Penerapan Finite Automata Nondeterministik pada Pencocokan String dengan Wildcard

Irfan Sidiq Permana - 13522007[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522007@std.stei.itb.ac.id

***Abstrak*— Pencocokan string merupakan salah satu permasalahan penting dan fundamental pada bidang ilmu komputer, sering dijumpai pada berbagai aplikasi di dunia nyata seperti pencarian teks, penambangan data, pengenalan pola, dan masih banyak lagi. Permasalahan pencocokan string ini sudah diteliti selama puluhan tahun dan banyak algoritma efisien yang telah ditemukan oleh para ahli, contohnya yang paling umum digunakan yaitu algoritma KMP (Knuth-Morris-Pratt) dan BM (Boyer-Moore). Namun algoritma pencocokan string pada umumnya, termasuk KMP dan BM, tidak mendukung pencocokan string dengan wildcard. Hal ini dikarenakan terdapat ketidakpastian pola pada simbol wildcard, sehingga pencocokan string dengan wildcard memerlukan algoritma yang dapat menganalisis beberapa kemungkinan rute sekaligus tanpa terjebak pada satu rute solusi. Pencocokan string dengan wildcard sendiri juga banyak digunakan pada dunia nyata, seperti pencarian file, kueri basis data, dan pencarian teks. Pada makalah ini, akan dibahas mengenai penerapan NFA (Non-Deterministic Finite Automata) untuk menyelesaikan permasalahan pencocokan string dengan wildcard.**

***Keywords*— Pencocokan string, *Wildcard*, *Non-Deterministic Finite Automata* (NFA)**

I. PENDAHULUAN

Pencocokan string adalah permasalahan dalam ilmu komputer yang bertujuan menemukan semua kemunculan atau posisi dari suatu pola dalam teks. Permasalahan ini sangat umum dijumpai di dunia nyata, misalnya pencarian teks pada *text editor*, pengenalan pola dalam pembelajaran mesin (*machine learning*), sistem mesin pencarian, pengolahan bahasa alami, dan masih banyak lagi. Pencocokan string tidak hanya terbatas pada tulisan saja, melainkan dapat pula digunakan untuk suara maupun gambar. Pencocokan string memiliki peran yang sangat penting dalam pengembangan teknologi, sehingga menjadi salah satu fokus utama bagi para peneliti sejak puluhan tahun yang lalu. Berbagai macam algoritma pencocokan string telah ditemukan oleh para peneliti, diantaranya yang paling terkenal dan umum digunakan adalah algoritma KMP (Knuth-Morris-Pratt) dan BM (Boyer-Moore). Kedua algoritma ini menyuguhkan penyelesaian dengan kompleksitas waktu yang jauh lebih baik dari algoritma *brute force*, sehingga kini dianggap sebagai algoritma yang cukup mangkus serta mudah digunakan untuk menyelesaikan permasalahan pencocokan string.

Selain pencocokan string dengan pola teks yang pasti, terdapat pula jenis pencocokan string dengan pola teks yang tidak pasti yaitu pencocokan string dengan *wildcard*. Pencocokan string dengan wildcard adalah proses mencari atau mencocokkan string menggunakan pola yang memiliki suatu karakter khusus, yang disebut dengan *wildcard*. Karakter wildcard ini dapat digantikan oleh sebuah karakter ataupun string sembarang, tergantung pada simbol karakter tersebut. Pencocokan string dengan *wildcard* memiliki banyak aplikasi di dunia nyata, seperti pencarian file pada CLI, kueri pada basis data seperti SQL, pemrosesan teks lanjutan, dan sebagainya. Sayangnya, algoritma pencocokan string pada umumnya (termasuk KMP dan BM) tidak mendukung pencocokan string dengan wildcard. Hal ini dikarenakan pencocokan string dengan wildcard lebih kompleks dari pencocokan string dengan pola pasti, dimana pencocokan wildcard memerlukan algoritma yang dapat menganalisis beberapa kemungkinan rute sekaligus tanpa terjebak pada satu rute solusi. Salah satu contoh algoritma yang dapat digunakan untuk pencocokan string dengan wildcard yaitu *brute force*, program dinamis (DP), dan Fast Fourier Transform (FFT).

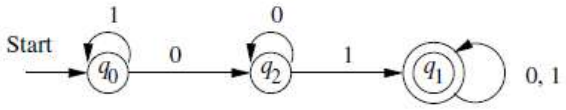
Pada makalah ini, akan dibahas mengenai penerapan NFA (Nondeterministic Finite Automata) untuk menyelesaikan pencocokan string dengan wildcard. NFA dapat digunakan untuk menyelesaikan permasalahan pencocokan string dengan wildcard karena cara kerja NFA yang menelusuri rute secara nondeterministik, yaitu menelusuri beberapa rute sekaligus sehingga tidak terjebak hanya pada satu rute layaknya algoritma string matching pada umumnya. Pembahasan makalah mencakup teori dasar singkat, implementasi algoritma disertai dengan *pseudocode*, serta hasil pengujian untuk menarik kesimpulan.

II. TEORI DASAR

A. Finite Automata

Finite Automata adalah model matematika mesin abstrak yang melakukan transisi secara deterministik, sehingga mesin hanya dapat berada dalam satu *state* pada suatu waktu. Sebuah finite automata didefinisikan dengan kumpulan state yang dimiliki, kumpulan simbol input (disebut juga alfabet), serta fungsi transisi dari suatu state menuju state yang lain. Finite automata akan merubah statenya ketika diberikan suatu input, dimana mesin akan berubah dari suatu state awal menuju state lain tergantung dari input yang sedang dibaca oleh finite

automata. Finite automata umumnya digambarkan dengan graf, dimana simpul dari graf menggambarkan state yang dimiliki oleh finite automata, sedangkan sisi berarah (panah) menggambarkan transisi yang mungkin dari suatu state menuju state lainnya.



Gambar 2.1. Contoh finite automata yang diilustrasikan dengan graf
Sumber: [1]

Finite automata memiliki satu state awal (*initial state*) dan satu atau lebih state akhir (*final state*). State awal dari automata direpresentasikan oleh simpul yang memiliki panah dengan label "start", sedangkan state akhir direpresentasikan oleh simpul dengan dua lingkaran. Contoh dari input yang dapat diterima automata ini yaitu string, dimana string merupakan kumpulan dari simbol. Automata akan membaca input misalnya string, lalu menelusuri statenya mulai dari state awal dengan melakukan transisi state sesuai dengan karakter yang sedang dibaca hingga akhirnya seluruh string telah selesai dibaca atau mesin terjebak pada keadaan state mati (*dead state*). Suatu dead state adalah suatu *non-final* state dimana untuk semua simbol yang mungkin diterima oleh automata selanjutnya, ia akan tetap pada state tersebut sehingga automata tidak akan pernah mencapai final state. Bila setelah membaca seluruh string automata berada pada final state, maka artinya automata "menerima" input string tersebut. Sebuah automata dikatakan "menerima" sebuah input string jika dan hanya jika setelah melakukan transisi mulai dari state awal hingga string terbaca seluruhnya, automata berada pada *final state* sebagai kondisi akhir automata.

Sebuah finite automata dapat direpresentasikan dengan *quintuple*:

$$A = (Q, \Sigma, \delta, q_0, F)$$

dimana Q merupakan kumpulan dari semua state yang dimiliki automata, Σ merupakan kumpulan simbol yang dapat dibaca automata, q_0 merupakan state awal dari automata, F merupakan kumpulan dari semua state akhir yang dimiliki automata, dan δ merupakan fungsi transisi dari automata. Sebuah fungsi transisi dari automata menerima parameter berupa state automata saat ini dan simbol yang sedang dibaca, lalu mengembalikan state hasil transisi setelah membaca simbol tersebut. Fungsi transisi dari finite automata deterministik (DFA) adalah deterministik, yang artinya untuk state tertentu dengan simbol tertentu yang sedang dibaca, DFA hanya memiliki satu kemungkinan state hasil transisi.

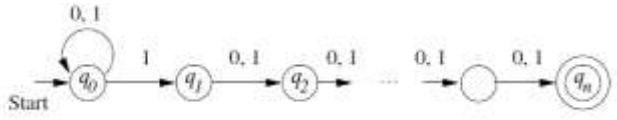
	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

Gambar 2.2. Contoh fungsi transisi sebuah DFA, setiap pasangan state dan simbol hanya memiliki satu kemungkinan state tujuan
Sumber: [1]

Sebuah "bahasa" (*language*) merupakan kumpulan dari semua string yang diterima oleh DFA. Sebuah DFA akan membaca simbol dari string input satu per satu dengan mengubah state dirinya sendiri secara deterministik sesuai dengan fungsi transisi yang dimilikinya. Bila DFA pada state tertentu tidak memiliki fungsi transisi yang sesuai dengan simbol yang sedang dibaca, maka DFA secara keseluruhan menolak string input tersebut.

B. Non-Deterministic Finite Automata (NFA)

Sebuah finite automata nondeterministik (NFA) merupakan sebuah automata layaknya DFA yang memiliki kumpulan state, simbol, dan fungsi transisi, namun memiliki kemampuan untuk berada pada beberapa state sekaligus dalam satu waktu. Kemampuan untuk berada pada beberapa state sekaligus ini memungkinkan DFA untuk "menebak" dalam melakukan pemrosesan input, menjadikannya mesin automata yang lebih *powerful* dari DFA. Gambar model dari NFA tidak jauh berbeda dengan DFA, bahkan bila dilihat secara sekilas mungkin terlihat sangat mirip. Perbedaan dari gambar DFA dan NFA terletak pada fungsi transisinya, dimana untuk suatu state dengan simbol tertentu DFA hanya memiliki satu panah sedangkan NFA dapat memiliki satu atau lebih panah.



Gambar 2.3. Contoh ilustrasi graf NFA, dimana untuk state q_0 dengan simbol 1 terdapat panah menuju q_0 dan q_1
Sumber: [1]

Sebuah finite automata nondeterministik (NFA) juga memiliki kemampuan lain yang tidak dimiliki oleh DFA, yaitu merubah state berdasarkan simbol epsilon. Simbol epsilon (ε) adalah simbol yang merepresentasikan transisi tanpa input. Ini artinya, NFA dapat merubah statenya tanpa perlu membaca simbol dari suatu string input. NFA yang memanfaatkan simbol epsilon dalam fungsi transisinya disebut sebagai ε-NFA (Epsilon Nondeterministik Finite Automata). Kemampuan simbol epsilon ini membuat ε-NFA menjadi lebih *powerful* lagi dari NFA, dimana ε-NFA dapat berada pada lebih banyak state sekaligus berkat transisi dengan simbol epsilon sehingga membuat kemampuan "menebak" dari ε-NFA jauh lebih kuat dari DFA maupun NFA.

	ϵ	a	b	c
$\rightarrow p$	$\{q, r\}$	$\emptyset$	$\{q\}$	$\{r\}$
q	$\emptyset$	$\{p\}$	$\{r\}$	$\{p, q\}$
$*r$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

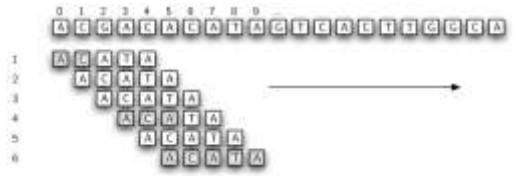
Gambar 2.4. Contoh fungsi transisi sebuah ε-NFA, setiap pasangan state dan simbol dapat memiliki beberapa kemungkinan state tujuan
Sumber: [1]

Walaupun begitu, ketiga automata ini sebenarnya adalah ekivalen. Ekivalen disini dalam artian sebuah NFA apapun pasti dapat dikonversi menjadi DFA yang menerima bahasa yang sama, begitu pula ε-NFA apapun pasti dapat dikonversi

menjadi NFA tanpa transisi epsilon. Dengan begitu, ketiga automata ini pasti dapat dikonversi menjadi satu sama lain. Kelebihan dari menggunakan NFA maupun ε-NFA daripada DFA adalah sebuah NFA maupun ε-NFA umumnya memiliki state yang jauh lebih sedikit dari DFA karena kemampuan "menebak", yang dimiliki oleh finite automata nondeterministik, sehingga konversi dari NFA ke DFA maupun ε-NFA ke DFA dapat menyebabkan jumlah state dan fungsi transisi meningkat secara eksponensial.

C. String Matching

String matching atau pencocokan string adalah permasalahan dalam ilmu komputer yang bertujuan menemukan semua kemunculan atau posisi dari suatu pola dalam teks. Permasalahan ini memiliki sangat banyak kegunaan di dunia digital, seperti misalnya pencarian teks pada *text editor*, pengenalan pola dalam pembelajaran mesin (*machine learning*), sistem mesin pencarian, pengolahan bahasa alami, dan masih banyak lagi.



Gambar 2.5. Contoh ilustrasi proses pencocokan string
Sumber: [2]

Dikarenakan sangat pentingnya peran pencocokan string dalam perkembangan teknologi, permasalahan pencocokan string menjadi pusat perhatian dari banyak peneliti puluhan tahun yang lalu. Banyak algoritma pencocokan string matching yang telah diusulkan oleh para peneliti, dengan tujuan mencari algoritma yang seefisien mungkin dalam melakukan pencocokan string. Berikut merupakan beberapa contoh algoritma pencocokan string yang telah ditemukan, disertai dengan kompleksitas waktu dan kompleksitas ruangnya:

Nama Algoritma	Kompleksitas Waktu	Kompleksitas Ruang
Brute force	O((n - m + 1) * m)	O(1)
KMP (Knuth-Morris-Pratt)	O(n + m)	O(m)
BM (Boyer-Moore)	Terbaik: O(n/m) Terburuk: O(nm) Rata-rata: O(n + m)	O(m + |Σ|)
Rabin-Karp	Terbaik: O(n + m) Terburuk: O(nm) Rata-rata: O(n + m)	O(1)
Aho-Corasick	O(n + z + k)	O(z)

Tabel 2.1. Contoh algoritma pencocokan string beserta kompleksitas waktu dan kompleksitas ruangnya

Seperti yang dapat dilihat pada tabel diatas, telah banyak usulan algoritma pencocokan string yang ditemukan oleh peneliti dengan kompleksitas waktu yang jauh lebih cepat dari *brute force*. Kebanyakan dari algoritma diatas memiliki rata-rata kompleksitas waktu O(n + m), dengan n menyatakan panjang string teks dan m menyatakan panjang string pola. Kompleksitas waktu ini dinilai cukup untuk menyelesaikan mayoritas kasus pencocokan string, dan sulit untuk menemukan algoritma yang lebih cepat dari O(n + m) untuk kasus rata-rata tanpa memerlukan optimasi yang jauh lebih mendetail dan spesifik serta teknik algoritma yang jauh lebih rumit.

D. Wildcard String Matching

Wildcard string matching atau pencocokan string dengan wildcard adalah proses mencari atau mencocokkan string menggunakan pola dengan menggunakan suatu karakter khusus, yang disebut dengan *wildcard*. Pencocokan string dengan *wildcard* memiliki banyak aplikasi di dunia nyata, seperti pencarian file pada CLI, kueri pada basis data seperti SQL, pemrosesan teks lanjutan, dan sebagainya. Karakter wildcard pada pola ini dapat digantikan oleh sebuah karakter ataupun string sembarang, tergantung pada simbol karakter tersebut. Berikut merupakan contoh karakter *wildcard* yang paling sering digunakan dalam pencocokan string menggunkana *wildcard*:

Karakter	Kegunaan
?	Dapat dipasangkan dengan karakter apapun
*	Dapat dipasangkan dengan string apapun, termasuk string kosong

Tabel 2.2. Karakter wildcard dan kegunaannya

Algoritma pencocokan string pada umumnya tidak mendukung pencocokan string dengan wildcard. Hal ini dikarenakan pencocokan string dengan wildcard memiliki faktor ketidakpastian pada pola apa yang menggantikan karakter wildcard, sehingga pencocokan wildcard memerlukan algoritma yang dapat menganalisis beberapa kemungkinan rute sekaligus tanpa terjebak pada satu rute solusi. Pencocokan string dengan wildcard juga umumnya membutuhkan waktu yang jauh lebih lama dari pencocokan string biasa, dikarenakan banyaknya kemungkinan ruang solusi yang harus diperiksa. Misalnya, algoritma brute force dengan algoritma program dinamis dapat digunakan untuk menyelesaikan pencocokan string wildcard, namun keduanya sama-sama memiliki kompleksitas waktu O(mn) dan kompleksitas ruang O(mn). Di sisi lain, pendekatan algoritma greedy memiliki kompleksitas waktu O(n) dan kompleksitas ruang O(1), namun tentunya algoritma greedy tidak menjamin kebenaran solusi karena pendekatan ini hanya memeriksa satu rute dari banyaknya kemungkinan rute yang ada. Contoh algoritma yang dirasa cukup mangkus untuk menyelesaikan persoalan pencocokan string dengan wildcard ini adalah Fast Fourier Transform (FFT), yang kini telah dikembangkan hingga dapat memiliki kompleksitas waktu mencapai O(n log m) [3]. Implementasi dari FFT sendiri dapat berbeda-beda, sehingga detail dari kompleksitas waktu FFT dapat berbeda tergantung dari implementasi yang digunakan.

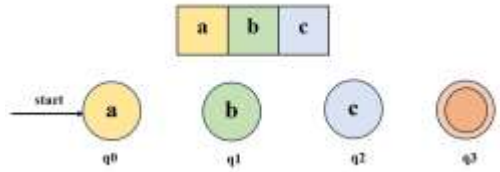
III. SOLUSI PERSOALAN

A. *Perancangan ε-NFA*

Tahap paling awal dalam menerapkan NFA untuk proses pencocokan string adalah merancang NFA itu sendiri. Seperti yang telah dibahas pada dasar teori, NFA didefinisikan oleh kumpulan state yang dimilikinya, kumpulan simbol yang dapat dibaca, dan fungsi transisi yang dimiliki.

a.) State

Tiap state pada rancangan NFA dipasangkan dengan tiap karakter pada pattern, ditambah satu state tambahan sebagai state akhir (final state). Dengan kata lain, untuk suatu pola dengan panjang *m*, NFA akan memiliki *m+1* jumlah state. Tiap state dimulai dengan indeks nol, yaitu $\{q_0, q_1, q_2, \ldots, q_m\}$ dimana q_m adalah final state sedangkan q_0 adalah start state yang sekaligus berkorespondensi dengan karakter pertama pada pola.



Gambar 3.1. Contoh state NFA yang dihasilkan pola "abc"
Sumber: Dokumen Penulis

b.) Simbol yang dibaca

Simbol yang dapat dibaca oleh NFA sebenarnya tidak dibatasi, namun untuk tujuan kepraktisan dapat diasumsikan simbol yang dibaca oleh NFA adalah seluruh karakter ASCII.

c.) Fungsi Transisi

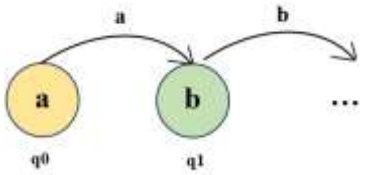
Fungsi transisi dari suatu state ke state yang lain disini dibagi menjadi 3 kasus berdasarkan karakter pola, yaitu:

- Karakter biasa

Untuk tiap state yang merepresentasikan karakter biasa pada pola, fungsi transisinya dapat didefinisikan sebagai berikut. Misalkan sebuah state q_k merepresentasikan suatu karakter *c*, maka fungsi transisi dari state tersebut adalah

$$\delta(q_k, c) = \{q_{k+1}\}$$

Contoh ilustrasi dapat dilihat sebagai berikut:



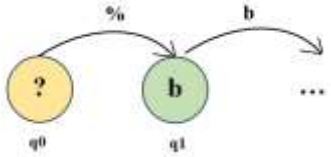
Gambar 3.2. Contoh fungsi transisi NFA yang dihasilkan pola "ab…"
Sumber: Dokumen Penulis

- Karakter *wildcard* '?'

Untuk tiap state yang merepresentasikan karakter wildcard '?' pada pola, fungsi transisinya dapat didefinisikan sebagai berikut. Misalkan sebuah state q_k merepresentasikan suatu karakter wildcard '?', maka fungsi transisi dari state tersebut adalah

$$\delta(q_k, \%) = \{q_{k+1}\}$$

dimana '%' melambangkan karakter apapun pada alfabet kecuali epsilon. Contoh ilustrasi dapat dilihat sebagai berikut:



Gambar 3.2. Contoh fungsi transisi NFA yang dihasilkan pola "?b…"
Sumber: Dokumen Penulis

- Karakter *wildcard* '*'

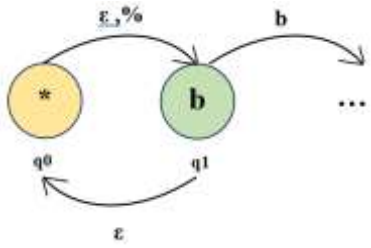
Untuk tiap state yang merepresentasikan karakter wildcard '?' pada pola, fungsi transisinya dapat didefinisikan sebagai berikut. Misalkan sebuah state q_k merepresentasikan suatu karakter wildcard '?', maka fungsi transisi dari state tersebut adalah

$$\delta(q_k, \varepsilon) = \{q_{k+1}\}$$

$$\delta(q_k, \%) = \{q_{k+1}\}$$

$$\delta(q_{k+1}, \varepsilon) = \{q_k\}$$

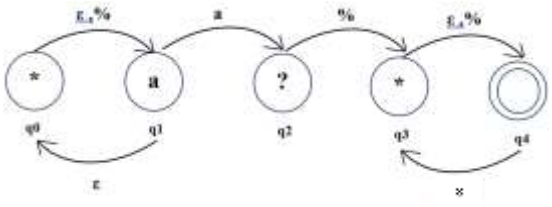
dimana 'ε' melambangkan karakter epsilon, yaitu melakukan transisi tanpa membaca simbol input. Contoh ilustrasi dapat dilihat sebagai berikut:



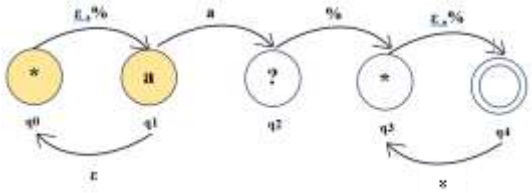
Gambar 3.2. Contoh fungsi transisi NFA yang dihasilkan pola "*b…"
Sumber: Dokumen Penulis

B. *Proses Pencocokan Pola*

Tahapan dari proses pencocokan pola dengan teks dilakukan sebagai berikut. Misalkan input teks adalah "abc", dengan pola yang ingin dicocokkan adalah "*a?*". Maka, DFA yang terbentuk adalah sebagai berikut:

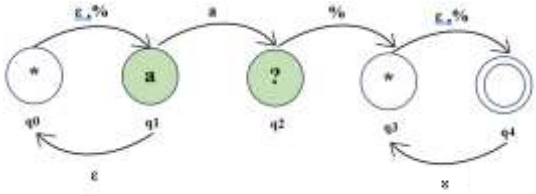


1. Sebelum memulai mencocokkan pola dengan string, lakukan transisi epsilon terlebih da hulu pada state awal untuk mencari semua posisi awal yang mungkin pada pola. Ini sama halnya dengan mencari klosur epsilon dari q_0. Semua hasil state dari klosur epsilon q_0 dijadikan sebagai simpul yang akan diekspan ketika pencocokan karakter pertama.

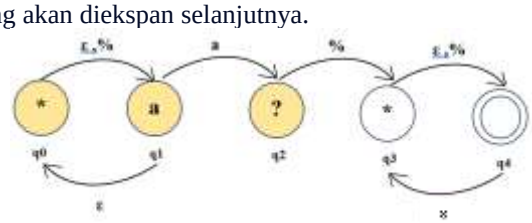


Pada gambar diatas, simpul kuning menyatakan state hasil klosur epsilon dari q_0, yaitu q_0 dan q_1. Hasil kumpulan state dari klosur epsilon kedepannya disebut sebagai simpul ekspan. Simpul ekspan adalah simpul yang akan dicocokkan dengan karakter pada teks. Kedua state ini menjadi simpul yang diekspan pada tahap berikutnya.

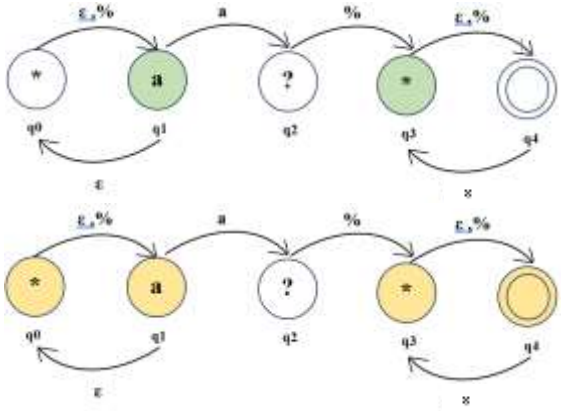
2. Untuk setiap simpul ekspan pada langkah 1, lakukan pencocokan karakter simpul dengan karakter pertama teks "abc" yaitu "a". Dari fungsi transisi $\delta(q_0, a) = \{q_1\}$ dan $\delta(q_1, a) = \{q_2\}$, diperoleh hasil simpul $\{q_1, q_2\}$. Hasil kumpulan state dari pencocokan karakter teks kedepannya disebut dengan simpul hidup. Simpul hidup adalah simpul yang akan dicari hasil klosur epsilonnya.



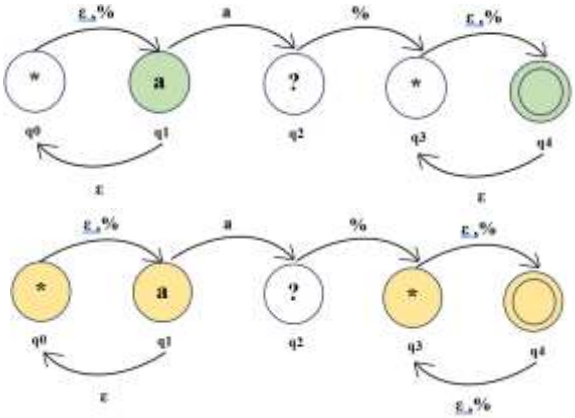
3. Untuk setiap simpul hidup pada langkah 2, lakukan transisi epsilon untuk mencari klosur epsilon dari tiap simpul. Dari fungsi transisi $\delta(q_1, \varepsilon) = \{q_0\}$, diperoleh ECLOSE($q_1$, q_2) = $\{q_0, q_1, q_2\}$. Kumpulan state hasil klosur epsilon ini kemudian dijadikan simpul ekspan yang akan diekspan selanjutnya.



4. Untuk setiap simpul ekspan pada langkah 3, pencocokan karakter simpul dengan karakter pertama teks "abcd" yaitu "b" sehingga diperoleh hasil simpul {q1, q3} yang dijadikan simpul hidup. Simpul hidup ini kemudian dicari klosur epsilonnya untuk memperoleh ECLOSE(q_1, q_3) = $\{q_0, q_1, q_3, q_4\}$ yang dijadikan simpul ekspan.



5. Begitu seterusnya hingga semua karakter pada teks selesai dibandingkan dan diperoleh simpul ekspan terakhir.



C. Verifikasi Hasil Pencocokan Pola

Setelah membandingkan seluruh karakter pada teks dan diperoleh kumpulan simpul ekspan terakhir, dapat ditentukan apakah NFA menerima teks input atau tidak.

- Bila dalam kumpulan simpul ekspan terakhir terdapat final state, maka NFA menerima input string tersebut.
- Bila tidak ada final state dalam kumpulan simpul ekspan terakhir, maka NFA menolak input string tersebut.

Dalam kasus contoh ini, final state yaitu q4 ada pada simpul ekspan terakhir (langkah 5), sehingga NFA dengan pola "*a?*"menerima teks input "abc".

Hal lain yang perlu diperhatikan adalah simpul ekspan bisa saja menjadi himpunan kosong (tidak ada simpul yang bisa diekspan) sebelum algoritma selesai membandingkan seluruh karakter. Pada kasus ini, final state tidak akan mungkin ada dalam hasil simpul ekspan terakhir sehingga NFA juga menolak input string yang diberikan.

D. Penerapan NFA pada Pencarian Substring Teks yang Sesuai Pola

Algoritma pattern matching diatas dapat pula diperluas untuk mencari kemunculan substring dari teks yang memenuhi pola. Salah satu cara yang mungkin yaitu dengan mengenumerasi semua substring pada suatu teks, lalu membandingkannya dengan pola menggunakan NFA diatas. Namun hal ini tentunya memakan waktu yang jauh lebih lama daripada membandingkan satu string input saja, dimana mengenumerasi seluruh substring pada teks memerlukan kompleksitas waktu sebesar $O(n^3)$. Oleh karena itu, penulis menambahkan batasan pencarian substring yang diimplementasikannya sebagai berikut:

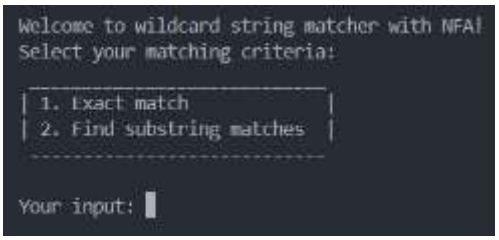
- Tidak ada solusi substring yang beririsan.
  Contoh: Untuk teks “aaa” dan pola “a*”, hasil solusi substringnya hanya {“a”, “a”, “a”} dan bukan {“a”, “a”, “a”, “aa”, “aa”, “aaa”}.
- Program selalu mencari substring terpendek yang memenuhi pola (kecuali jika pola hanya berisi karakter wildcard).
  Contoh: Untuk teks “aaaabaa” dan pola “a*a”, hasil solusi substringnya adalah {“aa”, “aa”, “aa”} dan bukan {“aaaa”, “aa”}.

Berdasarkan batasan-batasan tersebut, penulis mengimplementasikan algoritma pencarian substring pada teks sebagai berikut:

1. Tentukan karakter pertama dan terakhir pada pola yang bukan wildcard. Misalnya, untuk pola “?*sb*c?”, karakter non-wildcard pertama adalah “s” dan karakter non-wildcard terakhir adalah “c”. Bila semua karakter pada pola merupakan wildcard, maka seluruh teks dianggap menjadi solusi substring sehingga tidak perlu dilakukan pencarian.
2. Hitung masing-masing jumlah karakter wildcard “?” yang muncul sebelum karakter non-wildcard pertama (misalnya k_1) dan setelah karakter non-wildcard terakhir pola (misalnya k_2).
3. Cari indeks karakter pertama pada teks [0..*n*-1] yang cocok dengan karakter non-wildcard pertama pada pola, misalkan ditemukan pada indeks *i*. Selanjutnya, cari karakter pertama pada teks [*i*..*n-1*] yang cocok dengan karakter non-wildcard terakhir pada pola, misalkan ditemukan pada indeks *j*.
4. Cocokkan substring teks [$i–k_1$ … $j+k_2$] dengan pola menggunakan NFA diatas.
   - Bila substring cocok, tambahkan substring pada himpunan solusi lalu ulangi langkah 3 untuk teks pada posisi [$j+k_2+1$ … *n*-1].
   - Bila substring tidak cocok, cari karakter selanjutnya pada teks [*i*..*n-1*] yang cocok dengan karakter non-wildcard terakhir pada pola, Bila pencarian karakter selanjutnya sudah sampai akhir teks dan tidak ada substring yang cocok pada teks [*i*..*n-1*], ulangi langkah 3 untuk teks pada posisi [*i*+1 .. *n*-1].
5. Pencarian dilakukan hingga seluruh teks selesai diiterasi.

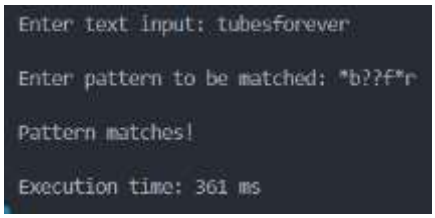
IV. HASIL PENGUJIAN

Setelah mengimplementasikan rancangan algoritma diatas dalam bentuk program C++, dilakukan pengujian dengan beberapa data uji. Berikut merupakan tampilan utama program:

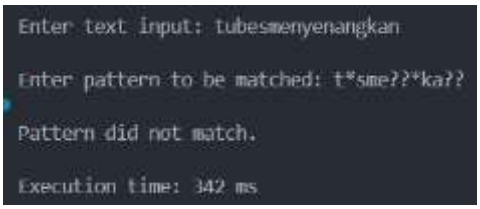


Gambar 4.1. Tampilan awal program
Sumber: Dokumen Penulis

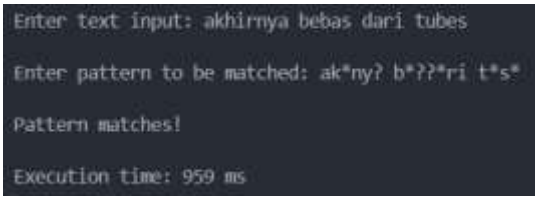
A. Exact Matching



Gambar 4.2. Hasil Data Uji 1 untuk Exact Matching
Sumber: Dokumen Penulis



Gambar 4.3. Hasil Data Uji 2 untuk Exact Matching
Sumber: Dokumen Penulis

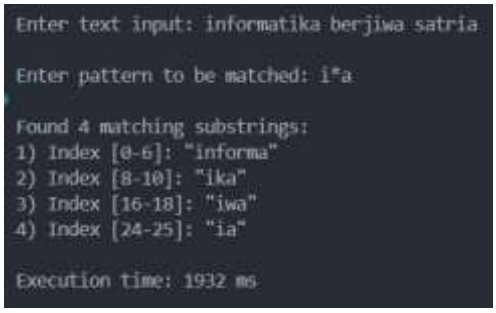


Gambar 4.4. Hasil Data Uji 3 untuk Exact Matching
Sumber: Dokumen Penulis

Panjang Teks	Panjang Pola	Jumlah Wildcard	Match/ Tidak	Waktu Eksekusi
12	7	4	Ya	361 ms
17	12	6	Tidak	342 ms
25	19	8	Ya	959 ms

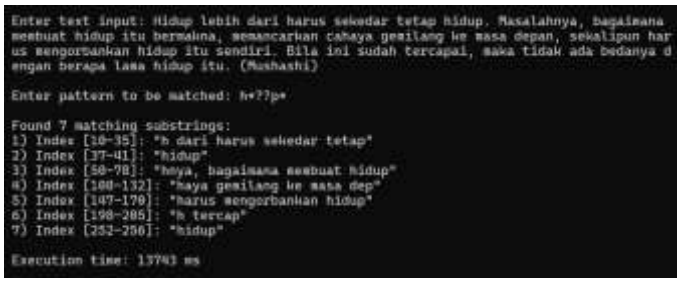
Tabel 4.1. Hasil Data Uji untuk Exact Matching
Sumber: Dokumen Penulis

B. Pencarian Substring yang Sesuai Pola



Gambar 4.5. Hasil Data Uji 1 untuk Pencarian Substring
Sumber: Dokumen Penulis

Gambar 4.6. Hasil Data Uji 2 untuk Pencarian Substring
Sumber: Dokumen Penulis



Gambar 4.7. Hasil Data Uji 3 untuk Pencarian Substring
Sumber: Dokumen Penulis

Panjang Teks	Panjang Pola	Jumlah Wildcard	Jumlah Match	Waktu Eksekusi
26	3	1	4	1932 ms
58	5	3	4	3644 ms
273	6	4	7	13743 ms

Tabel 4.2. Hasil Data Uji untuk Pencarian Substring
Sumber: Dokumen Penulis

V. KESIMPULAN

Pencocokan string dengan wildcard merupakan salah satu permasalahan komputasi yang memerlukan algoritma yang dapat menganalisis beberapa kemungkinan rute sekaligus tanpa terjebak pada satu rute solusi. Salah satu contoh penyelesaian dari permasalahan pencocokan string denga wildcard adalah dengan menggunakan NFA. Hal ini bisa dilakukan dikarenakan kemampuan NFA yang dapat berada dalam beberapa state sekaligus, sehingga tidak terjebak pada satu rute dan dapat melakukan pengecekan pada beberapa rute sekaligus. Berdasarkan hasil pengujian, diperoleh bahwa waktu eksekusi dari NFA bertambah dengan cepat seiring bertambahnya panjang teks, panjang pola, serta jumlah karakter wildcard.

PRANALA GITHUB

Source code program dapat dilihat pada pranala berikut:

https://github.com/IrfanSidiq/nfa_string_matching

PRANALA YOUTUBE

Include link of your video on YouTube in this section.

UCAPAN TERIMA KASIH

Penulis menyampaikan ucapan terima kasih kepada:

1. Tuhan Yang Maha Esa atas berkatnya sehingga penulis dapat menyelesaikan makalah ini dengan baik,
2. Orangtua penulis yang senantiasa selalu mendukung penulis,
3. Bapak Dr. Ir. Rinaldi Munir selaku dosen mata kuliah IF2211 Strategi Algoritma kelas K1 yang telah membimbing penulis dalam menulis jurnal dan mengajarkan segala ilmu yang relevan dengan topik penulis.

Akhir kata, penulis berharap makalah ini dapat bermanfaat tidak hanya bagi penulis namun juga bagi pihak yang membacanya.

REFERENCES

[1] Hopcroft, John E. (2006). "Introduction to Automata Theory, Languages and Computation, 3rd Edition".

[2] "Graph Revisited: Pattern Matching". https://runestone.academy/ns/books/published/pythonds3/Advanced/GraphsRevisited.html (diakses pada 12 Juni 2024)

[3] Clifford, Peter. (2006). "Simple Deterministic Wildcard Matching". https://www.informatika.bg/resources/StringMatchingWithFFT.pdf (diakses pada 12 Juni 2024)

VI. PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Juni 2024

Irfan Sidiq Permana 13522007