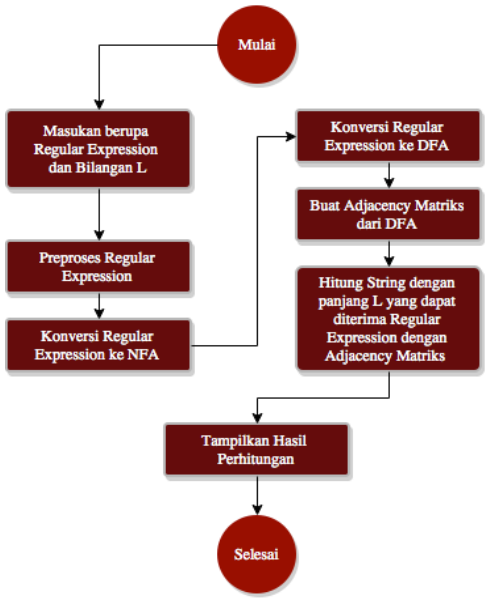
REGULAR EXPRESSION merupakan pola pencarian yang umum digunakan dalam pencarian ataupun validasi sebuah *string* ataupun kumpulan *string*. Sebagai contoh untuk pengecekan alamat email yang sah dan pencarian kata dalam dokumen. Untuk menyelesaikan permasalahan tersebut diperlukan cara untuk menginterpretasikan *Regular Expression* ke model yang dapat diterjemahkan ke dalam logika pemrograman. *Deterministic Finite Automata* (DFA) merupakan salah satu model *state machine* yang dikonversi dari *Nondeterministic Finite Automata* (NFA) yang masih memiliki beberapa *state* alternatif untuk suatu masukan yang unik. Konversi NFA ke DFA akan menghasilkan *state machine* yang memiliki *state* unik dengan masukan yang unik sehingga dapat diterjemahkan lebih mudah ke dalam logika pemrograman.

Topik Tugas Akhir ini diangkat dari permasalahan yang terdapat pada *Online Judge* SPOJ dengan nomor soal 10354 dengan kode CTSTRING [4]. Pada permasalahan ini diberikan dua buah masukan yaitu sebuah *string* yang merupakan *Regular Expression* dengan panjang maksimal 100 karakter dan hanya memiliki semesta karakter $\sum$ = {a,b} dan semesta operator = {“|”, “.”, “*”} dan sebuah bilangan L dengan batasan $1 < L < 10^9$. Dari dua masukan tersebut kita diminta untuk menentukan berapa banyak *string* dengan panjang L yang memenuhi *Regular Expression* yang diberikan.

II. DASAR TEORI DAN METODE

A. Alur Penyelesaian Permasalahan

Untuk menyelesaikan permasalahan yang diberikan, tahap yang harus ditempuh adalah konversi *regular expression* menjadi NFA kemudian melakukan konversi NFA ke DFA. Usai DFA terbentuk, maka didapatkan sebuah *directed graph* yang dapat digunakan untuk membuat sebuah *adjacency matrix* sebagai sarana perhitungan jumlah string yang dapat diterima oleh *regular expression* pada masukan program. Proses ini dapat dilihat pada **Gambar 1**.



Gambar 1 Proses penyelesaian permasalahan

B. Nondeterministic Finite Automata

Nondeterministic Finite Automata adalah sebuah *state machine* yang memiliki sekumpulan *state* berhingga *Q*, sekumpulan simbol Σ 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

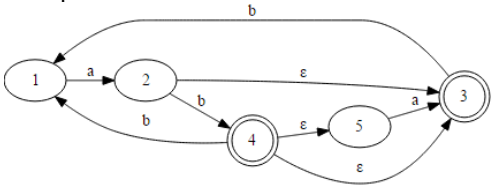
dimulai. NFA dapat direpresentasikan dengan sebuah tabel atau sebuah *graph* [1]. **Tabel 1** merepresentasikan sebuah NFA dengan 5 *state* dengan kumpulan simbol masukan $\Sigma = \{a, b\}$, *state* awal NFA dimulai dari *state* q_0 dan memiliki dua *state* selesai yakni $A = \{3,4\}$. NFA tersebut juga dapat direpresentasikan dalam bentuk *graph* seperti pada **Gambar 2**. NFA memiliki karakteristik khusus yakni sebuah *state* dapat berpindah ke *state* lain tanpa masukan simbol apapun yang dinotasikan dengan ε sebagai simbol masukan dalam **Tabel 1**.

Tabel 1 Representasi NFA dalam tabel

	a	b	ε
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 $Q' = \{4,1\}$.

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.



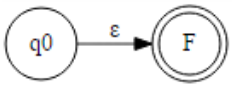
Gambar 2 Representasi NFA dalam directed graph

C. *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*.

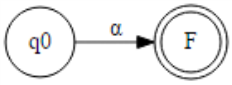
Metode ini menggunakan *automaton* yang spesifik untuk setiap operator yang digunakan dalam *regular expression*. Beberapa *automaton* tersebut adalah [1]

- *Automaton* untuk transisi epsilon ditunjukkan pada **Gambar 3**. *Automaton* ini berarti perpindahan state dari $q0$ ke F bisa terjadi tanpa masukan simbol apapun.



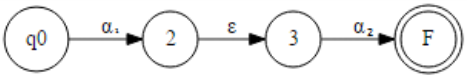
Gambar 3 *Automaton* transisi epsilon

- *Automaton* untuk sebuah masukan karakter ditunjukkan pada **Gambar 4**. *Automaton* ini berarti perpindahan state dari $q0$ ke F terjadi apabila karakter α dimasukkan sebagai simbol masukan ke NFA.



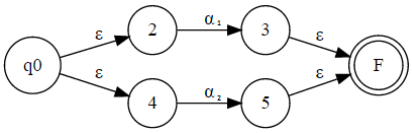
Gambar 4 *Automaton* dengan sebuah masukan karakter

- Automaton untuk operator "·" (*Concatenate*) ditunjukkan pada **Gambar 5**. *Automaton* ini berarti perpindahan *state* dari $q0$ ke F terjadi apabila NFA diberikan dua kali masukan simbol secara berurutan. untuk berpindah dari *state* q0 ke *state* 2 dengan masukan α_1, dan dari *state* 3 ke *state* F dengan masukan α_2



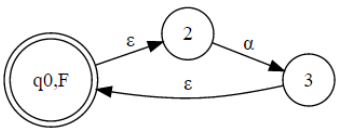
Gambar 5 *Automaton* operator *concatenate*

- *Automaton* untuk operator "|" (*Union*) ditunjukkan pada **Gambar 6**. *Automaton* ini berarti perpindahan *state* dari $q0$ ke F dapat terjadi apabila masukan yang diberikan merupakan salah satu dari simbol α_1 atau α_2.



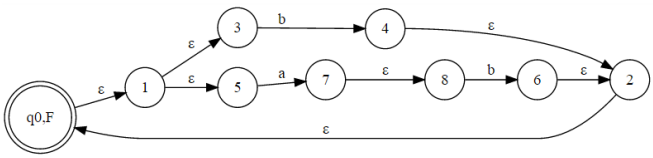
Gambar 6 *Automaton* operator Union

- *Automaton* untuk operator "*" (*Klenee star*) ditunjukkan pada **Gambar 7**. *Automaton* berarti perpindahan *state* dari $q0$ ke F dapat terjadi dengan tanpa masukan karena $q0$ adalah *state* yang sama dengan F atau dengan masukan simbol α.



Gambar 7 *Automaton* operator Klenee Star

Contoh NFA yang terbentuk dengan metode Thompson's untuk *regular expression* $\left(((a.b)|b)*\right)$ dapat dilihat pada **Gambar 8**.



Gambar 8 Contoh NFA

D. *Deterministic Finite Automata*

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[1].

E. Metode Subset Construction

Untuk melakukan konvesi NFA ke DFA didefinisikan dua buah fungsi, yakni

- 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].

F. Matrix Exponentiation

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 $D^2 x\, N$. 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 $D^2 x \log_2 N$. Teknik ini secara rekursif didefinisikan pada persamaan 1 [3].

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

III. UJI COBA DAN EVALUASI

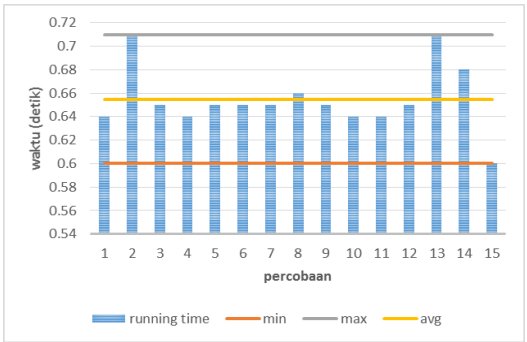
A. Uji Coba Kebenaran

Uji coba kebenaran merupakan uji coba yang dilakukan untuk menguji kebenaran implementasi metode yang digunakan pada program. Uji coba dilakukan dengan mengirim kode sumber pada situs SPOJ dengan kode permasalahan CTSTRING yang diangkat sebagai studi kasus Tugas Akhir ini. Setelah kode sumber dikirim, SPOJ akan memberikan umpan balik terhadap hasil berjalannya program. Hasil yang didapatkan dari SPOJ adalah *accepted*, waktu yang diperlukan untuk penyelesaian permasalahan adalah 0.60 detik dan memori yang dibutuhkan adalah 3MB seperti yang ditunjukkan pada **Gambar 9**. Hal tersebut menunjukkan bahwa implementasi yang dilakukan berhasil menginterpretasikan *regular expression* dan mengubahnya ke bentuk DFA untuk melakukan pengecekan *string* yang dapat diterima.

Gambar 9 Umpan balik dari situs SPOJ

Kemudian dilakukan uji coba dengan mengirim kode sumber sebanyak 15 kali pada permasalahan yang sama untuk melihat variasi hasil yang dihasilkan oleh program. Hasil uji coba tersebut dapat dilihat pada **Gambar 10**

Dari hasil uji coba yang dilakukan waktu yang dibutuhkan bervariasi antara 0.6 s.d. 0.71 dengan rata-rata 0.65 detik namun memori yang dibutuhkan program tetap 3 MB.



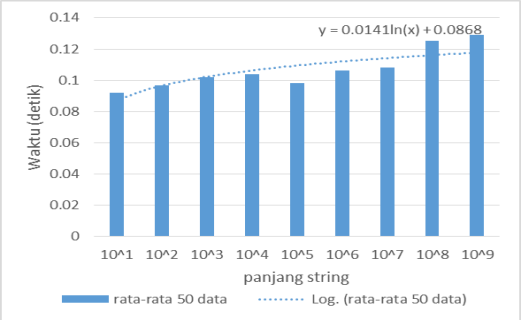
Gambar 10 Grafik hasil uji coba pada SPOJ sebanyak 15 kali

B. Uji Coba Kinerja

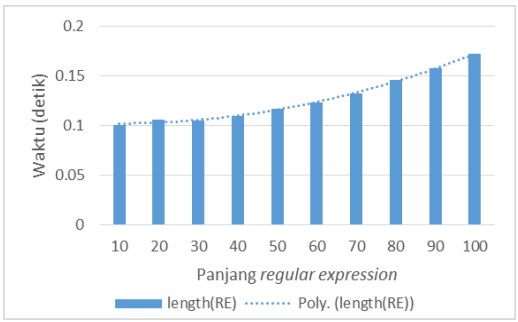
Hasil uji coba pengaruh panjang *string* L dapat dilihat di **Gambar 11**. Uji coba ini dimaksudkan untuk melakukan tes performa ketika *regular expression* dicocokan dengan sebuah *string*. Performa yang diharapkan dalam uji coba ini adalah logaritmik. Untuk *string* yang semakin panjang maka waktu yang dibutuhkan akan semakin lama.

Pada **Gambar 11** dapat dilihat bahwa dengan perbedaan L yang diberikan, semakin besar L maka semakin besar pula waktu yang diperlukan untuk mengeksekusi program, dan perkembangan waktu yang dihasilkan memenuhi waktu logaritmik seperti yang ditujukkan pada gambar. hal

dikarenakan algoritma perkalian matriks yang digunakan menggunakan algoritma dengan kompleksitas $O(\log(L)\, xN^2)$ dimana *L* adalah besar pangkat dan *N* adalah dimensi matriks.



Gambar 11 Grafik rata-rata hasil uji coba untuk L yang berbeda



Gambar 12 Grafik uji coba dengan panjang RE yang berbeda

Dari hasil uji coba yang dilakukan dapat dilihat bahwa besar atau kecilnya L memiliki pengaruh terhadap waktu yang diperlukan program untuk melakukan evaluasi permasalahan dengan kompleksitas logaritmik.

Uji coba yang dilakukan selanjutnya adalah dengan menggunakan *regular expression* acak dengan operator yang digunakan hanya operator *concatenate* yang dibangkitkan dengan sebuah program pembangkit uji kinerja. Operator *concatenate* digunakan karena memberikan hasil DFA yang lebih konsisten untuk setiap input dengan panjang *regular expression* yang berbeda, di mana semakin panjang *regular expression* maka *state* DFA yang dihasilkan juga akan semakin banyak pula. Parameter yang digunakan pada uji coba adalah panjang *regular expression*. Panjang yang diujikan adalah 10, 20, 30, 40, 50, 60, 70, 80, 90, dan 100 karakter. Bilangan L yang digunakan pada uji coba ini adalah 10^9.

Data hasil uji coba tersebut dapat dilihat pada **Gambar 12**. Uji coba ini dimaksudkan untuk melakukan tes performa konversi *regular expression* menjadi NFA dan kemudian DFA. Performa algoritma yang diharapkan adalah polinomial berderajat 2, Pada **Gambar 12** terlihat jika *regular expression* yang perlu dievaluasi semakin panjang makan semakin besar pula waktu yang diperlukan untuk melakukan konversi terhadap *regular expression* tersebut dan waktu berjalan program memenuhi kurva polinomial berderajat 2, hal ini dikarenakan saat pembentukan NFA operasi yang diperlukan adalah sebanyak $O(2xN)$ atau bisa dibilang linear untuk setiap operator *concatenate* yang dilakukan, sedangkan untuk konstruksi NFA ke DFA diperlukan operasi sebanyak $O(M^2)$ dimana M adalah jumlah *state* NFA.

IV. KESIMPULAN

Dari hasil uji coba program yang telah dilakukan dengan implementasi model NFA dan DFA untuk menginterpretasikan *regular expression* dapat ditarik beberapa kesimpulan sebagai berikut,

1. Implementasi yang dilakukan berhasil dengan benar menyelesaikan permasalahan studi kasus yang digunakan dalam penyusunan tugas akhir, yakni studi kasus SPOJ klasik 10354 dengan rata-rata waktu berjalan program sebesar 0.65 detik dan memori 3MB.
2. Untuk evaluasi *regular expression*, ditandai dengan besarnya L yang berbeda memiliki pengaruh terhadap waktu yang dibutuhkan program secara logaritmik yakni sebesar $O(M^2 \log(L))$, dimana M adalah dimensi *Adjacency Matrix* dari DFA.
3. Untuk konversi *regular expression* menjadi *state machine* NFA dan DFA, panjang *regular expression* memiliki pengaruh terhadap waktu yang dibutuhkan secara polinomial yakni $O(2N) + (2^N)$, $O(2N)$ untuk konversi *regular expression* menjadi NFA, dan $O(2^N)$ untuk konversi NFA menjadi DFA. Kompleksitas tersebut memiliki makna jika semakin panjang *regular expression* maka semakin besar pula waktu yang dibutuhkan.

UCAPAN TERIMA KASIH

Penulis M.Y.B. ingin mengucapkan puji syukur ke hadirat Allah S.W.T atas karunianya serta menyampaikan terima kasih yang sebesar-besarnya atas dukungan dan semangat yang diberikan dan membantu penulis baik secara langsung ataupun tidak dalam menyelesaikan Tugas Akhir ini kepada Bapak Arya Yudhi W. dan Bapak Rully Soelaiman yang telah banyak membantu penulis dalam menyelesaikan penelitian ini. Penulis juga ingin menyampaikan terimakasih kepada pihak-pihak lain yang tidak dapat disebutkan satu persatu atas bantuannya menyelesaikan penelitian ini.

DAFTAR PUSTAKA

[1] G. Navaro and M. Rafinot, Flexible Pattern Matching in Strings, Cambridge: Cambridge University Press, 2002.

[2] 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].

[3] 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].

[4] V. Jalan, "Count Strings," 9 September 2012. [Online]. Available: http://www.spoj.com/problems/CTSTRING/. [Accessed 10 Desember 2014].